Predicting with trees

Jeffrey Leek
Johns Hopkins Bloomberg School of Public Health

Key ideas

- Iteratively split variables into groups
- Evaluate "homogeneity" within each group
- Split again if necessary

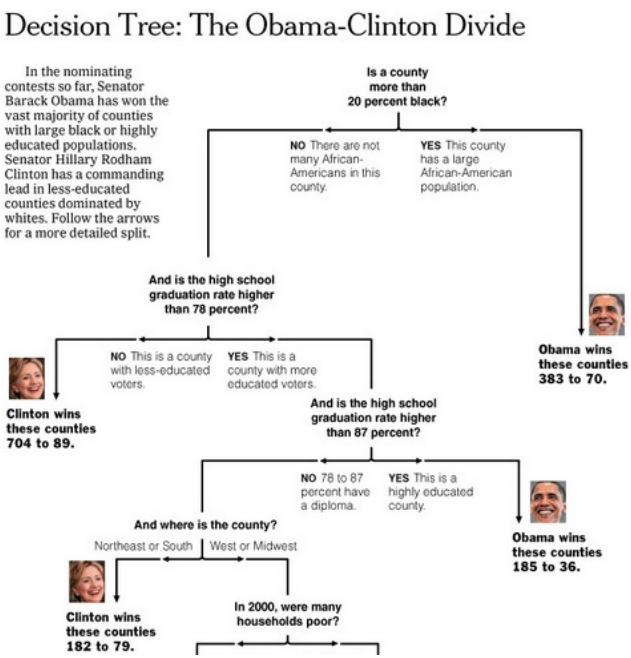
Pros:

- Easy to interpret
- Better performance in nonlinear settings

Cons:

- Without pruning/cross-validation can lead to overfitting
- Harder to estimate uncertainty
- Results may be variable

Example Tree



http://graphics8.nytimes.com/images/2008/04/16/us/0416-nat-subOBAMA.jpg

Basic algorithm

1. Start with all variables in one group
2. Find the variable/split that best separates the outcomes
3. Divide the data into two groups ("leaves") on that split ("node")
4. Within each split, find the best variable/split that separates the outcomes
5. Continue until the groups are too small or sufficiently "pure"

Measures of impurity

$$\hat{p}_{mk} = \frac{1}{N_m} \sum_{x_i \text{ in Leaf } m} \mathbb{I}(y_i = k)$$

Misclassification Error:

$$1 - \hat{p}_{mk(m)}; k(m) = \text{most; common; } k$$

- 0 = perfect purity
- 0.5 = no purity

Gini index:

$$\sum_{k \neq k'} \hat{p}_{mk} \times \hat{p}_{mk'} = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk}) = 1 - \sum_{k=1}^{K} p_{mk}^2$$

- 0 = perfect purity
- 0.5 = no purity

http://en.wikipedia.org/wiki/Decision_tree_learning

Measures of impurity

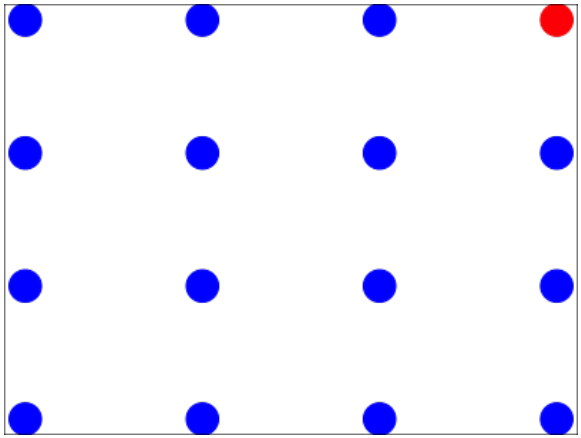
Deviance/information gain:

$$-\sum_{k=1}^{K} \hat{p}_{mk} \log_2 \hat{p}_{mk}$$

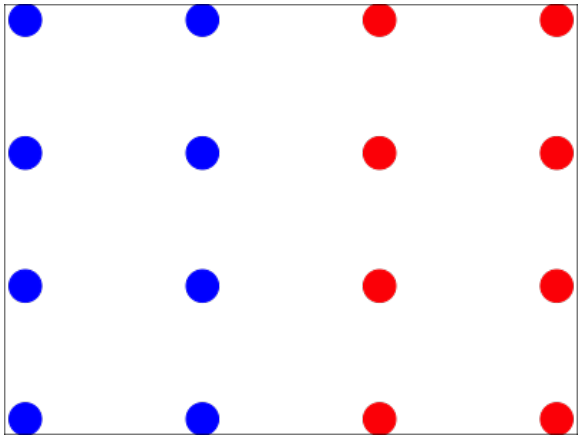
- 0 = perfect purity
- 1 = no purity

http://en.wikipedia.org/wiki/Decision_tree_learning

Measures of impurity



- **Misclassification:** 1/16 = 0.06
- **Gini:** $1 - [(1/16)^2 + (15/16)^2] = 0.12$
- **Information:**
  $-[1/16 \times \log2(1/16) + 15/16 \times \log2(15/16)] = 0.34$



- **Misclassification:** 8/16 = 0.5
- **Gini:** $1 - [(8/16)^2 + (8/16)^2] = 0.5$
- **Information:**
  $-[1/16 \times \log2(1/16) + 15/16 \times \log2(15/16)] = 1$

Example: Iris Data

```
data(iris); library(ggplot2)
names(iris)
```

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

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

```

    setosa versicolor  virginica
        50         50         50
```

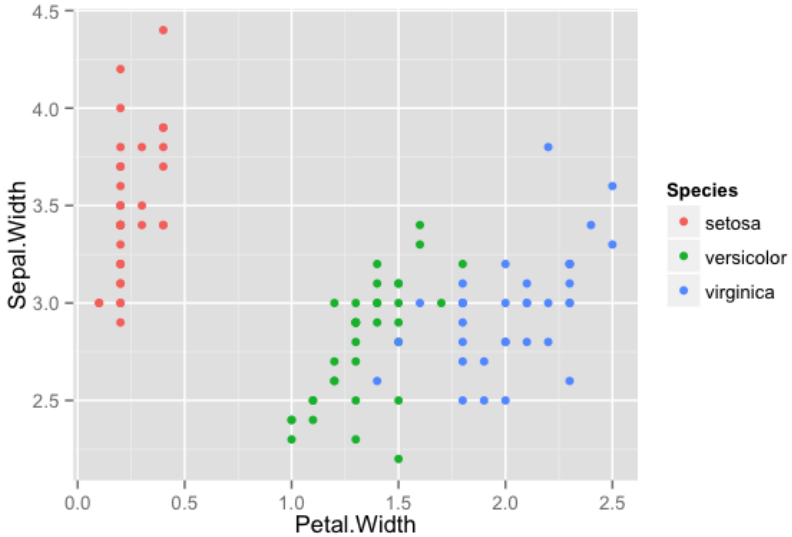
Create training and test sets

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

```
[1] 45  5
```

Iris petal widths/sepal width

```
qplot(Petal.Width,Sepal.Width,colour=Species,data=training)
```



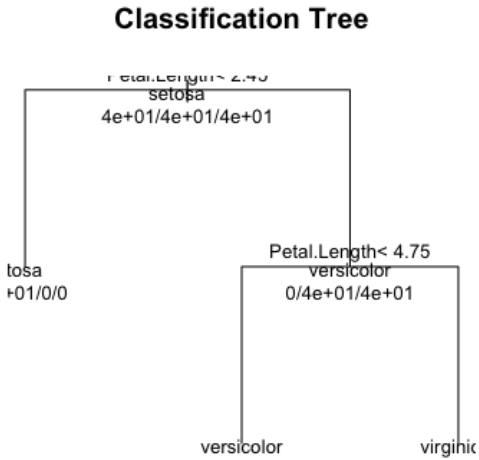
Iris petal widths/sepal width

```
library(caret)
modFit <- train(Species ~ .,method="rpart",data=training)
print(modFit$finalModel)
```

```
n= 105

node), split, n, loss, yval, (yprob)
      * denotes terminal node

1) root 105 70 setosa (0.3333 0.3333 0.3333)
  2) Petal.Length< 2.45 35  0 setosa (1.0000 0.0000 0.0000) *
  3) Petal.Length>=2.45 70 35 versicolor (0.0000 0.5000 0.5000)
    6) Petal.Length< 4.75 31  0 versicolor (0.0000 1.0000 0.0000) *
    7) Petal.Length>=4.75 39  4 virginica (0.0000 0.1026 0.8974) *
```

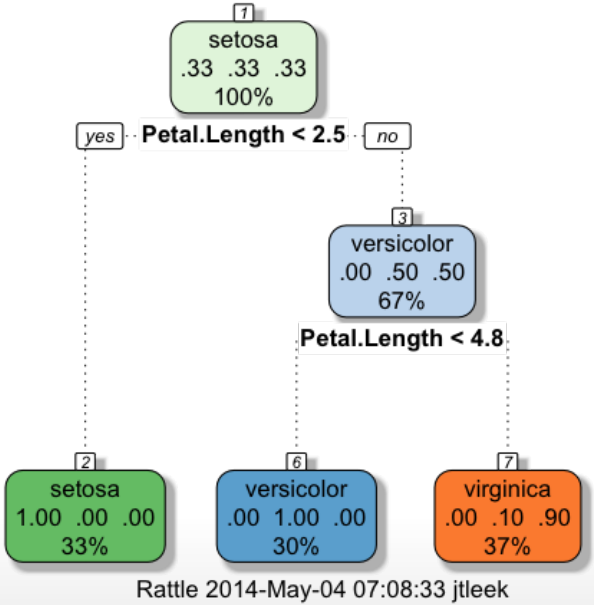
Plot tree

```
plot(modFit$finalModel, uniform=TRUE,
      main="Classification Tree")
text(modFit$finalModel, use.n=TRUE, all=TRUE, cex=.8)
```



Prettier plots

```
library(rattle)
fancyRpartPlot(modFit$finalModel)
```



Predicting new values

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

```
 [1] setosa     setosa     setosa     setosa     setosa     setosa     setosa     setosa
 [9] setosa     setosa     setosa     setosa     setosa     setosa     setosa     versicolor
[17] versicolor versicolor versicolor versicolor versicolor versicolor versicolor versicolor
[25] virginica  versicolor virginica  versicolor versicolor versicolor virginica  virginica
[33] virginica  versicolor virginica  virginica  virginica  virginica  virginica  virginica
[41] virginica  virginica  virginica  virginica  virginica
Levels: setosa versicolor virginica
```

Notes and further resources

- Classification trees are non-linear models
  - They use interactions between variables
  - Data transformations may be less important (monotone transformations)
  - Trees can also be used for regression problems (continuous outcome)
- Note that there are multiple tree building options in R both in the caret package - party, rpart and out of the caret package - tree
- Introduction to statistical learning
- Elements of Statistical Learning
- Classification and regression trees

Bagging

Jeffrey Leek
Johns Hopkins Bloomberg School of Public Health

Bootstrap aggregating (bagging)

Basic idea:

1. Resample cases and recalculate predictions
2. Average or majority vote

Notes:

- Similar bias
- Reduced variance
- More useful for non-linear functions

Ozone data

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

```
    ozone radiation temperature wind
17      1         8          59  9.7
19      4        25          61  9.7
14      6        78          57 18.4
45      7        48          80 14.3
106     7        49          69 10.3
7       8        19          61 20.1
```

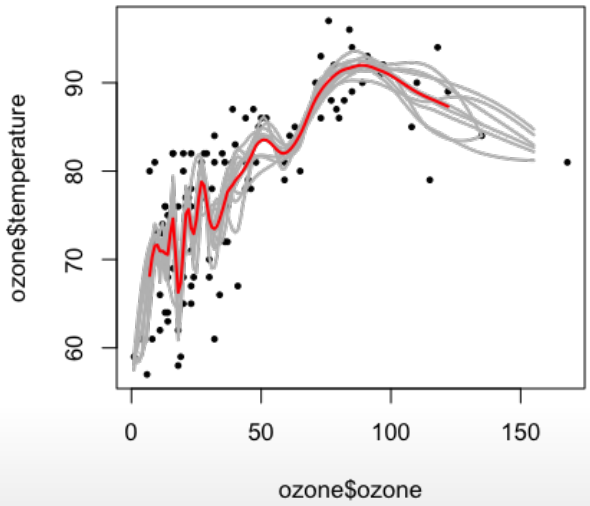
http://en.wikipedia.org/wiki/Bootstrap_aggregating

Bagged loess

```
ll <- matrix(NA,nrow=10,ncol=155)
for(i in 1:10){
  ss <- sample(1:dim(ozone)[1],replace=T)
  ozone0 <- ozone[ss,]; ozone0 <- ozone0[order(ozone0$ozone),]
  loess0 <- loess(temperature ~ ozone,data=ozone0,span=0.2)
  ll[i,] <- predict(loess0,newdata=data.frame(ozone=1:155))
}
```

Bagged loess

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



Bagging in caret

- Some models perform bagging for you, in `train` function consider `method` options
  - `bagEarth`
  - `treebag`
  - `bagFDA`
- Alternatively you can bag any model you choose using the `bag` function

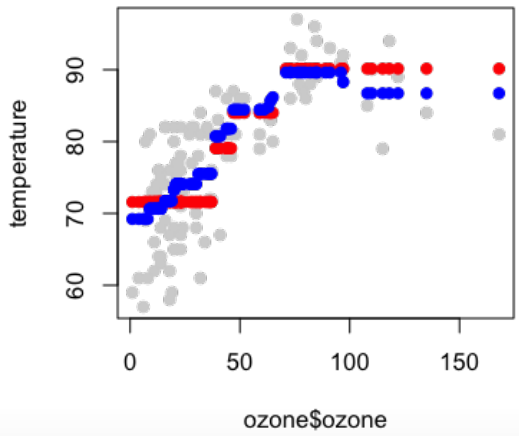
More bagging in caret

```
predictors = data.frame(ozone=ozone$ozone)
temperature = ozone$temperature
treebag <- bag(predictors, temperature, B = 10,
               bagControl = bagControl(fit = ctreeBag$fit,
                                       predict = ctreeBag$pred,
                                       aggregate = ctreeBag$aggregate))
```

http://www.inside-r.org/packages/cran/caret/docs/nbBag

Example of custom bagging (continued)

```
plot(ozone$ozone,temperature,col='lightgrey',pch=19)
points(ozone$ozone,predict(treebag$fits[[1]]$fit,predictors),pch=19,col="red")
points(ozone$ozone,predict(treebag,predictors),pch=19,col="blue")
```



Parts of bagging

```
ctreeBag$fit
```

```
function (x, y, ...)
{
    library(party)
    data <- as.data.frame(x)
    data$y <- y
    ctree(y ~ ., data = data)
}
<environment: namespace:caret>
```

Parts of bagging

```
ctreeBag$pred
```

```
function (object, x)
{
    obsLevels <- levels(object@data@get("response")[, 1])
    if (!is.null(obsLevels)) {
        rawProbs <- treeresponse(object, x)
        probMatrix <- matrix(unlist(rawProbs), ncol = length(obsLevels),
            byrow = TRUE)
        out <- data.frame(probMatrix)
        colnames(out) <- obsLevels
        rownames(out) <- NULL
    }
    else out <- unlist(treeresponse(object, x))
    out
}
<environment: namespace:caret>
```

Parts of bagging

```
ctreeBag$aggregate
```

```
function (x, type = "class")
{
    if (is.matrix(x[[1]]) | is.data.frame(x[[1]])) {
        pooled <- x[[1]] & NA
        classes <- colnames(pooled)
        for (i in 1:ncol(pooled)) {
            tmp <- lapply(x, function(y, col) y[, col], col = i)
            tmp <- do.call("rbind", tmp)
            pooled[, i] <- apply(tmp, 2, median)
        }
        if (type == "class") {
            out <- factor(classes[apply(pooled, 1, which.max)],
                levels = classes)
        }
        else out <- as.data.frame(pooled)
    }
    else {
        x <- matrix(unlist(x), ncol = length(x))
```

Notes and further resources

Notes:

- Bagging is most useful for nonlinear models
- Often used with trees - an extension is random forests
- Several models use bagging in caret's *train* function

Further resources:

- Bagging
- Bagging and boosting
- Elements of Statistical Learning

Random forests

Jeffrey Leek, Assistant Professor of Biostatistics
Johns Hopkins Bloomberg School of Public Health

Random forests

1. Bootstrap samples
2. At each split, bootstrap variables
3. Grow multiple trees and vote

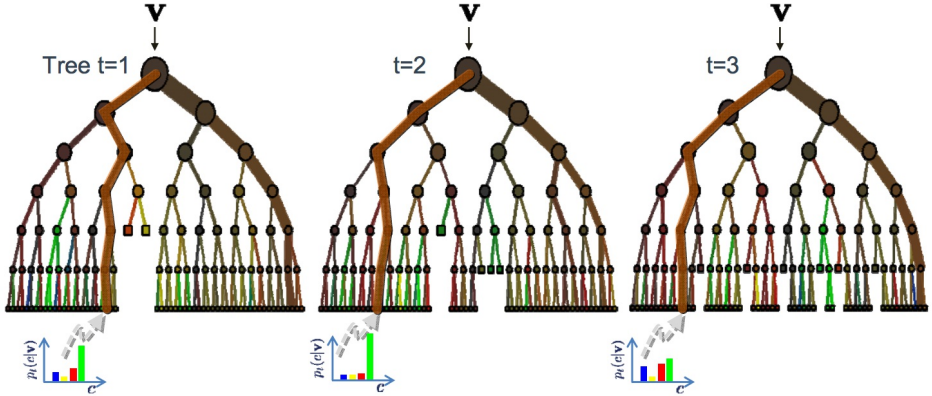
Pros:

1. Accuracy

Cons:

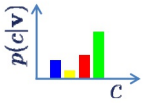
1. Speed
2. Interpretability
3. Overfitting

Random forests



The ensemble model

Forest output probability $p(c|\mathbf{v}) = \frac{1}{T}\sum_{t}^{T} p_t(c|\mathbf{v})$



http://www.robots.ox.ac.uk/~az/lectures/ml/lect5.pdf

Iris data

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

Random forests

```
library(caret)
modFit <- train(Species~ .,data=training,method="rf",prox=TRUE)
modFit
```

```
105 samples
  4 predictors
  3 classes: 'setosa', 'versicolor', 'virginica'

No pre-processing
Resampling: Bootstrap (25 reps)

Summary of sample sizes: 105, 105, 105, 105, 105, 105, ...

Resampling results across tuning parameters:

  mtry  Accuracy  Kappa  Accuracy SD  Kappa SD
  2     0.9       0.9    0.03         0.04
  3     0.9       0.9    0.03         0.05
  4     0.9       0.9    0.03         0.05
```

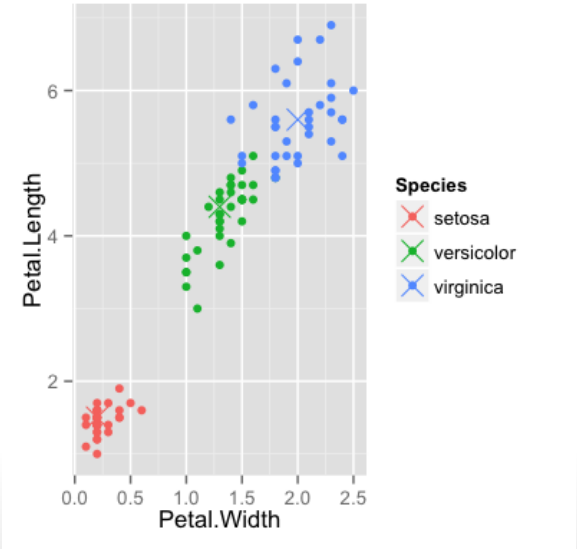
Getting a single tree

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

```
   left daughter right daughter split var split point status prediction
1              2              3         4        0.70      1          0
2              0              0         0        0.00     -1          1
3              4              5         4        1.70      1          0
4              6              7         3        4.95      1          0
5              8              9         3        4.85      1          0
6              0              0         0        0.00     -1          2
7             10             11         4        1.55      1          0
8             12             13         1        5.95      1          0
9              0              0         0        0.00     -1          3
10             0              0         0        0.00     -1          3
11             0              0         0        0.00     -1          2
12             0              0         0        0.00     -1          2
13             0              0         0        0.00     -1          3
```

Class "centers"

```
irisP <- classCenter(training[,c(3,4)], training$Species, modFit$finalModel$prox)
irisP <- as.data.frame(irisP); irisP$Species <- rownames(irisP)
p <- qplot(Petal.Width, Petal.Length, col=Species,data=training)
p + geom_point(aes(x=Petal.Width,y=Petal.Length,col=Species),size=5,shape=4,data=irisP)
```



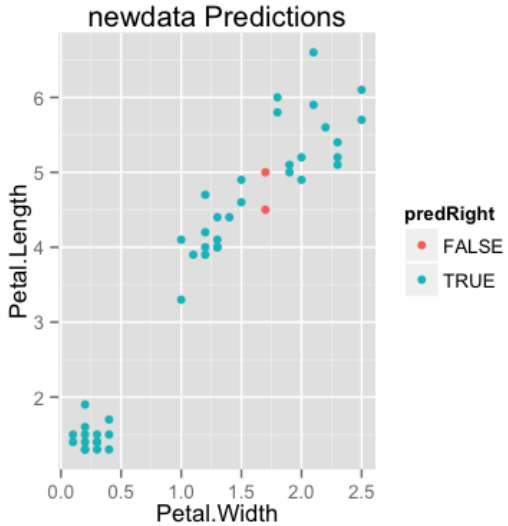
Predicting new values

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

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

Predicting new values

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



Notes and further resources

Notes:

- Random forests are usually one of the two top performing algorithms along with boosting in prediction contests.
- Random forests are difficult to interpret but often very accurate.
- Care should be taken to avoid overfitting (see rfcv funtion)

Further resources:

- Random forests
- Random forest Wikipedia
- Elements of Statistical Learning

Boosting

Jeffrey Leek
Johns Hopkins Bloomberg School of Public Health

Basic idea

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

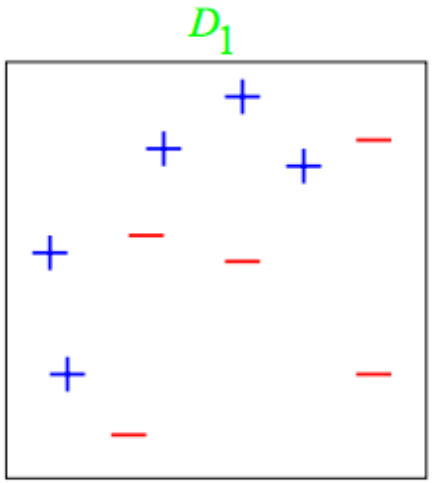
Basic idea behind boosting

1. Start with a set of classifiers $h_1, \ldots, h_k$
   - Examples: All possible trees, all possible regression models, all possible cutoffs.
2. Create a classifier that combines classification functions: $f(x) = \text{sgn}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$.
   - Goal is to minimize error (on training set)
   - Iterative, select one h at each step
   - Calculate weights based on errors
   - Upweight missed classifications and select next h

Adaboost on Wikipedia

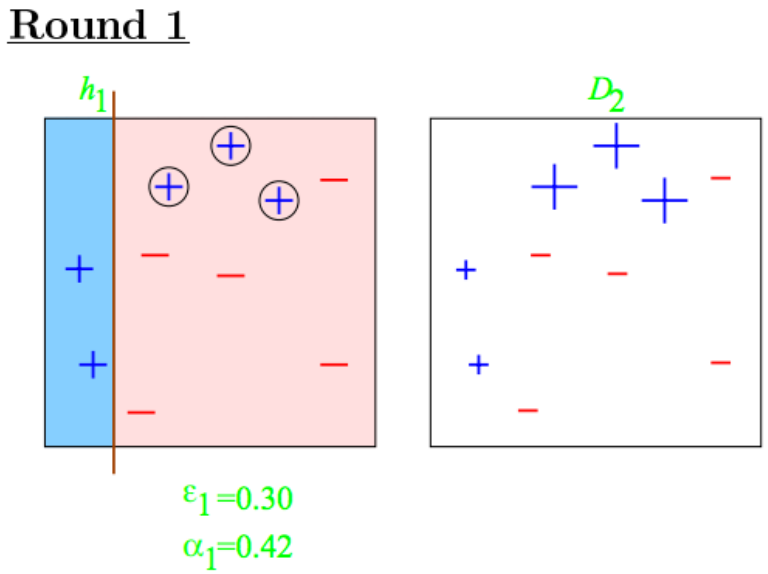
http://webee.technion.ac.il/people/rmeir/BoostingTutorial.pdf

Simple example



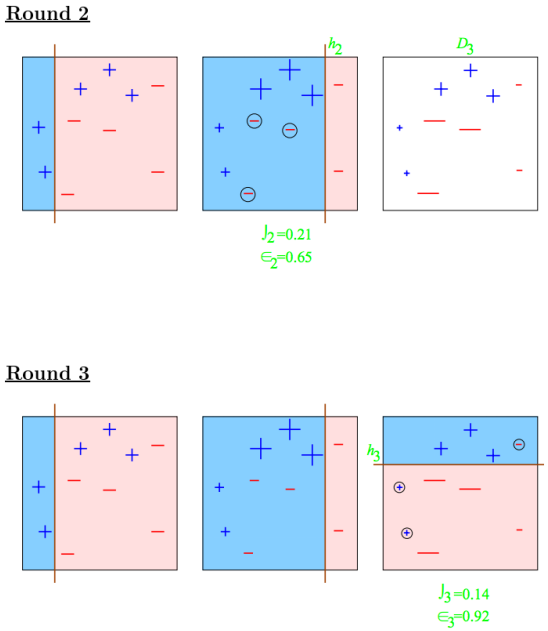
http://webee.technion.ac.il/people/rmeir/BoostingTutorial.pdf

Round 1: adaboost



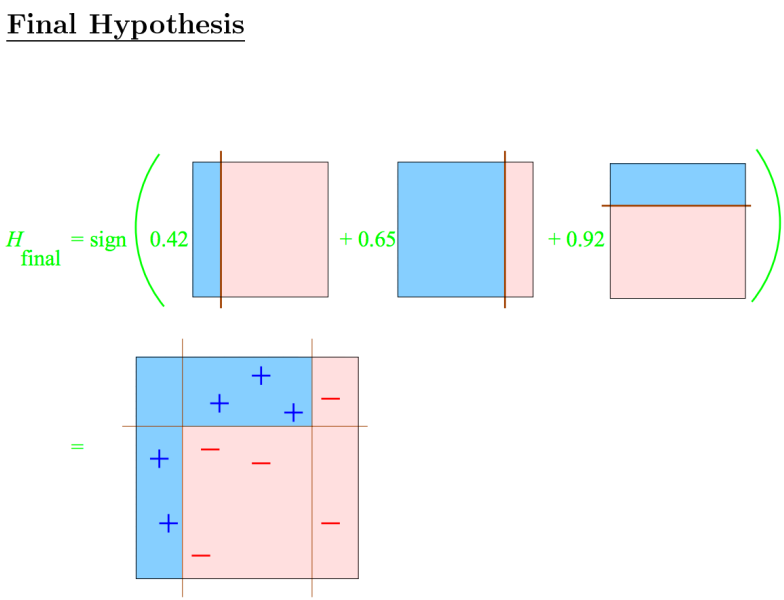
http://webee.technion.ac.il/people/rmeir/BoostingTutorial.pdf

Round 2 & 3



http://webee.technion.ac.il/people/rmeir/BoostingTutorial.pdf

Completed classifier



http://webee.technion.ac.il/people/rmeir/BoostingTutorial.pdf

Boosting in R

- Boosting can be used with any subset of classifiers
- One large subclass is gradient boosting
- R has multiple boosting libraries. Differences include the choice of basic classification functions and combination rules.
  - gbm - boosting with trees.
  - mboost - model based boosting
  - ada - statistical boosting based on additive logistic regression
  - gamBoost for boosting generalized additive models
- Most of these are available in the caret package

Wage example

```
library(ISLR); data(Wage); library(ggplot2); library(caret);
Wage <- subset(Wage,select=-c(logwage))
inTrain <- createDataPartition(y=Wage$wage,
                               p=0.7, list=FALSE)
training <- Wage[inTrain,]; testing <- Wage[-inTrain,]
```

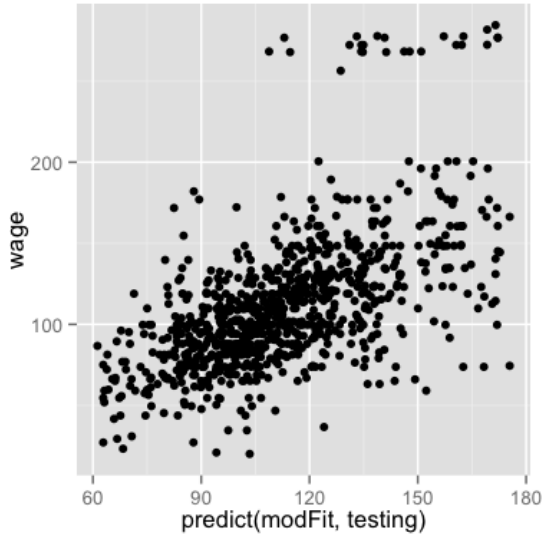
Fit the model

```
modFit <- train(wage ~ ., method="gbm",data=training,verbose=FALSE)
print(modFit)
```

```
2102 samples
  10 predictors

No pre-processing
Resampling: Bootstrap (25 reps)

Summary of sample sizes: 2102, 2102, 2102, 2102, 2102, 2102, ...

Resampling results across tuning parameters:

  interaction.depth  n.trees  RMSE  Rsquared  RMSE SD  Rsquared SD
  1                  50       30    0.3       1        0.02
  1                  100      30    0.3       1        0.02
  1                  200      30    0.3       1        0.02
  2                  50       30    0.3       1        0.02
  2                  100      30    0.3       1        0.02
  2                  200      30    0.3       1        0.02
```

Plot the results

```
qplot(predict(modFit,testing),wage,data=testing)
```



Notes and further reading

- A couple of nice tutorials for boosting
  - Freund and Shapire - http://www.cc.gatech.edu/~thad/6601-gradAI-fall2013/boosting.pdf
  - Ron Meir- http://webee.technion.ac.il/people/rmeir/BoostingTutorial.pdf
- Boosting, random forests, and model ensembling are the most common tools that win Kaggle and other prediction contests.
  - http://www.netflixprize.com/assets/GrandPrize2009_BPC_BigChaos.pdf
  - https://kaggle2.blob.core.windows.net/wiki-files/327/09ccf652-8c1c-4a3d-b979-ce2369c985e4/Willem%20Mestrom%20-%20Milestone%201%20Description%20V2%202.pdf

Model based prediction

Jeffrey Leek
Johns Hopkins Bloomberg School of Public Health

Basic idea

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

Pros:

- Can take advantage of structure of the data
- May be computationally convenient
- Are reasonably accurate on real problems

Cons:

- Make additional assumptions about the data
- When the model is incorrect you may get reduced accuracy

Model based approach

1. Our goal is to build parametric model for conditional distribution $P(Y = k|X = x)$
2. A typical approach is to apply Bayes theorem:

$$Pr(Y = k|X = x) = \frac{Pr(X = x|Y = k)Pr(Y = k)}{\sum_{\ell=1}^{K} Pr(X = x|Y = \ell)Pr(Y = \ell)}$$

$$Pr(Y = k|X = x) = \frac{f_k(x)\pi_k}{\sum_{\ell=1}^{K} f_\ell(x)\pi_\ell}$$

3. Typically prior probabilities π_k are set in advance.
4. A common choice for $f_k(x) = \frac{1}{\sigma_k\sqrt{2\pi}} e^{-\frac{(x-\mu_k)^2}{\sigma_k^2}}$, a Gaussian distribution
5. Estimate the parameters (μ_k, σ_k^2) from the data.
6. Classify to the class with the highest value of $P(Y = k|X = x)$

Classifying using the model

A range of models use this approach

- Linear discriminant analysis assumes $f_k(x)$ is multivariate Gaussian with same covariances
- Quadratic discrimant analysis assumes $f_k(x)$ is multivariate Gaussian with different covariances
- Model based prediction assumes more complicated versions for the covariance matrix
- Naive Bayes assumes independence between features for model building

http://statweb.stanford.edu/~tibs/ElemStatLearn/

Why linear discriminant analysis?

$$\log \frac{\Pr(Y = k|X = x)}{\Pr(Y = j|X = x)}$$

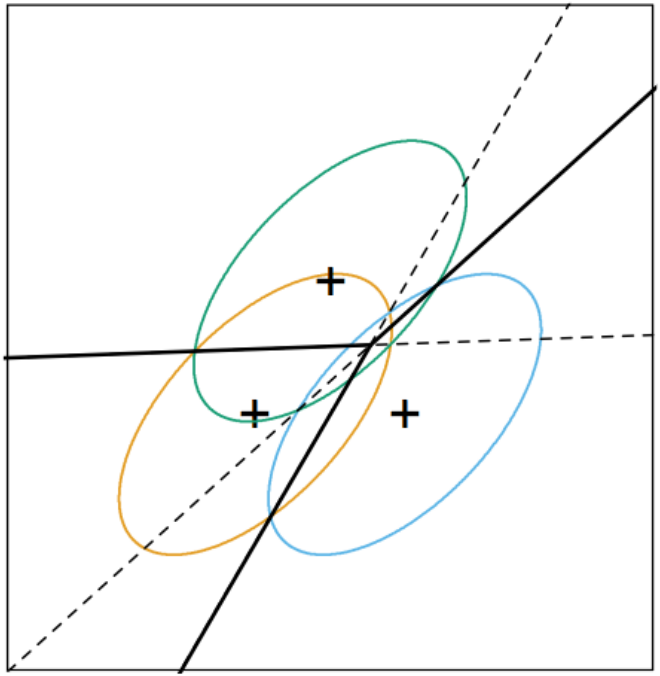
$$= \log \frac{f_k(x)}{f_j(x)} + \log \frac{\pi_k}{\pi_j}$$

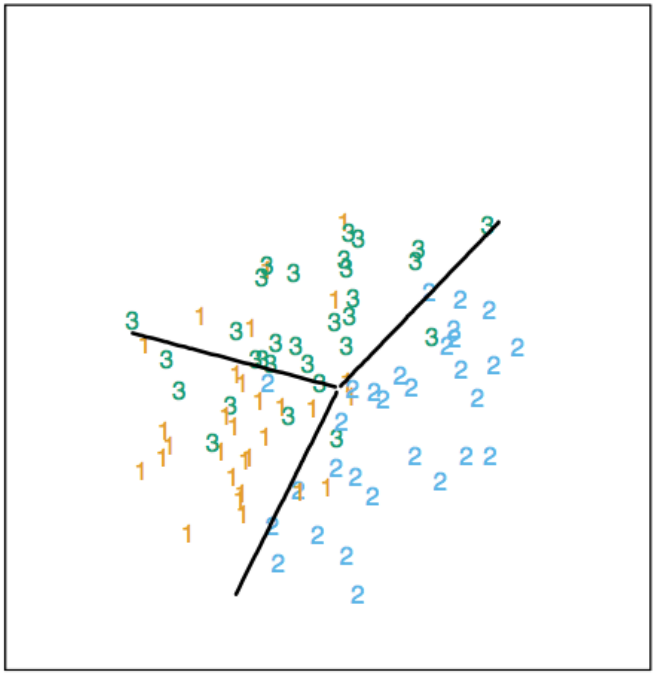
$$= \log \frac{\pi_k}{\pi_j} - \frac{1}{2}(\mu_k + \mu_j)^T \Sigma^{-1}(\mu_k + \mu_j)$$

$$+ x^T \Sigma^{-1}(\mu_k - \mu_j)$$

http://statweb.stanford.edu/~tibs/ElemStatLearn/

Decision boundaries





Discriminant function

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

- Decide on class based on $\hat{Y}(x) = \text{argmax}_k \delta_k(x)$
- We usually estimate parameters with maximum likelihood

Naive Bayes

Suppose we have many predictors, we would want to model: $P(Y = k|X_1, \ldots, X_m)$

We could use Bayes Theorem to get:

$$P(Y = k|X_1, \ldots, X_m) = \frac{\pi_k P(X_1, \ldots, X_m|Y = k)}{\sum_{\ell=1}^{K} P(X_1, \ldots, X_m|Y = k)\pi_\ell}$$

$$\propto \pi_k P(X_1, \ldots, X_m|Y = k)$$

This can be written:

$$P(X_1, \ldots, X_m, Y = k) = \pi_k P(X_1|Y = k)P(X_2, \ldots, X_m|X_1, Y = k)$$

$$= \pi_k P(X_1|Y = k)P(X_2|X_1, Y = k)P(X_3, \ldots, X_m|X_1, X_2, Y = k)$$

$$= \pi_k P(X_1|Y = k)P(X_2|X_1, Y = k) \ldots P(X_m|X_1 \ldots, X_{m-1}, Y = k)$$

We could make an assumption to write this:

$$\approx \pi_k P(X_1|Y = k)P(X_2|Y = k) \ldots P(X_m|, Y = k)$$

Example: Iris Data

```
data(iris); library(ggplot2)
names(iris)
```

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

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

```

    setosa versicolor  virginica
        50         50         50
```

Create training and test sets

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

```
[1] 45  5
```

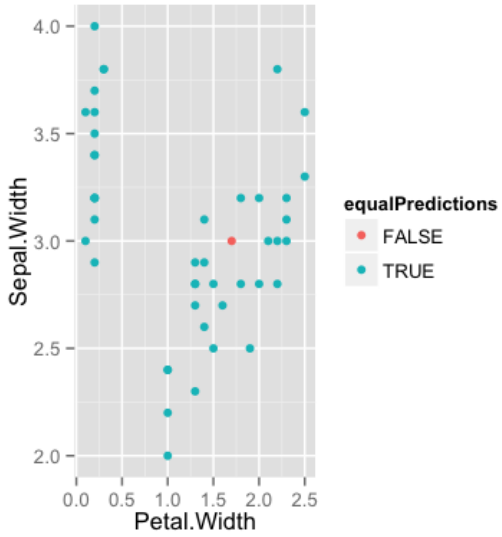
Build predictions

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

```
            pnb
plda         setosa versicolor virginica
  setosa         15          0         0
  versicolor      0         13         1
  virginica       0          0        16
```

Comparison of results

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



Notes and further reading

- Introduction to statistical learning
- Elements of Statistical Learning
- Model based clustering
- Linear Discriminant Analysis
- Quadratic Discriminant Analysis