Key Machine Learning

- In regression method, the most commonly used measure is the mean squared error. MSE is small if the predicted responses are very close to the true responses and will be large if some of the observations, the predicted and true responses differ substantially.
- Expected MSE: refers to the average test MSE that we would obtain if we repeatedly estimated f using a large number of training sets and tested each x0
- Variance refers to the amount by with f hat would change if we estimated it using a different training set. If a method has a high variance then small changes in the training data can result in large changes in f hat. More flexible statistical methods have higher variance
- Bias: error that is introduced by approximating a real-life problem. More flexible methods result in less bias.

K-Nearest Neighbors

- When K = 1, the decision boundary is overly flexible and finds patterns in the data that don't correspond to the Bayes decision boundary. Has low bias, but very high variance. As K grows, the method becomes less flexible and produces a decision boundary that is close to linear. Low variance but high bias

- Need to install "class" package

- The K-nearest neighbors (KNN) classifier is an example of a classification method that attempts to estimate the Bayes conditional distributions, and then classify a given observation to the class with highest estimated probability

- The knn() function requires 4 parameters:

    - 1. A matrix containing the predictors (the X's) associated with the training data
    - 2. A matrix containing the predictors (the X's) associated with the data for which we wish to make predictions, for the validate set, for the training set
    - 3. A vector containing the class labels (the Y's) for the training observations
    - 4. A value for K, the number of nearest neighbors to be # used by the classifier.

Linear Regression

- Residual – Difference between the ith observed response value and the ith response value that is predicted by the model
- RSS – Residual Sum of Squares
- Null hypothesis – There is no relationship between X and Y
- Alternative hypothesis – There is a relationship between X and Y
- If we see a small p-value then we can infer that there is an association between the predictor and the response – we reject the null hypothesis – that is, we declare a relationship to exist between X and Y – if the p-value is small enough.

- RSE – Residual Standard Error – is an estimate of the standard deviation of the error – roughly speaking, it is the average amount that the response will deviate from the true regression line. It is considered a measure of the lack of fit of the model.
- R-Squared - provides an alternative measure of fit. It takes the form a proportion – the proportion of variance explained.
- TSS – Total Sum of Squares – measures the total variance in the response Y – can be thought of as the amount of variability inherent in the response before the regression is performed. In contrast, RSS measures the variability that is left unexplained after performing the regression.
- F statistic – when there is no relationship between the response and predictors, one would expect the F-statistics to take on a value close to 1.
- Forward Selection – begin with the null model – the model that contains the intercept but no predictors. We then fit p simple linear regressions and add to the null model the variable that results in the lowest RSS. We then add to that model the variable that results in the lowest RSS for the new 2-variable model. This approach is continued until some stopping rule is satisfied.
- Backward selection – start with all variables in the model, and remove the variable with the largest p-value – which is the variable with the that is the least statistically significant. This continues until a stopping rule is reached
- Backward selection cannot be used if p > n
- Potential problems when we fit a linear regression model to a particular data set:
    - Non-linearity of the response-predictor relationships
    - Correlation of error terms
    - Non-constant variance of error terms
    - Outliers
    - High-leverage points
    - Collinearity
        - To assess, the best way is the compute the variance inflation factor (VIF)
        - The smallest possible value for VIF is 1, which indicates the complete absence of collinearity
        - A VIF that exceeds 5 or 10 indicates a problematic amount of collinearity
- In R:
    - lm(y ~ x, data) used to fit a simple linear regression model
        - Where y is the response, x is the predictor, and data is the data set in which these variables are kept.
    - If you do summary()
        - Gives the p-values, and standard errors for the coefficients. In addition, gives the R-Squared and F-statistic for the model
    - Confidence interval is done by confint()
    - predict() can be used to produce confidence intervals and prediction intervals (see bottom of page 111)
    - To draw a line: abline() and if want to draw a line with intercept a and slope b, type abline(a, b)
    - To increase the width of the regression line: lwd = 3 will increase it by a factor of 3
    - To create difference plotting symbols: use the pch option
    - par(mfrow = c(2,2)) divides the plotting window into a window of 2x2
    - For multiple linear regression, we also use the lm() function

CHOOSING THE BEST SUBSETTING METHOD

In order to select the best model with respect to test error, we need to estimate this test error. There are two common approaches:

1. We can indirectly estimate test error by making an ***adjustment*** to the training error to account for bias due to overfitting.
2. We can ***directly*** estimate the test error, using either a validation set approach or a cross-validation approach.

Adjustment Approaches

These approaches can be used to select among a set of models with different numbers of variables.

C_p

$$C_p = \frac{1}{n}\left(\text{RSS} + 2d\hat{\sigma}^2\right)$$

d = number of predictors
$\hat{\sigma}^2$ = estimate of variance of ε associated with each response

The C_p statistic adds a penalty of $2d\hat{\sigma}^2$ to the training RSS to adjust for the fact that the training error tends to underestimate the test error. When determining which is best, we choose the model with the **lowest** C_p value.

AIC

$$\text{AIC} = \frac{1}{n\hat{\sigma}^2}\left(\text{RSS} + 2d\hat{\sigma}^2\right)$$

The AIC criterion is defined for a large class of models fit by maximum likelihood. For least squares models, C_p and AIC are proportional to each other.

BIC

$$\text{BIC} = \frac{1}{n\hat{\sigma}^2}\left(\text{RSS} + \log(n)d\hat{\sigma}^2\right)$$

n = number of observations

The BIC statistic generally places a heavier penalty on models with many variables, and hence results in the selection of smaller models than C_p. Generally, we select the model that has the **lowest** BIC value.

Adjusted R^2

$$\text{Adjusted } R^2 = 1 - \frac{\text{RSS}/(n-d-1)}{\text{TSS}/(n-1)}$$

While RSS always decreases as more variables are added to the model, RSS/($n - d - 1$) may increase or decrease due to the presence of d in the denominator. The model with the **largest** adjusted R^2 will have only correct variables and no noise variables. Adjusted R^2 is not as motivated in statistical history as C_p, AIC, and BIC.

Alternate Calculations of AIC and BIC

$$AIC = 2d - 2l$$

$$BIC = l - \frac{d \cdot log(n)}{2}$$

l = log likelihood

Direct Estimates

We can compute the validation set error or the cross-validation error for each model under consideration, and then select the model for which the resulting estimated test error is smallest.

Advantages to Directly Estimating

- Provides a direct estimate of the test error
- Makes fewer assumptions about the true underlying model
- Can be used in a wider range of model selection tasks

One-Standard-Error Rule

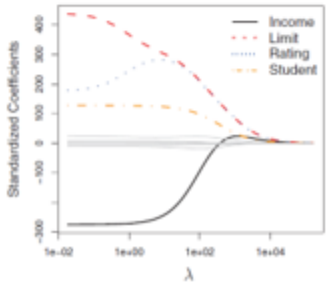
We first calculate the standard error of the estimated test MSE for each model size, and then select the smallest model for which the estimated test error is within one standard deviation of the lowest point on the curve. The reasoning behind this is that if a set of models appear to be more or less equally good, then we might as well choose the simplest model.

R Example (Hitters Data)

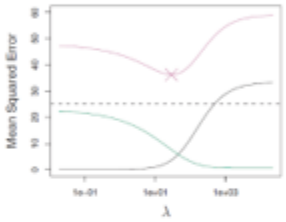
```
Hitters = na.omit(Hitters)
set.seed(1)
train = sample(c(TRUE,FALSE), nrow(Hitters), rep=TRUE)
test=(!train)
regfit.best=regsubsets(Salary~.,data=Hitters[train,],nvmax=19)
test.mat=model.matrix(Salary~.,data=Hitters[test,])
val.errors=rep(NA,19)
for(i in 1:19){
  coefi=coef[regfit.best,id=i)
  pred=test.mat[,names(coefi)]%*%coefi
  val.errors[i]=mean((Hitters$Salary[test]-pred)^2) }
(min<-which.min(val.errors))
coef(regfit.best,min)
regfit.best=regsubsets(Salary~.,data=Hitters,nvmax=19)
coef(regfit.best,10)
k=10
set.seed(1)
folds=sample(1:k,nrow(Hitters),replace=TRUE)
cv.errors=matrix(NA,k,19, dimnames=list(NULL, paste(1:19)))
for(j in 1:k){
 best.fit=regsubsets(Salary~.,data=Hitters[folds!=j,],nvmax=19)
 for(i in 1:19){
   pred=predict(best.fit,Hitters[folds==j,],id=i)
   cv.errors[j,i]=mean( (Hitters$Salary[folds==j]-pred)^2) }}
mean.cv.errors=apply(cv.errors,2,mean)
reg.best=regsubsets(Salary~.,data=Hitters, nvmax=19)
coef(reg.best,min)
(cv.err<-mean.cv.errors[min])
```

Shrinkage Methods: Ridge Regression

- **Previously:** The previous sections discussed fitting linear models to a subset of predictors. This was done with the goal of increasing model interpretability.
- **Shrinkage:** This section introduces the first of two shrinkage methods. The goal of a shrinkage method is to constrain the model's coefficient estimates to reduce model variance. This is done at the expense of model bias.
- **Ridge Regression:** The ordinary least squares regression method builds a model by minimizing the RSS.
  $RSS = \sum_{i=1}^{n}(y_i - \beta_o - \sum_{j=1}^{p}\beta_j x_{ij})^2$
  However, in ridge regression a shrinkage penalty is added.
  $Min -> RSS + \lambda\sum_{j=1}^{p}\beta_j^2$
- **Lambda 0 -> infinite:** When lambda is 0 the shrinkage penalty will have no effect, and produce a least squares model. As lambda approaches infinity it will force the regression coefficients to be 0.
- **Flexibility as Lambda grows:** as lambda grows from 0 (no effect) to infinity (large effect) the flexibility will decrease. This should make sense as regression is attempting to limit model variance. The figure below shows the relationship between the size of lambda and the size of the coefficients. As the size of the coefficients decreases so does the potential flexibility and variance of the model.



- **Bias Variance tradeoff:** The goal of ridge regression is to increase the model's predicitve ability by lowering it's variance. This comes at a trade off of increased model bias. In othewords, ridge regression is no longer attempting to provide the **BLUE** estimators. It is attempting to provide the **BLE** estimators. The graph below shows the relationship between lambda, model variance, and model bias.
- **Considerations:**



  - Ridge regression does not affect the intercept β_0.
  - Ridge regression should be performed on standardized predictor values.
  - Ridge regression does not perform variable selection. A full set of predictors is still used.
- **Ridge Regression in R:**
  - **glmnet()** can be used for ridge regression when the argument alpha = 0.
  - **lm.ridge()** is a ridge specific argument that can also be used.
  - **model.matrix**() is used to build a matrix of the predictors and will automatically convert qualitative variables to factors.
  - **Output:** The outputs give the regression coefficients associated with the input lambdas.
  - **cv.glmnet()** is used to perform cross validation on ridge regression when alpha = 0 and find the best lambda.

THE LASSO

PURPOSE	• Extending on linear model framework • Ways in which the simple linear model can be improved • Replacing the plain least squares fitting with some alternative fitting procedures
GOAL	• Use alternative fitting procedures to yield better prediction accuracy and model interpretability
PREDICTION ACCURACY	• Improvements in the accuracy with which we can predict the response for observations not used in the model training
MODEL INTERPRETABILITY	• Including irrelevant variables leads to unnecessary complexity in the resulting model • Techniques to automatically perform feature selection or variable selection (excluding irrelevant variables from a multiple regression model)

ALTERNATIVE APPROACHES TO LEAST SQUARES

Method	Approach	Techniques
Subset Selection	1. Identifying a subset of the p predictors that we believe to be related to the response 2. Fit the model using least squares on the reduced set of variables	Best Subset Selection Forward Stepwise Selection Forward Stepwise Selection
Shrinkage	1. Fit a model involving all p predictors 2. Estimated coefficients are shrunken towards zero relative to the least squares estimates	Ridge Regression Lasso
Dimension Reduction	1. Projecting the p predictors into M-dimensional subspace 2. Use projections as predictors to fit a linear regression model by least squares	Principal Components Partial Least Squares

SHRINKAGE MINIMIZATION TECHNIQUES

Method	Aims to Minimize…	Use When…	Code
Least Squares	$RSS = \sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij}\right)^2$	Use when variance isn't already a problem (already low)	lm(y~x, data=dataset)
Ridge Regression	$RSS + \lambda\sum_{j=1}^{p}\beta_j^2$	Use when all predictors are related to the response and coefficients are of roughly equal size	glmnet(x, y, alpha=0, lambda=grid)
Lasso	$RSS + \lambda\sum_{j=1}^{p}\lvert\beta_j\rvert$	Use when the predictors are divided between having substantial coefficients and small/zero coefficients	glmnet(x, y, alpha=1, lambda=grid)

PCA - Principal Component Analysis (Team 9)

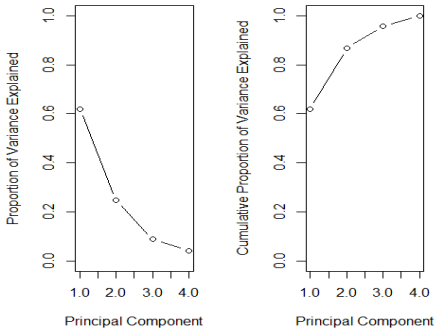
- **What is Dimension Reduction?** – reducing the number of variables in a data set to a more manageable amount (m) while preserving the majority of the data's variability

- **What does PCA do?** – reduce the dimensions/variance of a dataset while retaining as much model variability as possible

- **When do you use PCA?** – when you have a *large set of correlated predictors* with a *high variance*

- **What to keep in mind** – in using PCA, the created coefficient set (m) has a reduction in variance, but at the cost of introducing bias by constraining the coefficients (increases as p increases relative to n)

- **1st Principal Component** – The vector along which the data varies the most
    - 2nd Principal component is that in which the data varies next most, orthogonal to 1st (uncorrelated) and so forth
        - Each additional PC adds less explanation of variation than the previous, summing to 100%



$$Z_m = \sum_{j=1}^{p} \phi_{jm} X_j$$

- **PC Scores** -
    - Where z_m is the principal component score from a non-negative eigenvector from the covariance matrix
    - ϕ_{im} and ϕ_{i+1m} such that $\phi^2_{im} + \phi^2_{i+1m} = 1$
    - Remember, m < p because of *dimension reduction*

- **R (the important stuff)**
    - prcomp*(dataframe, scale = T)*
        - does PCA for you
        - Note: if you scale data prior, set scale = F
    - prcomp**$x**
        - Gives PC scores
    - prcomp**$rotation**
        - gives **ϕ's**
    - prcomp**$summary**
        - amount of variation explained by each PC

- **Example** – 5 Students, 4 Variables

	Maths	Science	English	Music
1	80	85	60	55
2	90	85	70	45
3	95	80	40	50
4	93	81	70	55
5	89	86	70	60

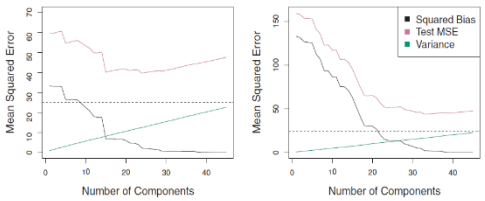
  - **ϕ's**

	PC1	PC2	PC3	PC4
Maths	0.5270317	-0.06995732	0.66451675	0.52512952
Science	-0.6128835	0.28517848	-0.06403710	0.73412959
English	-0.4859888	0.20318577	0.73889186	-0.42020139
Music	-0.3323043	-0.93407428	0.09140868	0.09339981

  - **Scores**

	PC1	PC2	PC3	PC4
[1,]	-1.2634785	-0.0760269	-1.2016540	-0.3234413
[2,]	-0.1400121	1.5970539	0.3562615	0.1004510
[3,]	2.2775901	-0.2780542	-0.5693981	0.2456426
[4,]	0.4584318	-0.4999870	0.9568735	-0.5495673
[5,]	-1.3325313	-0.7429858	0.4579171	0.5269151

Dimension Reduction Methods Furthered: PCR and PLS

Review: PCA	• Reduces Dimensions and variance of the data while retaining model variability • Use when dealing with a large set of correlated predictors, high variance
PCR	• Simplification of the Model • Use subset of principle components that explain the variance • Construct first M principle components, Z, using as predictors
	• Small number of Principle Components explain most variability • Fit a Least squares model to Z means better results than simple linreg • In right graph you can see relation to only 2 of the predictors
PLS (Partial Least Squares)	• A supervised alternative to PCR • Looks at Y's as opposed to variance in PCR • A dimension Reduction method
The Code **rm(list=ls())** **require(pls)** **#Need to do this first:** **Hitters<-na.omit(Hitters)** **names(Hitters)** **x<-model.matrix(Salary~.,Hitters)[,-1]** **y<-Hitters$Salary** **# Lab starts here** **set.seed(1)** **train<-sample(1:nrow(x), nrow(x)/2)** **test<-(-train)** **y.test<-y[test]** **set.seed(2)**	• Has the potential to reduce bias, but also has the potential to increase variance • More dependent on Training Y's • Can cancel out benefit of supervision
set.seed(1) **pls.fit<-plsr(Salary~., data=Hitters,subset=train,scale=TRUE,** **validation="CV")** **summary(pls.fit)** **validationplot(pls.fit,val.type="MSEP")** **pls.pred<-predict(pls.fit,x[test,],ncomp=2)** **mean((pls.pred-y.test)^2)** **pls.fit<-plsr(Salary~., data=Hitters,scale=TRUE,ncomp=2)** **summary(pls.fit)** **require(glmnet)** **library (glmnet)** **grid =10^ seq(10,-2, length =100)**	**ridge.mod =glmnet(x,y,alpha =0, lambda =grid)** **ridge.mod =glmnet(x[train ,],y[train], alpha =0, lambda =grid, thresh =1e-12)** **ridge.pred=predict(ridge.mod ,s=4, newx=x[test ,])** **mean((ridge.pred -y.test)^2)** **lasso.mod =glmnet(x[train ,],y[train],alpha =1, lambda =grid)** **set.seed (1)** **cv.out =cv.glmnet(x[train ,],y[train],alpha =1)** **bestlam =cv.out$lambda.min** **lasso.pred=predict(lasso.mod ,s=bestlam ,newx=x[test ,])** **mean((lasso.pred -y.test)^2)**

High Dimensional Data (HDD)

Basics

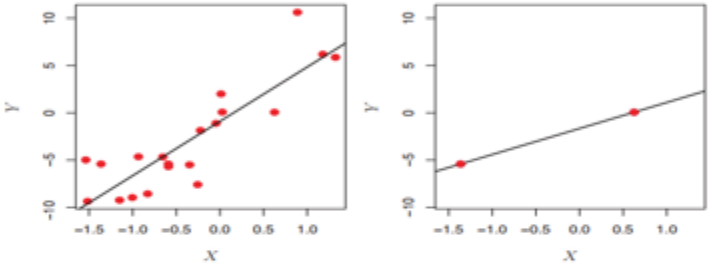
Definition: Data with more predictors (p) than observations (n), i.e. p>n.

Why it matters: HDD arises frequently in modeling scenarios with expensive or limited samples (like pharmaceutical trials) or many predictors (like tracking an individual's internet movements). Tech advances have enabled HDD prediction.

How to model HDD: Use subset selection (e.g. forward stepwise), shrinkage (e.g. lasso or ridge regression) or dimension reduction (like principal components regression) to reduce or eliminate unrelated predictors.

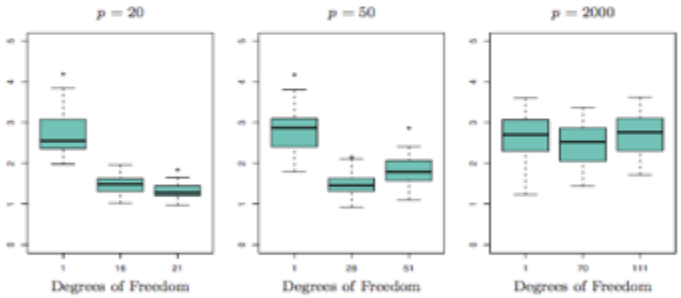
Pitfalls

HDD Challenges:
Traditional methods like linear, logistic, and LDA regression don't work because they will show a perfect fit regardless of whether a true relationship.



An HDD Regression

HDD Regression: Here we see how the lasso performs as p increases. Only ~20 predictors are truly correlated features, but more are included in the model as p goes up and lasso misses some irrelevant features. Test error usually increases as dimensionality goes up.



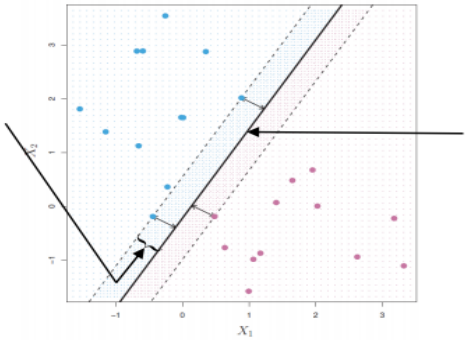
Interpreting Results from HDD Models

Consider Multicollinearity: Multicollinearity is when 3 or more predictor variables have high correlation with one another. This is another reason sub setting, regularization, and shrinkage are key.

Relevant Predictors Vary: Because there are so many features in a high dimensional setting, models will settle on different predictor variables between test sets. EX. If we are predicting blood pressure for a small number of individuals based on millions of DNA pairs, the relevant pairs will vary between groups (but both sets may still be valid).

The Maximal Margin Classifier

The ***margin*** is the distance from the solid line to either of the dashed lines.



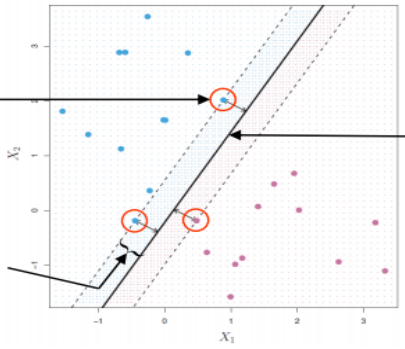
Maximal Margin Hyperplane is the mid-line of the widest "slab" that we can insert between the two classes

- Intuitively, the ***maximal margin hyperplane*** (also known as the ***optimal separating hyperplane***), is the separating hyperplane that is farthest from the training observations.
- To determine which of the candidate separating hyperplanes is the maximal margin hyperplane, we proceed as follows:
  - For each separating hyperplane, compute the distance from each training observation to that hyperplane; the smallest such distance is known as the ***margin*** for that hyperplane.
  - The maximal margin hyperplane is the separating hyperplane for which the margin is largest - that is, it is the hyperplane that has the farthest minimum distance to the training observations.

The Maximal Margin Classifier

The ***Support Vectors***

The ***margin*** is the distance from the solid line to either of the dashed lines



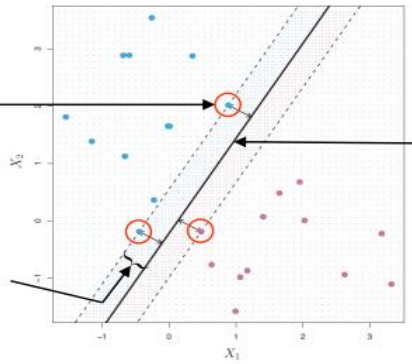
Maximal Margin Hyperplane is the mid-line of the widest "slab" that we can insert between the two classes

- Observe that three training observations are equidistant from the maximal margin hyperplane and lie along the dashed lines indicating the width of the margin.
- These three observations are known as ***support vectors***, since they are vectors in p-dimensional space (here $p = 2$) and they "support" the maximal margin hyperplane in the sense that if these points were moved slightly then the maximal margin hyperplane would move as well.

The Maximal Margin Classifier

The ***Support Vectors***

The ***margin*** is the distance from the solid line to either of the dashed lines



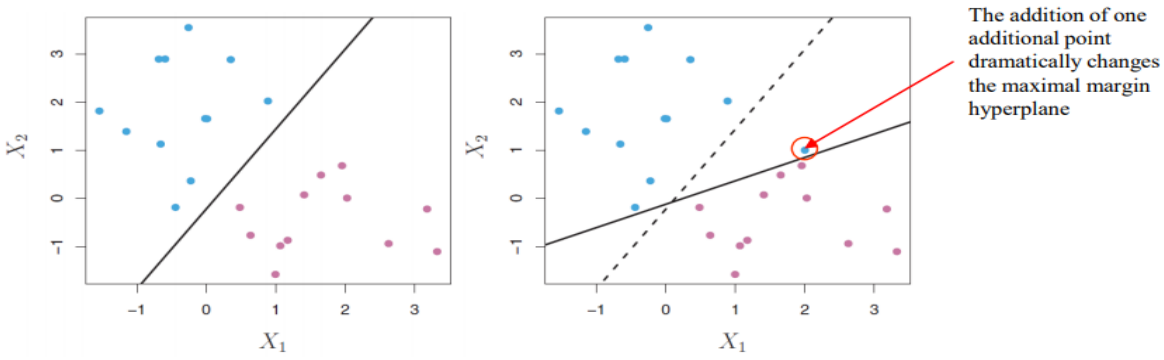
Maximal Margin Hyperplane is the mid-line of the widest "slab" that we can insert between the two classes

- Note that the maximal margin hyperplane depends directly on the support vectors, but not on the other observations.
- A movement to any of the other observations would not affect the separating hyperplane, provided that the observation's movement does not cause it to cross the boundary set by the margin.
- The fact that the maximal margin hyperplane depends directly on only a small subset of the observations is an important property that will arise later when we discuss the support vector classifier and support vector machines.

The Non-separable Case

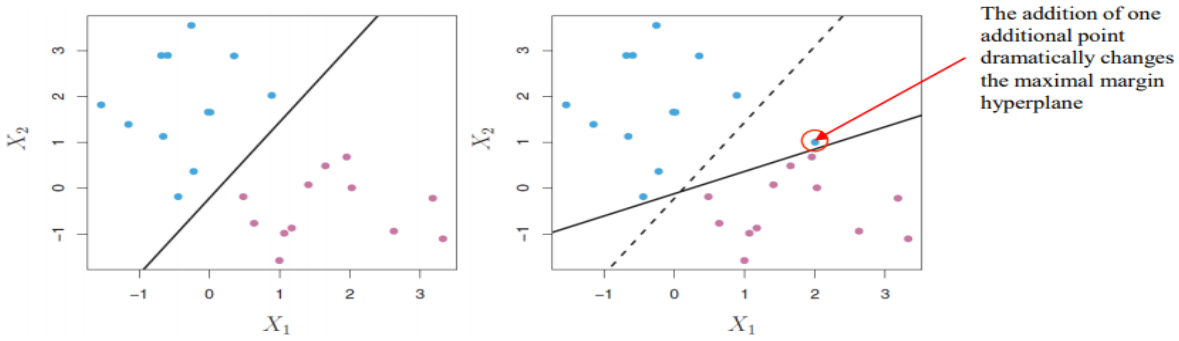
- The maximal margin classifier is a very natural way to perform classification, *if a separating hyperplane exists*.
- However, in most cases no separating hyperplane exists, and so there is no maximal margin classifier (and the optimization problem just discussed has no solution).
- In this case, we cannot *exactly* separate the two classes.
- However, we can extend the concept of a separating hyperplane in order to develop a hyperplane that *almost* separates the classes, using a so-called ***soft margin***.
- The generalization of the maximal margin classifier to the non-separable case is known as the ***Support Vector Classifier***.

Overview



- There are instances in which a classifier based on a separating hyperplane, even if it exists, might not be desirable.
- A classifier based on a separating hyperplane will necessarily perfectly classify all of the training observations; this can lead to sensitivity to individual observations.
- In the example above, the addition of a single observation in the right-hand panel leads to a dramatic change in the maximal margin hyperplane.

Overview



- The resulting maximal margin hyperplane is not satisfactory for two reasons:
  - It has only a tiny margin. This is problematic because as discussed previously, the distance of an observation from the hyperplane can be seen as a measure of our confidence that the observation was correctly classified.
  - The fact that the maximal margin hyperplane is extremely sensitive to a change in a single observation suggests that it may have overfit the training data.

- In this case, we might be willing to consider a classifier based on a hyperplane that does *not perfectly* separate the two classes, in the interest of
  - Greater robustness to individual observations
  - Better classification of *most* of the training observations
- That is, it could be worthwhile to misclassify a few training observations in order to do a better job in classifying the remaining observations.

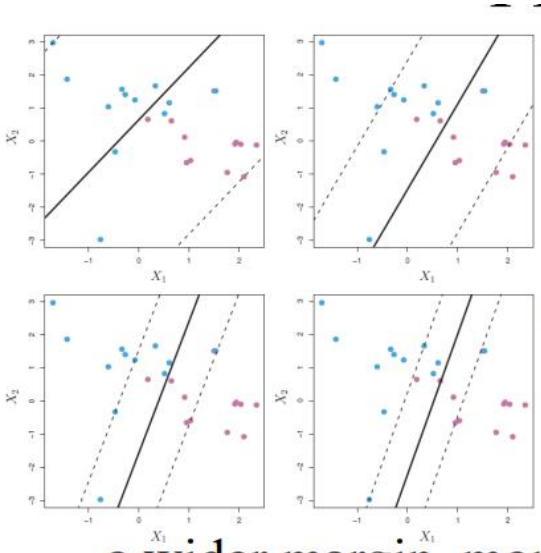
- The ***support vector classifier***, sometimes called a ***soft margin classifier***, does exactly this.
- Rather than seeking the largest possible margin so that every observation is both on the correct side of the hyperplane and also on the correct side of the margin, we instead allow some observations to be on the incorrect side of the margin, or even the incorrect side of the hyperplane (thus the term "soft")

Details of the Support Vector Classifier

- Consider the constraint $\sum_i^n \varepsilon_i \leq C$.
- *Since* C bounds the sum of the ε_i's, it determines the number and severity of the violations to the margin (and to the hyperplane) that we will tolerate.
- We can think of C as a *budget* for the amount that the margin can be violated by the n observations.
  - If $C = 0$ then there is no budget for violations to the margin, and it must be the case that all the ε_i's are 0, in which case the hyperplane we seek is just the maximal margin hyperplane optimization, which may not exist.
  - For $C > 0$ no more than C observations can be on the wrong side of the hyperplane, because if an observation is on the wrong side of the hyperplane then $\varepsilon_i > 1$, and the ε_i's must sum to no more than C.
- As the budget C increases, we become more tolerant of violations to the margin, and so the margin will widen.
- Conversely, as C decreases, we become less tolerant of violations to the margin and so the margin narrows.

- Note that only observations that either lie on the margin or that violate the margin will affect the hyperplane, and hence the classifier obtained.
  - In other words, an observation that lies strictly on the correct side of the margin does not affect the support vector classifier!
  - Changing the position of that observation would not change the classifier at all, provided that its position remains on the correct side of the margin.
- Observations that lie directly on the margin, or on the wrong side of the margin for their class, are known as ***support vectors***. These observations do affect the support vector classifier.

- The fact that only support vectors affect the classifier implies that C controls the bias-variance trade-off of the support vector classifier.
- When the tuning parameter C is large, then the margin is wide, many observations violate the margin, and so there are many support vectors.
- In this case, many observations are involved in determining the hyperplane.



- The largest value of C was used in the top left panel, and smaller values were used in the other panels.
- When C is large, there is a high tolerance for observations being on the wrong side of the margin, a wider margin, more bias, relatively more support vectors, and lower variance.
- As C decreases, the tolerance for observations being on the wrong side of the margin decreases, leading to narrower margins, lower bias, fewer support vectors, and higher variance.

Regression Splines

7.4 Regression Splines

Piecewise polynomials: Fit a different polynomial to different sections of your data

Knot: x value where you choose to change the coefficients to describe a new polynomial function

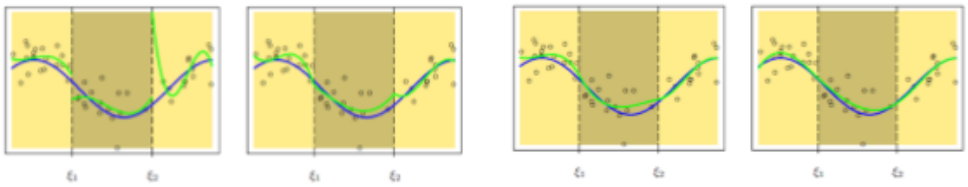
- More knots = more different functions describing your data → increased flexibility
- Pro over regular poly regression: rather than use a really high degree polynomial, you can use a cubic polynomial with more knots, which usually leads to more stable results (reduces variance)

To not end up with mismatched values at knots, you need to add **constraints**

- Ex1: constrain the function to be continuous at the boundary
- Ex2: constrain the fxn *& its derivative* to be continuous on the boundary
  - This will make the *slopes* equal on the boundary, so it will look smoother
- Ex3: a **boundary constraint** - fxn must be linear on the boundary

Spline of degree-d: a piecewise degree-d polynomial, with continuity in derivatives up to degree d-1 at each knot.

- For linear this just requires derivative 0 to be continuous – the fxn itself (Ex1 above)
- Cubic 0^{th}, 1^{st}, and 2^{nd} derivative must be continuous (pic 2, 3 & 4 respectively)



Natural spline: spline with boundary constraints (linear on boundary)

- Addresses wild edge behavior of polynomials, produces more stable results, reduces variance

Degrees of Freedom (given degree d polynomials, k knots) **Df = (K+1)(d+1)-Kd = K+d+1**

- K knots gives you k+1 polynomials describing your data
- (d+1) = coefficients per polynomial.
  - A 3^{rd} degree polynomial has 4 beta coefficients (for x, x^2, x^3, and the intercept: β_0)
- Subtract Kd because the continuity constraints take up some of the df

Spline-Basis Representation

- Choice of basis functions is kind of up to you, but most people do a basis for each degree ($b_1 = x$, $b_2 = x^2$, $b_3 = x^3$, etc.) and then use a *truncated power basis* function for each knot.
  - Math details beyond scope of class, but truncated power basis functions will produce continuity in the (degree-1) derivatives of the fxn – which satisfies spline constraints.

$$y_i = \beta_0 + \beta_1 b_1(x_i) + \beta_2 b_2(x_i) + \ldots + \beta_{K+3} b_{K+3}(x_i) + \epsilon_i$$

Truncated power basis fxn: $h(x, \xi) = (x - \xi)_+^3 = \begin{cases} (x-\xi)^3 & \text{if } x > 3 \\ 0 & \text{otherwise} \end{cases}$

Choosing knots: use cross validation, or set the "df" or "knots" arguments for bs() in R.

- You can place knots non-uniformly -- i.e. more knots where data changes more rapidly, but this is a less common approach to take.

R Review: library(splines) is needed. Use bs() fxn for regression splines and ns() fxn for natural splines

- Arguments to ns() and bs(): may want to set intercept = TRUE, can change "degree", "df", and "knots"

Smoothing Splines and Linear Regression

Smoothing Splines

Basic

Smoothing splines result from minimizing a RSS criterion subject to a smoothness penalty. If we don't put any constraints on $g(x_i)$, then we can always minimize RSS to zero by choosing the $g(x_i)$ that interpolates y_i. Thus, a more natural approach is to find such g(x) that minimize:

$$\sum_{i=1}^{n}(y_i - g(x_i))^2 + \lambda \int g''(t)^2 dt$$

λ is a nonnegative *tuning parameter*

$\sum_{i=1}^{n}(y_i - g(x_i))^2$ is the *loss function*

$\lambda \int g''(t)^2 dt$ is the *penalty term*

The penalty term reduces the variability of g(x). g''(t) is the second derivative of function g(x), which measures the roughness of the function. The larger the value of the second derivative, the wigglier the curvature of the function is (i.e. the second derivative of a straight line is zero). The $\int g''(t)^2 dt$ computes the total change of the slope of g(t). If g(x) is very smooth, then g'(t) is very close to zero, and thus the value of $\int g''(t)^2 dt$ is very small. Therefore, $\lambda \int g''(t)^2 dt$ encourages g(x) to be smooth.

Choosing Tuning Parameter λ

λ controls the roughness of smoothing spline, hence the *effective degrees of freedom* df_λ:

- When $\lambda \to 0$, penalty term has no effect
- When $\lambda \to \infty$, g(x) will be perfectly smooth
- As λ increases from 0 to ∞, df_λ decreases from n to 2

Can minimize cross validated RSS to find λ:

- Use LOOCV at the same cost of computing one single fit

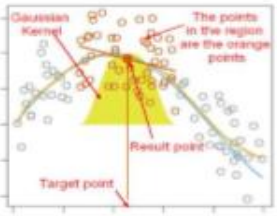
Built-in Functions in R

- splines library
- bs( )
- ns( )
- smooth.spline( )

Local Regression

Basic

It fits flexible non-linear function by computing the fit at a target point x_0 using only the nearby training observations. It is referred to as a memory-based procedure, because the fitting procedure is repeated for every training observation in the data set, and the entire training data set is needed.



Algorithm

1. Pick a target point x_0
2. Calculate span s = k/n
3. Assign a weight $K_{i0} = (x_i, x_0)$ to each point in neighborhood (span)
4. Fit a weighted least squares regression on the points in the neighborhood by finding the coefficients that minimize $\sum_{i=1}^{n} K_{i0}(y_i - \beta_0 - \beta_1 x_i)^2$
5. Calculate the fitted value at x_0 using $\widehat{\beta_0} + \widehat{\beta_1} x_0$

Considerations

1. How to define weighing function K
   - Using $K_{i0} = (1 - (\frac{d_i}{d_{max}})^3)^3$
2. What type of regression to fit in step 4 above
3. How to define the size of the span
   - Similar to λ in smoothing splines that controls the flexibility of non-linear fit
   - Use cross-validation to choose s or directly assign

Built-in Functions in R

- loess( )
- lowess( )
- locfit library

Modeling Methods discussed so far…

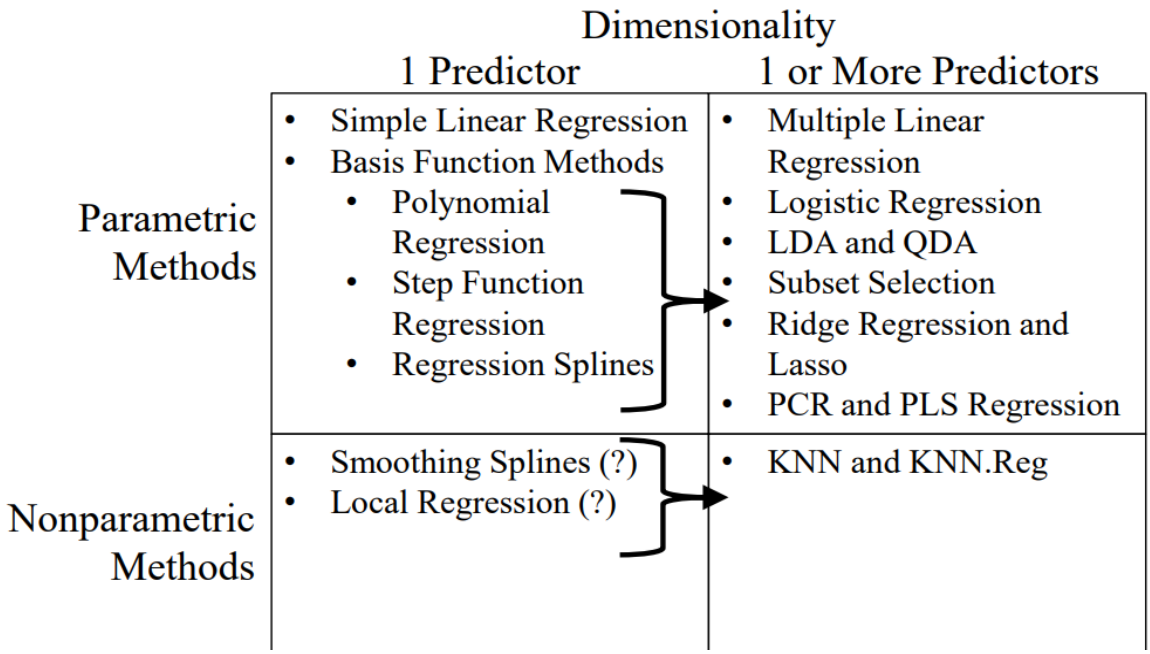
1. KNN and KNN.Reg
2. Simple Linear Regression
3. Multiple Linear Regression
4. Logistic Regression
5. LDA
6. QDA
7. Subset Selection (Best, Forward, Backward)
8. Ridge Regression
9. The Lasso
10. Principal Component Regression
11. Partial Least Squares Regression
12. Polynomial Regression (a Basis Function method)
13. Step Function Regression (also a Basis Function method)
14. Regression Splines (also a Basis Function method)
15. Smoothing Splines (**not** a Basis Function method)
16. Local Regression (**not** a Basis Function method)

	Dimensionality: 1 Predictor	Dimensionality: 1 or More Predictors

Parametric Methods	• Simple Linear Regression
	• Basis Function Methods
	  • Polynomial Regression
	  • Step Function Regression
	  • Regression Splines
		• Multiple Linear Regression
		• Logistic Regression
		• LDA and QDA
		• Subset Selection
		• Ridge Regression and Lasso
		• PCR and PLS Regression

Nonparametric Methods	• Smoothing Splines (?) • Local Regression (?)	• KNN and KNN.Reg

GAM's Do This…


Dimensionality
1 Predictor
1 or More Predictors
Parametric Methods
Simple Linear Regression
Basis Function Methods
Polynomial Regression
Step Function Regression
Regression Splines
Multiple Linear Regression
Logistic Regression
LDA and QDA
Subset Selection
Ridge Regression and Lasso
PCR and PLS Regression
Nonparametric Methods
Smoothing Splines (?)
Local Regression (?)
KNN and KNN.Reg

Generalized Linear Models (GLM's)

- Ordinary linear regression predicts the expected value of a given unknown quantity (the *response variable*, a random variable) as a linear combination of a set of observed values (*predictors*). This implies that a constant change in a predictor leads to a constant change in the response variable (i.e. a *linear-response model*).
- This is appropriate when the expected value of the response variable has a normal distribution. This follows from an assumption that ε, the irreducible error in the population, is normally distributed with mean 0 and variance σ^2.

- As an example…
  - A prediction model might predict that 10 degree temperature decrease would lead to 1,000 fewer people visiting the beach is unlikely to generalize well over both small beaches (e.g. those where the expected attendance was 50 at a particular temperature) and large beaches (e.g. those where the expected attendance was 10,000 at a low temperature).
  - The problem with this kind of prediction model would imply a temperature drop of 10 degrees would lead to 1,000 fewer people visiting the beach, a beach whose expected attendance was 50 at a higher temperature would now be predicted to have the impossible attendance value of -950.
  - Logically, a more realistic model would instead predict a constant *rate* of increased beach attendance (e.g. an increase in 10 degrees leads to a doubling in beach attendance, and a drop in 10 degrees leads to a halving in attendance). Such a model is termed an *exponential-response model* (or *log-linear model*, since the logarithm of the response is predicted to vary linearly).

- Similarly, a model that predicts a probability of making a yes/no choice (a Bernoulli variable) is even less suitable as a linear-response model, since probabilities are bounded on both ends (they must be between 0 and 1).
- For example…
  - Imagine, for example, a model that predicts the probability of a given person going to the beach as a function of temperature.
  - A reasonable model might predict that a change in 10 degrees makes a person two times more or less likely to go to the beach.
  - But what does "twice as likely" mean in terms of a probability? It cannot literally mean to double the probability value (e.g. 50% becomes 100%, 75% becomes 150%, etc.).
  - Rather, it is the *odds* that are doubling: from 2:1 odds, to 4:1 odds, to 8:1 odds, etc. Such a model is a *log-odds model*.

- Generalized linear models cover all these situations by allowing for the expected value of the response variables that have an arbitrary distribution (rather than simply normal distributions), and for an arbitrary function of the response variable (the *link function*) to vary linearly with the predicted values (rather than assuming that the response itself must vary linearly).
- For example, the case above of predicted number of beach attendees would typically be modeled with a Poisson distribution and a log link, while the case of predicted probability of beach attendance would typically be modeled with a binomial distribution and a log-odds (or *logit*) link function.

- The glm() function in R supports the following distributions (and associated link functions):

Family	Default Link Function
binomial	(link = "logit")
gaussian	(link = "identity")
Gamma	(link = "inverse")
inverse.gaussian	(link = "1/mu^2")
poisson	(link = "log")
quasibinomial	(link = "logit")
quasipoisson	(link = "log")

- Response variables that are counts normally use one of the Poisson or binomial distributions (depending on the degree of variance).
- Gamma distributions can address heteroskedasticity in non-negative data. The inverse-Gaussian distribution also relaxes homoskedastic assumptions.
- See GLMs.R for a simple example of a typical Poisson Regression model.

Regression Trees

I. Introduction: Regression Trees

For Regression Trees, we initially start with the training data set. Using a greedy, recursive binary splitting approach, we split the data set in a way that will minimize the RSS. Once this first split is made, for each region of the dataset, create another split minimizing the RSS.

Continue these steps until a criterion is met that will allow you to reach the terminal nodes; when a selected number of observations remain. As you label the tree, remember that the label indicates the left branch, while the opposite indicates the right branch.

Once all terminal nodes have been reached, these calculated regions may overfit the test set due to the greedy algorithm. The tree pruning process may reduce the number of branches (regions); therefore, reducing the overall RSS of the tree, as well as minimizing overfitting errors.

II. Partitioning the Dataset

Enumerating every possible tree, and testing whether it is a good model, will generally be too computationally expensive.

Instead, we use the observations (or the training subset) to search the space of possible trees to find a "good" model in a reasonable amount of time.

Recursive Binary Splitting

The algorithm that has been developed for decision tree induction is referred to as the top-down induction of decision trees, using a divide-and-conquer, or recursive partitioning, approach.

Greedy Algorithm

Reduces the computational complexity of search heuristics. Clearly, we need a measure of "goodness". We use this measure to decide when to split (The decision may be suboptimal).

Measure of Goodness

For regression trees, the measure of the goodness of a split is the familiar sum of the squared residuals (RSE). For any predictor j and split s, we define the pair of half planes:

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

Determine values of j and s that minimize the equation

$$\sum_i (y_i - y_{\hat{R}_1})^2 + \sum_i (y_i - y_{\hat{R}_2})^2$$

III. Model Performance

Using Cross Validation as a measure of performance, the K-fold process may produce good predictions on the training set, but is likely to overfit the data, leading to poor test set performance. A smaller tree with fewer splits might lead to lower variance and better interpretation at the cost of a little bias.

Subtree Selection

One possible alternative is to build the tree only so long as the decrease in the RSS due to each split exceeds some (high) threshold. This strategy will result in smaller trees, but is too short-sighted since a seemingly worthless split early on in the tree might be followed by an optimal split (a split that leads to a large reduction in RSS later on).

Tree Pruning

Therefore, a better strategy is to grow a very large tree and then prune it back in order to obtain a subtree. Select a subtree that leads to the lowest test error rate. However, estimating the cross-validation error for every possible subtree would be too cumbersome, since there is an extremely large number of possible subtrees. Instead, we need a way to select a small set of subtrees for consideration.

Tuning Parameters: rpart() Package

1) **split = argument**
   (i) Used to choose between different splitting functions
2) **minsplit = argument**
   (i) The minimum number of observations to compute a split. Changing the value of minsplit= allows us to eliminate some computation
3) **cp = argument**
   (i) Governs the minimum "benefit" gained at each split in order to make a split worthwhile. Saves on computing time by eliminating splits that add little value to the model
4) **minbucket = argument**
   (i) Sets the minimum number of observations in any leaf node.
5) **maxdepth = argument** (i) Sets the maximum depth of any node of the final tree.
6) **maxcompete = argument** (i) Limits the number of competing alternative splits retained in the output.

Classification Trees

Classification Trees limit the class of classifiers, thus sacrifice come accuracy, but have two key properties:

- Efficient to build
- Easy to interpret

Measuring Information:

- Because the dependent variable is categoric, you need to measure how good a certain partition of the dataset is.
- Classification Trees use an *information gain* measure for deciding between splits. (Regression uses RSE)
- **Entropy:** how evenly something is distributed in a system
  - **0:** contains only observations of the same value (**No information needed** to classify observations)
  - **1:** the observations are equally distributed across the two values of the target variable (**MAX info needed** to classify observations)
  - **0 < E < 1:** observations are split between the values of the target variable but the majority leans one way or another, i.e. 70/30 split (**Some extra information is needed**)

- The formula used to capture the entropy/information needed is: $\boldsymbol{info(D) = -p\log_2(p) - (1-p)\log_2(1-p)}$

- To calculate the information content of a particular split of the dataset: $\boldsymbol{info(D,S) = \frac{|D_1|}{|D|}I_1 + \frac{|D_2|}{|D|}I_2}$
  - Where: D_1 & D_2 = binary partitions of the dataset
    $D = D_1 \cup D_2$
    S = Particular Split
    I_1 & I_2 = the entropy of the respective datasets

- To measure the **gain** in information obtained by a particular split *S*: $\boldsymbol{gain(D,S) = info(D) - info(D,S)}$
  - The split that provides the greatest gain in information is the split we choose.

Loss Matrix:

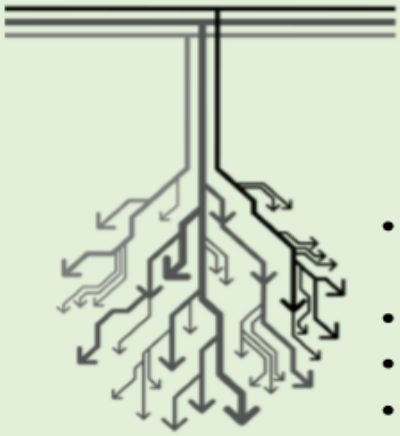
- When one type of error is more significant than another type of error, you can you a loss matrix to assign weights to the error types.
- If we favor Type I Error, we communicate this to the algorithm so that it will work harder to build a model to find all the positive cases, but at the expense of identifying false cases.

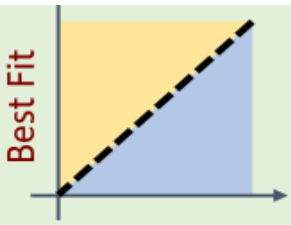
Classification Trees in R:

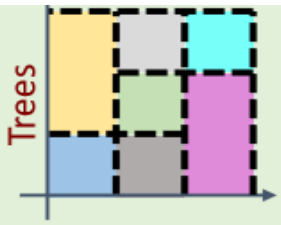
- The handout and slides provided by the Regression Trees group is a good reference for the rpart package.
- Package: rpart
  - Major Difference for Classification Trees:
    rpart (Target ~ decision variable, data, method="class",
    parms=list(split="information"),control= rpart.control(usesurrogate,maxsurrogate,minsplit))))
  - To build a maximal tree, use cp=0, minbucket=1, and minsplit=2
  - Complexity Table: *printcp(model)* OR *model$cptable*

- Package: rattle
  - fancyRpartPlot(model)
  - asRules(model)

Bagging and Random Forests



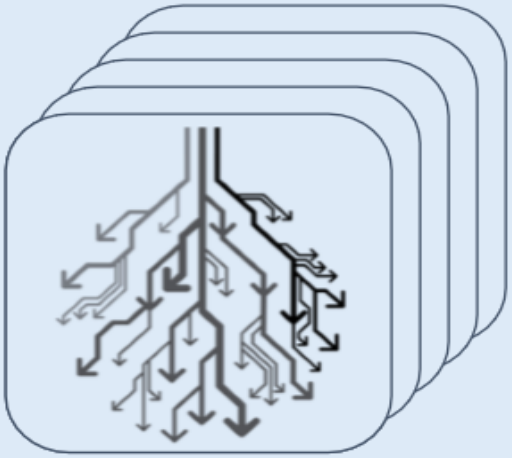


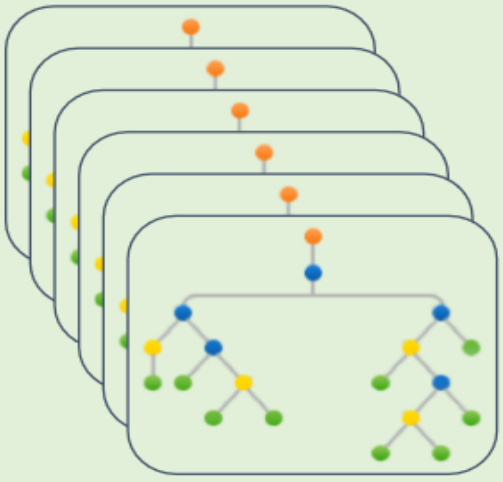


- Can be used on data which is linearly/ non linearly non separable
- Simple to understand and interpret
- Capable of both regression and classification
- Requires little data prep
- Uses white box model

Bagging

- A boot strap aggregation of samples with replacement
- All samples behave as training sets
- Each samples leaves out 33% observations
- Number of trees(samples) is not a critical parameter, going with a high number recommended
- **Avoids Overfitting** but **loses interpretability**
- Computationally inefficient





- Enhances bagging by introduction of variable selection
- Scores good in mixture of categorical with high cardinality and ordinal variables
- Useful in identifying variables that are significant from all that are available
- If not for decision making, can also be used for variable prioritization
- **Can deal with large number of predictors** but is **low on interpretability**

Random Forest

Boosting

Boosting relies on the use of a weak learning algorithm.

- A weak learning algorithm is one that is only slightly better than random guessing in terms of error rates (i.e., the model gets the decision wrong just less than 50% of the time).
- Through the process of boosting, it turns out that an ensemble of weak learners can lead to a strong aggregate model.

Steps for Boosting - Classification

1. Build a "weak" initial model from the training dataset

2. Any observations in the training data that the model incorrectly classifies will have their importance within the algorithm boosted

- This is done by assigning all observations an initial weight
- Observations that are wrongly classified by the model will have a greater weight for the building of the second decision stump, and so on.

3. By taking into account the weights of the observations in building the next model, the algorithm effectively tries harder each iteration to correctly classify "difficult" observations

4. The final model combines the "expertise's" of each individual model in such a way that the more accurate models carry more weight

Steps for Boosting - Regression

1. Build a "weak" initial model from the training dataset

2. Fit a second model to the residuals derived from the initial weak model

3. Fit small sequential trees to the residuals from the previous models

4. Add up predictions from all small trees to get a final prediction

Regression Boosting Approach – Visual:

The three tuning parameters of boosting

- d is the number of splits in each tree. d = 1 normally works well. When d = 1, the tree is called a "stump" because it consists of a single split
  - d is the interaction depth
  - d splits can involve at most d variables
- Unlike bagging and random forests, boosting can overfit if B is too large – this will occur slowly if overfitting occurs.
- Lambda is a small positive number and controls the rate at which boosting learns
  - Normally lambda is between .01 and .001
  - Small lambda = large B

Coding

Use the gbm package and the gbm() function

- Regression problem: distribution = "gaussian"
- Binary classification: distribution = "Bernoulli"

Other packages to use: Rattle, Adaboost, Gboost, Tree

Comparison of 3 ways to improve tree models

Bagging - Pros

- Improve prediction accuracy and avoids overfitting
- Out of bag (OOB) estimates can be used for model validation

Bagging - Cons

- Not as easy to visually interpret
- Computationally expensive

Random Forest - Pros

- Decorrelates trees (relative to bagged trees)
- Important when dealing with multiple features which may be correlated
- Reduced variance (relative to regular trees)
- Randomness causes to explore a whole lot of solution space

Random Forest - Cons

- Not as easy to visually interpret

Boosting - Pros

- It works and can easily handles qualitative (categorical) features

Boosting – Cons

- Unlike bagging and random forests, can overfit if number of trees is too large

Generalized Additive Models

- Here we explore an extension to multiple linear regression called GAMs.
- The "G" in GAM refers to "Generalized" - the idea that, as with GLM's in linear regression, the response variable can be a link function $g(y)$ such as the identity function, the logit function for modeling binomial probabilities, or the log function for modeling Poisson count data.
- The "AM" in GAM refers to "Additive Model", an approach that allows us to flexibly predict Y on the basis of several predictors, $X_1, \ldots, X_p$ by allowing non-linear, nonparametric functions f_j for each of the j variables, while maintaining *additivity*.
- Thus GAMs take on a form such that the conditional mean $\mu(X)$ of a response Y is related to an additive function of the predictors via a link function g:

$$g[u(X)] = \alpha + f_1(X_1) + f_2(X_2) + \cdots + f_p(X_p)$$

Here, the additive logistic regression model replaces each linear term by a more general functional form where each f_j is an unspecified smooth function.

- GAMs can be applied with both quantitative and qualitative responses (i.e.: both regression and classification).
- In the regression setting, a generalized additive model has the form

$$y \sim \alpha + f_1(X_1) + f_2(X_2) + \ldots + f_p(X_p)$$

where the f_j's can be unspecified smooth parametric or nonparametric functions.

- For example,
  - $f_1(X_1)$ could be a smoothing spline using predictor X_1
  - $f_2(X_2)$ could be a local regression using X_2
  - $f_3(X_3, X_4)$ could be a multinomial of degree 4 in X_3 and X_4 plus some interaction terms involving X_3 and X_4
  - $f_5(X_5)$ could be $\beta_0 + \beta_1 x_5$

- But with GAMs, we can also fit arbitrary nonparametric functions f_j - say smoothing splines, local regression, or (in the mgcv package) tensor-product based functions such as thin plate splines.
- In these cases, we can no longer use lm() or glm() (or indeed, any least-squares fitting approach) since such smoothers cannot be specified as simply a different basis representation of the X's.
- The GAMs approach provides an algorithm, called the Backfitting Algorithm, for simultaneously estimating all p of the unknown f_p functions.

GAMs/GLM for Classification

- In the classification setting, we used the logistic regression model for binary data (i.e. 2 classes) to relate the conditional probability of "yes" given an x, $\mu(X) = \Pr(Y = 1|X)$, to the predictors via a linear regression model and the logit link function:

the logit link function

$$\log\left(\frac{p(X)}{1-p(X)}\right) = \alpha + \beta_1 X_1 + \ldots + \beta_p X_p \quad \text{where } p(X) = \frac{e^{\beta_0+\beta_1 X}}{1+e^{\beta_0+\beta_1 X}}$$

- The GAMs logistic regression model replaces each linear term by a more general functional form

$$\log\left(\frac{p(X)}{1-p(X)}\right) = \alpha + f_1(X_1) + \ldots + f_p(X_p)$$

where again each f_j is an unspecified smooth function.

- The fact that the support vector classifier's decision rule is based only on a potentially small subset of the training observations (the support vectors) means that it is quite robust to the behavior of observations that are far away from the hyperplane.
- This property is distinct from some of the other classification methods that we have seen in preceding topics, such as linear discriminant analysis.
  - Recall that the LDA classification rule depends on the means μ_k of all of the observations within each class k, as well as the within-class covariance matrix $\sum$ computed using all of the observations.
    - That is, for more than one predictor, we computed

$$\delta_k(x) = x^T \Sigma^{-1} \mu_k - \frac{1}{2} \mu_k^T \Sigma^{-1} \mu_k + \log \pi_k$$

- In contrast, logistic regression has very low sensitivity to observations far from the decision boundary.
- In fact we will see that the support vector classifier and logistic regression are closely related.

The Challenge of Unsupervised Learning

- Unlike supervised learning, which has a clear objective (to predict something) and a clear way to measure the quality of the results (error rates, cross validation, test sets), unsupervised learning is often much more challenging.
  - The exercise tends to be more subjective, and there is often no simple goal for the analysis, such as prediction of a response.
- Unsupervised learning is often performed as part of an *exploratory data analysis*.

- The difficulty in assessing the results obtained from unsupervised learning methods is due to the absence of any universally accepted mechanism for performing cross validation or validating results on an independent data set.
  - In unsupervised learning, there is no way to check our work because we don't know the true answer - the problem is unsupervised.
- Consequently, the results are often mysterious – for example, we will not necessarily know why observations ended up in particular subgroups, for example.

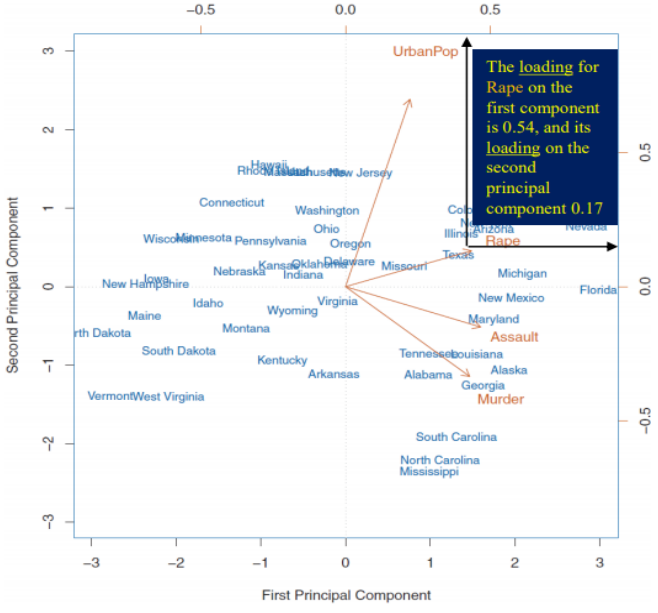
- Here is a concrete example of how clustering was used to improve decisions about how to set credit lines for new credit customers.
  - Analysts clustered existing GE Capital customers based on similarity in their use of their cards, payment of their bills, and profitability to the company.
  - They created five clusters and found that they represented very different consumer credit behavior, some desirable and some not. Customers in the five clusters can tolerate very different credit lines.
  - The problem with using this clustering immediately for decision making is that the data are not available when the initial credit line is set.
  - So GE Capital took this new knowledge and cycled back to the beginning of the machine learning process. They used the knowledge to define a precise predictive modeling problem using data that *are* available at the time of credit approval to predict the probability that a customer will fall into each of these clusters.
  - This predictive model was then used to improve initial credit line decisions.

Review of Earlier Discussions on Principal Components Analysis

- We discussed Principal Components last semester in the context of principal components regression (part of the Dimension Reduction topic).
- Briefly, when faced with a large set of (possibly correlated) variables, principal components allow us to summarize this set with a smaller number of representative variables that collectively explain most of the variability in the original set.
- The principal component directions were therefore described as "directions in feature space along which the original data are *highly variable*."
- We also mentioned that these directions also define lines and subspaces that are *as close as possible* to the data cloud.

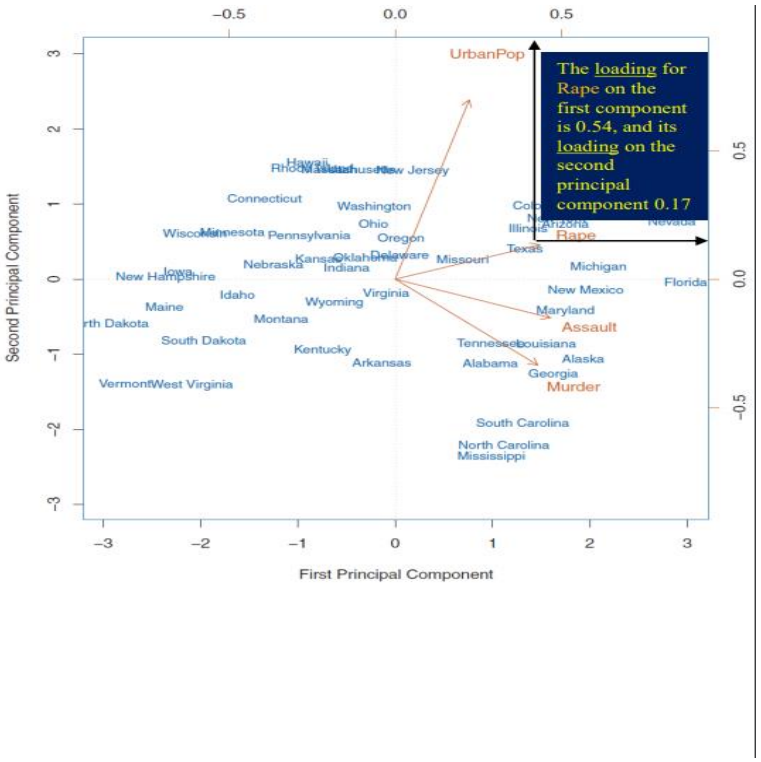
PCA and Visualization: Example

- The figure represents both the principal component scores and the loading vectors in a single ***biplot*** display.
  - *A biplot, displays both the principal component scores (information about the observations) and the principal component loadings (information about the variables)*
- The blue state names represent the scores for the first two principal components (axes on left and bottom)
- The orange arrows indicate the first two principal component loading vectors (with axes on the top and right).

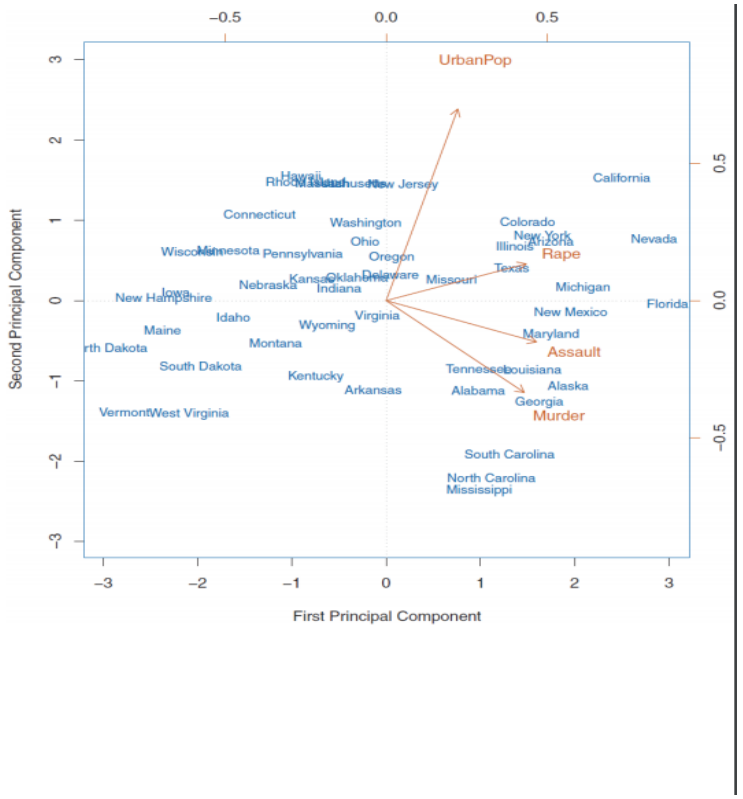


Interpreting the Biplot:

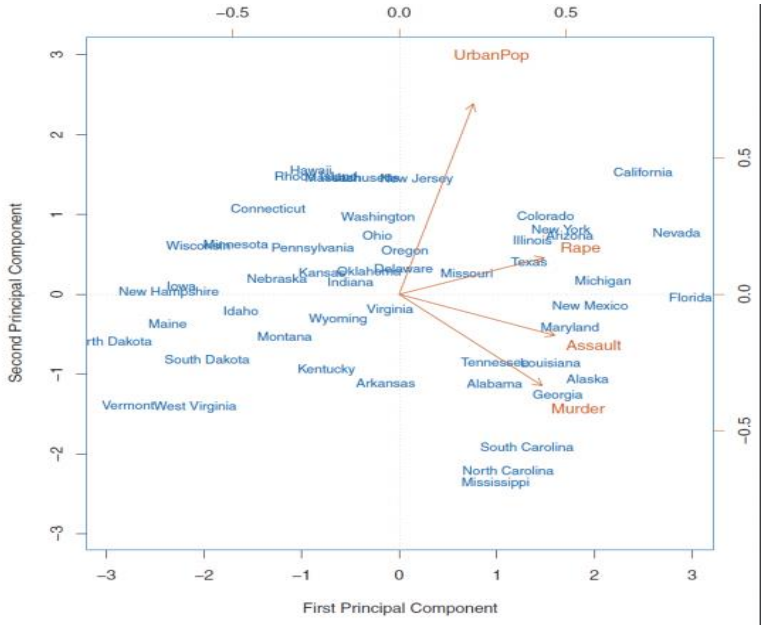
- Approximately true:
  - Data points: Projection on first two PCs
  - Distance in Biplot ~ True Distance
  - Projection of points onto arrow gives original (scaled) value of that variable
  - Arrow-length: Standard deviation of variable (if scaled, otherwise the variance)
  - Cosine of the angle between arrows: Correlation of the variables
- These approximations are exact when all principal components are used.



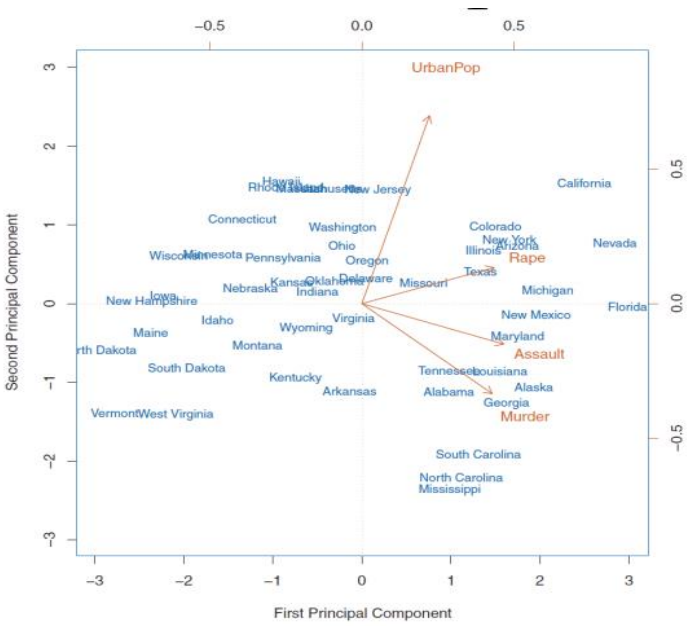
- In this plot, we see that the first loading vector places approximately equal weight on Assault, Murder, and Rape, with much less weight on UrbanPop.
- Hence this component roughly corresponds to a measure of overall rates of serious crimes (a so-called "latent variable").
- The second loading vector places most of its weight on UrbanPop and much less weight on the other three features.
- Hence, this component roughly corresponds to the level of urbanization of the state (another latent variable).



- Overall, we see that the crime-related variables (Murder, Assault, and Rape) are located close to each other, and that the UrbanPop variable is far from the other three.
- This indicates that the crime-related variables are correlated with each other - states with high murder rates tend to have high assault and rape rates - and that the UrbanPop variable is less correlated with the other three.



- We can examine differences between the states via the two principal component score vectors
- Our discussion of the loading vectors suggests that states with large positive scores on the first component, such as California, Nevada and Florida, have high crime rates, while states like North Dakota, with negative scores on the first component, have low crime rates.
- California also has a high score on the second component, indicating a high level of urbanization, while the opposite is true for states like Mississippi.



- States close to zero on both components, such as Virginia, have approximately average levels of both crime and urbanization.

More on PCA: Scaling the Variables

- We have already mentioned that before PCA is performed, the variables should be centered to have mean zero.
- Furthermore, *the results obtained when we perform PCA will also depend on whether the variables have been individually scaled* (each multiplied by a different constant).
  - This is in contrast to some other supervised and unsupervised learning techniques, such as linear regression, in which scaling the variables has no effect.

K-Means Clustering

Understanding K-Means Clustering

1. **Clustering** is an **unsupervised learning** method
   - X_i features of n observations
   - No Y response variable
   - Used in **exploratory analysis**
   - Hard to assess
     - Unknown true answer
     - No universally accepted validation mechanisms

2. Clustering lets us **find unknown subgroups within our data**
   - Ex: similar types of shoppers

3. There are **two types of clustering**:
   1. K-means
   2. Hierarchical

4. K-means clustering builds a **user specified quantity, *K*, clusters**
   - Each observation belongs to exactly 1 cluster
   - No clusters overlap
   - We don't know what *K* should be, so play around with it or ask your boss

5. We want **distinct clusters** that have **low within-cluster variation**
   - Within-cluster variation quantified using the Euclidean distance

$$W(C_k) = \frac{1}{|C_k|} \sum_{i,i' \in C_k} \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2$$

   - Minimize the total within-cluster variation across all clusters

6. We use a greedy algorithm to build the clusters
   - Randomly assign all observations to an initial cluster
   - Calculate the cluster centroids
     - The average of the observations in the cluster
   - Reassign observations to the cluster whose centroid is closest
   - Repeat until the assignments stop changing

7. Using a greedy algorithm means we could get **trapped in a local minimum**
   - Solution: run the algorithm multiple times
   - Choose the solution with the smallest total within-cluster variation

8. K-means can **handle different variable types**
   - Handles continuous, categorical and binary, and mixed data types
   - Calculates matching and dissimilarity metrics and uses them in its function

K-Means Clustering in R

mykmeans = **kmeans(x, #clusters)**
- Specify how many iterations of random assignments to try by adding the parameter nstart
  - kmeans(x, 2, nstart = 20)

mykmeans**$cluster**
shows cluster assignments

mykmeans**$tot.withinss**
shows total within cluster variation

mykmeans**$withinss**
shows the within cluster variation by cluster

mykmeans**$centers**
gives centroid values

mykmeans**$size**
tells cluster sizes

plot(x, col = km.out$cluster + 1) if in 2 dimensions

Team 1-01: Isabel, Len, Mark, and Michael F.

Hierarchical Clustering

- **What is Hierarchical Clustering?** – a technique in which clusters are built in a *hierarchy*

- **Two methods:**
  - **Agglomerative**
    - "Bottom-up"
    - Cluster points one-by-one
    - $O(n^3)$
  - **Divisive**
    - "Top-down"
    - Start with all points, separate points until each point is its own cluster
    - $O(2^n)$

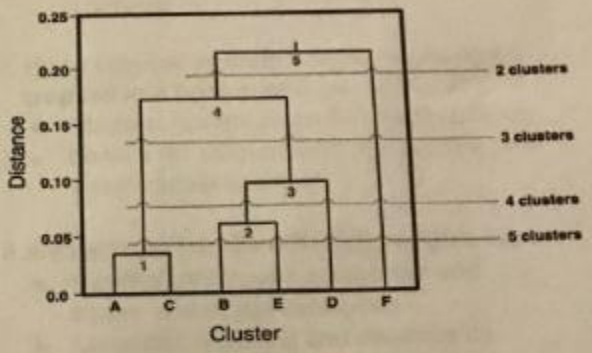
- **Note** – agglomerative might not lead to global best clustering

- **Distance** – measure used to determine where data points are in relation to each other
  - Usually Euclidean, but there can be other measurements used
    - Manhattan, etc.

Linkage – used to determine how clusters are created

Linkage	Description	Note
Complete	Maximal inter-cluster dissimilarity. Compute all pairwise dissimilarities between observations in two clusters, and record the largest of the dissimilarities.	Preferred
Single	Minimal inter-cluster dissimilarity. Compute all pairwise dissimilarities and record the smallest dissimilarities.	Can result in extended, trailing clusters in which single observations are fused one-at-a-time
Average	Mean inter-cluster dissimilarity. Compute all pairwise dissimilarities and record the average of these dissimilarities.	Preferred
Centroid	Dissimilarity between the centroid for cluster A and the centroid for cluster B.	Often used in genomics. Can result in undesirable inversions and difficulties in interpretation of dendrogram. Least preferred

- **Dendrogram** – visualization of the clustering
  - Y-axis shows distance between clusters
  - X-axis shows clusters

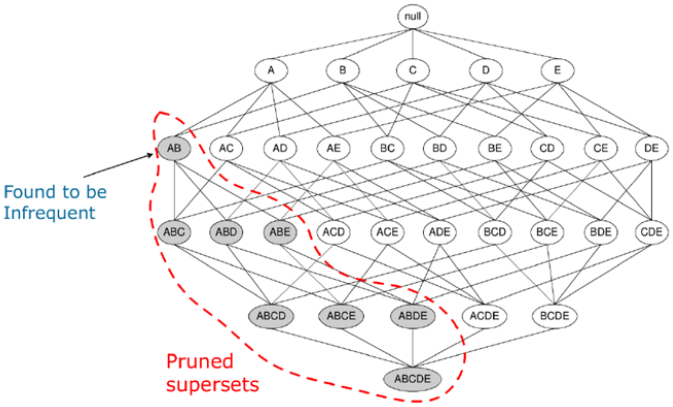


- **R – Code**
  - **hclust** - Hierarchical cluster analysis on a set of dissimilarities and methods for analyzing it.
  - **dist** - Computes and returns the distance matrix computed by using the specified distance measure to compute the distances between the rows of a data matrix.
  - **cutree** - Cuts a tree
  - **rect.hclust** - Draws rectangles around the branches of a dendrogram highlighting the corresponding clusters.
  - **heatmap** - Produce a heatmap with a dendrogram added to the left side and to the top.

Association Analysis

The Apriori Search Algorithm

- An algorithm for frequent itemset mining and association rule learning over transactional databases.
- **Phase I:** Generate ***frequent itemsets***
  - "Bottom up" approach - Start from individual item and extend one item at a time until no more itemset of the next level is frequent enough
  - ***Downward closure property***
    - All subsets of frequent itemset must also be frequent
    - No superset of an infrequent itemset can be a frequent itemset



  - ***Support*** - the frequency of itemsets appearing in all transactions
- **Phase II:** Build ***association rules*** (X → Y)
  - Generate all possible rules for each frequent itemset, and retain those with high ***confidence***
  - Number of association rules if n unique items are in a frequent itemset: $R_n = \sum_{i=1}^{n-1}\left(\binom{n}{i}\sum_{j=1}^{n-i}\binom{n-i}{j}\right)$
  - ***Confidence*** - conditional probability that Y (RHS of rule) appears in basket given that X (LHS) also appears

Calculating Measures:

- Support(X) = P(X), support(X U Y) = P(X U Y)
- Confidence(X → Y) = P(Y|X) = P(X U Y) / P(X) = support(X U Y) / support(X)
- Lift(X → Y) = support(X U Y) / support(X) * support(Y) = confidence(X → Y) / support(Y)
  - *Note: "X U Y" represents transactions where X **and** Y both appear*

R Package ("arules"):inspect()

- apriori()
- Lhs() and rhs()
- sort(rules,by=)

CARET PACKAGE (CLASSIFICATITON AND REGRESSION TRAINING)

Overview of the CARET Package

- This package alone is all you need to solve almost any supervised machine learning problem.
- To simplify the model building process, caret provides tools for almost every part of the model building process and provides a common interface to different machine learning methods.
- Some specific areas of model building and analysis that we have covered in class that caret may be helpful with moving forward include:
    - Data cleansing and splitting
    - Training models
    - Model evaluation
    - Model tuning
    - Data visualization

CARET Package Functions

- dummyVars—outputs a dummy variables object to apply to your data using predict()
- nearZeroVar—outputs an object with a few features; can be used to remove predictors w/ near zero variance
- findCorrelation—outputs an object with a few features; can be used to remove predictors above a specified correlation threshold
- findLinearCombos—outputs an object with a few features; can be used to remove predictors that form linear combinations
- **preProcess**—expansive function with many argument values and features. Most often used to create an object which can be applied to your data using predict() in order to execute specified transformations
- **predict**—very useful function in caret. In addition to applying objects to your data, predict may be used, as in base R, to create prediction objects with your models
- classDist—useful for creating centroids in a clustering setting
- **createDataPartition**—outputs an object, or indices, useful for partitioning data into train/test/etc. sets
- maxDissim—outputs an object useful for partitioning data based on dissimilarity of features. Best used when attempting to measure a model by outliers
- **trainControl**—expansive function with many argument values and features used to determine model tuning and measurement
- **train**—expansive function that sets up a grid of tuning parameters for a number of classification and regression models (see names(getModelInfo()))
- resamples—function which provides methods for collecting, analyzing and visualizing resampling results from a data set
- **names(getModelInfo())**—outputs a list of all possible model choices in caret
- **modelLookup**—returns a list of tunable parameters for chosen models

Drawbacks of the CARET Package

- Misuse! If you are not careful with your tuneControl() and train() functions, you will train a huge model over many parameter choices; this may take a LONG time. Be judicious in your decisions!

Helpful Links:

- http://topepo.github.io/caret/index.html
- https://cran.r-project.org/web/packages/caret/caret.pdf
- https://www.youtube.com/watch?v=PhjB5c2P_U0