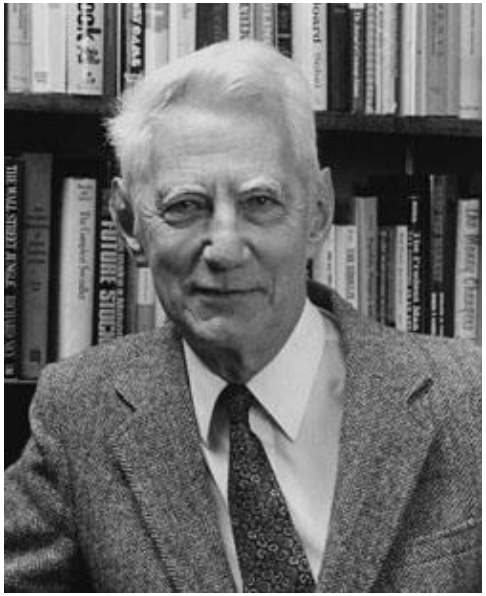
MIS 572:

Introduction to Big Data Analytics

Introduction to Statistical Learning

Yihuang K. Kang



"...we may have knowledge of the past
but cannot control it;
we may control the future
but have no knowledge of it..."

Claude E. Shannon

Introduction

- *Statistical learning* refers to concepts & techniques that help us understand hidden side of phenomena by learning from data. These techniques can generally be classified as *supervised* and *unsupervised* learning.

- The *supervised learning* involves building a model/algorithm that predicts an output based on one or multiple inputs. The "supervised" here is a concept of model learning from teachers (outcomes) until the model would best describe the relationships between inputs and outputs. The *unsupervised learning*, on the other hand, consider there are inputs but no supervising output. Instead, we are interested in learning relationships or structures from the data.

- There are other types of learnings, such as *reinforcement learning*, which consider learning from data with different graded outputs (reward or penalty, instead of definite answers/outputs). In this unit, however, we only consider techniques that cope with common supervised and unsupervised learning problems.

Introduction(cont.)

- We have been doing the statistical learning with R (e.g. fitting general linear models with *lm*()). In this unit, we will be focusing more on details of these techniques that cope with *regression* and *classification* problems, as well as those that *extract features* and *discover structures* within data. As always, we will "Think Big", and also discuss how to parallelize model learning processes and distribute the computations over a cluster of machines.

- Let's begin with thinking "what is *learning*?". And "what is *a learning problem?"*. Previously, we have discussed the *Model Thinking*. We understand that the goal of the learning is to be able to reason (predict) observations that we haven't seen before. And the quality of the learning could be determined by how close our prediction is to the true value, which is often represented as error functions or measures in statistical learning.

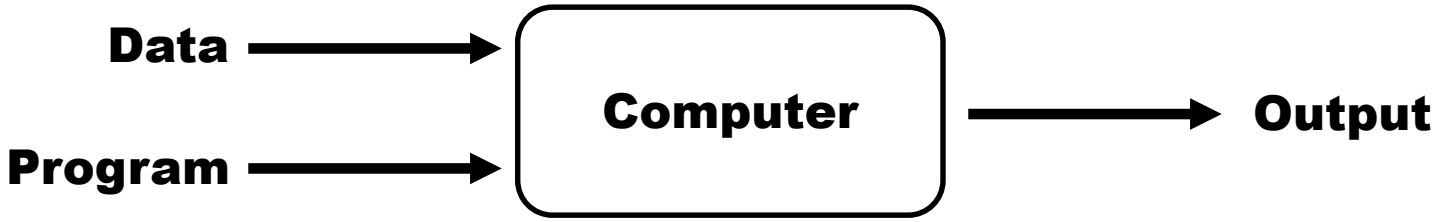
Introduction(cont.)

- Consider the way we learned to recognize a cat (if you could recall) . We actually didn't learn the concepts of a cat solely by its definitions. Instead, we learn "cat" (or in Chinese 貓, whatever you call it does not matter) by observing the characteristics and behaviors of a cat—*we learn from data (observations)*.

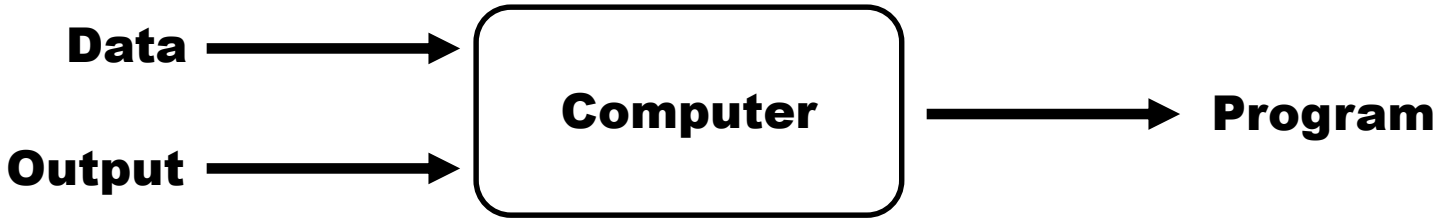
- Statistical Learning is about how to give machines abilities to learn without being definitely programmed—the abilities to recognize complex patterns, take reasonable actions, and even learn how to learn. Unfortunately, our brains are different from machines. The learning mechanisms of human and machines are also not the same (and they don't have to be the same!). However, we can still borrow ideas from human learning. Machines can learn from human teachers (e.g. we tell machines this is right or wrong) and from feedbacks on machines' own actions (e.g. do this instead of that, machines get more rewards).

Programming vs. Learning

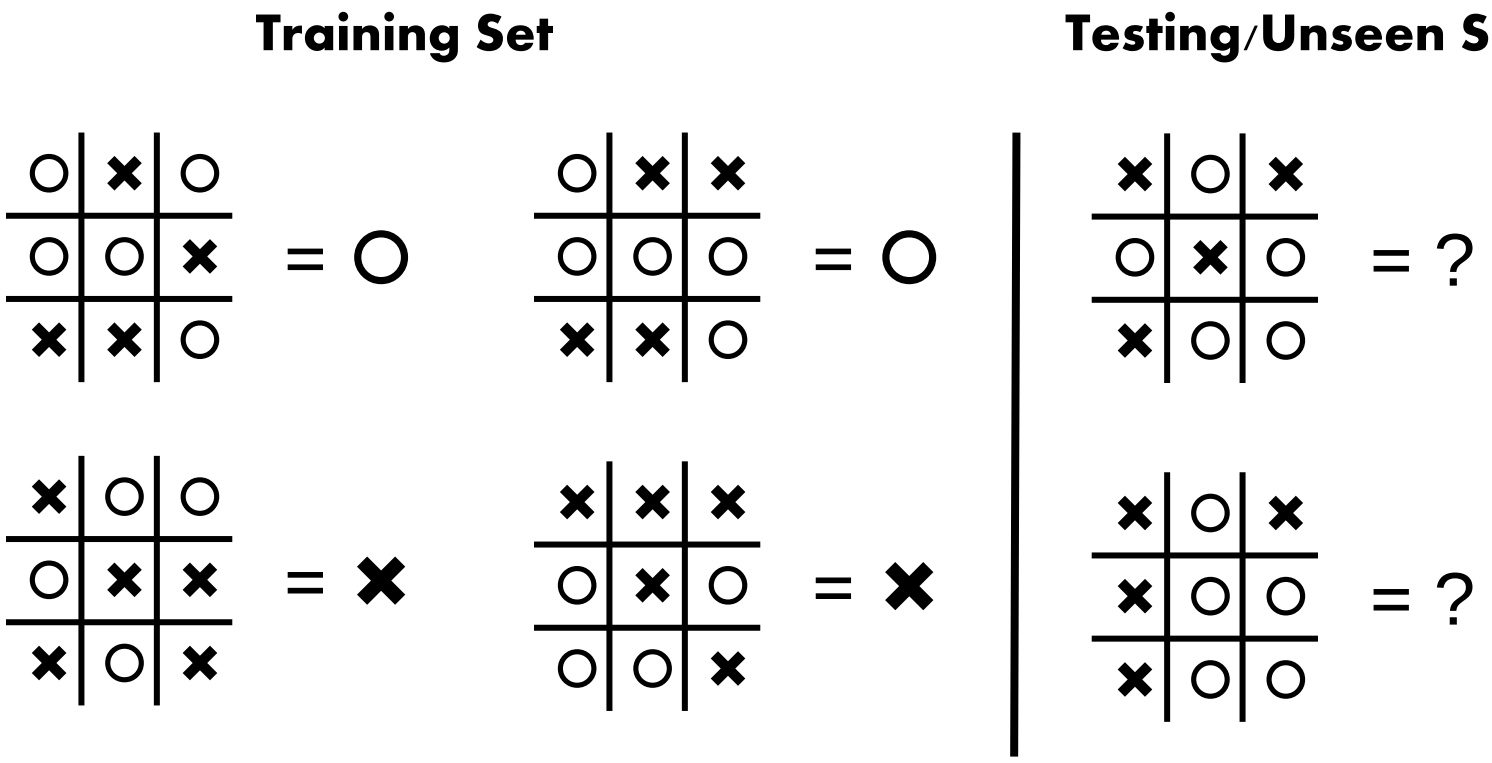
- Computer Programming



- Machine Learning

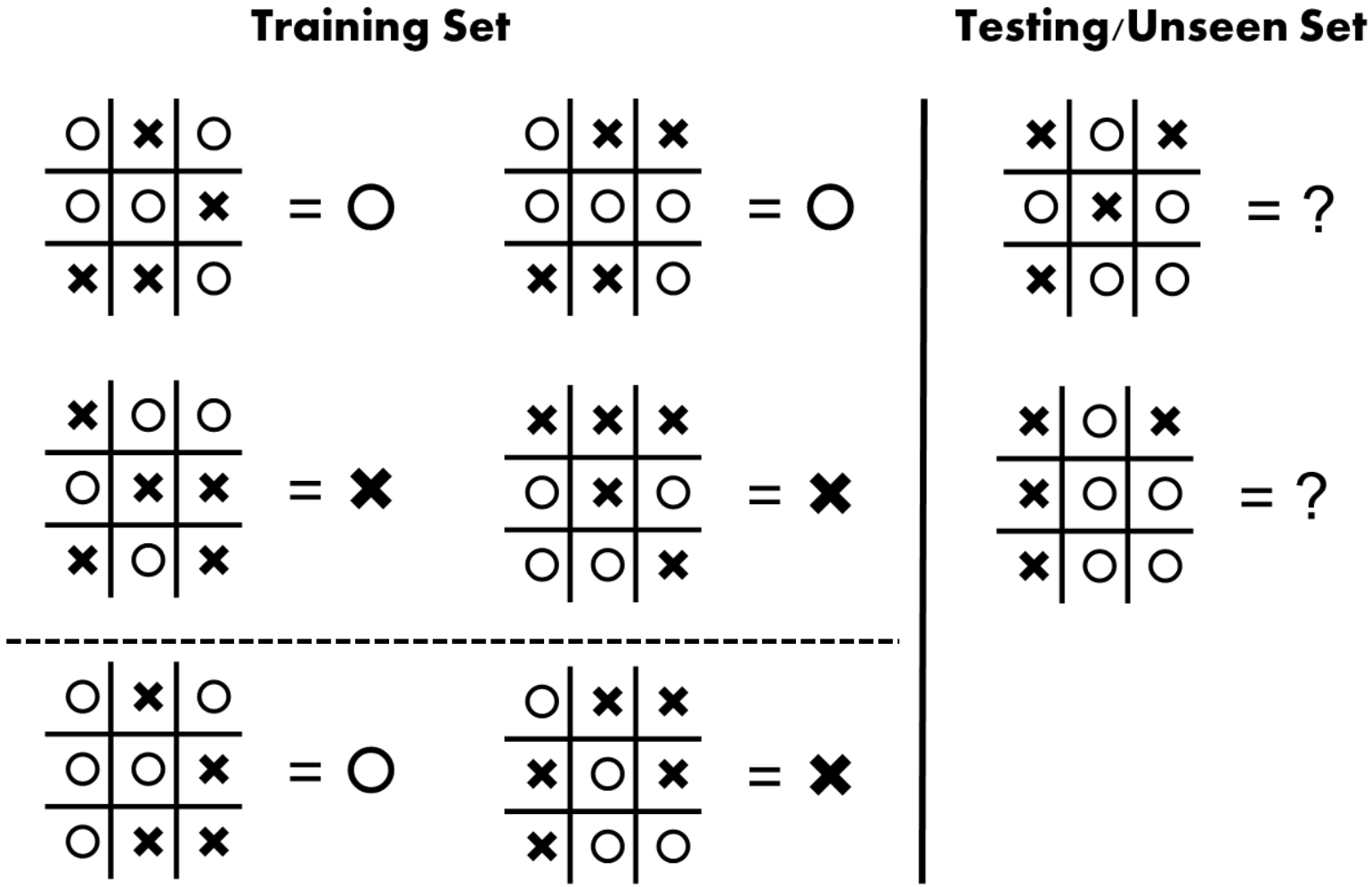


Learning Problem—A Toy Example



- What models (or patterns, rules, functions, …) did you learn from the above figure? Is your "model" different from others?

Learning Problem—A Toy Example(cont.)



Leaner and Learnability

- We may learn different models (classifiers, patterns, functions, ...etc.) from previous figures, as different backgrounds make us different *learners*. We also understand the goal of the learning is to create models that generalize well to the testing/unseen set. Some of you who created models that perform just slightly better than random guessing are, let's call, *weak learners*. On the other hand, those of you whose models perform relative better are *strong learners*. Actually, it doesn't matter whether we are weak or strong learners, as both can see this particular problem from different angles. And therefore, a committee of any types of learners (a big, aggregated, *meta learner*) can create *ensembles of models* that usually outperform models generated by a single learner, which also implies that, interestingly, if we can learn "OK" (weak) models, then we may learn "GOOD" (strong) ensemble of models.

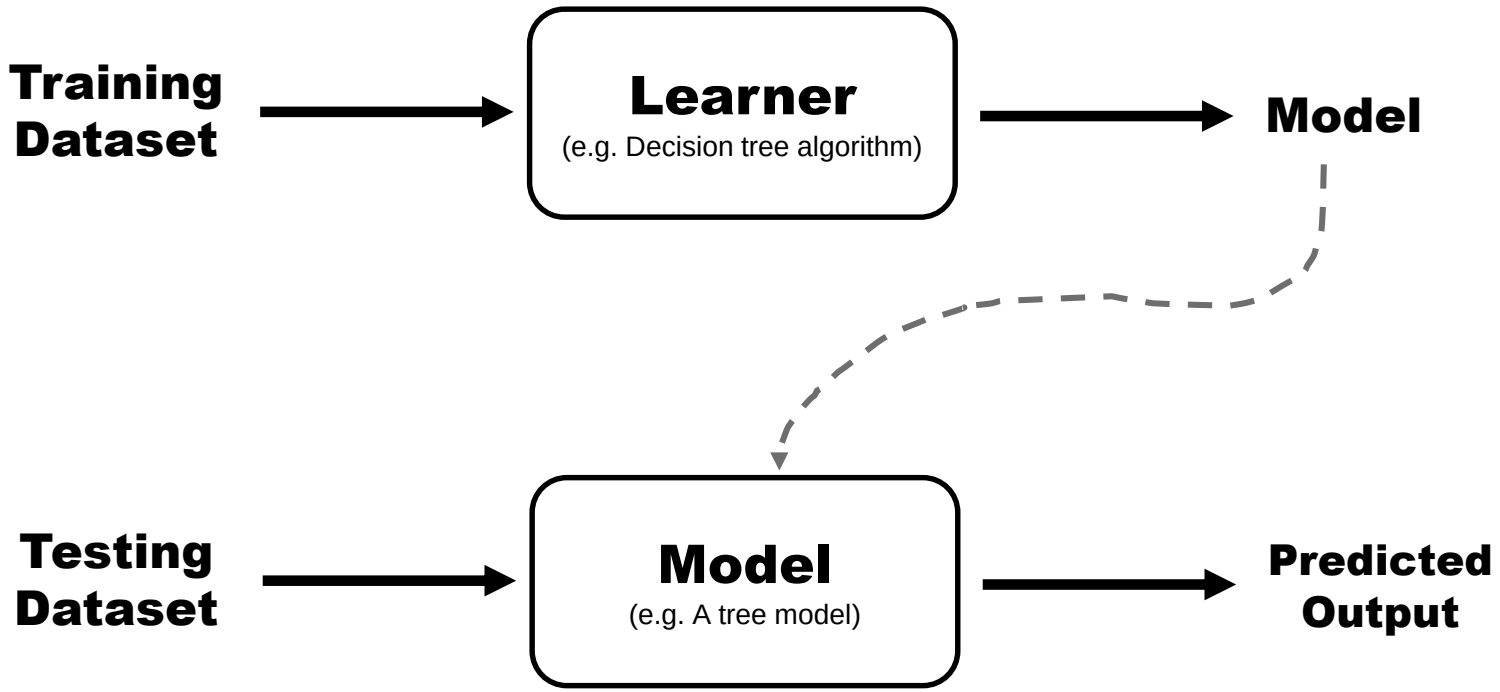
- Let's consider another concept about the learning. We here define a problem *learnable* if we can learn models with reasonable resources (e.g. time and samples). So, is previous visual problem learnable? And, is your model better than others? Some of you may say there is no model or solution to this problem, as your models always result in some misclassifications. Indeed, no model is perfect, but it doesn't mean "no model".

Leaner and Learnability(cont.)

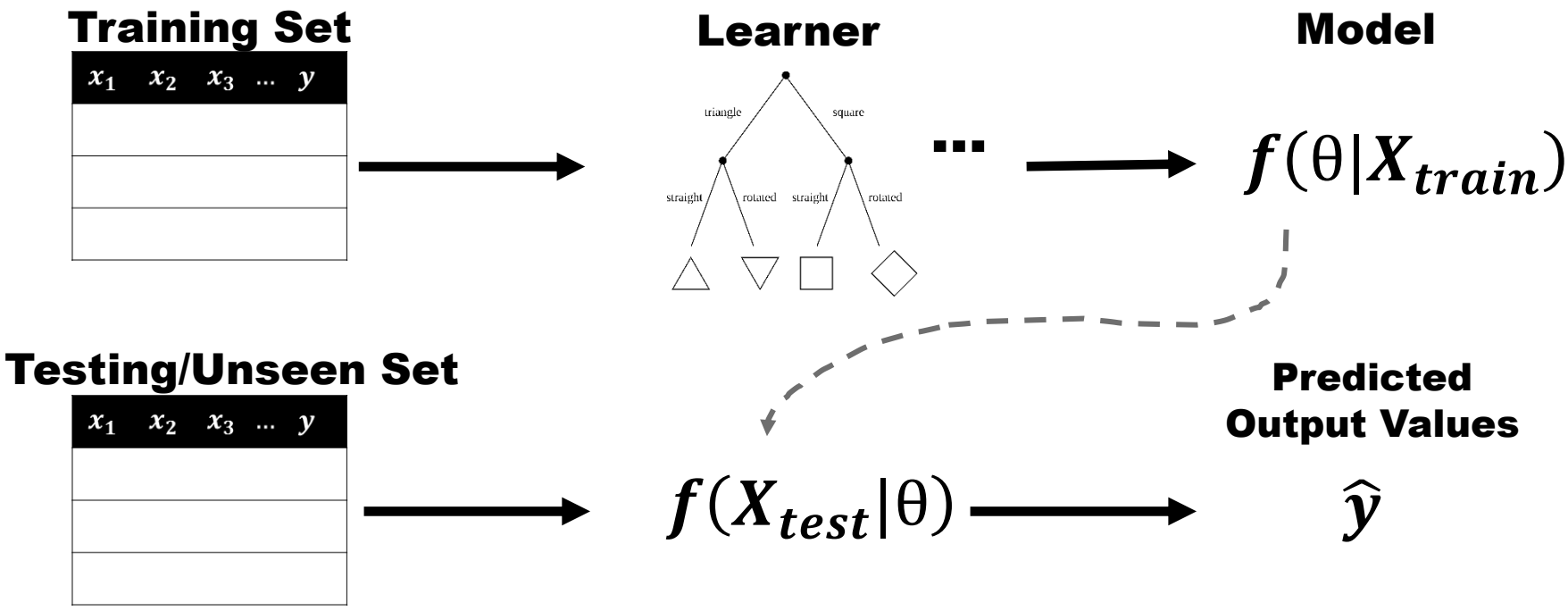
- Consider a real-world example that we prepare for a final exam. Let's say some of you have "low bias" against the contents of midterm, homework, and quiz (training set). And you believe that the final (testing/unseen set) should be similar to the training. So, you tend to spend too much time working on these contents. On the other hand, some of you believe the final should be significantly different from the training and thus tend to just skim the training. Most likely, both types of learners would fail in the final.

- So, how to succeed in the learning problem of the final? You may just study more and work really hard (more data), and/or, talk to your classmates and ask for their opinions (aggregate more models). I do not have an answer to it, but I'm sure that you would succeed in dealing with statistical learning problems with help of this class ☺. The first trick I can tell you is to embrace *the probabilistic nature of the outcome in your data and the errors of your models*.

Learner vs. Model



What Is "Learning"?



Error = Differences between y and $\hat{y}$

What Is "Learning"? (cont.)

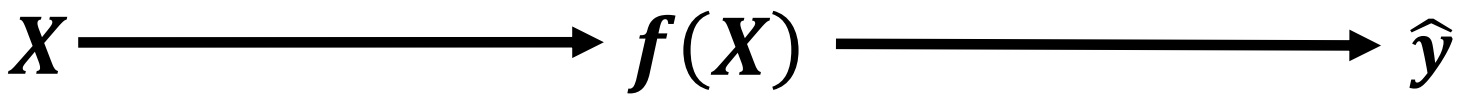
- There exists a complex function $\boldsymbol{f}(\boldsymbol{x}|\boldsymbol{\theta})$ that minimizes the differences between actual values $\boldsymbol{y}$ and predicted/estimated values $\hat{\boldsymbol{y}}$.

$$\boldsymbol{X} \longrightarrow \boldsymbol{f}(\boldsymbol{X}|\theta) \longrightarrow \hat{\boldsymbol{y}}$$

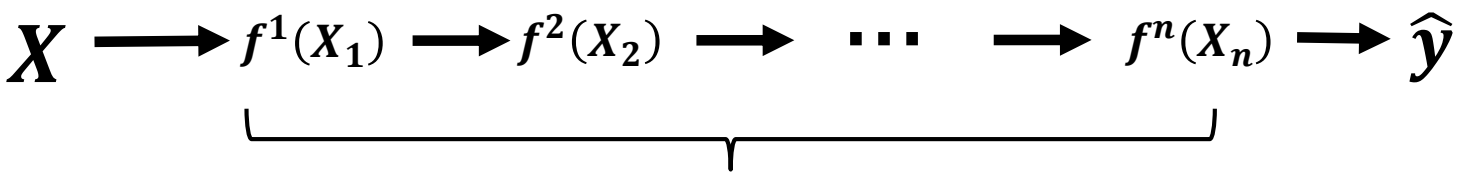
- Learning is about how to find this "$\boldsymbol{f}(\boldsymbol{x}|\boldsymbol{\theta})$"!

What Is "Learning"? (cont.)

What we would like to do…



What we're actually doing…



Manually—***Feature Engineering***

Automatically—***Feature Learning***

Learn Ensemble of Models

- To identify and learn the complex function $\boldsymbol{f(X)}$ manually is very difficult. Researchers have found that we learn and combine many models—ensemble of models, either deeper or diverser, may help.

$$\boldsymbol{X} \longrightarrow \boldsymbol{f^A\left(f^1(X), f^2(X), \ldots, f^n(X),\right)} \longrightarrow \boldsymbol{\hat{y}}$$

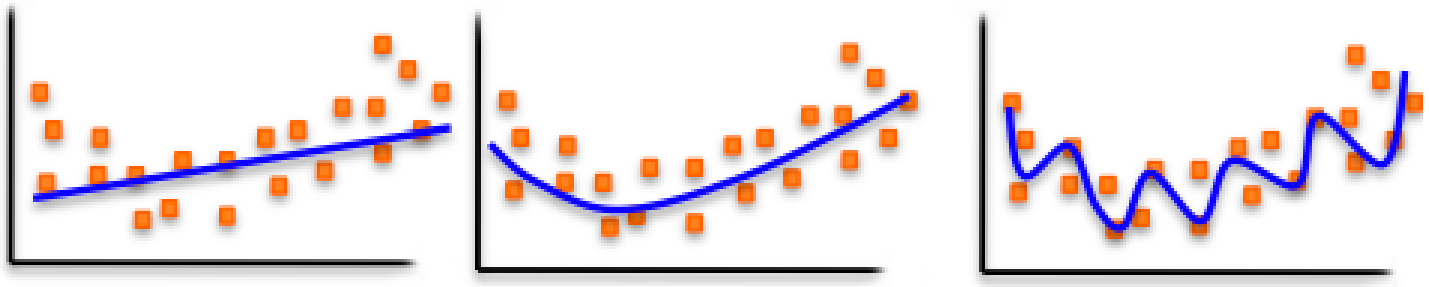
$$\boldsymbol{X} \longrightarrow \boldsymbol{f^4(f^3(f^2(f^1(x))))} \longrightarrow \boldsymbol{\hat{y}}$$

$$\boldsymbol{X} \longrightarrow \boldsymbol{f^A(f^{11}(f^1(x)), (f^{22}(f^2(x)))} \longrightarrow \boldsymbol{\hat{y}}$$

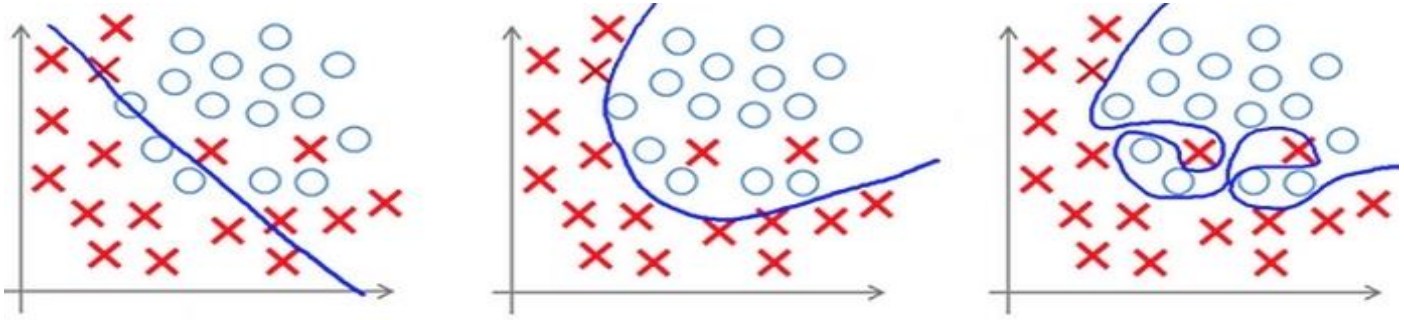
$$\vdots$$

Model Fitting Has Many Faces

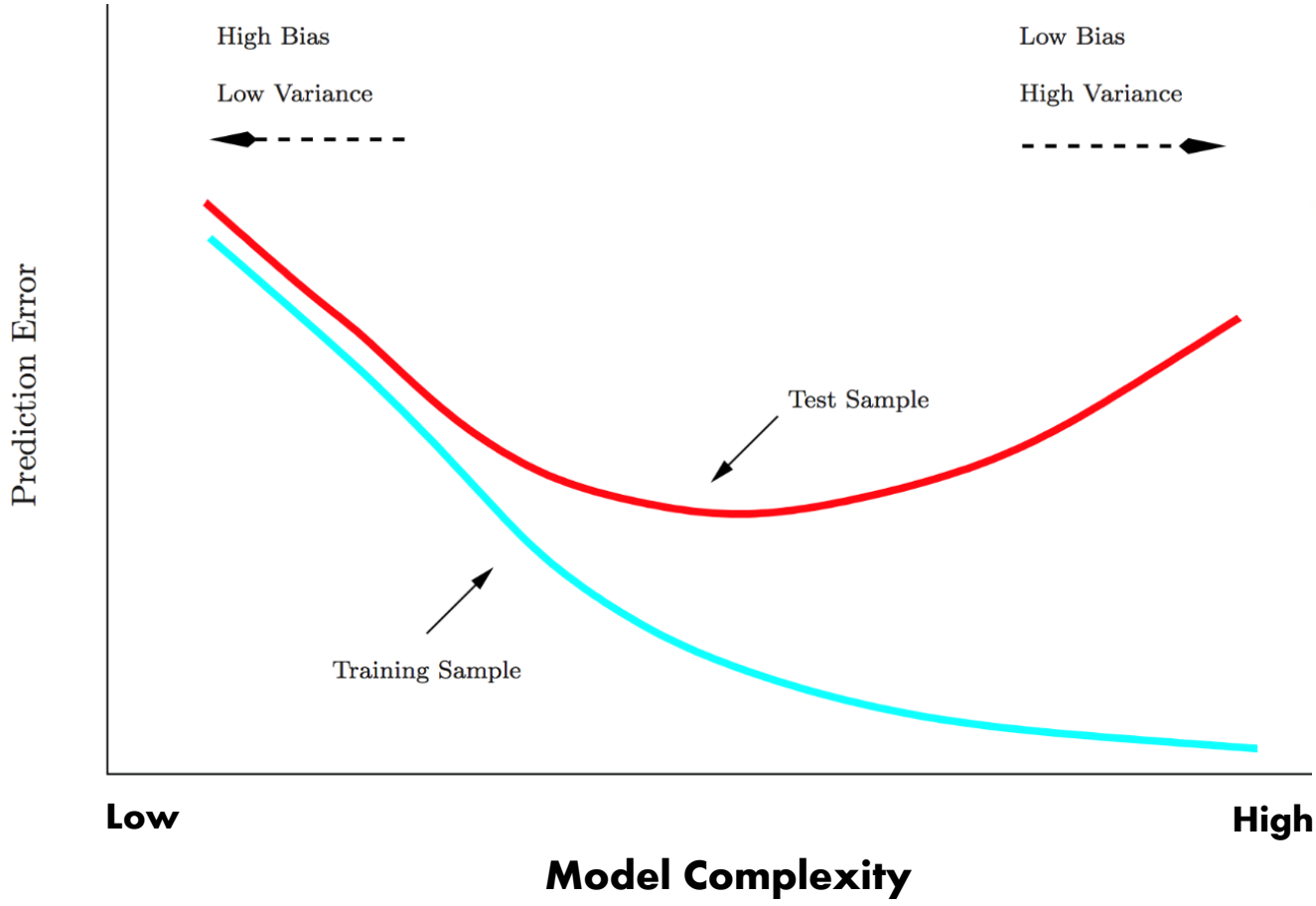
Regression:

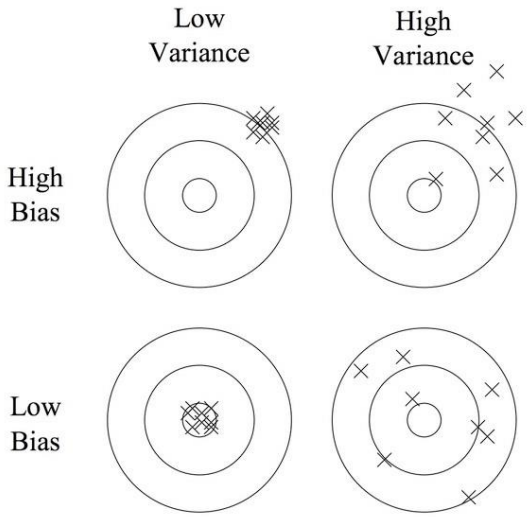


Classification:



Bias-Variance Tradeoff





Think Big!

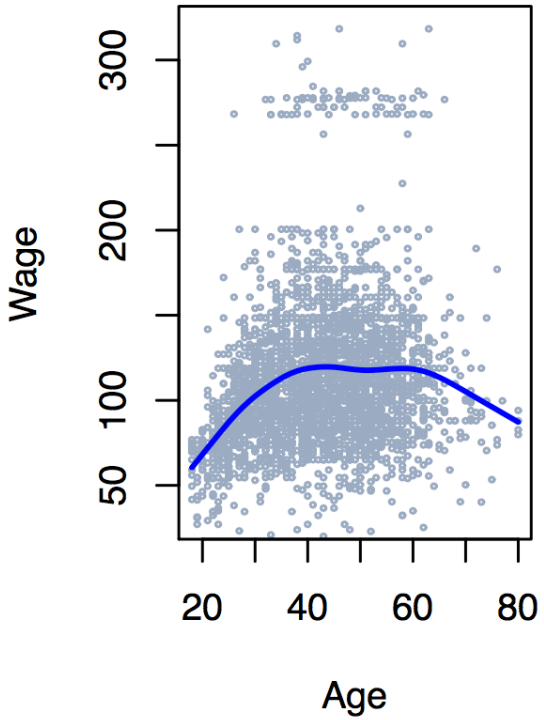
- The analysis of "Big Data" does remove intuitions, but it does not mean you have everything to help you make decisions. Something we just don't know, and there is no such thing as "**N=ALL**".

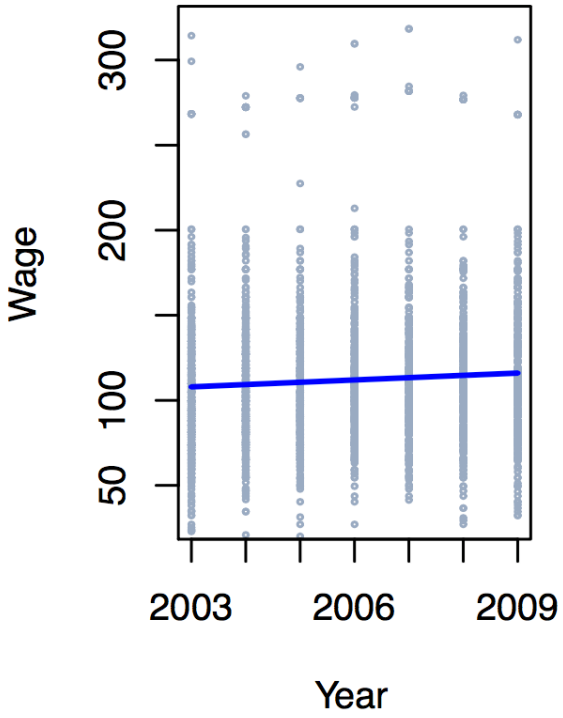
- Very often, your data is big, but data needed to answer your question is small. Your big data problem might actually be a small data problem in disguise, and sampling is much more important than you think.

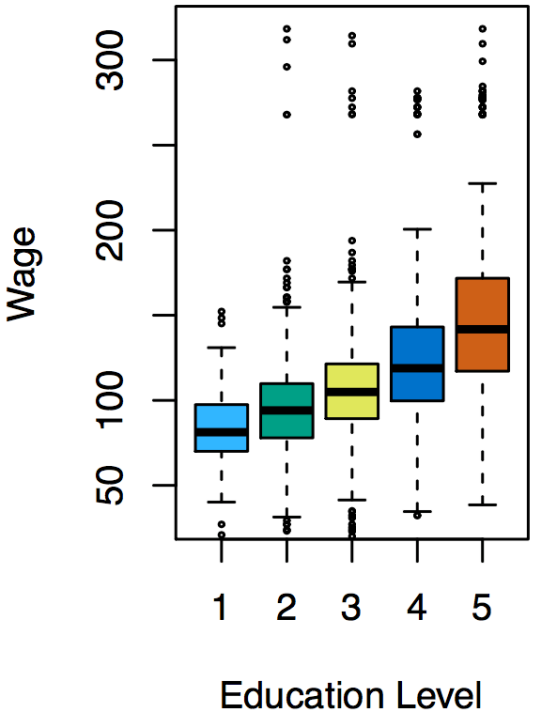
- Diversity of the learners is the key to a good model. Also, a mediocre model with more data usually performs better than a cleverer model with less data.

- *If a problem is proved strongly learnable, then it is weakly learnable* (Schapire 1990). It was later proved that, if we can learn "ok" models from a big dataset, then we can learn "better" ensembles of models.

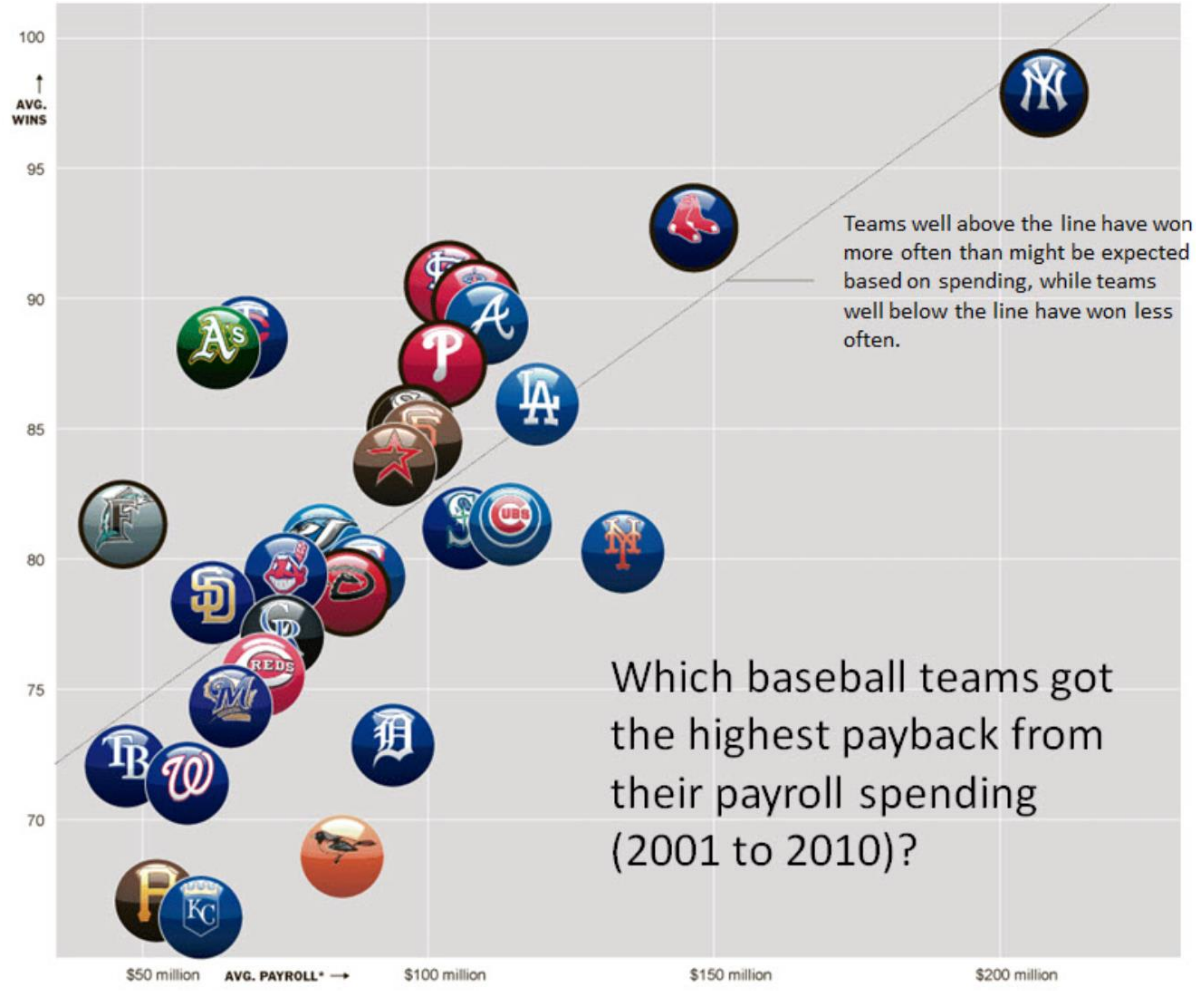
Discovering Relationships

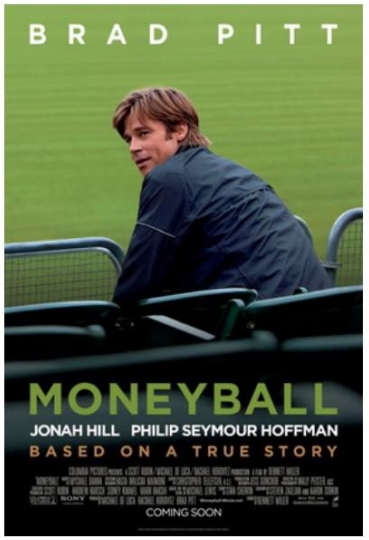




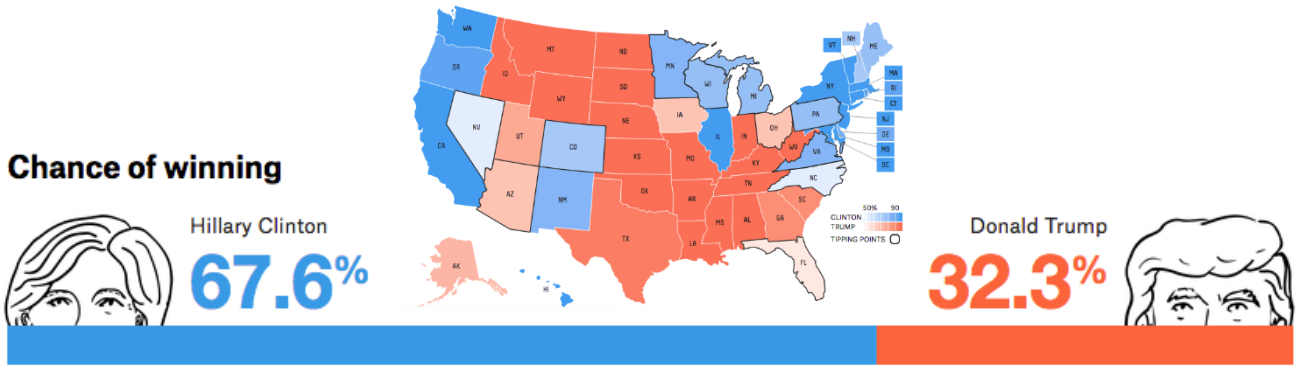


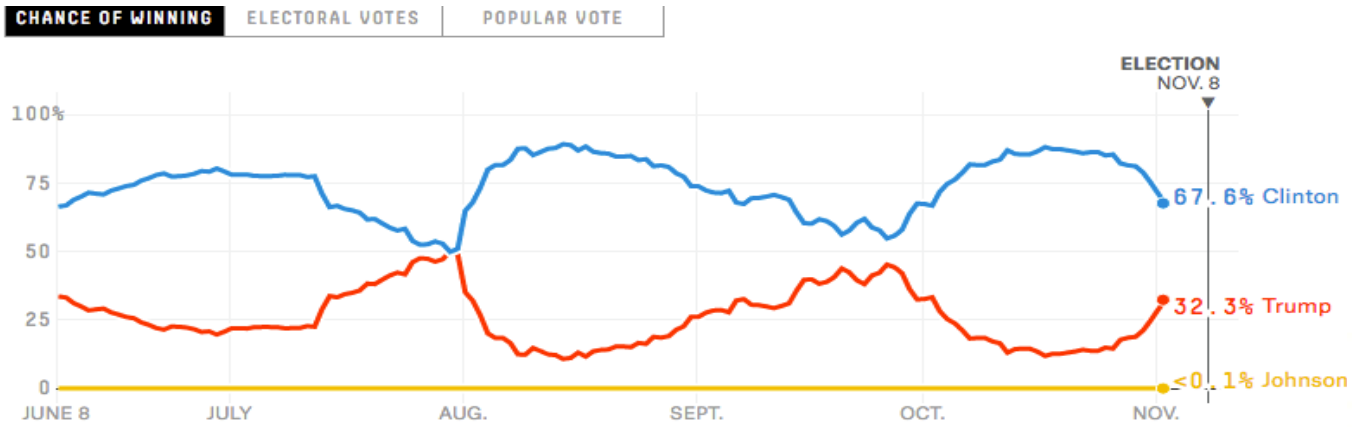
Sabermetrics & Sport Analytics

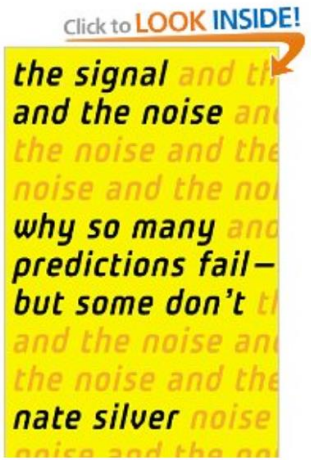




Prediction of Presidential Election

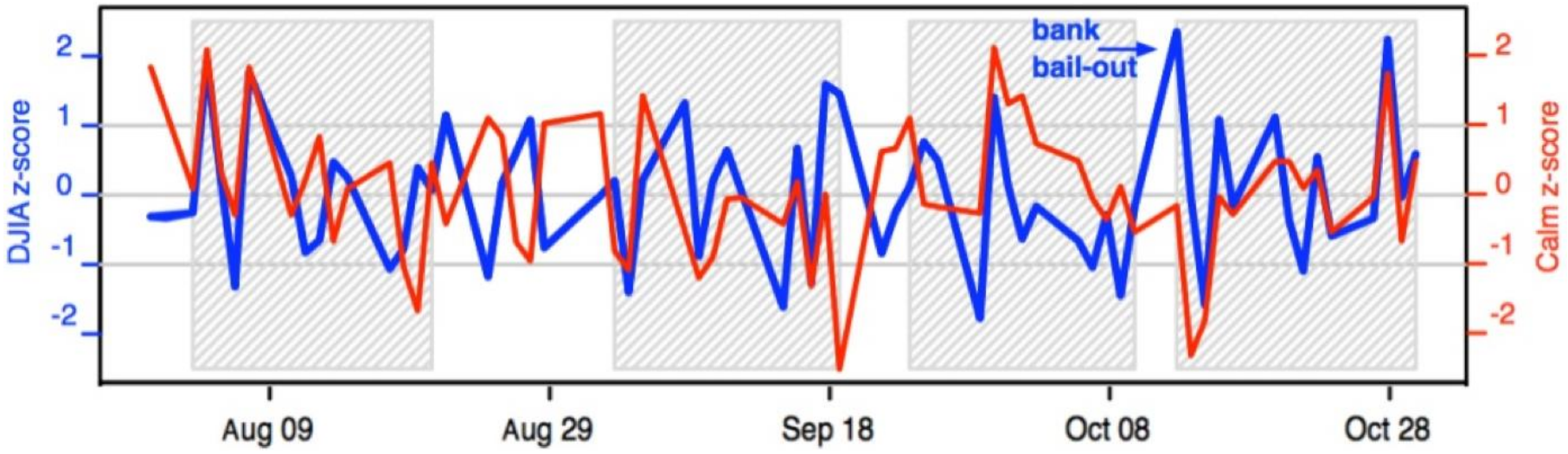






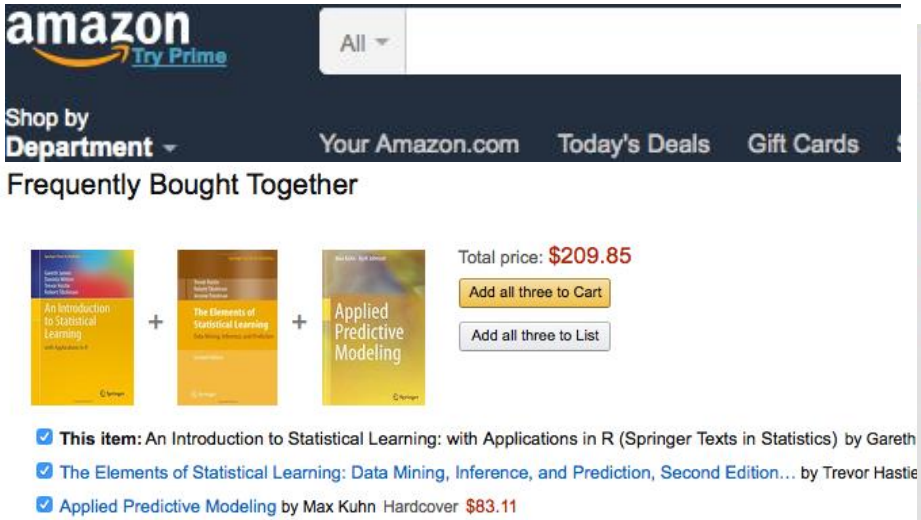
Twitter mood predicts the stock market

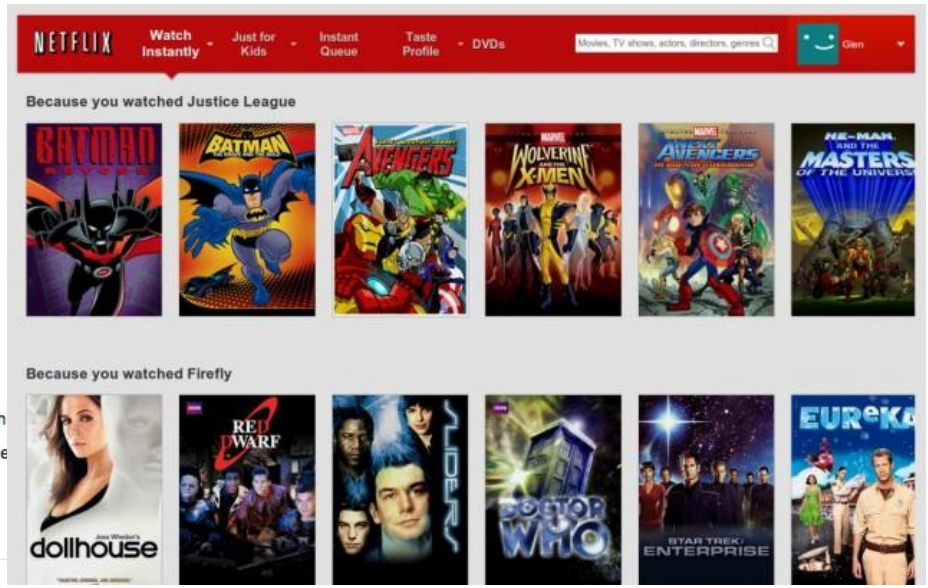
- General mood on Twitter predicts the rise or fall of the daily closing price of the Dow Jones Industrial Average with 86.7% accuracy.



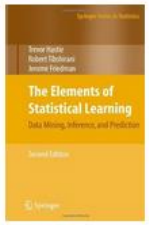
Source: J. Bollen, H. Mao, and X. Zeng, "Twitter mood predicts the stock market," J. Comput. Sci., vol. 2, no. 1, pp. 1–8, 2011.

Building Recommendation Engines





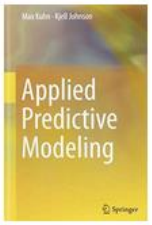
Customers Who Bought This Item Also Bought



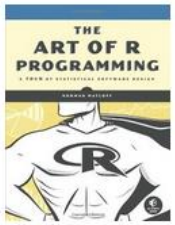
The Elements of Statistical Learning: Data Mining, Inference, and…
Trevor Hastie
61
#1 Best Seller in

Data Science from Scratch: First Principles with Python
Joel Grus
43
#1 Best Seller in Computer Programming…

Learning From Data
› Yaser S. Abu-Mostafa
97
Hardcover

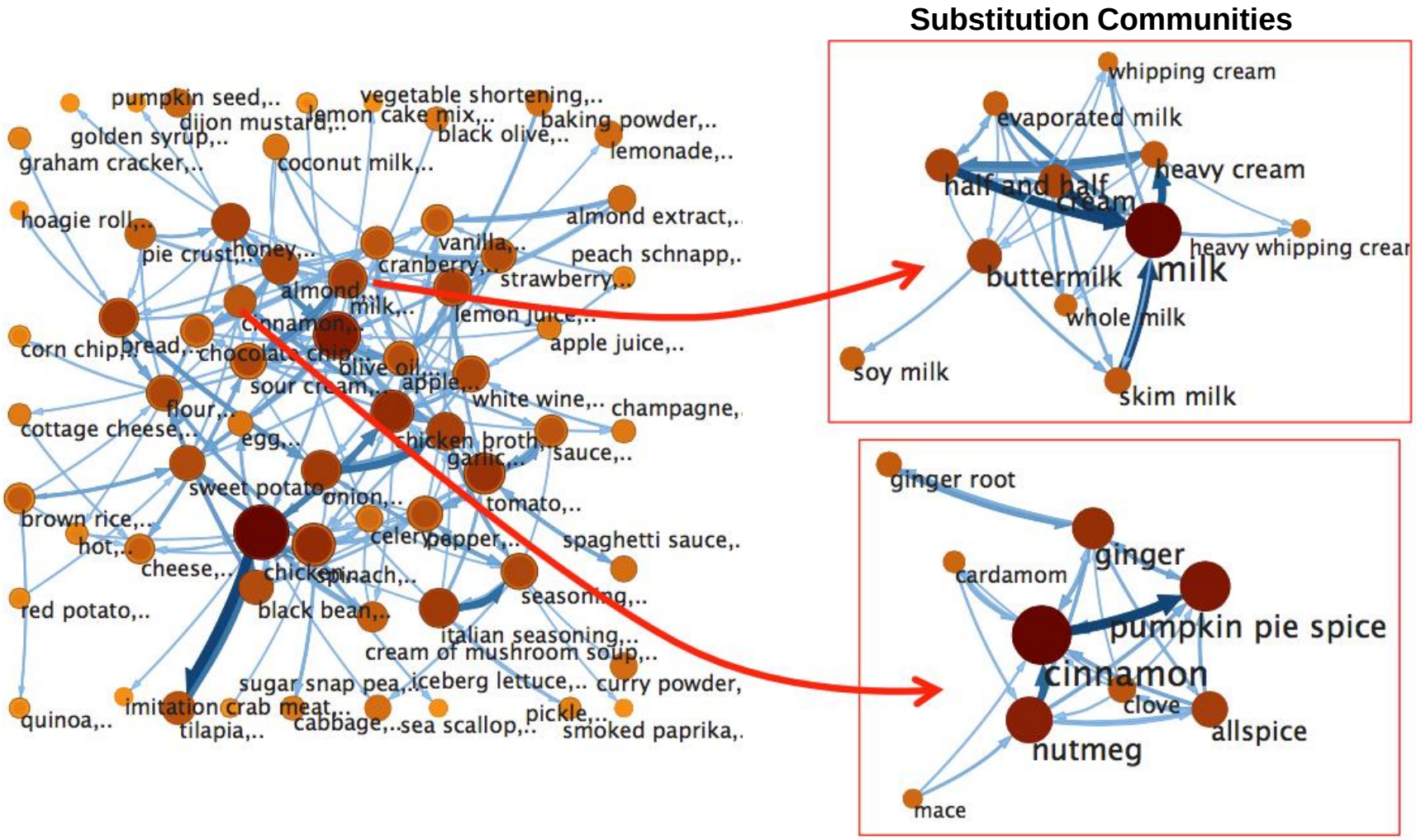


Applied Predictive Modeling
› Max Kuhn
44
#1 Best Seller in Biostatistics



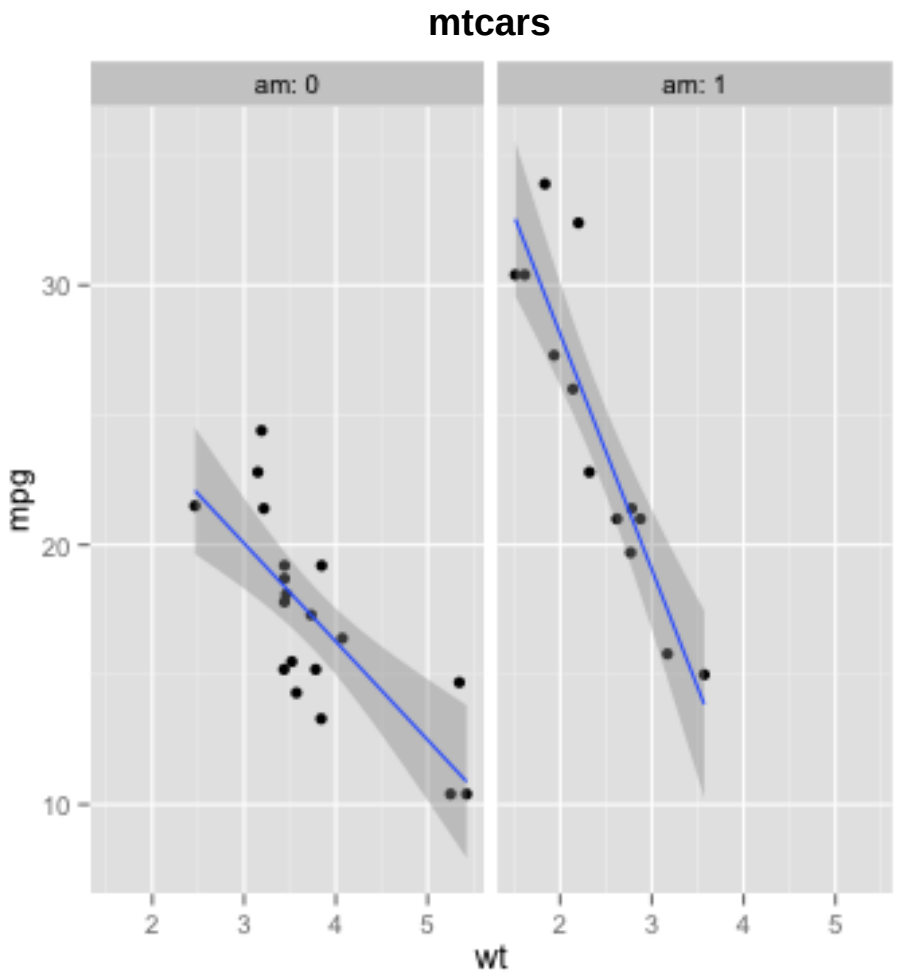
The Art of R Programm
A Tour of Statistical Software Design
Norman Matloff
104
Paperback

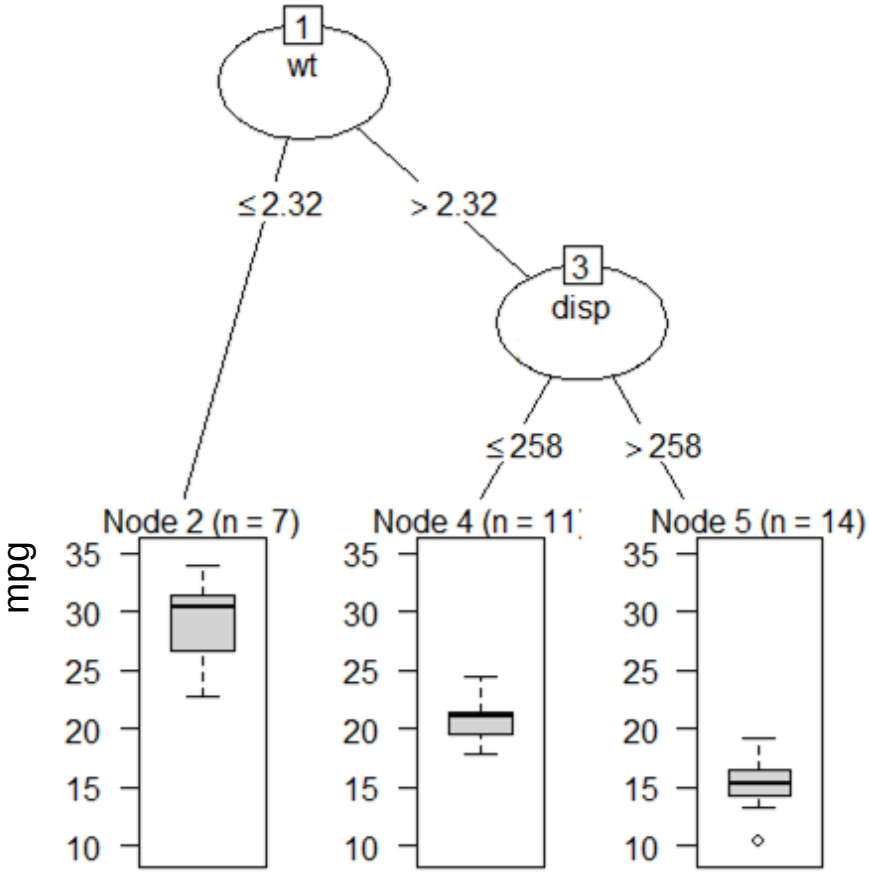
Community Detection in Network



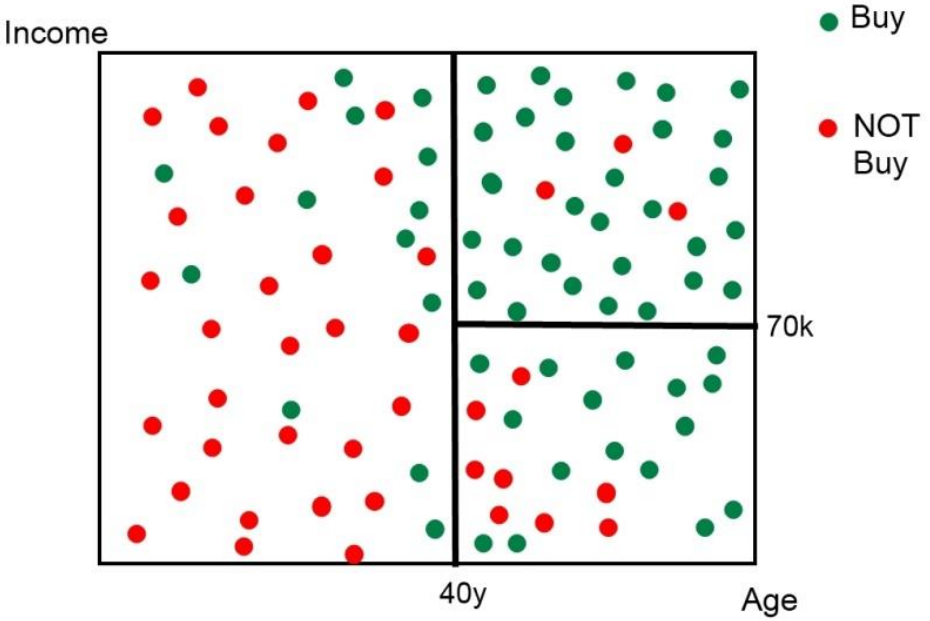
Source: C.-Y. Teng, Y.-R. Lin, and L. A. Adamic, "Recipe recommendation using ingredient networks," in Proceedings of the 4th Annual ACM Web Science Conference, 2012, pp. 298–307.

Regression


mtcars
am: 0
am: 1
mpg
30
20
10
2
3
4
5
wt


1
wt
≤ 2.32
> 2.32
3
disp
≤ 258
> 258
Node 2 (n = 7)
Node 4 (n = 11
Node 5 (n = 14)
mpg
35
30
25
20
15
10

Classification

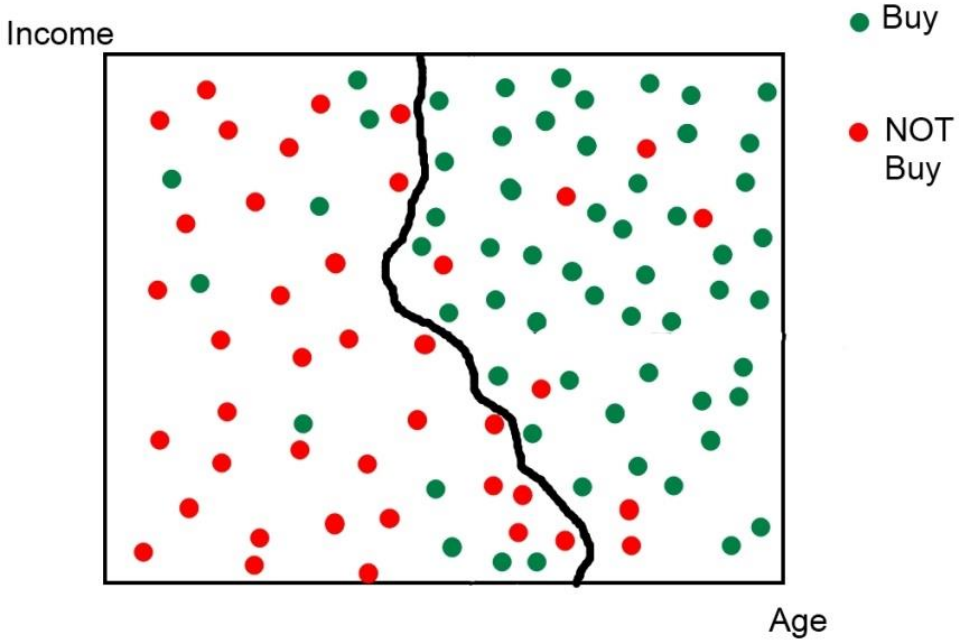


Misclassification Rate = 15% ??

Easy to understand the model by those "rules":

1: IF Age>= 40y AND Income >= 70k
THEN Buy = True
2: IF Age>= 40y AND Income < 70k
THEN Buy = True
3: IF Age < 40y
THEN Buy = False

Simple!!

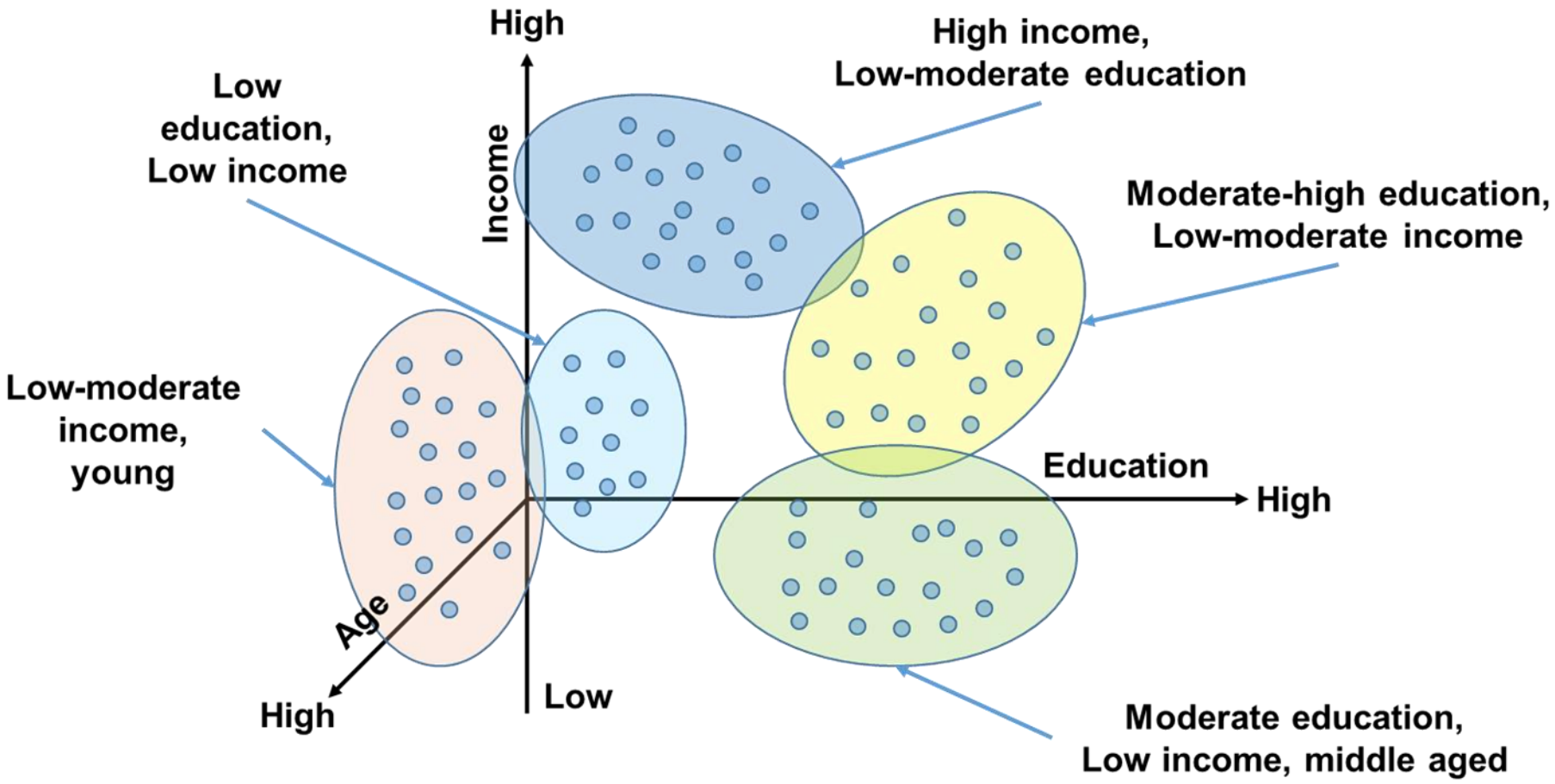


Misclassification Rate = 10% ?? (usually lower)

Rules ? What is that?

Obscure!!
Rule "extraction"?!

Clustering


High
Income
Low
High income,
Low-moderate education
Low
education,
Low income
Moderate-high education,
Low-moderate income
Low-moderate
income,
young
Education
High
Age
High
Moderate education,
Low income, middle aged

Top 10 Data Mining Algorithms

Supervised Learning	Unsupervised Learning
C4.5 Decision Tree	*k*-means
Support Vector Machines	Apriori
AdaBoost	Expectation-Maximization
k-nearest neighbor	PageRank
Naïve Bayes	
Classification and Regression Tree	

Top 10 Data Mining "Mistakes"

...(data mining) success is improved by learning from experience; especially our mistakes. So go out and make mistakes early!

John Elder

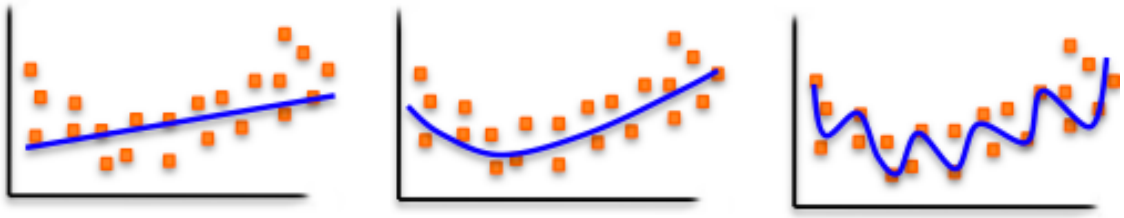
Top 10 Data Mining "Mistakes" (cont.)

0) Lack proper data

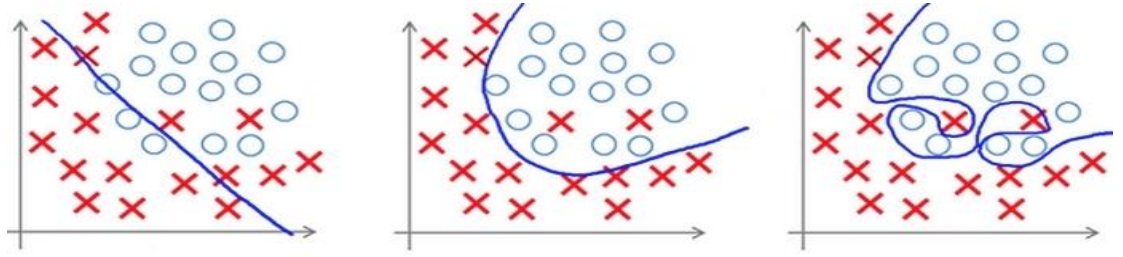
We sometimes just have no important data, labeled cases, or observations of interest. We're not considering this a "mistake" but it often results in the failure of a data mining project.

1) Focus on training (overfitting and underfitting)

Regression



Classification



Top 10 Data Mining "Mistakes" (cont.)

2) Rely on one technique

"All you got is a hammer?" For many reasons, most researchers and practitioners focus too narrowly on a few types of modeling techniques.

3) Ask the wrong question

It's also essential to have an appropriate model goal. Ask yourself, "what is the goal of the data analytics project?

4) Listen (only) to the data

"*Data is just a quantitative, pale echo of the events of our society*" . Don't tune out received wisdom while letting the data speak.

5) Accept (data) leaks from the future

Suspect "perfect" models. You may have mixed training and testing (or future) data!

Top 10 Data Mining "Mistakes" (cont.)

6) Discount pesky cases

Unusual data (e.g. outliers, "special" cases, etc..) may greatly affect your analysis results, but they may also be the new discovery. Don't just remove or skip them. "To keep or not to keep, that's the question."

7) Extrapolate

Models are unreliable outside the bounds of any known data. Admit it. Our models just cannot be generalized that well.

8) Answer every inquiry

Models are always NOT perfect. If something we really don't know, just say "I don't know".

9) Sample casually

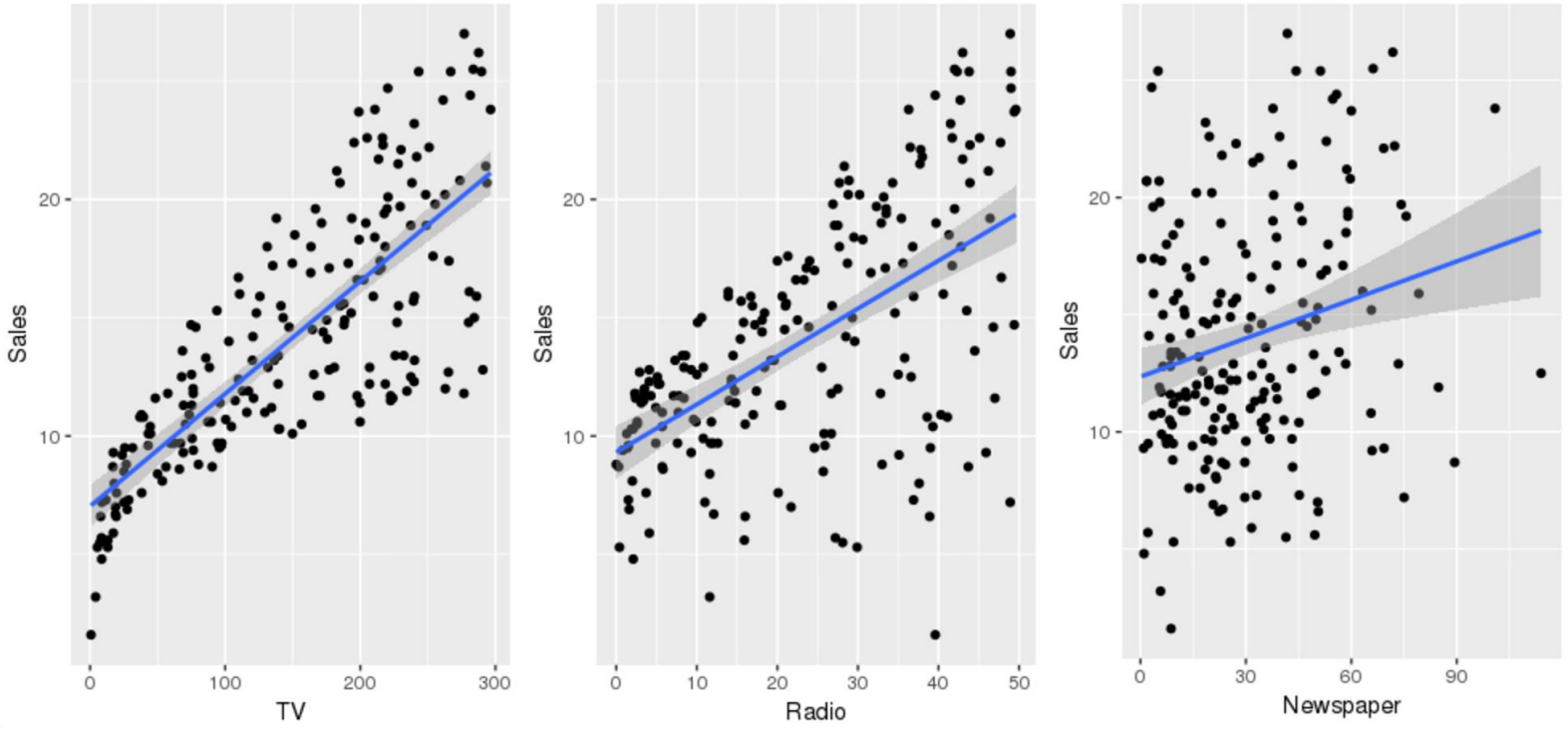
Randomize your data well. A good strategy is to “shake before baking"!

10) Believe "the best model"

"*...all models are wrong, but some are useful*.."

Statistical Learning—An Example

- Consider a simple example that we have a dataset about sales and advertisement budget for a product in 200 different markets. We're trying to predict *Sales* (in 1,000 units) with budget(in 1,000 dollars) already spent in 3 different media—*TV*, *Radio*, and *Newspaper*. We can separately plot the data points to get some rough idea about their relationships.



Statistical Learning—An Example (cont.)

- Note that, in the figure, we consider *bivariate* linear regression models used to describe a kind of linear relationship. Such relationship among predictors and outcome can be described by models (or algorithms) with rules, steps, or functions—*parameters (weights)* with values that help define models. Back in the day in Statistics, the process of finding parameter values of a model that best fits a dataset is called *parameter estimation*.

- Take previous linear models as an example. The parameter values(βs) of below regression model, i.e. *Sales = f(TV)*, can be estimated by techniques such as *Maximum Likelihood Estimation, Least-Squares Estimation, and (Stochastic) Gradient Descent*.

$$\widehat{Sales} = \beta_0 + \beta_1 * TV$$

Statistical Learning—An Example (cont.)

```
# Get advertising data
advertise = read.csv("http://www-bcf.usc.edu/~gareth/ISL/Advertising.csv")
# log-likelihood function
LL = function(beta0, beta1){
  # Residuals
  resid = advertise$Sales - (advertise$TV * beta1) - beta0
  # Likelihood for the residuals
  ll = dnorm(resid, 0, 1);
  # Sum of the log likelihoods for all of the data points
  return(-sum(log(ll)));
}
# Maximum likelihood estimation of parameters (betas)
stats4::mle(LL, start = list(beta0 = 0, beta1 = 0))
...
Coefficients:
     beta0      beta1
7.03262463 0.04753648
# Or, just use lm() to make our life easier.
lm(Sales ~ TV, advertise) # Note that lm() uses QR factorization instead
...
Coefficients:
(Intercept)           TV
    7.03259      0.04754
```

Statistical Learning—An Example (cont.)

- You may wonder, why not just put all variables as predictors into the model. Yes, a model below might better predict the *Sales*.

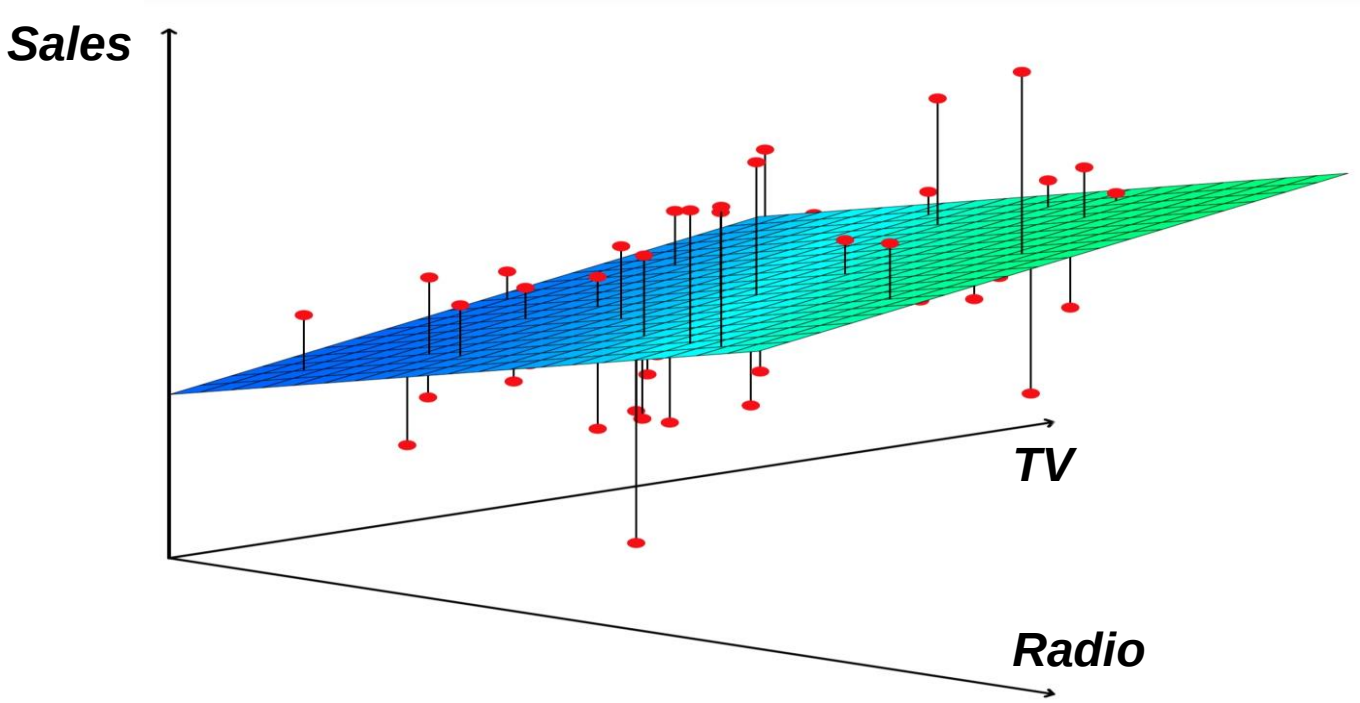
$$Sales = f(TV, Radio, Newspaper)$$

```
ad_fit = lm(Sales ~ TV + Radio + Newspaper, data = advertise);
summary(ad_fit); ad_fit
...
TV           0.045765   0.001395  32.809   <2e-16 ***
Radio        0.188530   0.008611  21.893   <2e-16 ***
Newspaper   -0.001037   0.005871  -0.177     0.86
...
Coefficients:
(Intercept)           TV        Radio    Newspaper
   2.938889     0.045765     0.188530    -0.001037
```

- It seems that *TV* & *Radio* are "good predictors", but we're still not 100% sure whether there is any other *effects* that also contribute to the variance of the *Sales*. By "effects", we mean subset of variables, combinations (new variables) of multiple variables, or any other transformations of variables that might have impact on the *Sales*. Such process of selecting variables relevant to an outcome variable is also known as *feature selection*.

Statistical Learning—An Example (cont.)

- For simplicity, let's consider *TV* and *Radio* only in below figure. We expect that almost all the data points are not on the regression "line" (In this example, it's actually a *hyperplane*!), i.e. all the predicted values almost do not match the actual values of the *Sales*.



Statistical Learning—An Example (cont.)

- The difference between the predicted and actual values of the *Sales* are known as *errors*/*residuals*. The residuals are theoretically expected/assumed to be *approximately normal* and *independently distributed* with mean = 0 and some constant variance, according to assumptions of regression analysis.
- Just like what we did for typical regression analysis, we could do some further analysis on the residuals, such as normality tests, to verify the applicability of our linear model.

```
resid = advertise$Sales - predict(ad_fit, advertise)
# or just enter "residuals(ad_fit)"

# normal Q-Q plot
qqnorm(resid); qqline(resid)
```

Statistical Learning—An Example (cont.)

- In Statistical Learning, however, our goal is slightly different from what we did for the typical statistical analysis. We care more about how the model performs when it is applied to other future/unseen/testing datasets—*accuracy of prediction*, instead of *model interpretability*.
- One of the popular measure used in the performance evaluation of regression models is *Root Mean Square Error* (RMSE), as defined:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(\hat{y} - y)^2}{n}}$$

where $(\hat{y} - y)$ is the *residuals* and n is the number of observations.

```
rmse_lm = function(f, d){
  m = lm(formula = f, data = d,na.action = na.omit);
  return(  round(sqrt(mean(resid(m) ^ 2)),4)) ;
} # RMSE of the training set
```

Statistical Learning—An Example (cont.)

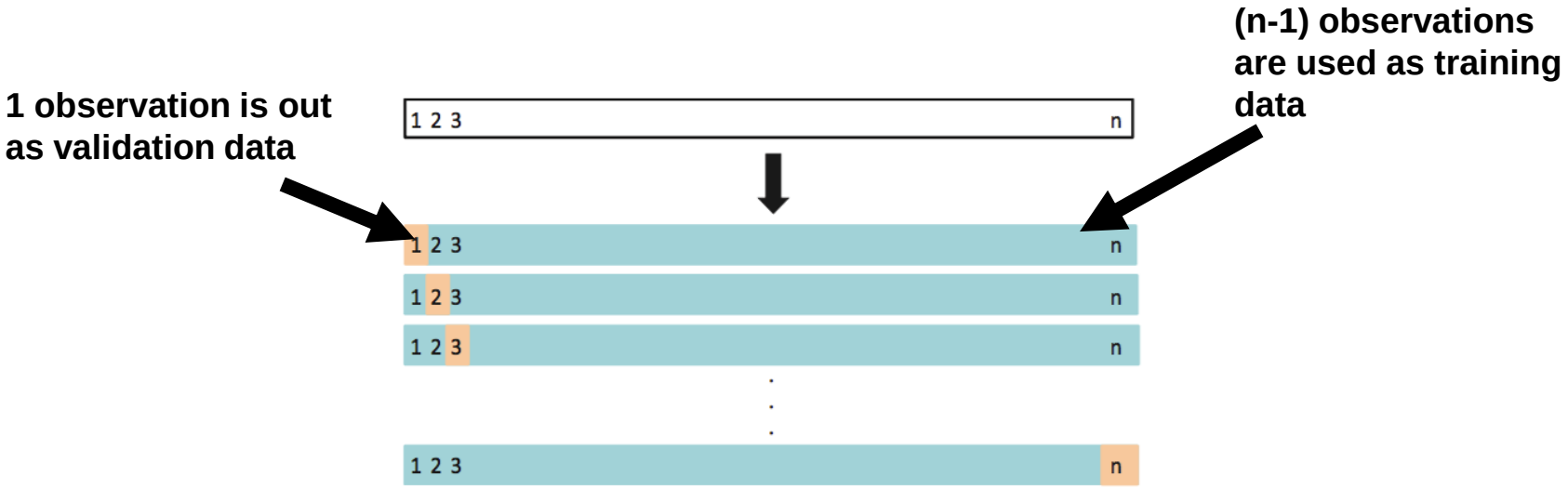
- Lower RMSE usually means better model. However, we should not evaluate our model based on the residuals of the training data, as the model might be characterized too much noise/detail of the training data. And therefore the model may not generalized well to the unseen/testing data.

- Thanks to the modern computers with cheaper computations, we can now use a *resampling* technique called *Cross-Validation* (CV) to get (hopefully) unbiased error measures, such as RMSE for regression and *Misclassification Rate* for classification problems.

- The concept of the CV is to estimate an error (or accuracy) measure (e.g. RMSE) by holding out a subset of training observations (aka validation data) from the model learning process. As the subset is not used in the process, the error (e.g. residual in regression problems) is therefore an approximately unbiased estimate of the testing error.

Leave-One-Out Cross-Validation

- In this unit, we will be discussing two popular CV techniques—*Leave-One-Out Cross-Validation* (LOOCV) and *k*-fold CV (they are basically the same technique!). Let's first introduce LOOCV into our feature selection process. LOOCV repeatedly leave one observation *i* out as validation data and learn a model using training data that contains $n - 1$ observations. Then, the error is estimated later by averaging the error of predicting the *i* observation.



Leave-One-Out Cross-Validation(cont.)

- Yes, no technique is perfect. You may have noticed that, for *n* observations, we need to fit *n* models, which requires significant amount of computations when *n* is big. However, LOOCV does have some advantages. One advantage is that, compared to other validation approaches that randomly split data into training and validation data, LOOCV tends NOT to overestimate the testing error as it eliminates possible bias due to randomness.
- Let's get back to previous *advertise* data. Now, we'd like to evaluate the performance of our linear models using LOOCV with different combinations of variables. Specifically, we need an LOOCV version of RMSE for any *lm()*.

```
LOOCV_lm_rmse = function(f, d){ # f: formuala, d: data
  numOfRec = nrow(d); # Number of records, n
  rmse_vec = rep(0,numOfRec); # n RMSEs
  reponse_var = all.vars(f)[1]; # Get the name of the response/target variable

  for(i in 1:numOfRec){
    m = lm(formula=f, data=d[- i,], na.action = na.omit);
    rmse_vec[i] = (d[[reponse_var]][i]) - predict(m, newdata=d[i,]);
  }
  # Return a string with the LOOCV RMSE
  return( paste("LOOCV RMSE for lm(", format(f),") =",
                round(sqrt(mean(rmse_vec ^ 2)),4)) );
}
```

Leave-One-Out Cross-Validation(cont.)

- The LOOCV version of RMSE is now used to evaluate the model performance. From the below output, we can see that the RMSE is relatively lower when all 3 variables are in the linear model.

```
Map(function(x) LOOCV_lm_rmse(x, advertise),
    c(Sales ~ TV, Sales ~ Radio, Sales ~ Newspaper,
         Sales ~ TV + Radio + Newspaper))

"LOOCV RMSE for lm( Sales ~ TV ) = 3.2774"
"LOOCV RMSE for lm( Sales ~ Radio ) = 4.2995"
"LOOCV RMSE for lm( Sales ~ Newspaper ) = 5.1255"
"LOOCV RMSE for lm( Sales ~ TV + Radio + Newspaper ) = 1.7167"
```

- The RMSEs can also help us select variables useful in predicting the outcome. You might consider enumerating all possible combinations of variables and letting computers to do the feature selection job for you. Notice that, however, when learning models from a big dataset with p variables, there are $2^p - 1$ combinations (exhaustive feature search)—required computation grows exponentially. We may not be able to get a result within a reasonable time!

Introduction to Feature Selection

- Feature selection is a science as well as an art. There is no "the best" selection process/algorithm that can be applied to any datasets. However, there does have some strategies we may consider:
  - It is always a good start to talk to domain experts so as to get some senses of your data and variables of interest. Then use the most recent, powerful, and off-the-shelf supervised learning techniques to get *rankings of variable importance*.
  - Different learning techniques might result in different rankings of the variable importance. Conflicting importance rankings often implies complicated interactions and correlations among predictors. Try to resolve these problem by identifying such connections, removing irrelevant (low importance) variables, and/or reducing dimensionality and numerosity of your data.
  - Although *simplicity (and complexity!) does not imply accuracy*, still, DO NOT try to find "the best model" (i.e. model with the lowest error). Instead, choose relatively simple, parsimonious, and interpretable models. Such models with fewer variables and/or parameters often perform better when they are applied to real-world prediction tasks!

Hands-on Feature Selection

- Let's get back to the *advertise* data again. Consider two simple techniques first that rank the importance of variables by *absolute standardized regression coefficients* and *t-statistics* of general linear models.

```
# Standardized variables using scale()
z_advertise = data.frame(scale(advertise))
z_advertise_lm = lm(Sales ~ TV + Radio + Newspaper, z_advertise)
data.frame(abs_std_coef = abs(z_advertise_lm$coefficients[-1]))

            abs_std_coef
TV          0.753065912
Radio       0.536481550
Newspaper   0.004330686

# Rank based on absolute t-statistics. Use function in package "caret"
caret::varImp(z_advertise_lm)
           Overall
TV         32.8086244
Radio      21.8934961
Newspaper   0.1767146
```

Hands-on Feature Selection(cont.)

- The rankings are identical. *TV* and *Radio* are relatively important compared to *Newspaper*. Also, the value of *Newspaper* importance (and standardized coefficient) is significantly lower then those for *TV* and *Ratio*. We might consider re-calculating the RMSEs again with and without *Newspaper*.

```
Map(function(x) LOOCV_lm_rmse(x, advertise),
    c(Sales ~ TV + Radio, Sales ~ TV + Radio + Newspaper))
"LOOCV RMSE for lm( Sales ~ TV + Radio) = 1.7061"
"LOOCV RMSE for lm( Sales ~ TV + Radio + Newspaper ) = 1.7167"
```

- Seems like we should just remove *Newspaper* to get "the best model" However, both simple variable importance techniques, unfortunately, cannot help us discover interactions between *TV* and *Radio* .

```
LOOCV_lm_rmse(Sales ~ TV * Radio, advertise)
"LOOCV RMSE for lm( Sales ~ TV * Radio ) = 0.9627"
summary(lm(Sales ~ TV * Radio, advertise))
...
Coefficients:
...
TV          1.910e-02  1.504e-03  12.699   <2e-16 ***
Radio       2.886e-02  8.905e-03   3.241   0.0014 **
TV:Radio    1.086e-03  5.242e-05  20.727   <2e-16 ***
```

Hands-on Feature Selection(cont.)

- We could just manually identify such interactions with helps of domain experts. For a big dataset with hundreds of variables, however, we certainly need techniques, such as *Forward Selection* and *Recursive Feature Elimination*, that automatically do the selection tasks for us. Don't worry about it now. We will soon discuss how they work.
- Consider another dataset *ISLR*::*Auto*. We would like to know what variables have significant impacts on *mpg*. Here we apply previous techniques along with *RandomForest variable importance* to the data.

```
sort(abs(lm_Auto$coefficients)[-1], decreasing = T)

weight     originJapanese   horsepower   displacement  originEuropean  cylinders      acceleration
4.0872671  2.9325397        2.3608132    1.1957694     1.1255451       0.9657934      0.0882399

cbind(caret::varImp(lm(mpg ~ . , data = Auto_xy)),
      importance(randomForest(mpg ~ ., data = Auto_xy)))

             Overall       IncNodePurity
cylinders    1.401195      3915.672
displacement 1.335385      6462.076
horsepower   3.699779      4445.469
weight       6.055225      5642.350
acceleration 0.272361      1419.728
origin       4.221362       975.282
```

Hands-on Feature Selection(cont.)

- RandomForest has become one of data analysts' standard toolbox used to identify important variables in recent decades. In this example, we can see the rankings are significantly different, which suggests there may have some "connections" (e.g. multicollinearity) among predictors. For real-world applications of linear models (e.g. identifying important variables), such connections must be handled carefully!
- In this case, we may do some further analyses on predictors, such as calculating *Variance Inflation Factors* or doing bivariate analysis. Correlation analysis might help. For discrete variables, Chi-square tests might also help. Some statisticians suggest quantifying such connections by computing *Mutual Information* in bits among variables.

```
Hmisc::rcorr(as.matrix(Auto_xy[, ! colnames(Auto_xy) %in% c("name",
"year")]),type = "spearman")
# Comparing models (with interaction "displacement:cylinders"
# and no "acceleration")
Map(function(x) LOOCV_lm_rmse(x, Auto_xy),
    list(mpg ~ ., mpg ~ . + displacement:cylinders, mpg ~ . +
displacement:cylinders - acceleration))
"LOOCV RMSE for lm( mpg ~ . ) = 4.2024"
"LOOCV RMSE for lm( mpg ~ . + displacement:cylinders ) = 4.0431"
"LOOCV RMSE for lm( mpg ~ . + displacement:cylinders - acceleration ) = 4.0262"
```

Hands-on Feature Selection(cont.)

- If you are a big fan of the *p*-value or really need an explanation (or excuse) of what variables are "significantly important", check out and compare the analysis results of aforementioned models.

```
Map(function(x) summary(lm(x, Auto_xy)),
    c(mpg ~ ., mpg ~ . + displacement:cylinders, mpg ~ . +
        displacement:cylinders - acceleration))
```

- And yes, if you're tired of such manual feature selection process, some R packages have provided functions that automatically do that for you. But, notice that these functions may not give you the best subset of variables and also not able to detect the "bad connections" within predictors!

```
library("caret"); library("doMC");
registerDoMC(cores = detectCores() - 1) # DO NOT use all CPU cores
# Recursive Feature Elimination algorithm with lm()
rfe_auto = caret::rfe(x = Auto_xy[,-1], y = Auto_xy$mpg,
  metric = "RMSE", rfeControl = rfeControl(method = "LOOCV",
    functions =  lmFuncs,  allowParallel = T)); rfe_auto$optVariables
[1] "weight"  "origin"  "horsepower"   "cylinders"  "displacement" "acceleration"

# It's telling you all variables are important!?
```

More about Feature Selection Process

- Before the buzzword "Big Data" is created, statisticians working in different fields have been using computers to automate the feature selection processes. As mentioned previously, the most popular techniques are *Forward Selection* and *Recursive Feature Elimination* (Backward Selection).

- Forward Selection generally starts the selection process with one or a few variables of interests, add new predictors to the model one at a time, and then stops when the addition of variables would no longer improve the model.

- Recursive Feature Elimination (RFE), however, starts a model with all predictor variables, and then delete them one at a time until removing variables would degrade the performance (e.g. higher RMSE) of the model.

Forward Selection

1) Begin with identifying *variables of interests*. These variables may always be "forced-in", as they are important variables in your research or studies.

2) Fit p simple linear regressions and add the variable that results in the lowest error (e.g. RMSE) to the model

3) Add the variable that results in the lowest measure among all two-variable models.

4) Continue this process (keep adding variables) until some stopping rule is satisfied, e.g. all remaining variables have a *p-value* below 0.1.

Recursive Feature Elimination(RFE)

1) Start with fitting a model with all variables.

2) Remove the variable with the largest *p*-value or lowest variable importance (e.g. refer to a ranking list from a RandomForest model may be a good start). The variable should be the least statistically significant.

3) Fit a new model with $(p - 1)$-variable model.

4) Continue step 2 and 3 until a stopping rule is reached. For instance, we may stop when all remaining variables have a significant *p*-value < 0.05 or the model performance(e.g. RMSE) cannot be improved any further.

Which Selection Process?

- You may wonder which feature selection process is better? The answer is "it depends". We should always take the size (*n* observations by *p* variables) of our dataset, the goal of the data analysis, and the statistical learning techniques we use into consideration.

- In the age of the "Big Data" (say we have hundreds of variables), people tend to use *ensemble learning* algorithms with RFE in *exploratory data analysis* and let a cluster of computers to find the best subset of variables. On the other hand, *confirmatory data analysis* tasks often involve forward selection along with typical statistical tools of inference, significance, and confidence, as people usually have variables of interests and look for evidences that support their views.

- However, no technique is perfect. Experienced data engineers often mix both approaches and use different statistical learning algorithms to verify their findings from data. They also suspect those models that are "too good to be true"!

A Real-world Hybrid Selection Approach

```
# To predict "murder rate" of 50 states in the U.S. (1977)
data(state); murder = data.frame(state.x77)
# Lists of variable importance could give you some senses of the data
varImp(lm(Murder ~ ., data = murder))
murder_rf = randomForest(Murder ~ ., data = murder, ntree = 100)
importance(murder_rf)
# Let's try RFE
caret::rfe(x = murder[,-5], y = murder$Murder, metric = "RMSE",
  rfeControl = rfeControl(method = "LOOCV", functions =  lmFuncs,
    allowParallel = T))
# Hands-on hybrid selection process. Step 1 - bivariate analysis
Map(function(x) {fit = lm(x, data = murder); anova(fit)$"Pr(>F)"[1]},
        Map(as.formula, paste("Murder ~", colnames(murder)[-5])))
# Step 2 - fit model with those variables with p-values < 0.1
summary(lm(Murder ~ Population + Illiteracy + Life.Exp + HS.Grad +
Frost, murder))
# Step 3 - fit model again but remove those with p-values < 0.1
summary(lm(Murder ~ Population + Illiteracy + Life.Exp + HS.Grad ,
murder))
# We may continue this process, or just stop here then start
# identifying the "bad connections" among predictors.
```

Group Exercise #1

- Please check out our course page in NSYSU Cyber University. Go get your team members and begin hacking!

Going beyond Linear Models

- Classical linear model has many advantages (e.g. easy-to-interpret), and we have been using it to solve regression problems. However, you may have noticed that it often underperforms those statistical learning algorithms (e.g. the RandomForest) when our dataset is big (e.g. $n < p$ problem).

- Due to recent explosion of data growth, the weaknesses of typical statistical modeling methods like linear models have caused huge controversy. Statisticians have been trying to improve them by creating new parameter estimations techniques and/or relaxing assumptions of these methods (e.g. linear independences among predictors).

Going beyond Linear Models(cont.)

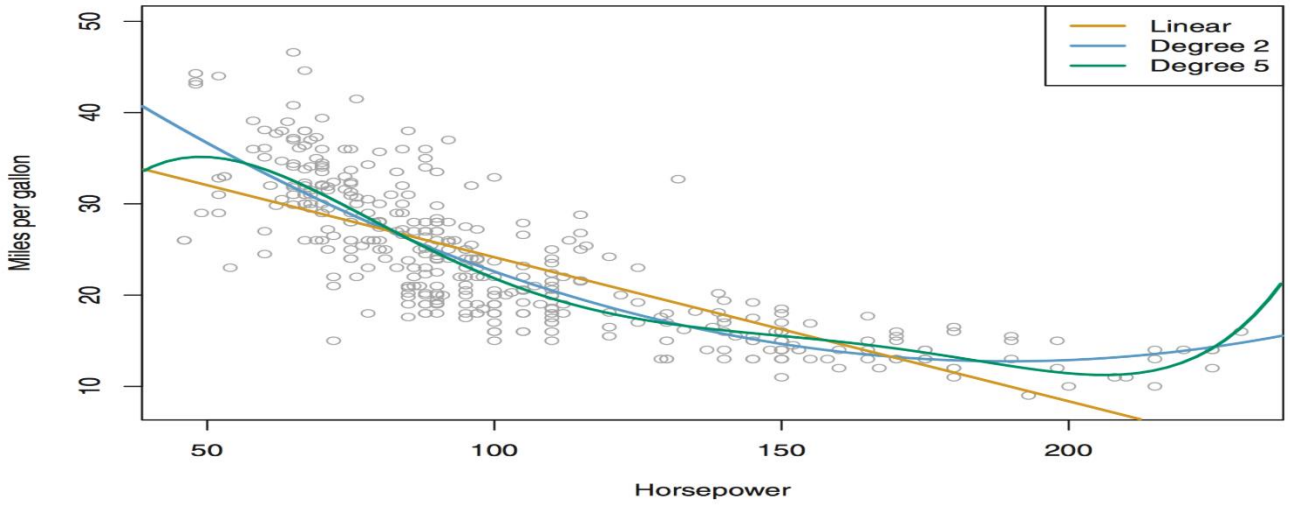
- In this unit, we will be discussing some methods that expand the scope of linear models to solve below problems:
  - **Non-linearity:** kernel smoothing, splines and generalized additive models; nearest neighbor methods
  - **Interactions**: Tree-based methods, bagging, random forests, and boosting (these also capture non-linearities)
  - **Regularized fitting**: Ridge regression and LASSO
  - **Classification problems**: Logistic Regression, Support Vector Machines

Polynomial Regression

- The linear regression assumes a linear relationship between the response and predictors. But in some cases, the true relationship between the response and the predictors may be non-linear. A very simple way to directly extend the linear model to accommodate non-linear relationships is to use *polynomial regression*, which is defined:

$$\hat{y} = \beta_0 + \beta_1 * X_1 + \beta_2 * X_1^2 + \cdots + \beta_p * X_1^p$$

- Let's consider bivariate relationship between *mpg* and *horsepower* (raised to integer power of 5) in previous *Auto* data as an example.



Polynomial Regression(cont.)

- Seems clear that the relationship is *non-linear*—it's most likely a curved relationship. A simple approach for incorporating non-linear associations in a linear model is to include transformed versions of the predictors in the model. For example, the points in the figure seem to have a quadratic shape, suggesting that a model of the form

$$\widehat{mpg} = \beta_0 + \beta_1 * horsepower + \beta_2 * horsepower^2$$

might provide a better model. We can easily fit this model in R:

```
# polynomial regression
Auto_xy_hp2 = transform(Auto_xy, horsepower2 = horsepower ^ 2 )
summary(lm(mpg ~ horsepower + horsepower2, Auto_xy_hp2))
# Or, just use I()
summary(lm(mpg ~ horsepower + I(horsepower ^ 2), Auto_xy))
...
horsepower      -0.4661896  0.0311246  -14.98   <2e-16 ***
I(horsepower^2)  0.0012305  0.0001221   10.08   <2e-16 ***
```

Polynomial Regression(cont.)

- It looks like including *horsepower^2* led to a big improvement in the model. If so, why not just includes *horsepower^3* , *horsepower^4* , or even more? Yes, as you may expect, we would most likely create a complicated model in such cases, and this kind of models may not be generalized well.
- So, try to identify curvilinear relationships by plotting your data or evaluating model fit results. Unless our data shows such clear relationships, we should not consider polynomials of higher degree.

```
# Use lm() with poly() to fit polynomial regression models
summary(lm(mpg ~ poly(horsepower, degree = 5), data = Auto_xy))
...
poly(horsepower, degree = 5)1 -120.1377     4.3259 -27.772  < 2e-16 ***
poly(horsepower, degree = 5)2   44.0895     4.3259  10.192  < 2e-16 ***
poly(horsepower, degree = 5)3   -3.9488     4.3259  -0.913  0.36190
poly(horsepower, degree = 5)4   -5.1878     4.3259  -1.199  0.23117
poly(horsepower, degree = 5)5   13.2722     4.3259   3.068  0.00231 **
...
# Seems like fitting to the degree of 2 is reasonable.
```

Regression for Classification Problems

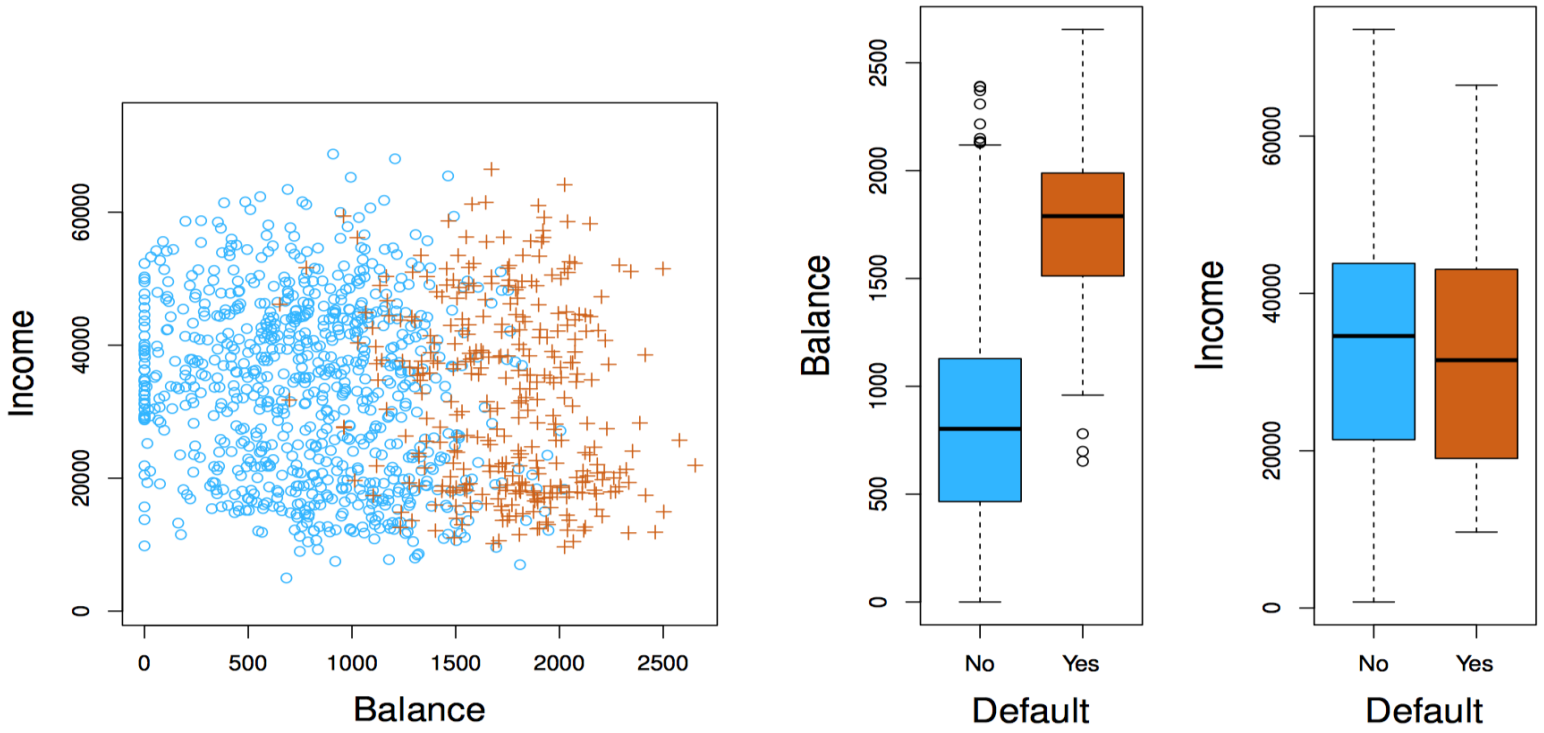
- We have been trying to predict continuous variables and solve regression problems using linear regression. How about predicting categorical/discrete target variables?

- Categorical variables are those with a set of unordered values, such as eye color ∈ {brown, blue, green}, email ∈ {spam, ham}, and buy ∈ {Yes, No}. To predict values of categorical response Y given a set of features X is considered *Classification problems*.

- Often, we are more interested in estimating the probabilities that X belongs to each category/value of Y—the classification task to build a classification function $C(X)$ that takes features X as the input and predicts its category/value for Y.

Regression for Classification Problems(cont.)

- Let's start by considering a dataset *ISLR::Default* consisting 4 variables: (*default*, *student*, *balance*, *income*). We're trying to predict whether a customer would default on their credit card debt, and what variables are "significant" when predicting *default*.



Logistic Regression

- The *default* is a binary variable (**0** for **No**, **1** for **Yes**). We can certainly use linear regression by considering the "probability of default" as the outcome. If the predicted value > 0.5 (50%), we classify it as "Yes" (we'll talk more about setting this "threshold").

- However, linear regression might produce probabilities less than zero or bigger than one. Applying logistic function to the predicted values of our linear models, *Logistic Regression,* could solve this problem.

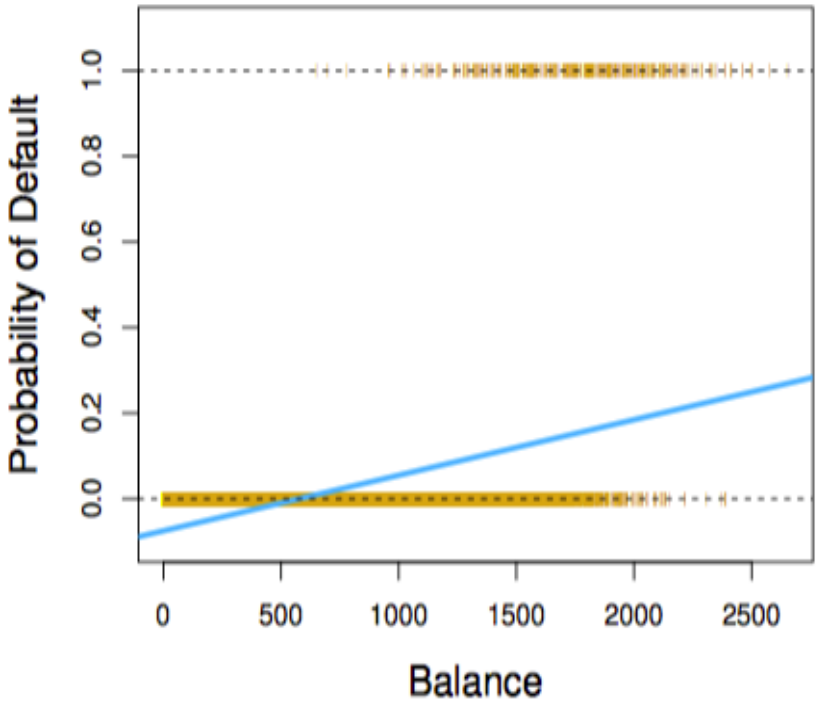
- Let's consider using *balance* to predict *default* , $p(X)=P(Y= 1 \mid X)$, for example. Logistic regression uses the form:

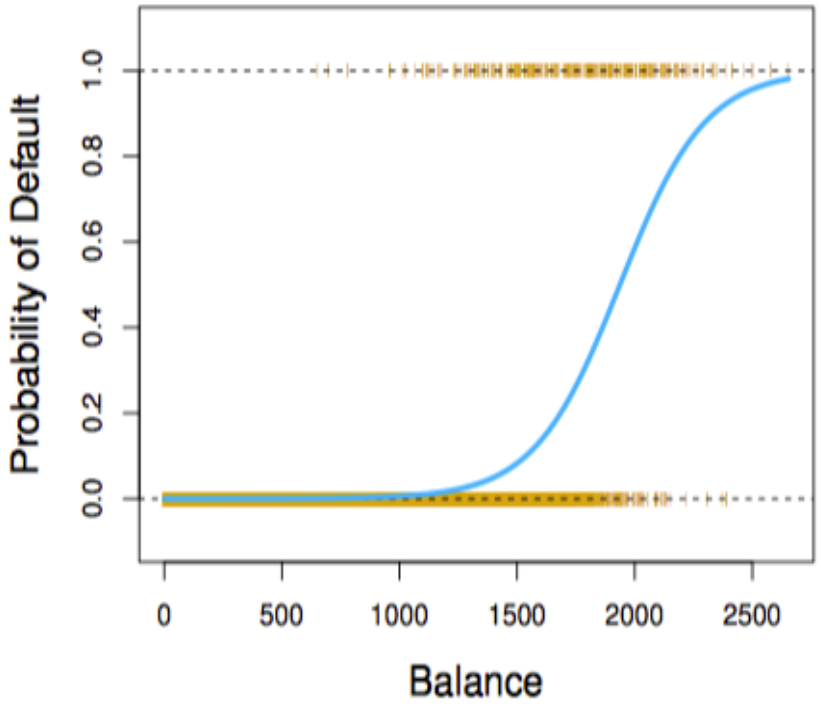
$$p(X) = \frac{e^{\beta_0+\beta_1 X}}{1 + e^{\beta_0+\beta_1 X}}$$

where $p(X)$ will have values between 0 and 1.

Logistic Regression(cont.)

- We can see from below figure. Logistic regression ensures that our estimate for $p(X)$ lies between 0 and 1.





Logistic Regression(cont.)

- In R, we can easily fit logistic models by using *glm*().

```
library("ISLR"); data("Default")
glm_default = glm(default ~ balance, data = Default,
   family = "binomial"); summary(glm_default)
...
Coefficients:
              Estimate Std. Error z value Pr(>|z|)
(Intercept) -1.065e+01  3.612e-01  -29.49   <2e-16 ***
balance      5.499e-03  2.204e-04   24.95   <2e-16 ***
```

- So, how can we use the model to predict the probability of *default=Yes*, say the *balance* is $2000?

$$P(default = Yes|balance = 2000\ ) = \frac{e^{-10.65\ +\ 0.0055\ *2000}}{1 + e^{-10.65\ +\ 0.0055\ *2000}} = 0.586$$

Logistic Regression(cont.)

- Note that the previous model is for continuous predictors, how about categorical ones? Let's consider using *student* instead.

```
glm_default = glm(default ~ student, data = Default,
   family = "binomial"); summary(glm_default)
...
Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept) -3.50413    0.07071  -49.55  < 2e-16 ***
studentYes   0.40489    0.11502    3.52 0.000431 ***
...
```

- The probability of *default=Yes* when a customer is a student (*student = Yes*) or not (*student = No*) can be computed as:

$$P(default = Yes|student = Yes) = \frac{e^{-3.5041+0.4049*\mathbf{1}}}{1+e^{-3.5041+0.4049*\mathbf{1}}} = 0.0431$$

$$P(default = Yes|student = No) = \frac{e^{-3.5041+0.4049*\mathbf{0}}}{1+e^{-3.5041+0.4049*\mathbf{0}}} = 0.0292$$

Interpretation of Logistic Regression

- Logistic regression has become one of the most popular statistical modeling techniques in scientific research, especially in Business, Public Health, and Medicine. Many researchers are fascinated by its *p-value*, *relative risk*, and *odd ratio,* probably because these measures can be easily interpreted. However, logistic regression does have limitations when it is used in analyzing big data. We will be discussing more about it soon. Now, let us just find out why logistic regression is so charming.
- Here we begin with a crosstab "*student* by *default*", and compute the percentages of being default for two groups, (*student* = *No*) and (*student* = *Yes*). Then, the *odd ratio* (OR) of being default for "student", compared to "non-student" can be easily computed as:

```
p = prop.table(xtabs( ~ student + default, Default), margin = 1); p
             default
student         No        Yes
    No  0.97080499 0.02919501
    Yes 0.95686141 0.04313859

odds = p[,2] / p[,1]
odds[2] / odds[1] # odd ratio of being default for "student"
[1] 1.499133
```

Interpretation of Logistic Regression(cont.)

- The OR is approximately 1.5, which suggests that the odds of being default for "students" is 1.5 times larger than odds for the reference group ("NOT student"). Note that such interpretation of OR create a "magic"—researchers could determine whether a particular *exposure* (e.g. being a student) is a *risk factor* for a particular outcome (being default), and also compare the *magnitude* (e.g. 1.5 times) of various risk factors for that outcome.

- Notice that we here only fit bivariate models, *default ~ student*. For real-world applications, obviously, we would most likely consider putting the rest of independent variables into the model. Let's say we have done a series of feature selection processes on predictors and found the best model as below.

```
# Let's say "default ~ balance + student" is our best model
glm_default = glm(default ~ balance + student, data = Default,
   family = "binomial"); summary(glm_default)
...
balance      5.738e-03  2.318e-04  24.750  < 2e-16 ***
studentYes  -7.149e-01  1.475e-01  -4.846 1.26e-06 ***
```

Interpretation of Logistic Regression(cont.)

- Note that we here consider the OR after *adjusting for* other variables, i.e. we'd like to know the OR of "being a student" after considering *balance*. By definition of OR, as

$$OR = \frac{P(y=1|x_1=exposed)}{P(y=1|x_1=unexposed)} = \frac{p(x_1=1)/(1-p(x_1=1))}{p(x_1=0)/(1-p(x_1=0))} = \frac{e^{\beta_0+\beta_1 X_1+\cdots+\beta_p X_p}}{e^{\beta_0+0+\cdots+\beta_p X_p}}$$

$$\Rightarrow \log(OR) = \log\left(\frac{e^{\beta_0+\beta_1 X_1+\cdots+\beta_p X_p}}{e^{\beta_0+0+\cdots+\beta_p X_p}}\right) = (\beta_0 + \boldsymbol{\beta_1 X_1} + \cdots + \beta_p X_p) - (\beta_0 + \mathbf{0} + \cdots + \beta_p X_p) = \beta_1 X_1 = \beta_1$$

we can easily get the *adjusted OR* by exponentiating β.

```
glm_default$coefficients
 (Intercept)       balance    studentYes
-10.749495878   0.005738104  -0.714877620
exp(-0.714877620)
0.489252 # adjusted OR for "student=yes"
```

- We're a little surprised that, after adjusting for the *balance*, the OR is 0.4892, compared to 1.5 of the "crude" bivariate model, which also suggests that students tend NOT to be in default if we also take the balance they carry into consideration!

Interpretation of Logistic Regression(cont.)

- You may have frequently seen such OR & adjusted OR along with statistics & p-values in many scientific literatures. Many researchers present analysis results in this way when dealing with data with binary outcomes. So, as you may expect, there certainly has R packages (e.g. *epiDisplay*) that do all aforementioned calculations for you.

```
epiDisplay::logistic.display(glm_default)

Logistic regression predicting default : Yes vs No
                     crude OR(95%CI)         adj. OR(95%CI)          P(Wald's test)
balance (cont. var.) 1.0055 (1.0051,1.0059)  1.0058 (1.0053,1.0062)  < 0.001   ...
student: Yes vs No   1.5 (1.2,1.88)          0.49 (0.37,0.65)        < 0.001   ...
```

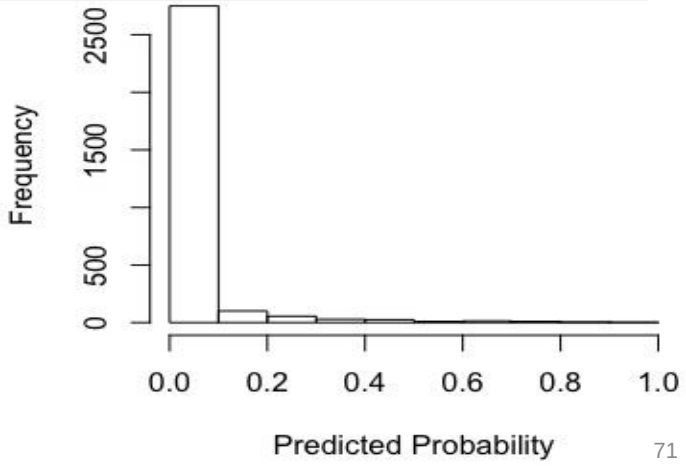
- Researchers love this "magic" of binary logistic regression, as it can be easily comprehended by most people. However, this magic have failed in the age of Big Data. Due to recent explosion of data, typical logistic regression modeling based on MLE has been proved unstable and slow when estimating model parameters from large dataset and dealing with large classification or prediction tasks. Good news is that techniques, such as *regularizations,* have been proposed to fine-tune the process of parameter estimations. We'll be talking more about them later!

Logistic Regression for Classification

- With the logistic regression model, we are able to compute probability of "*default=Yes*" given a set of predictor values (*balance* & *student*).

```
# Split "Default" dataset into training and testing data
set.seed(1); numOfRows = nrow(Default) # Number of observations
train_idx = sample(1:numOfRows,numOfRows * 0.7) # 70% (7,000) as training
test_idx = setdiff(1:numOfRows, train_idx) # 3,000 as testing

default_LR = glm(default ~ student + balance,
   data = Default[train_idx,], family = "binomial") # Fit logit model
test_pred_prob = predict(default_LR, newdata = Default[test_idx,],
   type = "response") # Generate predicted probability
hist(test_pred_prob, xlab="Predicted Probability") # Plot your data!
```

- Notice that most of predicted probabilities of being in default are below 0.1. It's simply because there is a severe *Class Imbalance*, i.e. "*default=yes*" is a *rare event*. It often occurs when numbers of instances of classes are significantly different.



Model Evaluation for Classification

- A severe class imbalance problem may affect model performance. But it can be easily identified beforehand by checking the distribution of discrete outcome. Techniques are proposed to cope with this problem. Most of them are based on evaluations of a *Confusion Matrix*—crosstab of predicted by actual response values. For binary outcome, a confusion matrix and its related measures are defined as:

		Actual Class	
		Yes	**No**
Predicted Class	**Yes**	True Positive (TP)	False Positive (FP)
	No	False Negative (FN)	True Negative (TN)

$$\mathbf{N\ (Num.\ of\ Obs.) = TP + FP + FN + TN}$$

$$\mathbf{Accuracy = 1 - Misclassification\ Rate = \frac{TP + TN}{N}}$$

$$\mathbf{Sensitiviy = True\ Positive\ Rate = \frac{TP}{TP + FN}}$$

$$\mathbf{Specificity = True\ Negative\ Rate = \frac{TN}{FP + TN}}$$

$$\mathbf{False\ Positive\ Rate = 1 - Specificity = \frac{FP}{FP + TN}}$$

Model Evaluation for Classification(cont.)

- As the "*default=yes*" is rare (3.34%), if the cutoff is too high (50%), the model might not be able to capture and predict such rare event ("True Positive" might be too low). One possible solution is to find a new "optimal cutoff" that balances both high sensitivity and specificity. Let's consider three cutoff, 0.8, 0.5 and 0.1. What are their confusion matrices?

```
# The probability of "default=yes" is actually very low.
prop.table(table(Default[train_idx,]$default))
No         Yes
0.96657143 0.03342857
# 3 confusion matrices for 3 cutoffs
pred_class = Map(function(x) factor(ifelse( test_pred_prob > x,
   "Yes", "No")), c(0.8, 0.5, 0.1) )
CMs = Map(function(x) table(relevel(x,"Yes"),
   relevel(Default[test_idx, "default"], "Yes") ) , pred_class)
CMs
[[1]]
       Yes   No
  Yes    7    1
  No    92 2900

[[2]]
       Yes   No
  Yes   28   12
  No    71 2889

[[3]]
       Yes   No
  Yes   71  179
  No    28 2722
```

Model Evaluation for Classification(cont.)

- We can see that changing the cutoff just to make trade-offs different types of errors (numbers go up and down). It may change the sensitivity and specificity, but it does not necessarily increase the overall predictive power of the model. Also notice that, in real-world applications, the "optimal cutoff" should be derived from *an independent sample* (NOT from training or testing datasets). An optimal cutoff can be identified by finding the highest sum of "sensitivity + specificity", i.e. to find maximum *Youden's J Statistic*, which is defined:

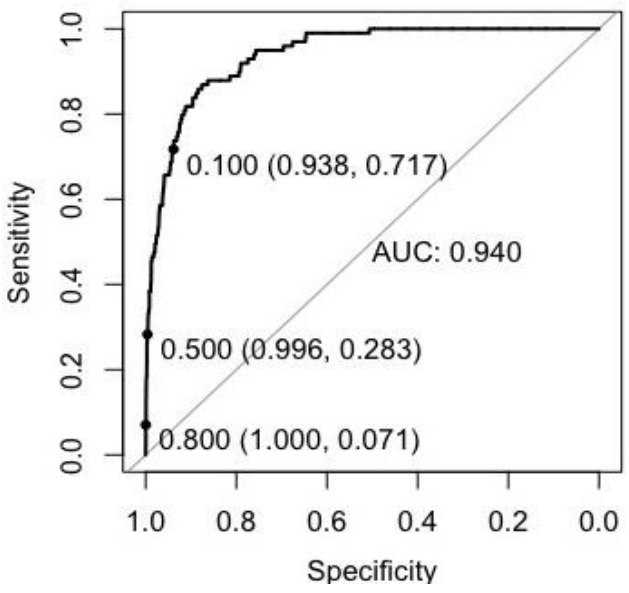
$$\boldsymbol{J = \text{Sensitivity} + \text{Specificity} - 1}$$

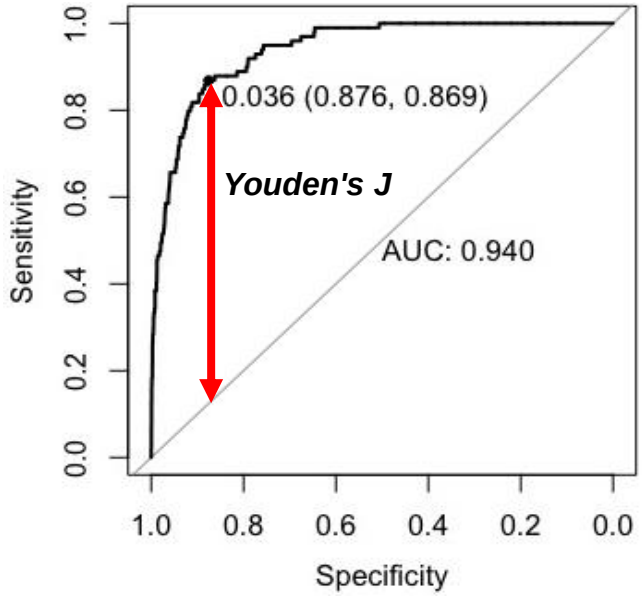
- So, how to evaluate the "overall" classification model performance? A popular technique is to plot *Receiver Operating Characteristic* (ROC) curve and see how the models perform given different cutoffs. To quantify performance, some researchers prefer the *Area Under the ROC Curve* (AUC), which is between 0 and 1.

Model Evaluation for Classification(cont.)

- In R, package *ROCR* and *pROC* should suit your need.

```
library("pROC")
default_glm_roc = roc(Default[test_idx, "default"], test_pred_prob)
# ROC for 3 cutoffs
plot.roc(default_glm_roc,print.thres = c(0.8, 0.5, 0.1), print.auc=T)
# Optimal threshold/cutoff based on Youden's Index
# Notice that, again, in real-world applications, you should use an independent
# sample (instead the testing data in our sample code) to identify this cutoff.
plot.roc(default_glm_roc,print.thres = "best",
   print.thres.best.method="youden", print.auc = T)
```





k-fold Cross-Validation

- Indeed, we should not rely on result of ROC & AUC only to evaluate classification model performance. To properly estimate test error associated with fitting a model on a dataset, we could get cross-validation of error measures by alternatively holding out a part of the dataset as a validation set and computing average error.

- In LOOCV, which can be considered a special case of *k-fold Cross-Validation (k*-fold CV*)*, one observation is excluded as validation set and *n* models are created. *k*-fold CV, however, randomly splits data into *k* pieces, repeatedly holds out 1/*k* observations as validation set, and then fits & uses *k* models to estimate errors.

- Some literatures have suggested that, to evaluate model performance, 10- to 20- fold CV should result in reasonably good estimates of errors, and also balance the variance of estimates and complexity of computations.

k-fold Cross-Validation(cont.)

- Let's get back to our *Default* data and see how *k*-fold CV helps us measure the performance of classification models.

```
# Get k-fold CV confusion matrix for Logistic Regression model
# f: formula, d: data, k: number of folds, cutoff: cutoff point 0-1
 k_fold_CV_logit = function(f, d, k, cutoff){
  numOfRec = nrow(d) # number of observations
  reponse_var = all.vars(f)[1] # name of the response variable
  # k indices used to split data into k parts
  sample_idx_k = rep(sample(1:k),round(numOfRec / k) + 1)[1:numOfRec]
  # k models for k subsets of data
  k_fits = Map( function(x) glm(f, d[sample_idx_k != x, ],
             family = "binomial"), 1:k)
  # Predicted & actual classes for each hold-out subset
  predActualClass = Map(function(x){
    predictedProb = predict(k_fits[[x]], d[sample_idx_k == x,],
                    type = "response")
    predictedClass = ifelse(predictedProb > cutoff, 1, 0)
    return(data.frame("predictedClass" =  predictedClass,
           "actualClass" = d[sample_idx_k == x, reponse_var] ) )
    }, 1:k)
  # A data frame with all predicted & actual classes
  output_DF = Reduce(function(x, y) rbind(x, y), predActualClass)
  output_DF$predictedClass = factor(output_DF$predictedClass,
      levels=c(0,1),labels = c("No", "Yes"))
  return( table(output_DF$predictedClass, output_DF$actualClass))
}
```

k-fold Cross-Validation(cont.)

```
# Different cutoffs
Map(function(cutoff) k_fold_CV_logit(default ~ student + balance,
   Default[train_idx,], 10, cutoff), list(0.8, 0.5, 0.1))
```

```
[[1]]                [[2]]                [[3]]
       No  Yes              No  Yes              No  Yes
  No  6762  211        No  6735  156        No  6386   57
  Yes    4   23        Yes   31   78        Yes  380  177
```

```
# Different "k"s
Map(function(k_fold) k_fold_CV_logit(default ~ student + balance,
   Default[train_idx,], k_fold, 0.5), list(5, 10, 20))
```

```
[[1]]                [[2]]                [[3]]
       No  Yes              No  Yes              No  Yes
  No  6736  155        No  6735  156        No  6735  156
  Yes   30   79        Yes   31   78        Yes   31   78
```

```
# Accuracy of prediction for models with different feature set
Map(function(x) sum(diag(k_fold_CV_logit(x, Default[train_idx,], 10,
   0.5))) / nrow(Default[train_idx,]), list(default ~ student,
      default ~ balance, default ~ student + balance, default ~
         student + balance + income ))
```

Group Exercise #2

- Please check out our course page in NSYSU Cyber University. Happy hacking!

Try It!

- Please download *titanic* dataset from below link http://biostat.mc.vanderbilt.edu/wiki/pub/Main/DataSets/titanic.txt. Try to identify most relevant variables and create your best logistic models that predict "survived". Also please report *p*-values and crude/adjusted odd ratios. Do you see anything interesting?

- Please write a function in MapReduce that computes *k*-fold cross-validation accuracy of prediction for logistic regression models.

Further Beyond Linearity

- The world is complex, and the truth is almost never linear. Linearity assumptions simplify the tasks of modeling our worlds of interests. It works good enough, but loses some flexibilities.

- In the recent decades, computer scientists have been developing solutions to a variety of prediction and optimization problems. Different from views of typical statisticians, their interpretations of the worlds were considered special in his/her time, but later proven revolutionary. They tend to focus on how to solve problems, without making assumptions on data generations, by simulating systems and explaining the world with a series of systematic steps—*algorithms*.

- Here, we will be discussing one of such learning algorithms—*Tree-based learning*, which perform comparably well without losing ease, flexibility, and interpretability.

Tree-based Methods

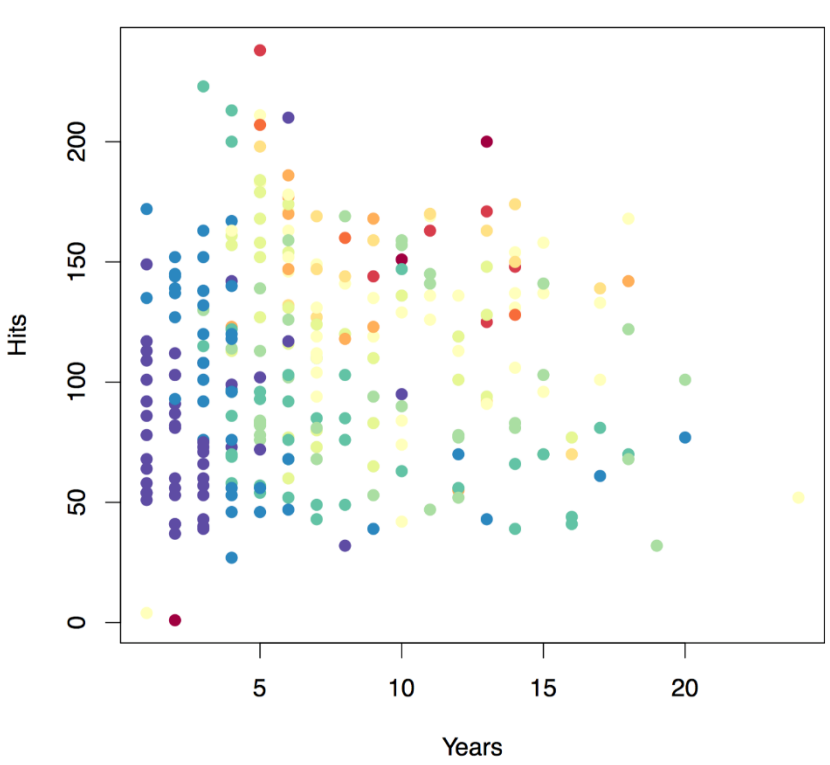
- The basic idea of tree-based approaches is to stratify or segment predictor spaces into a number of simple *regions*. Since the set of splitting rules used to segment the predictor space can be summarized in a tree, these types of approaches are also called *decision trees*, which are simple and useful for interpretation. However, they typically are not competitive with the best modern supervised learning approaches in terms of prediction accuracy.

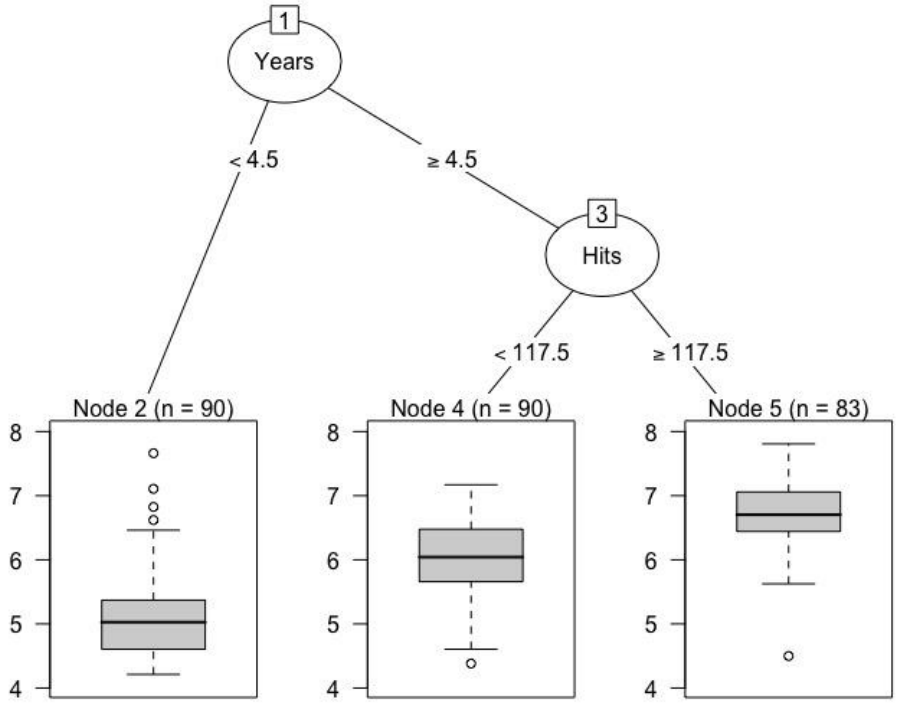
- Researchers have improved the tree approaches by combining a large number of trees, which often results in dramatic improvements in prediction accuracy, at the expense of some loss of interpretation.

- In this unit, we will be introducing one of popular tree-based methods—*Classification and Regression Trees* (*CART*) along with *Random Forest*, which can deal both classification and regression problems.

Classification and Regression Trees

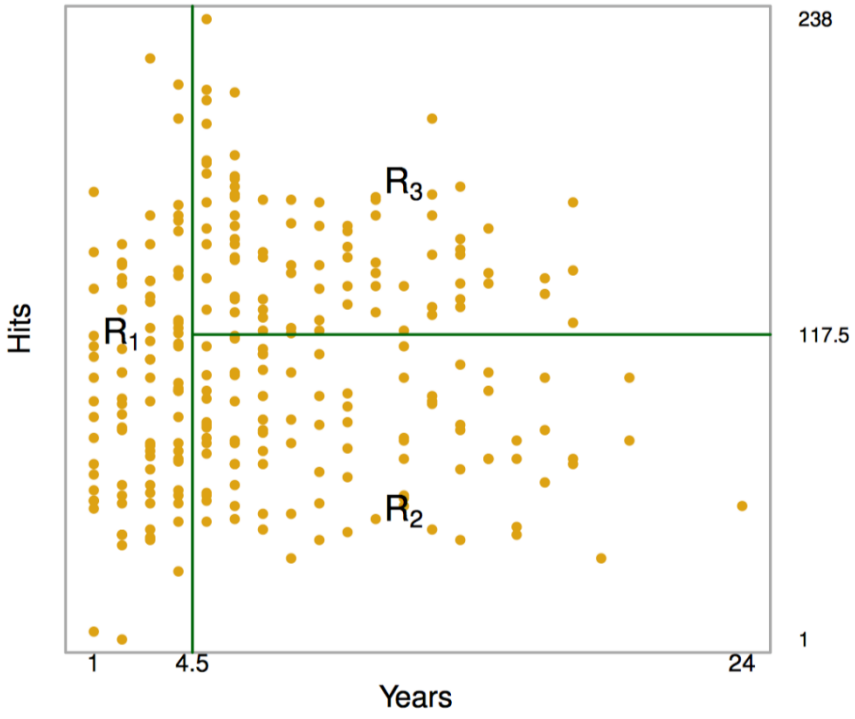
- Let's begin with a simple stratification task. Consider a baseball salary dataset *ISLR::Hitters*. *Salary* is color-coded from low (blue, green) to high (yellow,red). And the *log of Salary* is used later in this example so as to get parameters insensitive to the outliers and scale.





Regression Tree

- For the *Hitters* data, we can build a regression tree for predicting the log salary of a baseball player, based on the number of years (*Years*) that he has played in the MLB and the number of hits (*Hits*) that he made in the previous year.
- We can see that the tree splits data space into 3 regions (R_j) that denote 3 different leaf nodes. Each region is defined/identified by a rule from root to the leaf node. For example, the splitting rule for R2 is "**IF** *Years* >= 4.5 **AND** *Hits* < 117.5 **THEN** *log of Salary* = 6". For every observation that falls into a region R_j, we make the same prediction, which is simply the *mean* of the response values for the training observations in R_j (e.g. 6 for R2).



Regression Tree Learning

- The tree is surely easy to display & interpret. But tree models are usually deeper and more complex in real-world applications. In a case that we train a tree with multiple predictors, how to choose predictor and then divide predictor space into regions (or "high-dimensional boxes")?

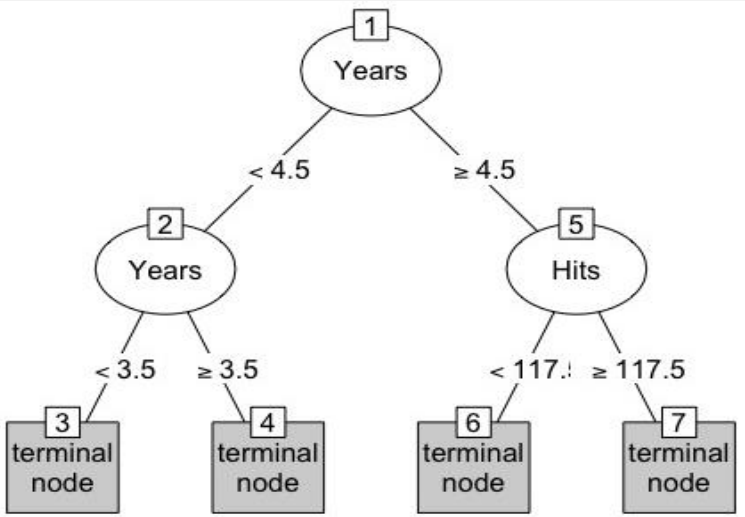
- Regression tree learning process takes a top-down & recursive binary splitting approach. It successively splits the predictor space by selecting a predictor with a cutpoint such that splitting the predictor space into J regions that minimize RSS, as:

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

where $\hat{y}_{R_j}$ is the mean response for the training observations within jth region.

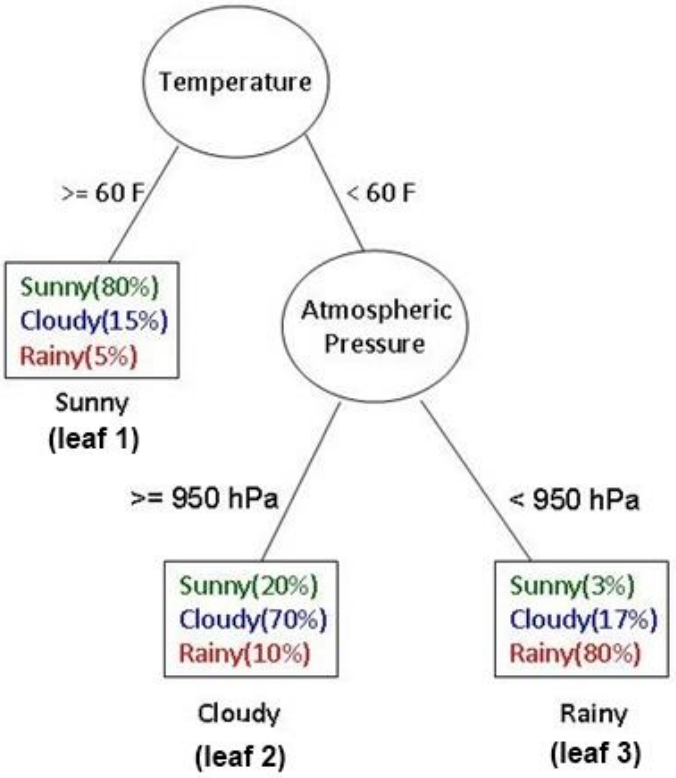
Regression Tree Learning (cont.)

```
# Using package "rpart"
library(ISLR); library(rpart); library(rpart.plot); data(Hitters)

# Building regression tree with depth = 2
hitSalary_RT = rpart(log(Salary) ~ Years + Hits, Hitters,
                     control = rpart.control(maxdepth = 2))

hitSalary_RT
n=263 (59 observations deleted due to missingness)
node), split, n, deviance, yval
* denotes terminal node
1) root 263 207.15370 5.927222
  2) Years< 4.5 90  42.35317 5.106790
    4) Years< 3.5 62  23.00867 4.891812 *
    5) Years>=3.5 28  10.13439 5.582812 *
  3) Years>=4.5 173  72.70531 6.354036
    6) Hits< 117.5 90  28.09371 5.998380 *
    7) Hits>=117.5 83  20.88307 6.739687 *

# Plot it!
library(partykit)
plot.party(as.party(hitSalary_RT))
```



Classification Tree

- Consider another example that we would like to build a tree to predict *Weather Condition* with *Temperature* and *Atmospheric Pressure*—a classification problem.



Weather Condition (Sunny, Cloudy, Rainy)	Temperature (F)	Atmospheric Pressure (hPa)
Sunny	65	980
Sunny	70	990
Sunny	55	800
Cloudy	50	960
Cloudy	72	850
Sunny	69	950
Cloudy	75	800
Rainy	60	840
Rainy	61	930
Cloudy	70	970

Classification Tree Learning

- Similar to the process of building regression tree, classification tree recursively and successively splits predictor spaces into regions. But here we choose a predictor & a cutpoint that increases *purity* of regions/leaf nodes instead. The purity is simply the percentage of the predominant class in a region, e.g. 80% for "Rainy" in leaf 3 in previous tree.

- Classification Tree uses *Gini Diversity Index* (GDI) to measure the purity, which is defined:

$$Gini(D) = 1 - \sum_{i=1}^{m} p_i^2$$

$$Gini_V(D) = \sum_{i=1}^{m} \frac{|D_i|}{|D|} Gini(D_i)$$

$$\Delta Gini(V) = Gini(D) - Gini_V(D)$$

where m is the number of classes of the target variable (e.g. 3 weather conditions, m = 3), p_i is the percentage of observations that belongs to a class i, $|D_i|$ is the number of observations that belongs to a class i in a region, and V is an independent variable as a predictor.

Classification Tree Learning (cont.)

- For simplicity, let's consider previous weather data but all variables are categorical instead. The goal is to choose a variable V and a nominal value (categorical variable) or a cutpoint (continuous variable) that maximize $\Delta Gini(V)$, i.e. to find the minimum $Gini_v\ (D)$.

$$Gini(Weather) = 1 - \left(\frac{4}{10}\right)^2 - \left(\frac{4}{10}\right)^2 - \left(\frac{2}{10}\right)^2 = 0.64$$

$$Gini_{Temperature}(Weather) = \frac{4}{10}\left(1 - \left(\frac{3}{4}\right)^2 - \left(\frac{1}{4}\right)^2\right) + \frac{4}{10}\left(1 - \left(\frac{3}{4}\right)^2 - \left(\frac{1}{4}\right)^2\right) + \frac{2}{10}\left(1 - \left(\frac{2}{2}\right)^2 - \left(\frac{0}{2}\right)^2\right)$$

$$= 0.15 + 0.15 + 0 = 0.3$$

$$\Delta Gini(Temperature) = 0.64 - 0.3 = 0.34$$

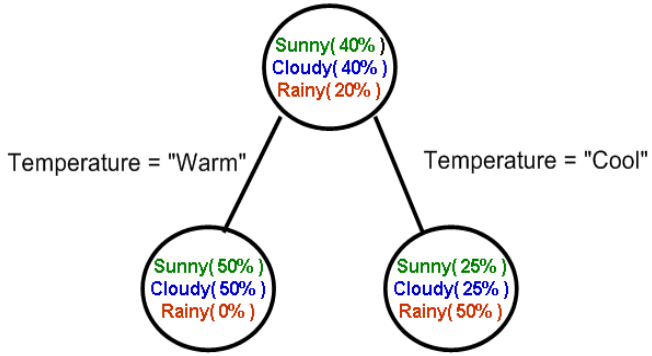
$$Gini_{Atmospheric\ Pressure}(Weather) = \frac{4}{10}\left(1 - \left(\frac{3}{4}\right)^2 - \left(\frac{1}{4}\right)^2\right) + \frac{4}{10}\left(1 - \left(\frac{2}{4}\right)^2 - \left(\frac{2}{4}\right)^2\right) + \frac{2}{10}\left(1 - \left(\frac{1}{2}\right)^2 - \left(\frac{1}{2}\right)^2\right)$$

$$= 0.15 + 0.2 + 0.1 = 0.5$$

$$\Delta Gini(Atmospheric\ Pressure) = 0.64 - 0.5 = 0.14$$

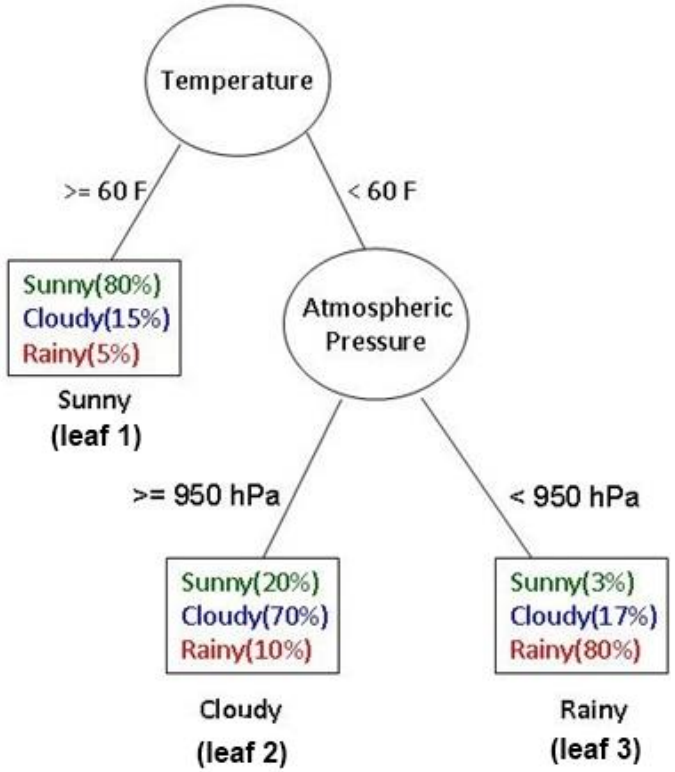
Weather Condition	Temperature	Atmospheric Pressure
Sunny	Warm	High
Sunny	Warm	High
Sunny	Cool	Low
Cloudy	Cool	High
Cloudy	Warm	Low
Sunny	Warm	High
Cloudy	Warm	Low
Rainy	Cool	Low
Rainy	Cool	High
Cloudy	Warm	High

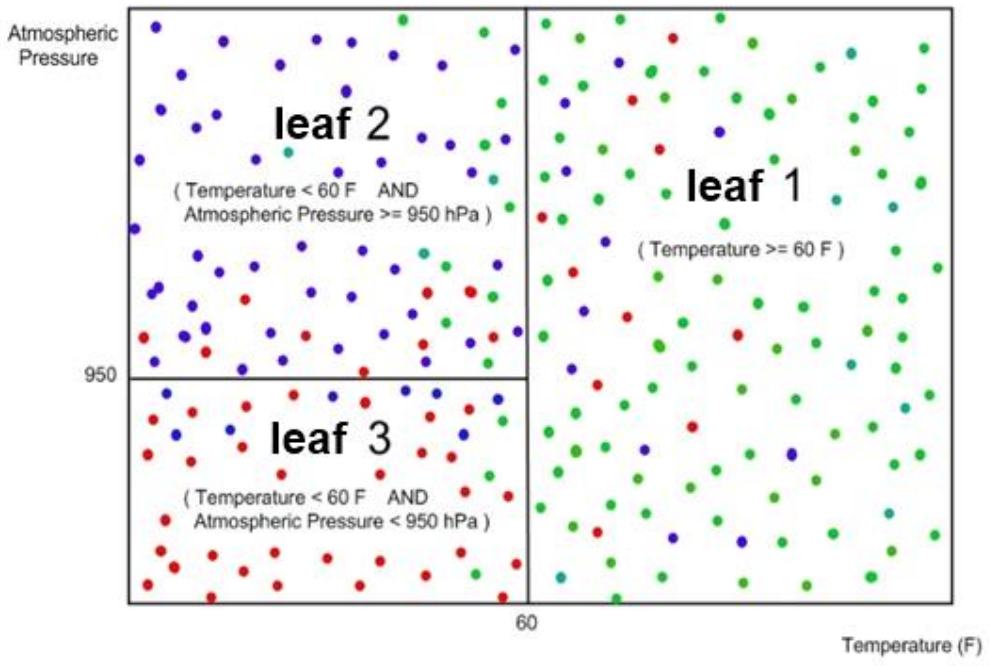
- We therefore choose *Temperature* (*Gini* = 0.34), not the *Atmospheric Pressure* (*Gini* =0.14), as the first split to create the tree.



Classification Tree Learning (cont.)

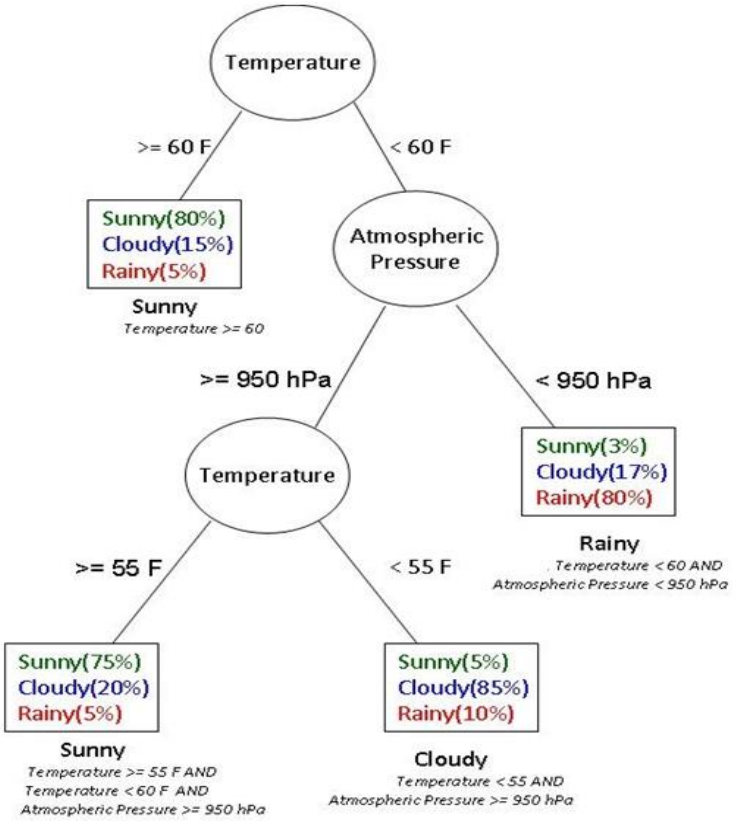
- Let's get back to previous tree model. The tree divides the predictor space into 3 regions/leave nodes. We can also see that these splits do increase the purity of regions.

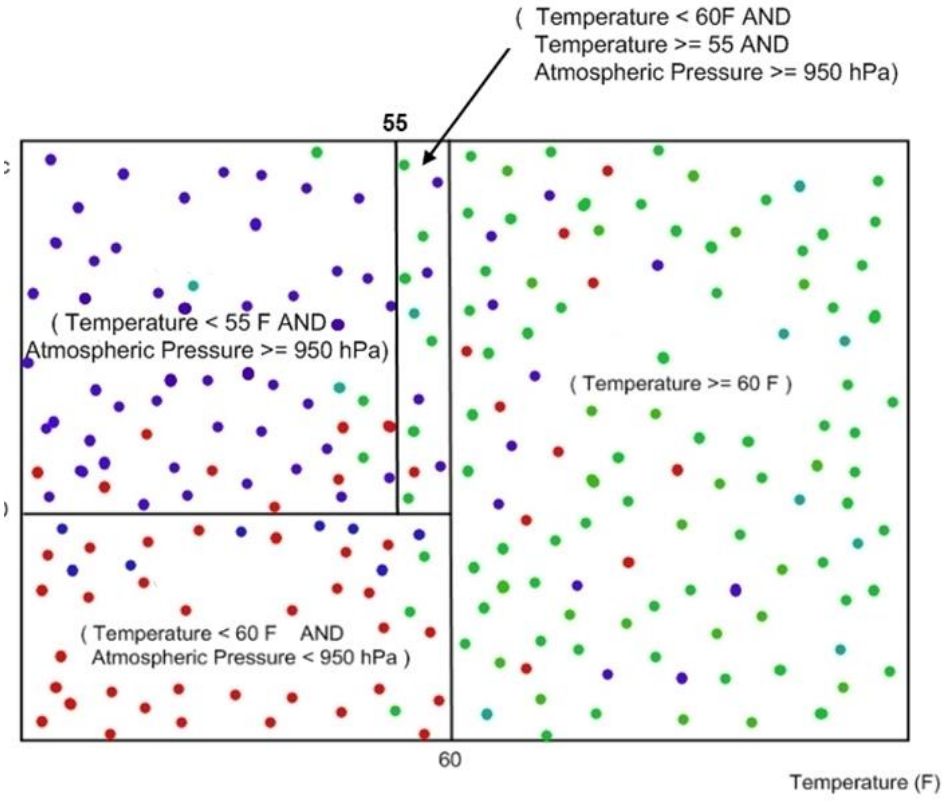




Tree Splitting

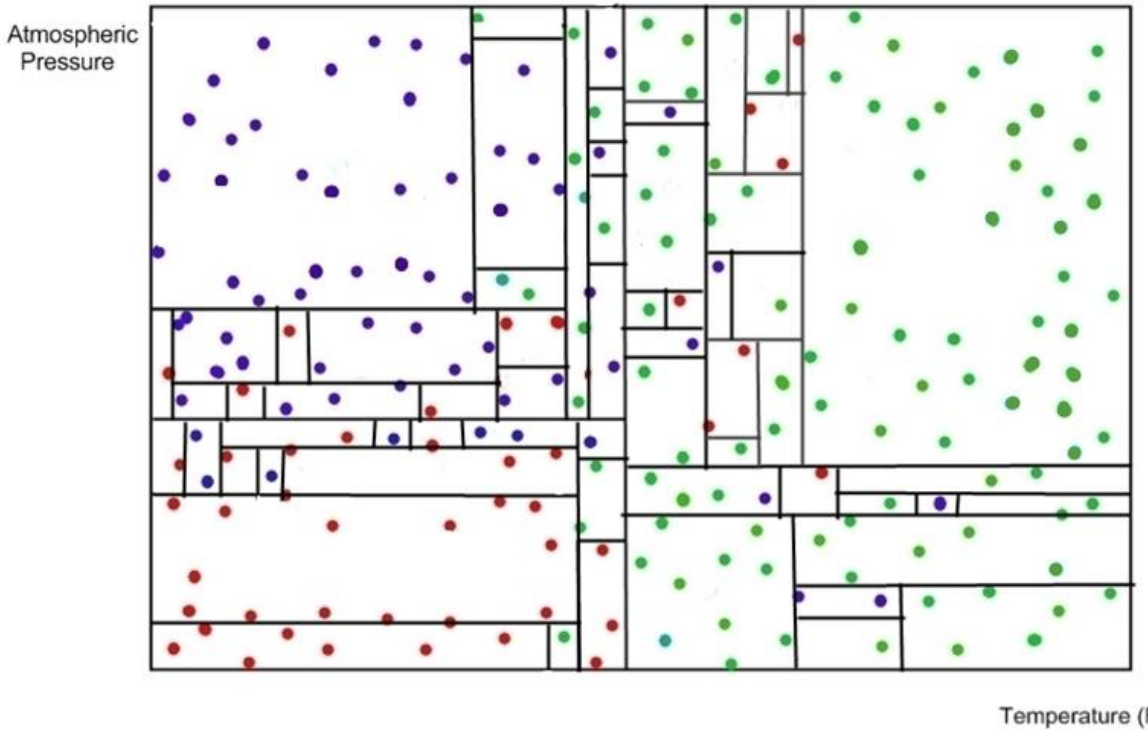
- We can certainly keep the tree growing and improving purity. In this case, we would get more leaf nodes but the rules for each leaf get more complicated.





Tree "Over-splitting"

- If we just let the tree fully grow, we will get a over-complicated tree such that the RSS is very low (for regression tree) or the purity is 100% (for classification tree). As you might expect, such trees have overfit the data. So, how to choose a better tree?



Tree Pruning

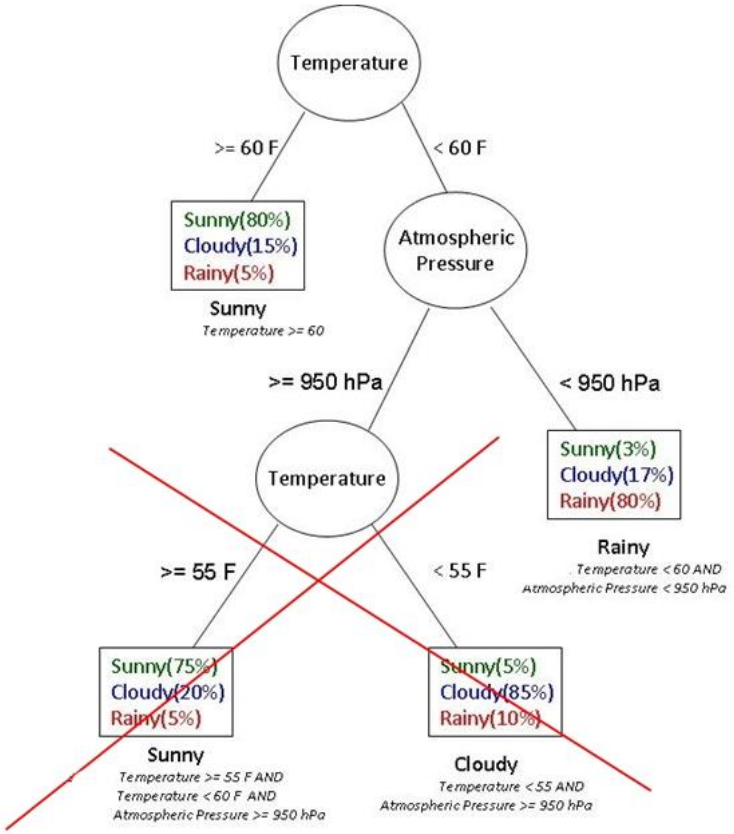
- Yes, cross-validation may help. However, a fully-grown tree may have hundreds of leaves. It requires significant amount of computation to "prune" the tree node by node. To cope with this problem, researchers have proposed two different tree pruning techniques.

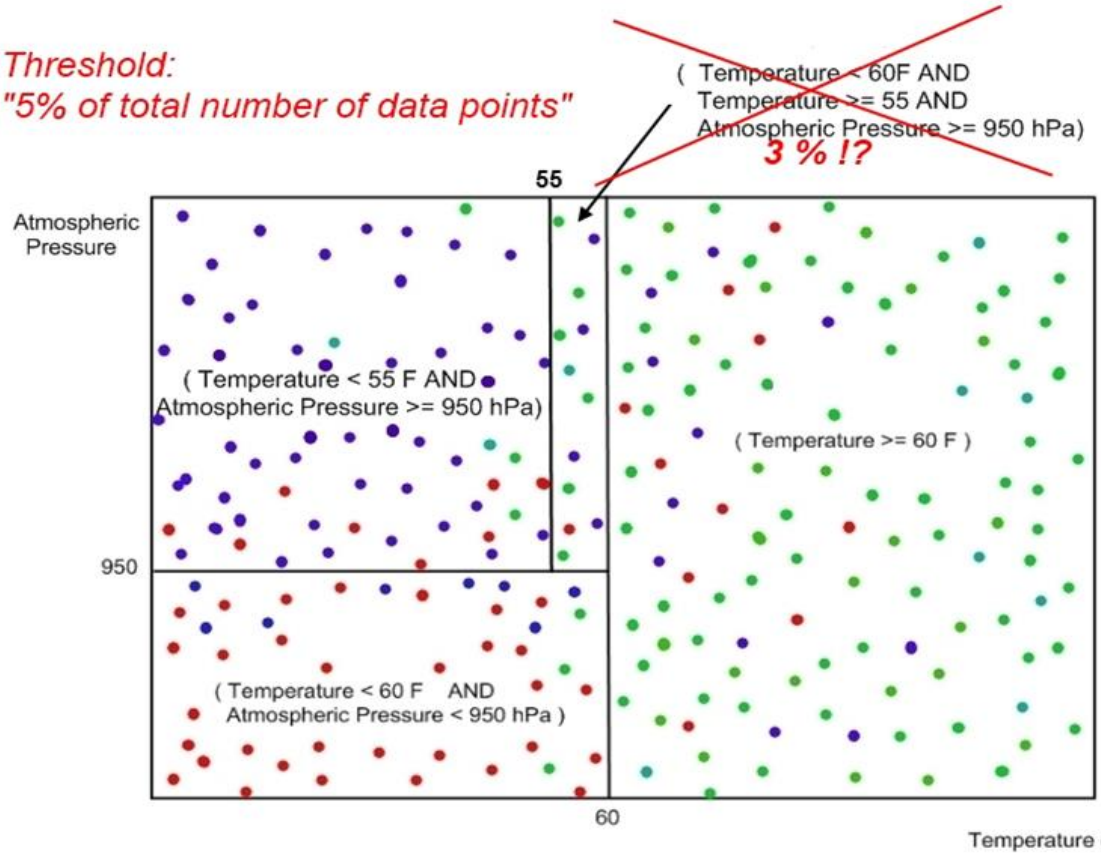
- *Pre-pruning* is to limit the growth of a tree by setting thresholds on tree growing parameters—such as the depth of a tree, the minimum number of data points in a leaf node, and the minimum number of data points for a node to attempt splitting.

- *Post-pruning* is to compute "costs" of removing subtrees of a grown tree, and prune the subtree with minimum cost—a.k.a *cost-complexity pruning* algorithm.

Pre-Pruning

- Pre-pruning is often used to halt the tree growth early by setting a threshold.





Cost-Complexity Post-Pruning

- The *cost-complexity post-pruning* is a process to keep finding a subtree with an inner (non-leaf) node that has minimum cost, and pruning it if necessary. For regression tree, the cost is defined as:

$$Cost = \frac{RSS_{pruned} - RSS_{orig}}{|T_{orig}| - |T_{pruned}|}$$

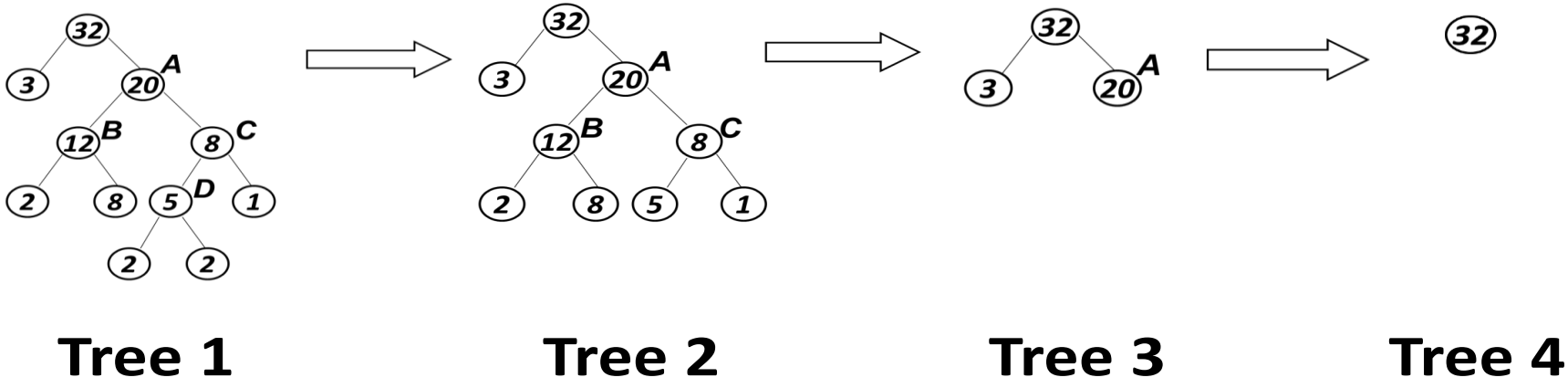
  where $|T|$ is the number of leaf nodes of the tree. Also note that the *RSS* is literally the *sum of total errors* of a tree.

- For classification tree, on the other hand, the errors is the *total misclassified data points*. The cost can be calculated with *Misclassification Rates* (MR), as:

$$Cost = \frac{MR_{pruned} - MR_{orig}}{|T_{orig}| - |T_{pruned}|}$$

Cost-Complexity Post-Pruning (cont.)

- Here is an example. Let's consider pruning a classification tree. Suppose Tree 1 is a fully-grown tree used to classify a dataset with **100** data points. The numbers in nodes denote the number of misclassified data points. We can then get 4 trees after the post-pruning process.



Cost-Complexity Post-Pruning (cont.)

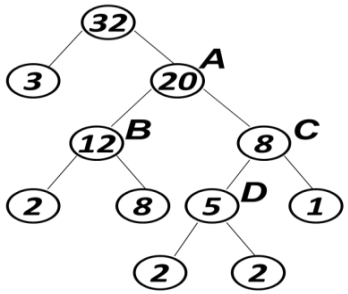
- Here we demonstrate the first tree pruning. In the very beginning, we have a fully-grown tree (Tree 1) with 4 subtrees (**A**, **B**, **C**, and **D**) and 6 leaves. The cost-complexity algorithm first calculates the cost for each of the subtrees if they are pruned, then prunes the subtree with minimal cost. In Tree 1, the cost for each of the subtrees is:

$$Cost_A = \frac{\frac{(3+20)}{100} - \frac{(3+2+8+2+2+1)}{100}}{6-2} = 0.0125$$

$$Cost_B = \frac{\frac{(3+12+2+2+1)}{100} - \frac{(3+2+8+2+2+1)}{100}}{6-5} = 0.02$$

$$Cost_C = \frac{\frac{(3+2+8+8)}{100} - \frac{(3+2+8+2+2+1)}{100}}{6-4} = 0.015$$

$$Cost_D = \frac{\frac{(3+2+8+5+1)}{100} - \frac{(3+2+8+2+2+1)}{100}}{6-5} = 0.01$$



Tree 1

The subtree **D** would thus be pruned.

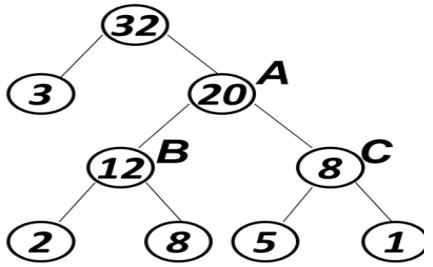
Cost-Complexity Post-Pruning (cont.)

- Now we have 3 inner nodes. And their costs are:

$$Cost_A = \frac{\frac{(3+20)}{100} - \frac{(3+2+8+5+1)}{100}}{5-2} = 0.0166$$

$$Cost_B = \frac{\frac{(3+12+5+1)}{100} - \frac{(3+2+8+5+1}{100}}{5-4} = 0.02$$

$$Cost_C = \frac{\frac{(3+2+8+8)}{100} - \frac{(3+2+8+5+1)}{100}}{5-4} = 0.02$$



Tree 2

- So, we next prune subtree **A** to get Tree 3. We can always continue to prune the tree until a tree with only a root node (Tree 4).

Hands-on Tree Learning

- Let's go back to our *Hitters* data and find a better tree!

```
# Regression tree with 10-fold CV. "cp = 0" indicates a fully-grown tree
set.seed(1); hitSalary_RT = rpart(log(Salary) ~ Years + Hits, Hitters,
                      control = rpart.control(xval = 10, cp = 0))
rpart.plot(hitSalary_RT, digits = 3) # Plot tree
# printcp(): Print errors and cost-complexity parameters
# nsplit: number of splits, rel error: RSS for training data
# xerror: k-fold CV RSS, xstd: k-fold CV standard error
# All columns are re-scaled by root node error (divided by 0.78766)
printcp(hitSalary_RT)
Root node error: 207.15/263 = 0.78766
n=263 (59 observations deleted due to missingness)
        CP nsplit rel error  xerror     xstd
1  0.4445745      0   1.00000 1.01115 0.066083
2  0.1145455      1   0.55543 0.56976 0.060607
3  0.0444602      2   0.44088 0.46771 0.058638
4  0.0183127      3   0.39642 0.43830 0.065139
5  0.0169020      4   0.37811 0.42861 0.065530
6  0.0110721      5   0.36121 0.42204 0.066461
7  0.0096474      6   0.35013 0.45081 0.067790
8   0.0085782      7   0.34049 0.44711 0.067613
...
# Plot cost-complexity parameters (CP) to help select the best tree
plotcp(hitSalary_RT)
```

Hands-on Tree Learning(cont.)

```
# If we'd like a CP table with "real" values:
realCPTable = data.frame(printcp(hitSalary_RT))
Map(function(x) realCPTable[,x] <<- realCPTable[,x] * 0.78766,
   list(1,3,4,5 ))
```

- Also notice that the cost-complexity parameters (*cp*) of package *rpart* is calculated as:

$$cp = \frac{Error_{pruned} - Error_{orig}}{|T_{orig}| * Error_{root}}$$

  where *Error* is RSS for regression tree and MR for classification tree.

- Seems that a tree with 2 or 3 splits (3 or 4 leaves) performs relatively good (more splits does not result in significant decrease of RSS). Let's say we'd like the tree with 3 splits. We can then prune the tree:

```
# Always prune tree with cp threshold higher then cp of the tree we need
hitSalary_RT_3split = prune(hitSalary_RT,cp = 0.019)
printcp(hitSalary_RT_3split)
# Plot the new tree
rpart.plot(hitSalary_RT_3split)
```

Hands-on Tree Learning(cont.)

- The function *rpart*() can automatically select the best (although it's not necessary "the best") tree for you. The selection criterion is "*k*-fold CV Error + 1 SE (standard error)". Take previous fully-grown tree as an example.

```
# k-fold CV + 1 SE
realCPTable$xerr_1std = realCPTable$xerror + realCPTable$xstd
realCPTable
           CP nsplit rel.error    xerror       xstd xerr_1std
1  0.350173515      0 0.7876600 0.7989775 0.05204908 0.8510266
2  0.090222907      1 0.4374865 0.4442035 0.04638970 0.4905932
3  0.035019532      2 0.3472636 0.3644106 0.04502710 0.4094377
4  0.014424165      3 0.3122440 0.3291886 0.04590225 0.3750909
5  0.013313012      4 0.2978199 0.3278793 0.05043186 0.3783112
6  0.008721079      5 0.2845069 0.3254647 0.05080204 0.3762668
7  0.007598884      6 0.2757858 0.3436915 0.05173687 0.3954284
8  0.006756734      7 0.2681869 0.3431666 0.05143842 0.3946050
9  0.003685938      8 0.2614302 0.3395059 0.05104336 0.3905492
...
# Let rpart() select its best tree.
# Just remove the parameter cp in rpart.control().
set.seed(1); hitSalary_RT = rpart(log(Salary) ~ Years + Hits, Hitters,
                        control = rpart.control(xval = 10))
# Plot the tree (6 splits, 7 leaves.)
rpart.plot(hitSalary_RT)
```

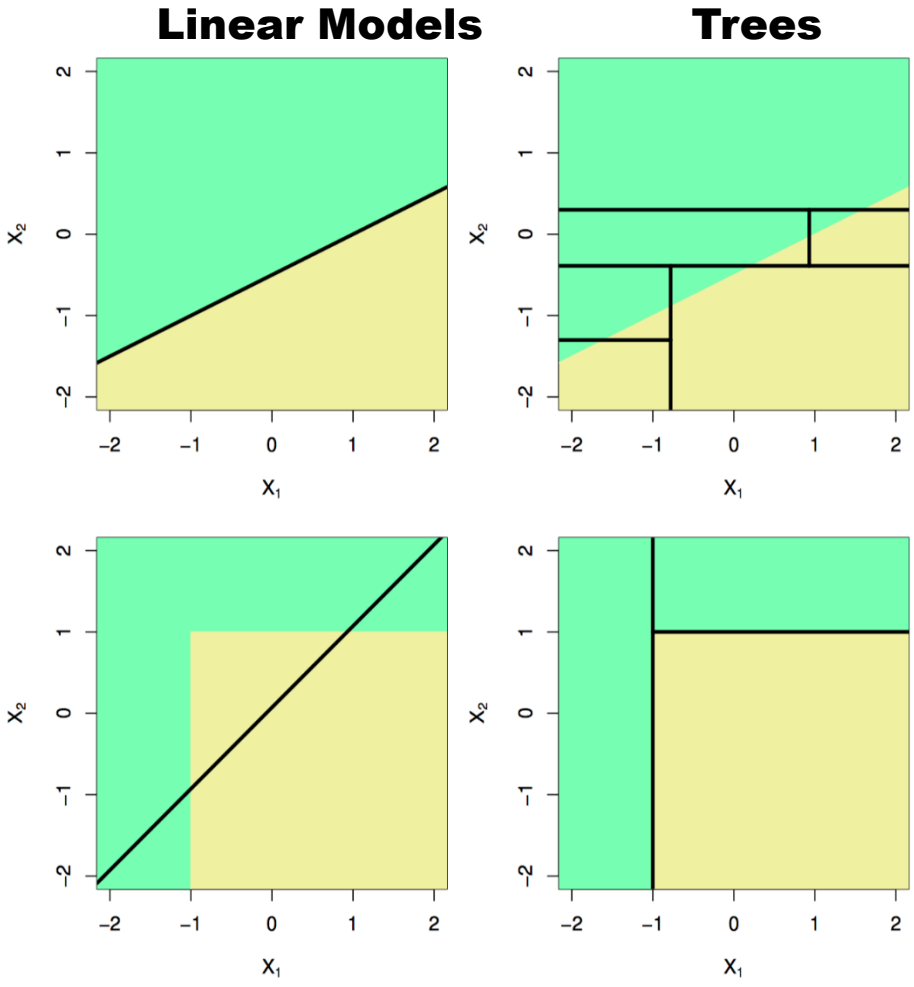
Hands-on Tree Learning(cont.)

- Consider another example, *Heart* data, for classification. As discussed previously, each new split should increase the purity of the classification tree. And the goal here is to find a tree that results in relatively low cross-validation misclassification rate (CV-MR).

```
# Download heart dataset, skip the first column
heart = read.table("http://www-bcf.usc.edu/~gareth/ISL/Heart.csv",
   header = T, sep = ",")[,-1]
# Fully-grown tree with 13 leaves
set.seed(1)
heart_CT = rpart(AHD ~ ., data = heart, control =
   rpart.control( cp = 0, xval = 10))
# Show all possible cost-complexity prunings
plotcp(heart_CT); printcp(heart_CT)
# Let rpart select the best tree (a tree with 5 splits?)
seet.seed(1)
heart_best_CT = rpart(AHD ~ ., data = heart, control =
   rpart.control(xval = 10))
printcp(heart_best_CT)
# Plot it!
rpart.plot(heart_best_CT,extra = 1)
```

Advantages and Disadvantage of Trees

- Trees are comparably easy to be interpreted by people. They can also handle categorical predictors without the need to create dummy variables.
- Unfortunately, trees generally do not have the same level of predictive accuracy as some of the other regression and classification approaches. Another drawback of the trees is that a small change in the data (e.g. remove one column) can cause a large change in the final estimated tree, which is why tree models often suffer the problem of high variance.
- However, by aggregating many trees—to create a "forest", the predictive performance of trees can be substantially improved!



Creating a Forest

- Researchers have found that *an ensemble of multiple models outperforms any individual model*, partly because the various errors of the models "average out"—variance reduction. One of such ensemble techniques is called *Bootstrap Aggregation*, or *Bagging*.

- The basic concepts of the bagging trees is to repeatedly resample (sample with replacement) the training dataset, learn models from these bootstrapped data, and then average (or get the "majority votes" of) all the predicted responses used to predict *out-of-bag* (OOB) data.

- You may have notice that bagging trees also "leaves observations out" and compute prediction errors from these left-out observations. Yes, given sufficient number of bootstrapped samples *B*, the errors of the OOB data is equivalent to the errors of LOOCV and is used to evaluate the performance of a forest.

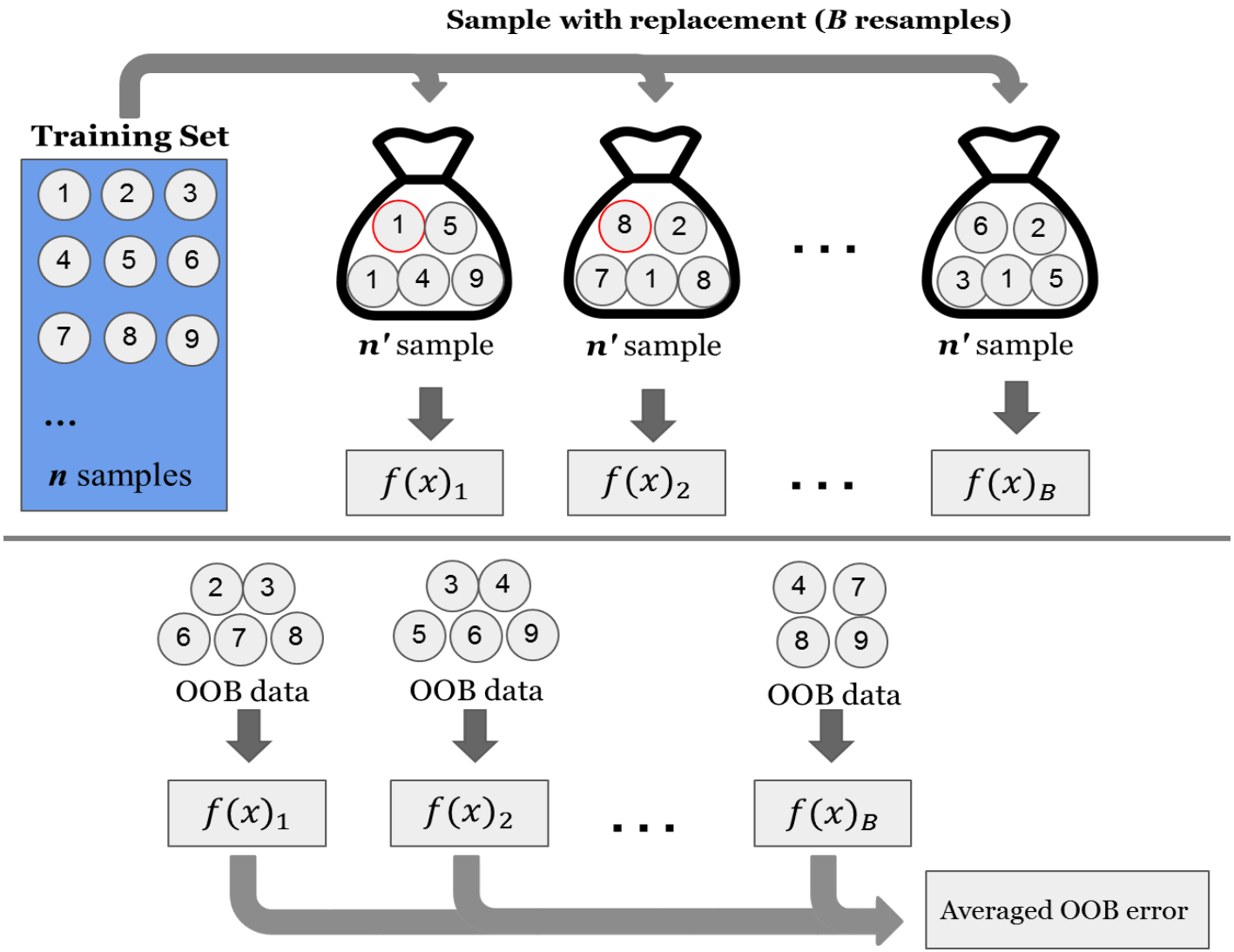
Creating a Forest(cont.)

- Also notice that, the OOB errors are often computed from 1/3 of the observations—*0.632 bootstrapping*. In this case, we re-sample the *whole* training data with n observations($n' = n$). About 36.8% of observations will not be selected for training (i.e. ending up as the OOB data), and the remaining 63.2% will be used as the training data. The 36.8% is simply the probability that an observation is not being chosen $(1 - \frac{1}{n})$ as the training data.

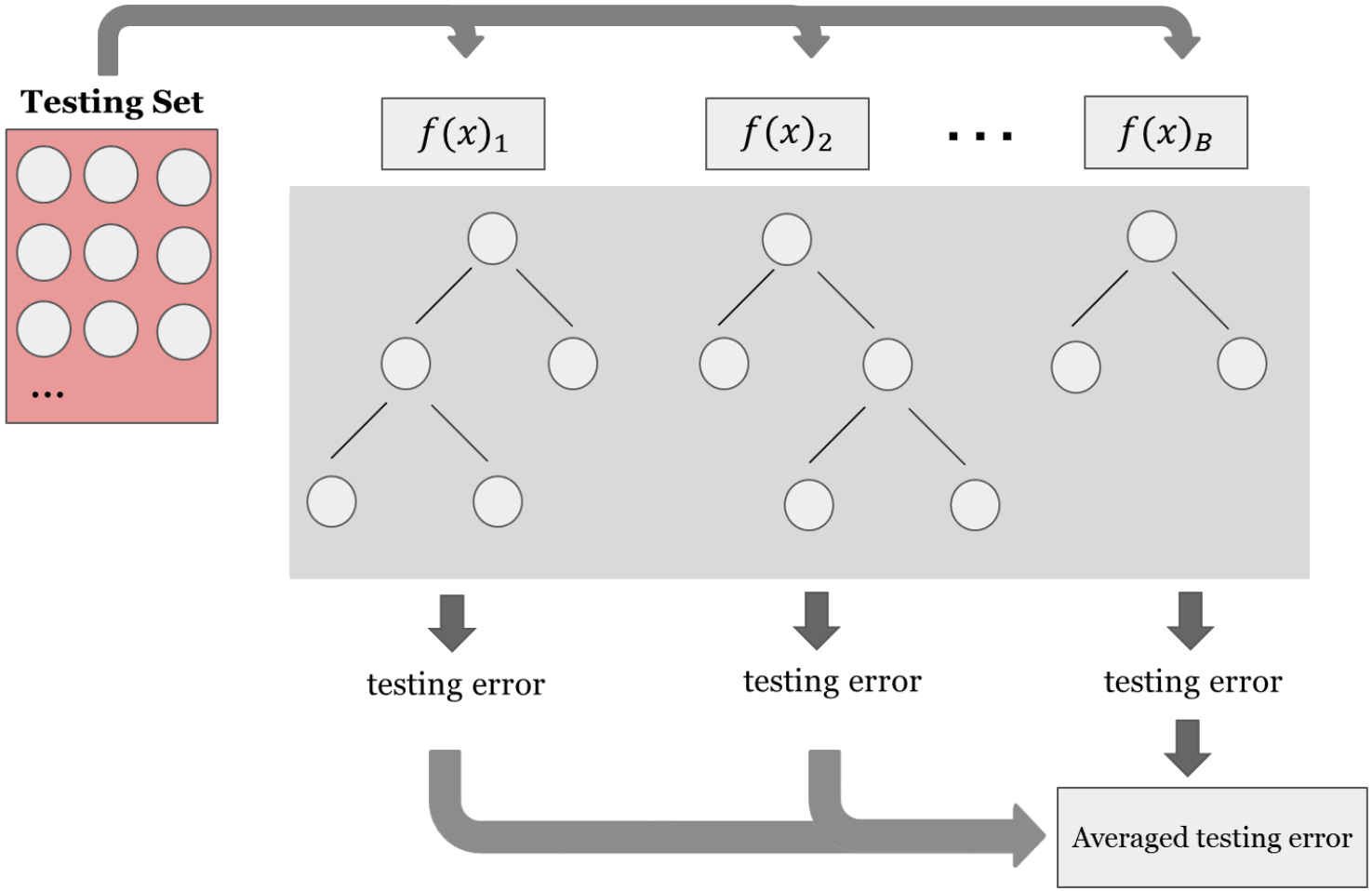
$$(1 - \frac{1}{n})^n \sim 0.368 = e^{-1}$$

- It sounds that, if we do not care about model interpretability, a forest instead of one single tree does a good job for us. Yes, bagging trees may improve model performance. However, all bagged trees may look similar as all predictors are used to learn the trees. That is, those strong predictors are often in the top splits, which may result in high correlations of predictions and therefore lower the accuracy of predictions.

Bootstrap Aggregating, Bagging



Bootstrap Aggregating, Bagging(cont.)



Introduction to Random Forest

- To solve this problem, researchers have proposed building the forest by randomly selecting a subset of features (approximately $p/3$ and $\sqrt{p}$ features for regression and classification respectively, where p is the number of total features)—a *Random Forest* that could "decorrelates" the bagged trees. The random forest learning process can be summarized as:

A. For i = 1 to *B (number of bootstrapped samples)*:
   a) Draw a bootstrapped sample from the training data.
   b) Fully grow a tree from the bootstrapped data by the following steps recursively until a tree pre-pruning threshold (if any) is reached.
      i. Select m ($p/3$ or $\sqrt{p}$) variables at random from the p variables.
      ii. Pick the best variable/split-point among the m.
      iii. Split the node into two child nodes.

B. Output the ensemble of B trees.

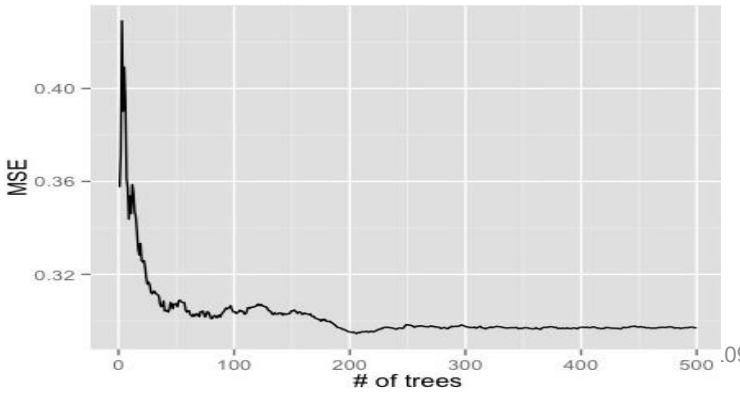
C. Make prediction at new points using average of the predictions (Regression) or majority of votes (Classification).

Hands-on Random Forest

- In R, we can easily build & evaluate a random forest using functions in package *randomForest*. Let's apply the forest learning to our *Hitters* data.

```
# Split dataset into training and testing
library(ISLR); library(randomForest); data(Hitters)
Hitters = na.omit(Hitters);numOfRows = nrow(Hitters)
set.seed(1); train_idx = sample(1:numOfRows,size = numOfRows * 0.7)
test_idx = setdiff(1:numOfRows, train_idx)
# Build a random forest with 500 trees (by default)
set.seed(1)
hitSalary_RF = randomForest(log(Salary) ~ Years + Hits, Hitters[train_idx,])
# Plot the "Errors vs. # of trees". Or just enter "plot(hitSalary_RF)"
qplot(1:500, hitSalary_RF$mse, geom="line") +labs(x= "# of trees", y= "MSE")
```

- It seems that a forest with 200 trees should balance the performance of the model and the need of computation.



Hands-on Random Forest(cont.)

- Let's see how forest models actually perform compared to other modeling techniques, say Regression Tree and Linear Regression Model?

```
# Get RMSEs for 5 different models
hitters_RMSE = Map(function(f){
  set.seed(1)
  fit = f(log(Salary) ~ Years + Hits, Hitters);
  pred_response = predict(fit, newdata = Hitters[test_idx,]);
  return(sqrt(mean( (pred_response - log(Hitters[test_idx,"Salary"]) ) ^2)) );},
  c("RF_50"= function(f, d) randomForest(formula = f, data = d, ntree=50),
       "RF_200"=function(f, d) randomForest(formula = f, data = d, ntree=200),
       "RF_500" = function(f, d) randomForest(formula = f, data = d, ntree=500),
       "RT" = rpart, "LM" = lm))
```

```
$RF_50
[1] 0.318371

$RF_200
[1] 0.3143926

$RF_500
[1] 0.3063058

$RT
[1] 0.4313982

$LM
[1] 0.5225531
```

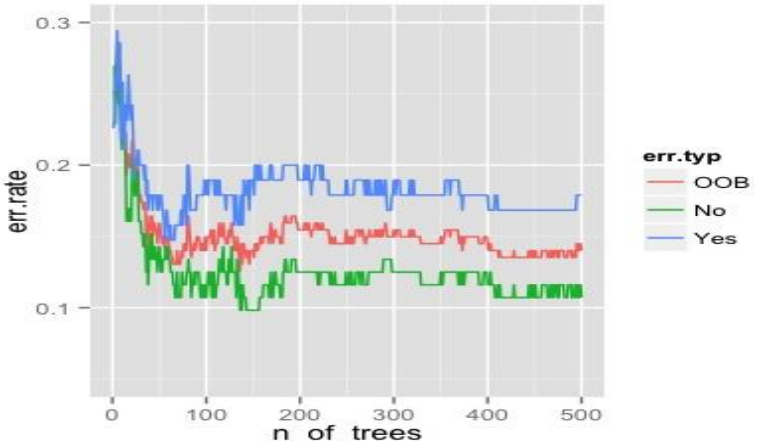
- The result suggests more trees may increase the accuracy of prediction. However, it is not always the case. If the number of features increases, the result may be unstable due to the randomness of feature subset selections.

Hands-on Random Forest(cont.)

- Consider applying the random forests to classification problem. Again, we here plot the "error rates vs. the number of trees" to evaluate the forests.

```
# Heart data again!
heart = na.omit(heart); numOfRows = nrow(heart)
set.seed(1); train_idx = sample(1:numOfRows,size = numOfRows * 0.7)
test_idx = setdiff(1:numOfRows, train_idx)
heart_RF = randomForest(AHD ~ ., heart[train_idx,])
heart_RF_err = reshape2::melt(heart_RF$err.rate)
colnames(heart_RF_err) = c("n_of_trees", "err.typ","err.rate")
qplot(n_of_trees, err.rate, color=err.typ,data = heart_RF_err, geom="line") +
   ylim(0.05, 0.3)
heart_RF
```

```
...
No. of variables tried at each split: 3

        OOB estimate of  error rate: 14.01%
Confusion matrix:
     No Yes class.error
No  100  12   0.1071429
Yes  17  78   0.1789474
```



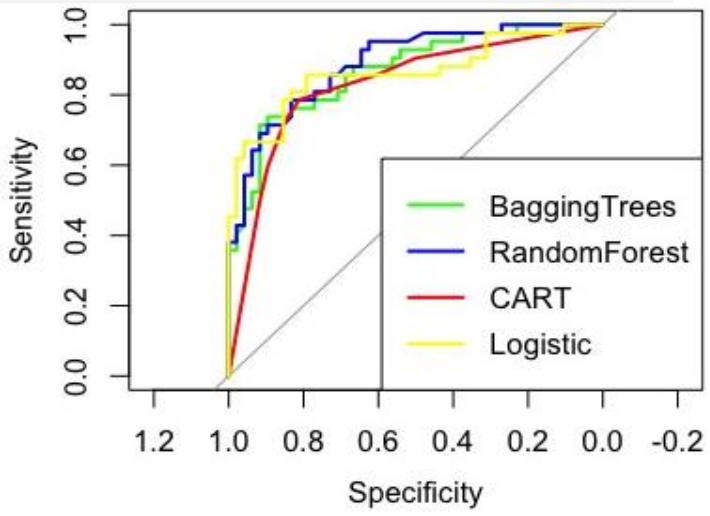
Hands-on Random Forest(cont.)

- The trained forest with default number of trees (*ntree* = 500) gives relatively stable errors (MR is around 14%). Let's see how it actually performs compared to other techniques for classification.

```
library(pROC) # ROC and AUCs for testing data
heart_measures = Map(function(f){
  set.seed(1);
  fit = f(AHD ~ ., heart[train_idx,])

  # Get predicted probability for ROCs
  if(class(fit)[1] == "glm"){
    pred_prob = predict(fit, newdata = heart[test_idx,],
                type = "response");
  }else{
    pred_prob = predict(fit, newdata = heart[test_idx,],
                type = "prob")[, "Yes"];
  }
  return(roc(heart[test_idx, "AHD"], pred_prob));
 }, c("RF_500" = randomForest,"CT" = rpart,
       "Logistic" = function(f, d) glm(formula = f, data = d,
           family="binomial") ))
```

Hands-on Random Forest(cont.)

```
# By setting mtry = # of all variables, we can get "bagging trees" result
set.seed(1)
heart_BT = randomForest(AHD ~ ., heart[train_idx,], mtry = (ncol(heart) - 1))
heart_BT_pred_prob = predict(heart_BT,
                      newdata = heart[test_idx,], type = "prob")[, "Yes"];
heart_BT_roc = roc(heart[test_idx, "AHD"], heart_BT_pred_prob)
# Plot ROCs
plot.roc(heart_BT_roc, col = "green")
plot.roc(heart_measures[[1]], add = T, col = "blue")
plot.roc(heart_measures[[2]], add = T, col = "red")
plot.roc(heart_measures[[3]], add = T, col = "yellow")
legend("bottomright", legend=c("BaggingTrees",
  "RandomForest", "CART","Logistic"),
  col=c("green", "blue", "red", "yellow"), lwd=2)

# Get all AUCs
heart_BT_roc$auc
Area under the curve: 0.8683
Map(function(x) x$auc, heart_measures)
$RF_500
Area under the curve: 0.8899
$CT
Area under the curve: 0.8299
$Logistic
Area under the curve: 0.8641
```



Using Random Forest

- Random Forest has become one of the most popular off-the-shelf learning algorithm. In the recent years, it is even a baseline algorithm used in comparison with newly proposed techniques. However, it does have some weakness you should consider.

- More trees in the forest would most likely increase its accuracy of predictions, but learning more trees certainly requires significant amount of computations. Some literatures suggest a range between *64 and 128 trees* in a forest, which may obtain a good balance between accuracy and computation complexity. Thanks to recent cheaper computations, however, researchers recommend *adding trees until we get a stable OOB estimate of errors*.

- Another issues about using random forest is that, as it keeps randomly selecting *m* out of *p* features during its learning process, *its performance may be unstable when p is large*. Again, one simple solution is to keep adding more trees until the errors of OOB data become stable!

Unsupervised Learning

- We have been discussing several supervised learning methods. In such the cases, we observe a set of features $X_1, X_2, \ldots, X_p$ as predictor variables as well as a response/outcome variable Y . We understand that the goal is to predict Y using $X_1, X_2, \ldots, X_p$.

- Here we instead focus on *unsupervised learning*, where we observe only the features $X_1, X_2, \ldots, X_p$. As we do not have an associated response variable Y, we are concerned about the structures or patterns learned from the data.

- The goal here is to discover interesting patterns among $X_1, X_2, \ldots, X_p$. For example, is there an informative way to visualize the data? Can we discover subgroups among the variables or among the observations?

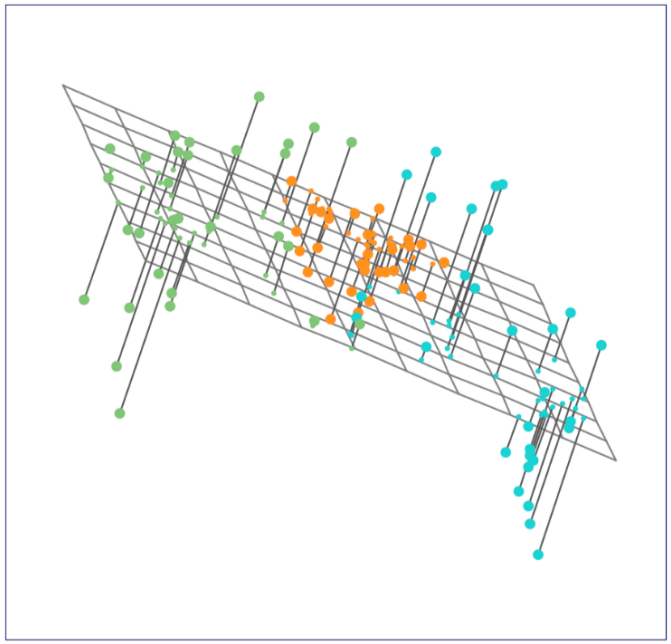
Challenge of Unsupervised Learning

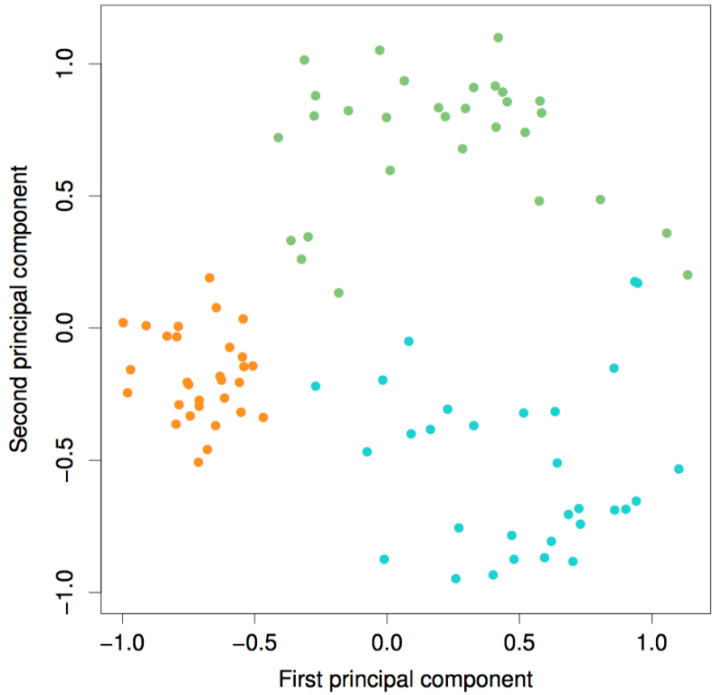
- Unsupervised learning is more subjective than supervised learning, as there is no simple goal (e.g. better predicting an outcome) for the analysis. Another big challenge is that it is hard to obtain labeled (well-defined outcome) data—labeled data often requires human intervention and cost relatively high. For example, it is difficult to automatically assess the overall sentiment of a movie review: is it favorable or not?

- Techniques for unsupervised learning are growing importance in many fields, e.g. subgroups of breast cancer patients grouped by their gene expression measurements; groups of shoppers characterized by their browsing and purchase histories; movies grouped by the ratings assigned by movie viewers.

Dimensionality Reduction

- In many applications, even if the data is of high dimensions, it may come from an intrinsic lower dimensional space that represent features of higher-level concepts. By distilling the important information from the data, we could remove noise within the data, reveal the hidden patterns (signals), and reduce the size of the data.

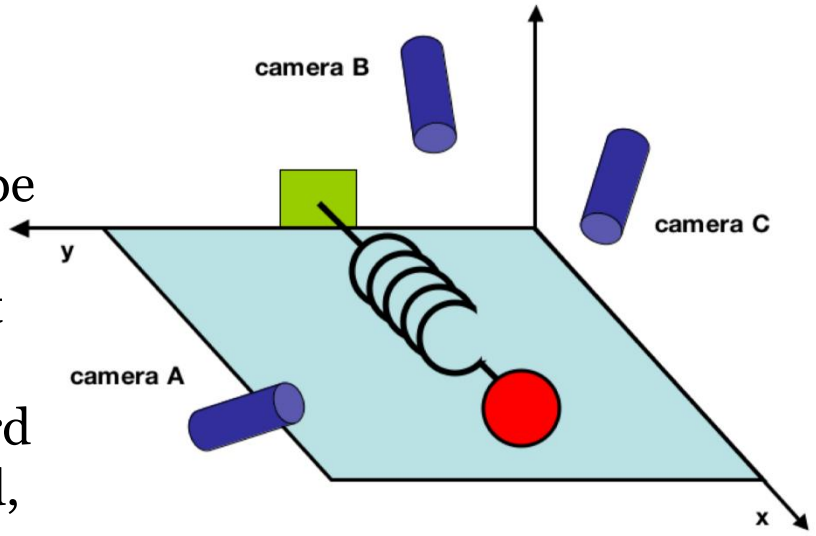




Principal Components Analysis

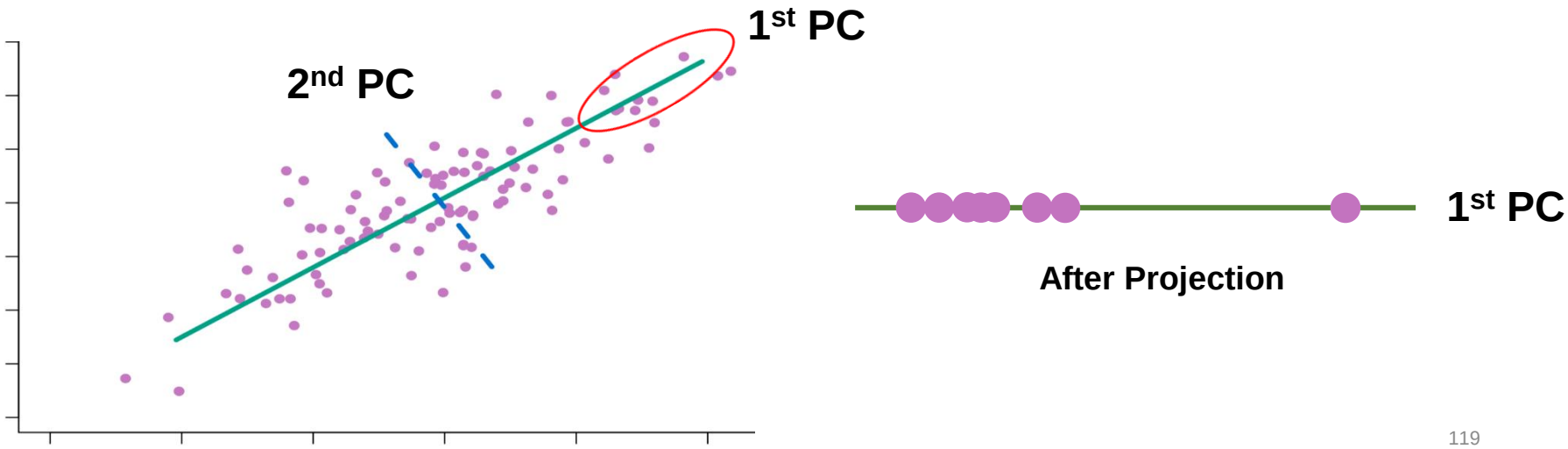
- *Principal Components Analysis* (PCA) has become one of the most popular dimensionality reduction techniques. It produces a low-dimensional representation of a dataset by finding a sequence of linear combinations of the variables $(X_1, X_2, \ldots, X_p)$ that have maximal variance, and are mutually uncorrelated.

- Consider an example measuring a toy ball's dynamics. We expect the ball should move back a forth along the x-axis—the underlying dynamics should be expressed as a function of a single variable x. However, suppose we do not know the dynamics of the ball, to measure the dynamics, we should record more dimensions than we actually need, as the ball may move along other axes!



Principal Components Analysis (cont.)

- Again, the goal of PCA is to replace a large number of correlated variables with a smaller number of uncorrelated variables while capturing as much information in the original variables as possible. In the previous ball dynamics example, using PCA may help determine whether the ball dynamics along the *x*-axis is the most important.
- Below figure shows how PCA works. It re-expresses (projects) data onto new basis vectors (*n*th principal component) to maximize variance, which may extract important part of the data without losing too much information.



Principal Components Analysis (cont.)

- Let's define PCA. The derived new variables, called *n*th principal components (PCs), are *normalized* linear combinations of the observed variables $(X_1, X_2, \dots, X_p)$. Specifically, the 1st principal component

$$PC_1 = a_1X_1 + a_2X_2 + \dots + a_pX_p$$

  is the weighted combination of the p observed variables that accounts for the most variance in the original set of variables. The 2nd principal component accounts for the most variance in the original variables, under the constraint that it is orthogonal (uncorrelated) to the 1st principal component. Each subsequent PCs maximizes the variance accounted for, while remaining uncorrelated with all previous components. Theoretically, you can extract up to $min(n - 1, p)$ PCs for a dataset with n observations by p variables.

- Also notice another constraint that the sum of squares of the *loadings* $(a_1, a_2, \dots, a_p)$, $\sum_i^p a_i^2 = 1$ is used here to avoid getting arbitrarily large variance of PCs.

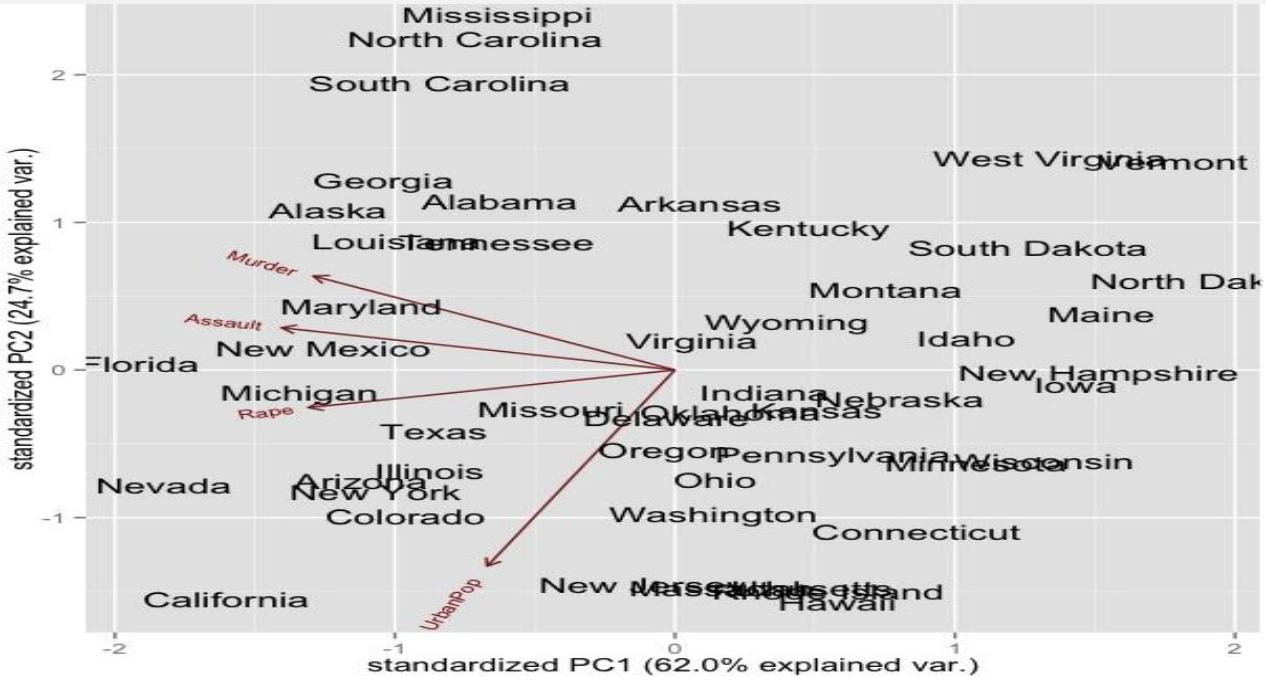
Computation of Principal Components

- Suppose we have an n by p dataset X. Since we are only interested in the variance, the covariance matrix of X could therefore play as a "transformation matrix" that de-correlates the data and obtain the PCs. Specifically, we can do eigen-decomposition of the covariance matrix to obtain PCs with their loading vectors. Take R built-in dataset *USArrests* as the example. The PC score vectors have length n = 50, and the loading vectors have length p = 4. PCA was performed after standardizing each variable to have mean 0 and standard deviation 1.

```
# Principal component analysis on "USArrests"
data(USArrests)
X = scale(as.matrix(USArrests)) # "Center" (standardize) the data
cov_X = cov(X) # Covariance matrix
# Eigen decomposition of the covariance matrix
eigen_judge = eigen(cov_X)
eigen_judge$vectors[,1] # 1st PC loadings (weights)
# 1st PC score for each observation
as.matrix(X) %*% matrix(eigen_judge$vectors[,1], ncol = 1)
# sum of squared loadings is 1
sum(eigen_arrest$vectors[,1] ^ 2)
```

Computation of Principal Components (cont.)

```
# A set of PCA functions help you do PCA. No matrix operations needed.
pc_arrest = princomp(X, scores = T) # PCA using princomp()
pc_arrest$loadings; pc_arrest$scores # loadings & scores of each PC
pc_arrest$sdev ^2 / sum((pc_arrest$sdev)^2) # % of explained variance
biplot(pc_arrest) # A biplot should help you understand more about PCs
# library("devtools"); install_github("vqv/ggbiplot")
ggbiplot(pc_arrest,labels = rownames(X),labels.size = 5 )
```



How many PCs to extract?

- You may wonder how many PCs we should consider in applications? Several criteria are available for deciding how many components to retain in a PCA:

1. Depends on domain experts' prior experience and/or theory

2. Select the number of components needed to account for some threshold cumulative amount of variance in the variables. For example, we set the threshold "80%", and the first two PCs explain 86% variance in previous PCA on *USArrests* data.

3. Examines the eigenvalues of the p by p correlation matrix among the variables. Kaiser–Harris criterion suggests retaining components with eigenvalues greater than 1. Components with eigenvalues less than 1 explain less variance than contained in a single variable.

Limitations of PCA

- PCA also makes assumptions and therefore has some limitations/drawbacks we should be aware of:

1. PC scores are linear combinations of weighted features, which assume linearity of the problems.

2. PCs with larger variances imply higher degrees of importance. PCA assumes that PCs with large associated variances represent important patterns/signals, while those with lower variances represent noise, which is incorrect in some applications.

3. PCs are orthogonal, which suggests that PC scores (new features) should represent different concepts. It is not always the case in real-world applications. For example, in *topic modeling* of text analytics, there may have some overlapping abstract topics that consist of the same terms (e.g. "interest rate" may be part of topic "Business" and "Finance" in large corpus of news articles).

Applications of PCs

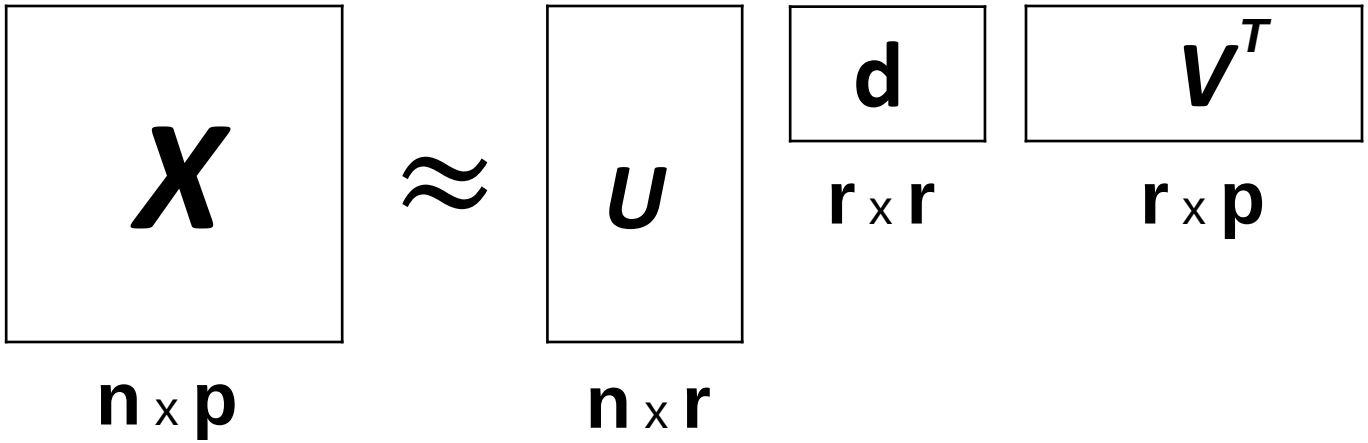
- You may wonder, except the dimensionality reduction, what these PCs can do for us? Yes, we have introduced the *biplot*. PCs are used to present information (connections) between observations and features graphically. These PCs may be used to identify clusters in a set of dataset.

- Also, the scores of PCs can actually be used as predictors and even the outcomes in supervised learning algorithms, although another close-related technique, *Factor Analysis* (FA), is more appropriate in such cases.

- PCA is often used in data reduction to represent *the total variable variance* using fewer variables, but not to detect the latent construct or factors among variables. FA, on the other hand, is based on *correlation structures of features*. It is often used to create conceptual factors that reproduce the inter-correlations among variables.

Singular Value Decomposition

- *Singular Value Decomposition* (SVD) is a generalization of the eigen-decomposition and a matrix factorization technique widely used in many different fields and applications. It decomposes an n by p data matrix $\boldsymbol{X}$ into the product of three matrices—a low-rank approximation with rank $\boldsymbol{r}$, as:



where $\boldsymbol{U}$ and $\boldsymbol{V}$ can be regarded as *how much information of* $\boldsymbol{X}$ *about rows/observations and columns/variables* respectively. And diagonal matrix $\boldsymbol{d}$ can be considered *the importance of the information*.

Latent Semantic Analysis using SVD

- Here we consider a popular application of SVD—*Latent Semantic Analysis* used to analyze relationships between a set of documents and the terms they contain. The relationships are often represented as a *Document-Term Matrix* (DTM) that contains term frequencies in corresponding documents. SVD is here used to find a low-rank approximation to the DTM—a lower-dimensional semantic map used in applications to find relations among documents and terms.

- R does provide a set of text analytics packages that allow you to do various matrix factorizations and DTM manipulations. Here, however, we only consider a simple DTM as the example.

```
# A simple DTM
dtm = data.frame("programming"=c(2,0,3,0,3), "database"=c(3,3,4,0,0),
                 "SQL" = c(1,2,5,0,0), "finance" = c(0,2,0,5,0),
                 "rate" = c(0,2,0,2,0),
                 "market" = c(0,1,0,3,0), "data" = c(1,2,0,2,2))
rownames(dtm) = paste("doc", 1:5, sep = "_")
# SVD on the DTM
dtm_svd = svd(dtm)
```

Latent Semantic Analysis using SVD (cont.)

- Similar to PCA, the first few columns/rows of matrix $\boldsymbol{U}/\boldsymbol{V}$ contain the most important information (higher variances) about documents/terms extracted from the DTM. We may just consider these columns/rows in later applications. For example, we can project the first 2 columns/rows of matrix $\boldsymbol{U}/\boldsymbol{V}$ onto 2-D plane to form a semantic map, which could help us identify similarities or relationships among documents and terms.

```
# Create a data frame used to plot the terms/documents
# in 2-D semantic map
dtm_loc = data.frame("doc_term" = c(rownames(dtm),colnames(dtm)),
                     "dt_type" = c(rep("document",5), rep("term",7)),
                     "x1" = c(dtm_svd$u[,1], dtm_svd$v[,1]),
                     "x2" = c(dtm_svd$u[,2], dtm_svd$v[,2]) )

ggplot(dtm_loc, aes(x = x1, y = x2, color=factor(dt_type))) +
   geom_point(size = 3) + geom_text(aes(label=doc_term),hjust=0,
   vjust=0) + xlim(-0.75,0) + xlab("Topic 1") + ylab("Topic 2")
```

Latent Semantic Analysis using SVD (cont.)

- We can do further analysis on the data points in the 2-D space below. For example, we understand that the new axes represent the higher-level concepts of documents and terms—they are abstract "topics" extracted from the DTM. However, one big drawback of such approach is that these topics must be uncorrelated/independent, which is a bit unrealistic, as topics may share similar terms in real-world texts. We will soon talk about it in the following unit about text analytics.

SVD and PCA

- We understand that PCA is simply to identify one (or multiple) new basis, which is a linear combination of the original basis, that best represents (maximizes variance) our data. Previously, we used eigen-decomposition on covariance matrix so as to find new basis vectors and decorrelate our data. SVD is considered a generalization of eigen-decomposition. It can also decompose the data matrix (as eigen-decomposition does) without lose some information. Specifically, SVD can find new ways (axes) to represent the columns (variables) while retain the information about the "rows" (observations) of a dataset.

```
# PCA using SVD
data(USArrests); X = scale(as.matrix(USArrests)) # Center the data
svd_X = svd(X)

# We can recover the original scaled matrix
X_prime = svd_X$u %*% diag(c(svd_X$d[1:4]) ) %*% t(svd_X$v)

# Low-rank approximation (say rank = 2) or "noise reduction"
# of the original matrix
X_prime = svd_X$u %*% diag(c(svd_X$d[1:2], 0, 0) ) %*% t(svd_X$v)
# Each columns in V contains loadings of each PC
svd_X$v
```

SVD and PCA(cont.)

```
# Each column in U contains information about the observations
# Let's plot them onto a 2-D "concept map".
# We may evaluate the similarity of states in terms of PC1 and PC2
U = data.frame(svd_X$u)
rownames(U) = rownames(USArrests)
colnames(U) = paste("PC_", 1:4, sep = "")
ggplot(U, aes(x = PC_1, y = PC_2, )) + geom_point(size = 3)
  + xlim(-0.3, 0.4) + geom_text(aes(label=rownames(U)),
    hjust=0, vjust=0, size = 4) + xlab("PC 1") + ylab("PC 2")
```