Introduction to Classification & Regression Trees

ISLR Chapter 8

November 4, 2019

Classification and Regression Trees

Carseat data from ISLR package

Classification and Regression Trees

Carseat data from ISLR package

- Binary Outcome `High` 1 if Sales > 8, otherwise 0

Classification and Regression Trees

Carseat data from ISLR package

- Binary Outcome `High` 1 if Sales > 8, otherwise 0
- Fit a Classification tree model to `Price` and `Income`

Classification and Regression Trees

Carseat data from ISLR package

- Binary Outcome `High` 1 if Sales > 8, otherwise 0
- Fit a Classification tree model to `Price` and `Income`
- Pick a predictor and a cutpoint to split data

$$X_j \leq s \text{ and } X_k > s$$

to minimize deviance (or SSE for regression) - leads to a root node in a tree

Classification and Regression Trees

Carseat data from ISLR package

- Binary Outcome `High` 1 if Sales > 8, otherwise 0
- Fit a Classification tree model to `Price` and `Income`
- Pick a predictor and a cutpoint to split data

$$X_j \leq s \text{ and } X_k > s$$

to minimize deviance (or SSE for regression) - leads to a root node in a tree

- continue splitting/partitioning data until stopping criterion is reached (number of observations in a node > 10 and within node deviance > 0.01 deviance of the root node)

Classification and Regression Trees

Carseat data from ISLR package

- Binary Outcome `High` 1 if Sales > 8, otherwise 0
- Fit a Classification tree model to `Price` and `Income`
- Pick a predictor and a cutpoint to split data

$$X_j \leq s \text{ and } X_k > s$$

to minimize deviance (or SSE for regression) - leads to a root node in a tree

- continue splitting/partitioning data until stopping criterion is reached (number of observations in a node > 10 and within node deviance > 0.01 deviance of the root node)
- Prediction is mean or proportion of successes of data in terminal nodes

Classification and Regression Trees

Carseat data from ISLR package

- Binary Outcome `High` 1 if Sales > 8, otherwise 0
- Fit a Classification tree model to `Price` and `Income`
- Pick a predictor and a cutpoint to split data

$$X_j \leq s \text{ and } X_k > s$$

to minimize deviance (or SSE for regression) - leads to a root node in a tree

- continue splitting/partitioning data until stopping criterion is reached (number of observations in a node > 10 and within node deviance > 0.01 deviance of the root node)
- Prediction is mean or proportion of successes of data in terminal nodes
- Output is a decision tree

Classification and Regression Trees

Carseat data from ISLR package

- Binary Outcome `High` 1 if Sales > 8, otherwise 0
- Fit a Classification tree model to `Price` and `Income`
- Pick a predictor and a cutpoint to split data

$$X_j \leq s \text{ and } X_k > s$$

to minimize deviance (or SSE for regression) - leads to a root node in a tree

- continue splitting/partitioning data until stopping criterion is reached (number of observations in a node > 10 and within node deviance > 0.01 deviance of the root node)
- Prediction is mean or proportion of successes of data in terminal nodes
- Output is a decision tree
- regression or classification function is nonlinear in predictors

Classification and Regression Trees

Carseat data from ISLR package

- Binary Outcome `High` 1 if Sales > 8, otherwise 0
- Fit a Classification tree model to `Price` and `Income`
- Pick a predictor and a cutpoint to split data

$$X_j \leq s \text{ and } X_k > s$$

to minimize deviance (or SSE for regression) - leads to a root node in a tree

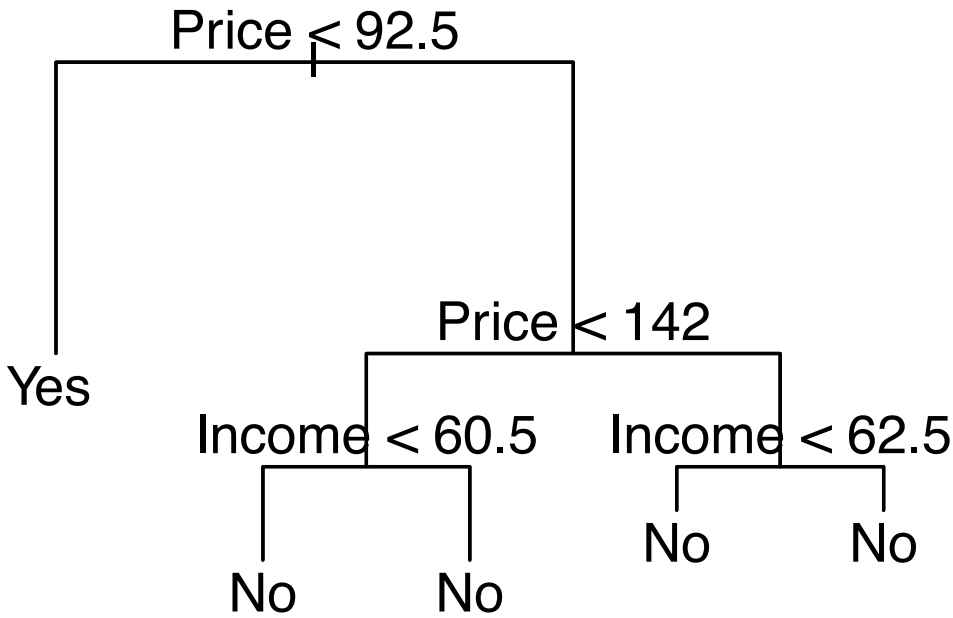
- continue splitting/partitioning data until stopping criterion is reached (number of observations in a node > 10 and within node deviance > 0.01 deviance of the root node)
- Prediction is mean or proportion of successes of data in terminal nodes
- Output is a decision tree
- regression or classification function is nonlinear in predictors
- Captures interactions

Carseat Example

```
library(tree)
data(Carseats)
  Carseats = mutate(Carseats, High=factor(ifelse (
  Carseats$Sales <=8,
    "No",
    "Yes "))
  )

tree.carseats = tree(High ~ Price + Income,
                     data=Carseats)
```

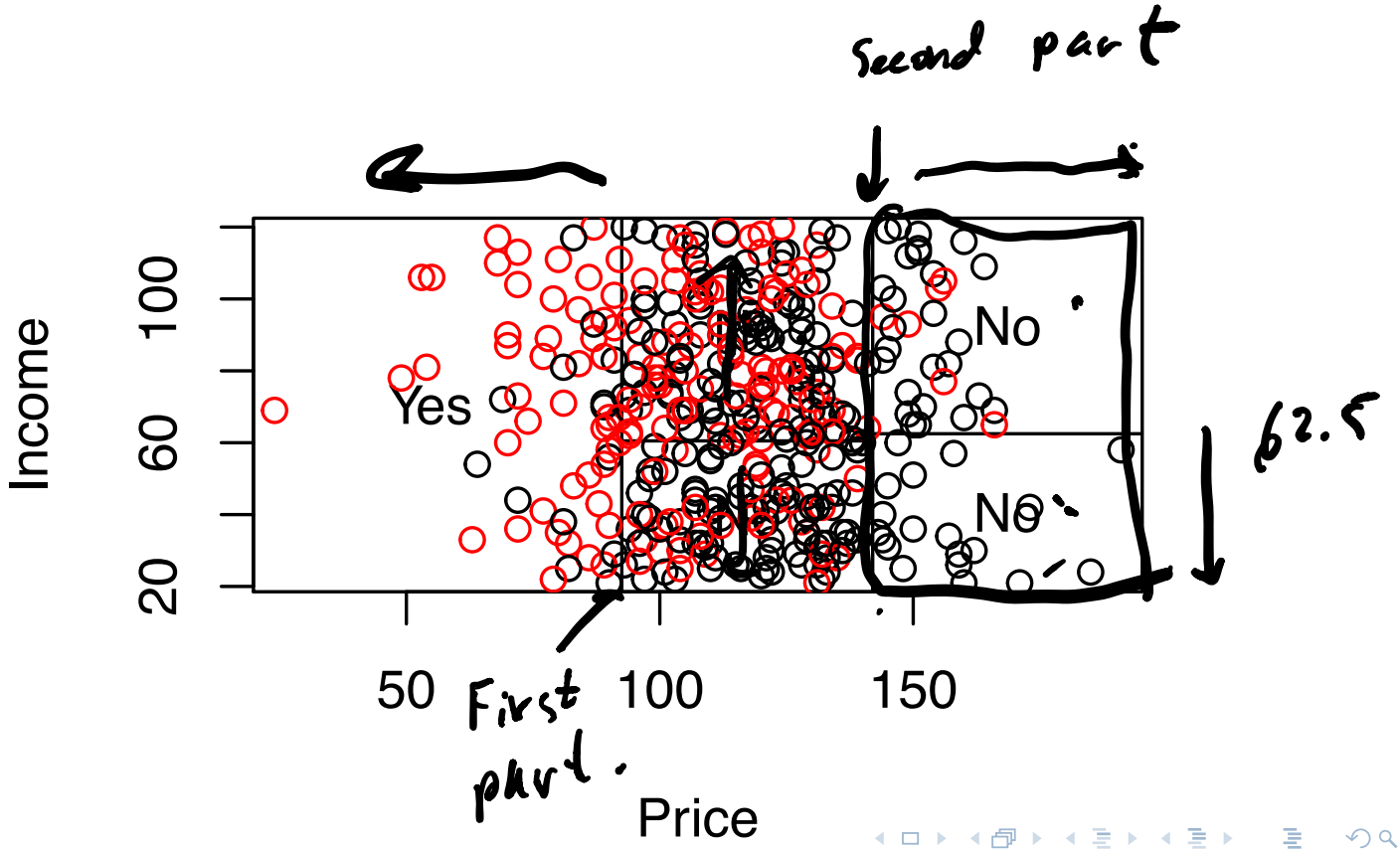
Carseat Example

```
plot(tree.carseats)
text(tree.carseats)
```



Partition

```
partition.tree(tree.carseats)
points(Carseats$Price,Carseats$Income, col=Carseats$High)
```



Splits

```
tree.carseats

## node), split, n, deviance, yval, (yprob)
##       * denotes terminal node
##
##  1) root 400 541.50 No ( 0.5900 0.4100 )
##    2) Price < 92.5 62  66.24 Yes  ( 0.2258 0.7742 ) *
##    3) Price > 92.5 338 434.80 No ( 0.6568 0.3432 )
##      6) Price < 142 287 382.10 No ( 0.6167 0.3833 )
##       12) Income < 60.5 113 128.70 No ( 0.7434 0.2566 )
##       13) Income > 60.5 174 240.40 No ( 0.5345 0.4655 )
##      7) Price > 142 51  36.95 No ( 0.8824 0.1176 )
##       14) Income < 62.5 19   0.00 No ( 1.0000 0.0000 ) *
##       15) Income > 62.5 32  30.88 No ( 0.8125 0.1875 ) *
```

Summary

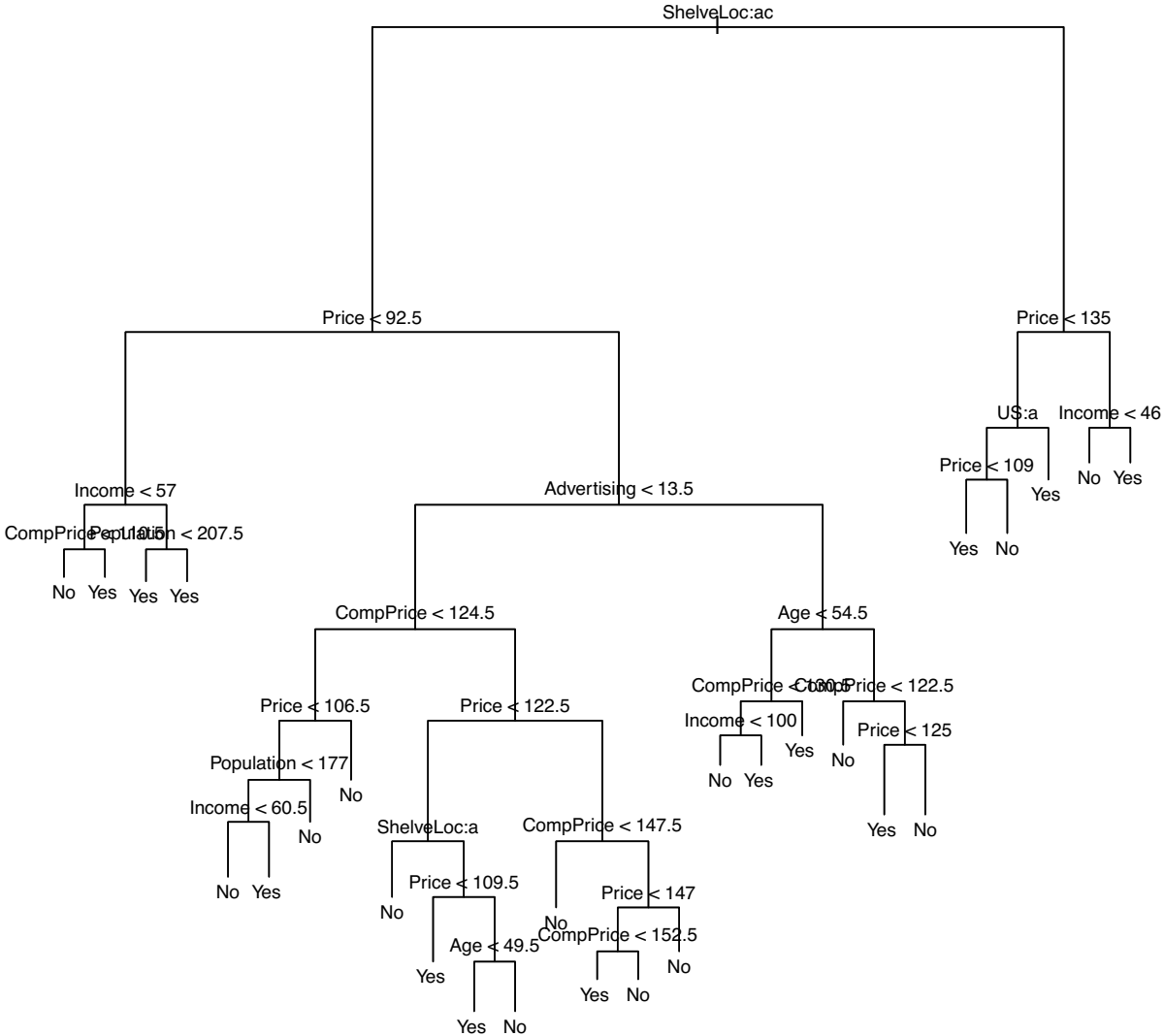
```
 summary(tree.carseats)

##
## Classification tree:
## tree(formula = High ~ Price + Income, data = Carseats)
## Number of terminal nodes:  5
## Residual mean deviance:  1.18 = 466.2 / 395
## Misclassification error rate: 0.325 = 130 / 400
```

All Variables

```
tree.carseats =tree(High ~   . -Sales, data=Carseats )
summary(tree.carseats)

##
## Classification tree:
## tree(formula = High ~ . - Sales, data = Carseats)
## Variables actually used in tree construction:
## [1] "ShelveLoc"   "Price"       "Income"      "CompPrice
## [6] "Advertising" "Age"         "US"
## Number of terminal nodes:  27
## Residual mean deviance:  0.4575 = 170.7 / 373
## Misclassification error rate: 0.09 = 36 / 400
```

Overfitting?

Tree


ShelveLoc:ac
Price < 92.5
Price < 135
Income < 57
Advertising < 13.5
US:a
Income < 46
Price < 109
No
Yes
Yes
Yes
No
CompPrice < 118.5
Population < 207.5
No
Yes
Yes
Yes
CompPrice < 124.5
Age < 54.5
Price < 106.5
Price < 122.5
Income < 100
Price < 125
Yes
No
No
Yes
Yes
No
Population < 177
No
Income < 60.5
No
No
Yes
ShelveLoc:a
CompPrice < 147.5
Price < 109.5
No
Price < 147
No
Age < 49.5
CompPrice < 152.5
No
Yes
Yes
No
Yes
No

Classification Error

```
set.seed (2)
train=sample (1: nrow(Carseats ), 200)
Carseats.test=Carseats [-train ,]

tree.carseats =tree(High ~  . -Sales,
                    data=Carseats,subset=train )
tree.pred=predict (tree.carseats ,Carseats.test ,type ="cla
table(tree.pred , Carseats.test$High)
```

```
##
##  tree.pred  No Yes
##        No  104  33
##        Yes  13  50
```

confusion matrix

```
 (33 + 13) /200    # classification error
```

```
## [1] 0.23
```

Cost-Complexity Pruning

1. Grow a large tree on training data, stopping when each terminal node has fewer than some minimum number of observations

Cost-Complexity Pruning

1. Grow a large tree on training data, stopping when each terminal node has fewer than some minimum number of observations
2. Prediction for region m is the Class c that $max_c \hat{\pi}_{mc}$

Cost-Complexity Pruning

1. Grow a large tree on training data, stopping when each terminal node has fewer than some minimum number of observations
2. Prediction for region m is the Class c that $max_c \hat{\pi}_{mc}$
3. Snip off the least important splits via cost-complexity pruning to the tree in order to obtain a sequence of best subtrees indexed by cost parameter k,

Cost-Complexity Pruning

1. Grow a large tree on training data, stopping when each terminal node has fewer than some minimum number of observations
2. Prediction for region m is the Class c that $max_c \hat{\pi}_{mc}$
3. Snip off the least important splits via cost-complexity pruning to the tree in order to obtain a sequence of best subtrees indexed by cost parameter k,

$$\frac{N_{miss}}{N} - k|T|$$

criteria for model selection

misclassification error penalized by number of terminal nodes

Cost-Complexity Pruning

1. Grow a large tree on training data, stopping when each terminal node has fewer than some minimum number of observations
2. Prediction for region m is the Class c that $max_c \hat{\pi}_{mc}$
3. Snip off the least important splits via cost-complexity pruning to the tree in order to obtain a sequence of best subtrees indexed by cost parameter k,

$$\frac{N_{miss}}{N} - k|T|$$

   missclassification error penalized by number of terminal nodes
4. Using K-fold cross validation, compute average cost-complexity for each k

Cost-Complexity Pruning

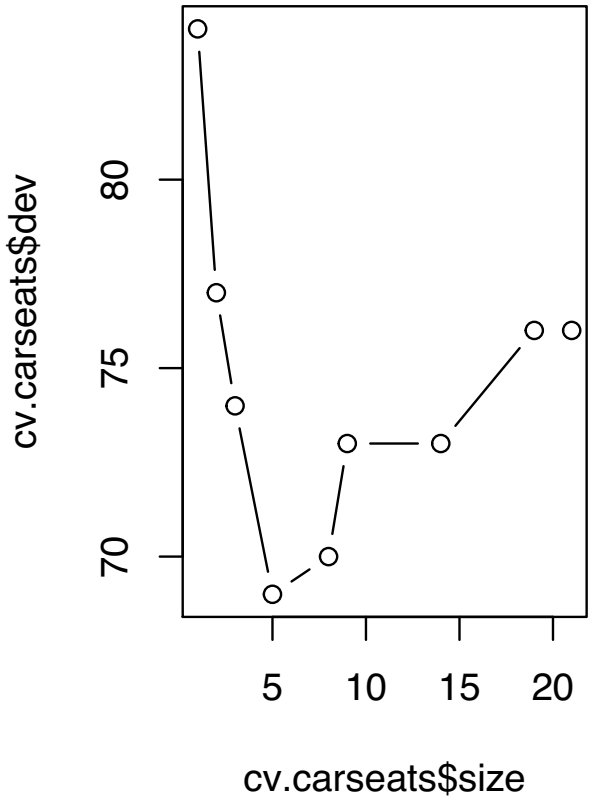
1. Grow a large tree on training data, stopping when each terminal node has fewer than some minimum number of observations
2. Prediction for region m is the Class c that $max_c \hat{\pi}_{mc}$
3. Snip off the least important splits via cost-complexity pruning to the tree in order to obtain a sequence of best subtrees indexed by cost parameter k,

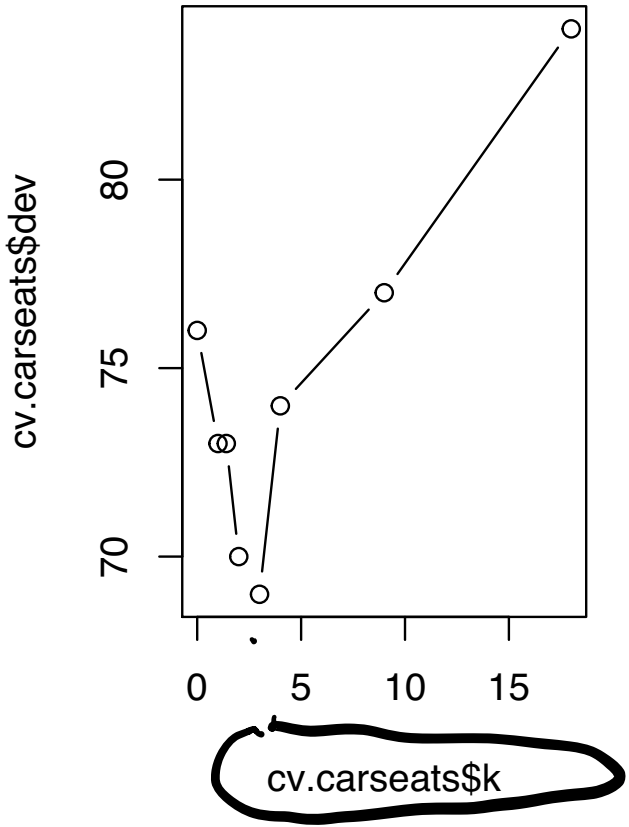
$$\frac{N_{miss}}{N} - k|T|$$

   missclassification error penalized by number of terminal nodes
4. Using K-fold cross validation, compute average cost-complexity for each k
5. Pick subtree with smallest penalized error

Pruning via Cross Validation)

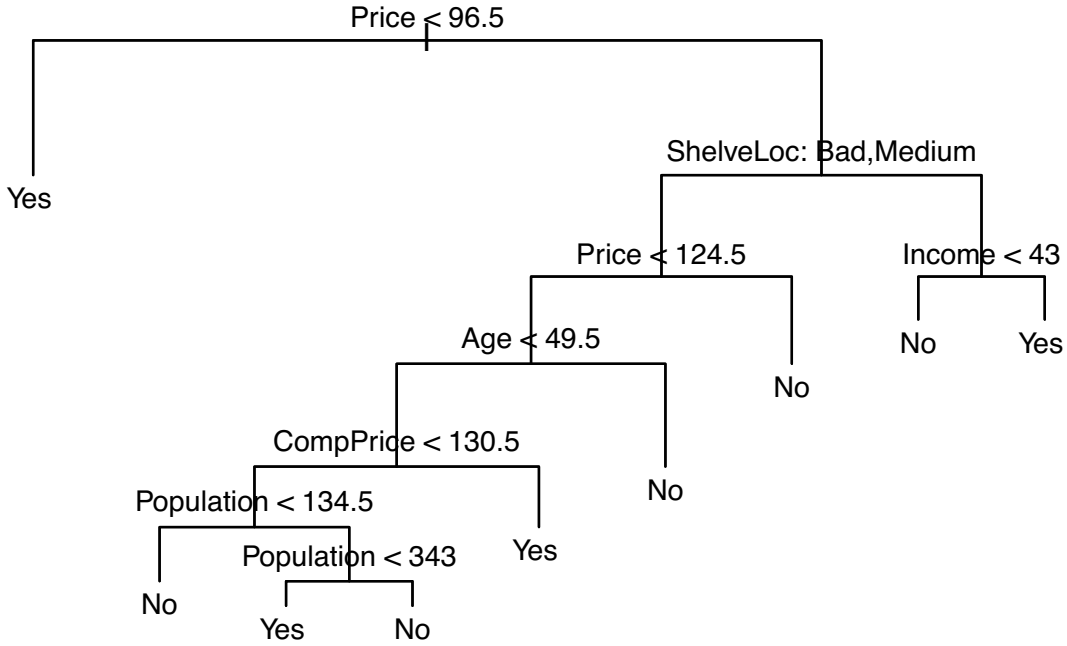
```
set.seed(2)
cv.carseats = cv.tree(tree.carseats, FUN=prune.misclass)
```





Pruned

```
prune.carseats = prune.misclass(tree.carseats ,best =9)
```



Miss-classification after Selection

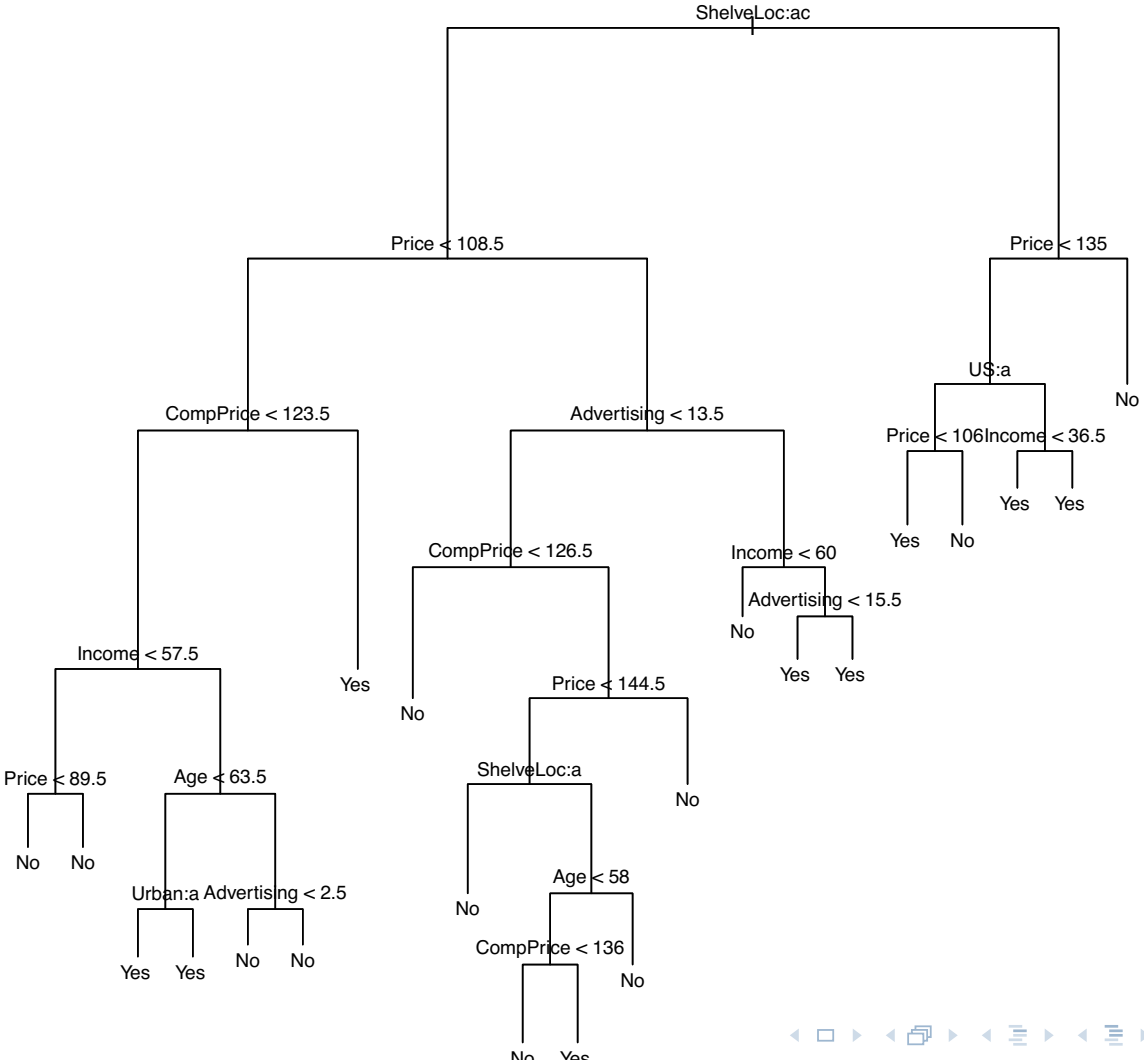
```
tree.pred=predict(prune.carseats , Carseats.test,
                  type="class")
table(tree.pred ,Carseats.test$High)
```

```
##
## tree.pred No Yes
##       No   97   25
##       Yes  20   58
```

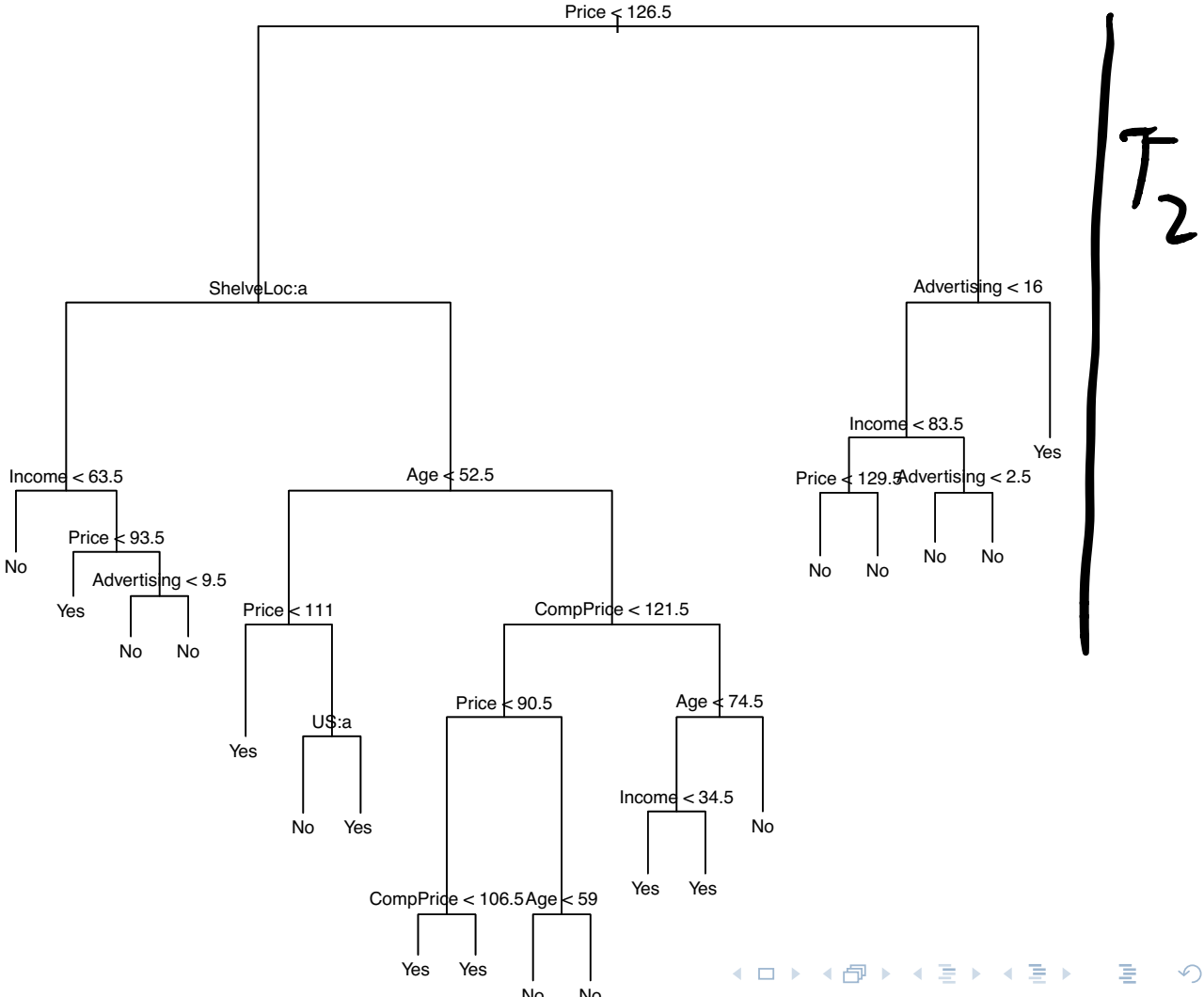
```
(97 + 58)/200  # classified Correctly
```

```
## [1] 0.775
```

Tree with Random Split of Data



Tree with another Random Split of Data



Bagging: Bootstrap Aggregation | Aggregating trees

- Splitting data into random partitions and fitting a tree model on each half may lead to very different predictions (high variability)

$$\bar{T} = \text{Avg}(T_1, T_2)$$

Argue: $\text{Var}(\bar{T}) \ll \text{Var}(T_1)$

$\text{Var}(\bar{T}) \ll \text{Var}(T_2)$

$\text{Bias}(\bar{T}) \approx \text{Bias}(T_1)$

$\simeq \text{Bias}(T_2)$

Bagging: Bootstrap Aggregation

- Splitting data into random partitions and fitting a tree model on each half may lead to very different predictions (high variability)
- Reduce variability by averaging over multiple training sets!

Bagging: Bootstrap Aggregation

- Splitting data into random partitions and fitting a tree model on each half may lead to very different predictions (high variability)
- Reduce variability by averaging over multiple training sets!
- do not have access to multiple training sets so create them via bootstrap samples (sample of size n with replacement)

Bagging: Bootstrap Aggregation

- Splitting data into random partitions and fitting a tree model on each half may lead to very different predictions (high variability)
- Reduce variability by averaging over multiple training sets!
- do not have access to multiple training sets so create them via bootstrap samples (sample of size n with replacement)
  - Generate B bootstrap sample of observations from the single training data

Bagging: Bootstrap Aggregation

- Splitting data into random partitions and fitting a tree model on each half may lead to very different predictions (high variability)
- Reduce variability by averaging over multiple training sets!
- do not have access to multiple training sets so create them via bootstrap samples (sample of size n with replacement)
  - Generate B bootstrap sample of observations from the single training data
  - Calculate predictions for the bth sample $\hat{f}^b(x)$

Bagging: Bootstrap Aggregation

- Splitting data into random partitions and fitting a tree model on each half may lead to very different predictions (high variability)
- Reduce variability by averaging over multiple training sets!
- do not have access to multiple training sets so create them via bootstrap samples (sample of size n with replacement)
  - Generate B bootstrap sample of observations from the single training data
  - Calculate predictions for the bth sample $\hat{f}^b(x)$
  - Bagging (Bootstrap Aggregation) estimate is

$$\hat{f}_{\text{bag}}(x) = \frac{1}{B}\sum \hat{f}^b(x)$$

- Trees are grown deep so little bias (although could prune)

Avoid pruning.

Model Averaging vs.

Model Selection

Bagging: Bootstrap Aggregation

- Splitting data into random partitions and fitting a tree model on each half may lead to very different predictions (high variability)
- Reduce variability by averaging over multiple training sets!
- do not have access to multiple training sets so create them via bootstrap samples (sample of size n with replacement)
  - Generate B bootstrap sample of observations from the single training data
  - Calculate predictions for the bth sample $\hat{f}^b(x)$
  - Bagging (Bootstrap Aggregation) estimate is

$$\hat{f}_{\text{bag}}(x) = \frac{1}{B}\sum \hat{f}^b(x)$$

- Trees are grown deep so little bias (although could prune)
- Reduce variance by averaging many trees across the bootstrap samples

Bagging: Bootstrap Aggregation

- Splitting data into random partitions and fitting a tree model on each half may lead to very different predictions (high variability)
- Reduce variability by averaging over multiple training sets!
- do not have access to multiple training sets so create them via bootstrap samples (sample of size n with replacement)
  - Generate B bootstrap sample of observations from the single training data
  - Calculate predictions for the bth sample $\hat{f}^b(x)$
  - Bagging (Bootstrap Aggregation) estimate is

$$\hat{f}_{\text{bag}}(x) = \frac{1}{B} \sum \hat{f}^b(x)$$

- Trees are grown deep so little bias (although could prune)
- Reduce variance by averaging many trees across the bootstrap samples

Next Class

Trees are simple to understand, but not as competitive with other supervised learning approaches for prediction/classification.

Next Class

Trees are simple to understand, but not as competitive with other supervised learning approaches for prediction/classification.
Ways to improving Trees through multiple trees in some ensemble:

- Bagging

Next Class

Trees are simple to understand, but not as competitive with other supervised learning approaches for prediction/classification.
Ways to improving Trees through multiple trees in some ensemble:

- Bagging
- Random Forests

Next Class

Trees are simple to understand, but not as competitive with other supervised learning approaches for prediction/classification.
Ways to improving Trees through multiple trees in some ensemble:

- Bagging
- Random Forests
- Boosting

Next Class

Trees are simple to understand, but not as competitive with other supervised learning approaches for prediction/classification.
Ways to improving Trees through multiple trees in some ensemble:

- Bagging
- Random Forests
- Boosting
- BART (Bayesian Additive Regression Trees)

Next Class

Trees are simple to understand, but not as competitive with other supervised learning approaches for prediction/classification.
Ways to improving Trees through multiple trees in some ensemble:

- Bagging
- Random Forests
- Boosting
- BART (Bayesian Additive Regression Trees)

Combining trees will yield improved prediction accuracy, but with loss of interpretability.

Random Forests and Related Tree Ensembles

ISLR Chapter 8

November 8, 2019

Outline

Ways to improving trees through multiple trees in some ensemble:

- Bagging

Outline

Ways to improving trees through multiple trees in some ensemble:

- Bagging
- Random Forests

Outline

Ways to improving trees through multiple trees in some ensemble:

- Bagging
- Random Forests
- Boosting

Outline

Ways to improving trees through multiple trees in some ensemble:

- Bagging
- Random Forests
- Boosting
- BART (Bayesian Additive Regression Trees)

Outline

Ways to improving trees through multiple trees in some ensemble:

- Bagging
- Random Forests
- Boosting
- BART (Bayesian Additive Regression Trees)

Combining trees will yield improved prediction accuracy, but with loss of interpretability.

Random Forests

Closely related to Bagging, but attempts to de-correlate the trees used in the average

Random Forests

Closely related to Bagging, but attempts to de-correlate the trees used in the average

- $\hat{Y} = \sum(\hat{Y}^b)$,
  $\text{Var}(\sum(\hat{Y}^b)) = \sum(\text{Var}(\hat{Y}^b)) + 2\sum_{b,b'} \text{Cov}(\hat{Y}^b, \hat{Y}^{b'})$ $= \text{Var}(\hat{Y})$
- take B bootstrap samples

Random Forests

Closely related to Bagging, but attempts to de-correlate the trees used in the average

- $\hat{Y} = \sum(\hat{Y}^b)$,
  $\text{Var}(\sum(\hat{Y}^b)) = \sum(\text{Var}(\hat{Y}^b)) + 2\sum_{b,b'} \text{Cov}(\hat{Y}^b, \hat{Y}^{b'})$
- take B bootstrap samples
- Each time a split in a tree is considered, a random sample of m predictors is chosen as split candidates from the set of p predictors.

Random Forests

Closely related to Bagging, but attempts to de-correlate the trees used in the average

- $\hat{Y} = \sum(\hat{Y}^b)$,
  $\text{Var}(\sum(\hat{Y}^b)) = \sum(\text{Var}(\hat{Y}^b)) + 2\sum_{b,b'} \text{Cov}(\hat{Y}^b, \hat{Y}^{b'})$
- take B bootstrap samples
- Each time a split in a tree is considered, a random sample of m predictors is chosen as split candidates from the set of p predictors.
- a new sample is taken at each split

Random Forests

Closely related to Bagging, but attempts to de-correlate the trees used in the average

- $\hat{Y} = \sum(\hat{Y}^b)$,
  $\text{Var}(\sum(\hat{Y}^b)) = \sum(\text{Var}(\hat{Y}^b)) + 2\sum_{b,b'} \text{Cov}(\hat{Y}^b, \hat{Y}^{b'})$
- take B bootstrap samples
- Each time a split in a tree is considered, a random sample of m predictors is chosen as split candidates from the set of p predictors.
- a new sample is taken at each split
- Recommended m around $\sqrt{p}$

Random Forests

Closely related to Bagging, but attempts to de-correlate the trees used in the average

- $\hat{Y} = \sum(\hat{Y}^b)$,
  $\text{Var}(\sum(\hat{Y}^b)) = \sum(\text{Var}(\hat{Y}^b)) + 2\sum_{b,b'} \text{Cov}(\hat{Y}^b, \hat{Y}^{b'})$
- take B bootstrap samples
- Each time a split in a tree is considered, a random sample of m predictors is chosen as split candidates from the set of p predictors.
- a new sample is taken at each split
- Recommended m around $\sqrt{p}$
- Random Forests with $m = p$ is Bagging

at each split pick a random subset of features

Random Forests

Closely related to Bagging, but attempts to de-correlate the trees used in the average

- $\hat{Y} = \sum(\hat{Y}^b)$,
  $\text{Var}(\sum(\hat{Y}^b)) = \sum(\text{Var}(\hat{Y}^b)) + 2\sum_{b,b'} \text{Cov}(\hat{Y}^b, \hat{Y}^{b'})$
- take B bootstrap samples
- Each time a split in a tree is considered, a random sample of m predictors is chosen as split candidates from the set of p predictors.
- a new sample is taken at each split
- Recommended m around $\sqrt{p}$
- Random Forests with $m = p$ is Bagging
- Predictions are based on average over the bootstrap samples

Random Forests

Closely related to Bagging, but attempts to de-correlate the trees used in the average

- $\hat{Y} = \sum(\hat{Y}^b)$,
  $\text{Var}(\sum(\hat{Y}^b)) = \sum(\text{Var}(\hat{Y}^b)) + 2\sum_{b,b'} \text{Cov}(\hat{Y}^b, \hat{Y}^{b'})$
- take B bootstrap samples
- Each time a split in a tree is considered, a random sample of m predictors is chosen as split candidates from the set of p predictors.
- a new sample is taken at each split
- Recommended m around $\sqrt{p}$
- Random Forests with $m = p$ is Bagging
- Predictions are based on average over the bootstrap samples

Bagging Example

```
suppressMessages(library(randomForest))
bag.carseats = randomForest(High ~ . -Sales,
                            data=Carseats, subset = train,
                            mtry=10, importance =TRUE)
#  mtry = 10 (the number of predictors)
yhat.bag = predict(bag.carseats,newdata = Carseats.test)
tab = table(yhat.bag,Carseats.test$High)
tab

##
## yhat.bag  No Yes
##      No   104   24
##      Yes   13   59

class.bag = (tab[1,1] + tab[2,2])/sum(tab)
class.bag

## [1] 0.815
```

RandomForests

```
 rf.carseats = randomForest(High ~ . -Sales,data=Carseats,
                            subset =train,
                            mtry=3, importance =TRUE)
yhat.rf= predict(rf.carseats ,newdata =Carseats.test)
tab = table(yhat.rf,Carseats.test$High)
tab

##
## yhat.rf  No Yes
##     No  110   24
##     Yes   7   59

(tab[1,1] + tab[2,2])/sum(tab)

## [1] 0.845
```

Variable Importance Measures in Bagging & RandomForests

- For Regression Trees use the total reduction in Sum of Squares Error due to splits with that variable averaged over all trees

Variable Importance Measures in Bagging & RandomForests

- For Regression Trees use the total reduction in Sum of Squares Error due to splits with that variable averaged over all trees
- For Classification Trees use the total reduction in Gini Index due to splits of the that variable averaged over all trees. Gini Index is defined as

$$G = \sum_{k=1}^{K} \hat{\pi}_{mk}(1 - \hat{\pi}_{mk})$$

where K is the number of classes for region m or use the reduction in deviance

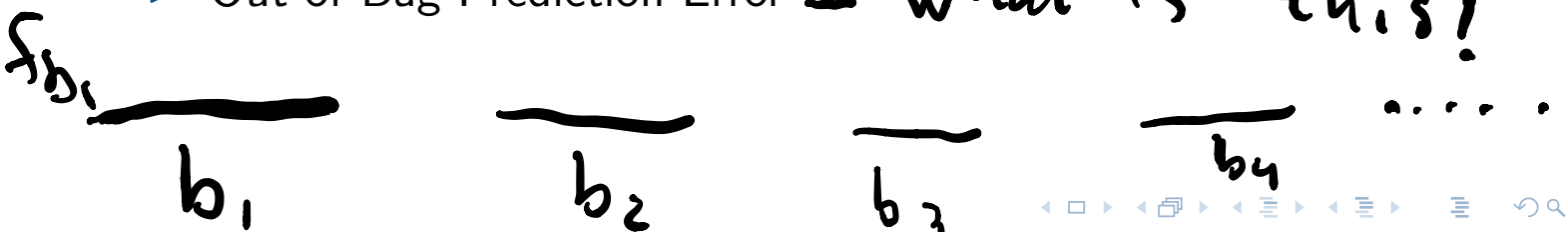
Variable Importance Measures in Bagging & RandomForests

- For Regression Trees use the total reduction in Sum of Squares Error due to splits with that variable averaged over all trees
- For Classification Trees use the total reduction in Gini Index due to splits of the that variable averaged over all trees. Gini Index is defined as

$$G = \sum_{k=1}^{K} \hat{\pi}_{mk}(1 - \hat{\pi}_{mk})$$

where K is the number of classes for region m or use the reduction in deviance

- Out of Bag Prediction Error $\approx$ CV — What is this?



Variable Importance Measures in Bagging & RandomForests

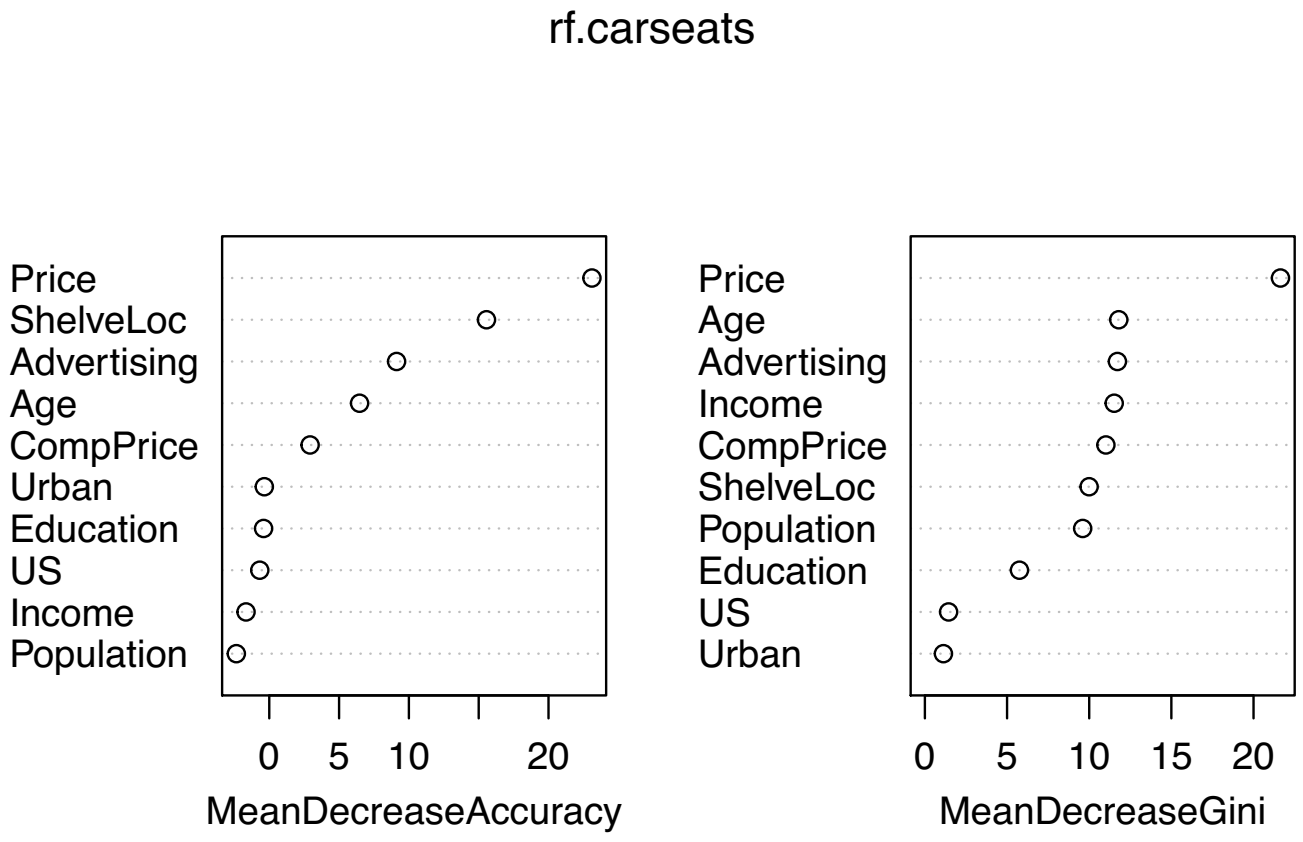
- For Regression Trees use the total reduction in Sum of Squares Error due to splits with that variable averaged over all trees
- For Classification Trees use the total reduction in Gini Index due to splits of the that variable averaged over all trees. Gini Index is defined as

$$G = \sum_{k=1}^{K} \hat{\pi}_{mk}(1 - \hat{\pi}_{mk})$$

  where K is the number of classes for region m or use the reduction in deviance
- Out of Bag Prediction Error
- May be normalized by dividing by the maximum

Variable Importance Measures in RandomForests

```
varImpPlot(rf.carseats)
```



Boosting

Algorithm 8.2 *Boosting for Regression Trees*

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

$0 < \lambda < 1$

multiplicative update will add an explicit expansion

linear comb. of trees

Boosting

- Mean is a sum of B trees

Boosting

- Mean is a sum of B trees
- Boosting can over-fit if B is too large

Boosting

- Mean is a sum of B trees
- Boosting can over-fit if B is too large
- The number of splits, d, in each tree controls the complexity of the boosted ensemble; $d = 1$ corresponds to models of stumps

Boosting

- Mean is a sum of B trees
- Boosting can over-fit if B is too large
- The number of splits, d, in each tree controls the complexity of the boosted ensemble; $d = 1$ corresponds to models of stumps
- d is the interaction depth; d splits can involve d variables. (default in `gbm` package is 4)

Boosting

- Mean is a sum of B trees
- Boosting can over-fit if B is too large
- The number of splits, d, in each tree controls the complexity of the boosted ensemble; $d = 1$ corresponds to models of stumps
- d is the interaction depth; d splits can involve d variables. (default in `gbm` package is 4)

Boosting Example 'library(gdm)'

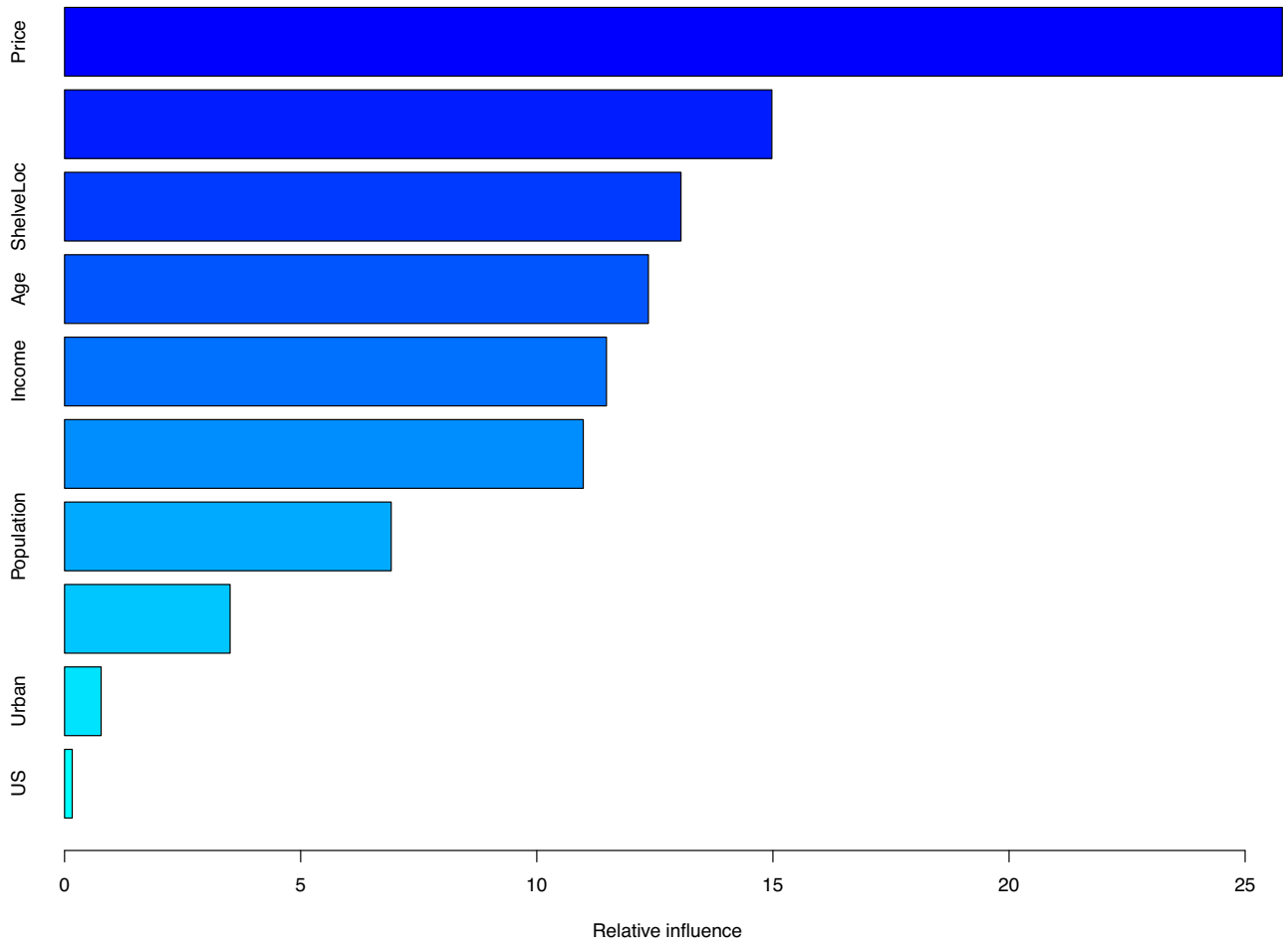
```
boost.car = gbm(I(as.numeric(High)-1) ~ . -Sales,
                data=Carseats[train ,],
                distribution="bernoulli",
                n.trees =5000, interaction.depth =4)
pihat.boost = predict(boost.car, newdata=Carseats.test,
                      n.trees=5000, type="response")
yhat.boost = ifelse(pihat.boost > .5, 1, 0)
tab = table(yhat.boost,Carseats.test$High)
tab

##
## yhat.boost  No Yes
##          0 108   19
##          1   9   64

(tab[1,1] + tab[2,2])/sum(tab)

## [1] 0.86
```

Variable Importance: Boosting

```
summary(boost.car, plotit=FALSE)
```

```
##                        var    rel.inf
## Price                Price 25.7868489
## CompPrice        CompPrice 14.9777415
## ShelveLoc        ShelveLoc 13.0530405
## Age                    Age 12.3632640
## Income              Income 11.4755044
## Advertising Advertising 10.9841631
## Population   Population  6.9170104
## Education     Education  3.5053057
## Urban             Urban  0.7745811
## US                   US  0.1625404
```

Variable Importance: summary(boost.car)



Bayesian Additive Regression Trees

Gaussian Model: Single Tree model

$$Y = g(x, T, M) + \epsilon$$

Where T is a tree and $M = (\mu_1, \ldots \mu_b)^T$ is the vector of means at the terminal nodes of the tree given by T

Bayesian Additive Regression Trees

Gaussian Model: Single Tree model

$$Y = g(x, T, M) + \epsilon$$

Where T is a tree and $M = (\mu_1, \ldots \mu_b)^T$ is the vector of means at the terminal nodes of the tree given by T

BART represents the mean function as

$$Y = \sum_j g(x, T_j, M_j) + \epsilon$$

as a sum of trees.

Bayesian Additive Regression Trees

Gaussian Model: Single Tree model

$$Y = g(x, T, M) + \epsilon$$

Where T is a tree and $M = (\mu_1, \ldots \mu_b)^T$ is the vector of means at the terminal nodes of the tree given by T

BART represents the mean function as

$$Y = \sum_j g(x, T_j, M_j) + \epsilon$$

as a sum of trees.
Priors control the complexity of each tree; back-fitting as in boosting.

Bayesian Additive Regression Trees

Gaussian Model: Single Tree model

$$Y = g(x, T, M) + \epsilon$$

Where T is a tree and $M = (\mu_1, \ldots \mu_b)^T$ is the vector of means at the terminal nodes of the tree given by T

BART represents the mean function as

$$Y = \sum_j g(x, T_j, M_j) + \epsilon$$

as a sum of trees.
Priors control the complexity of each tree; back-fitting as in boosting.

BART: Bayesian Additive Regression Trees

```
library(BART); set.seed(42)
X.Seats = model.matrix(High ~ . -Sales,
                       data = Carseats)[, -1]
bart.carseats = pbart(x.train=X.Seats[train, ],
                      y.train=as.numeric(Carseats$High[train])-1,
                      printevery=1000L)
pred.bart = predict(bart.carseats,
                    newdata=X.Seats[-train,])
pihat.bart = pred.bart$prob.test.mean
yhat.bart = ifelse(pihat.bart > .5, 1, 0)
tab = table(yhat.bart, Carseats.test$High)
```

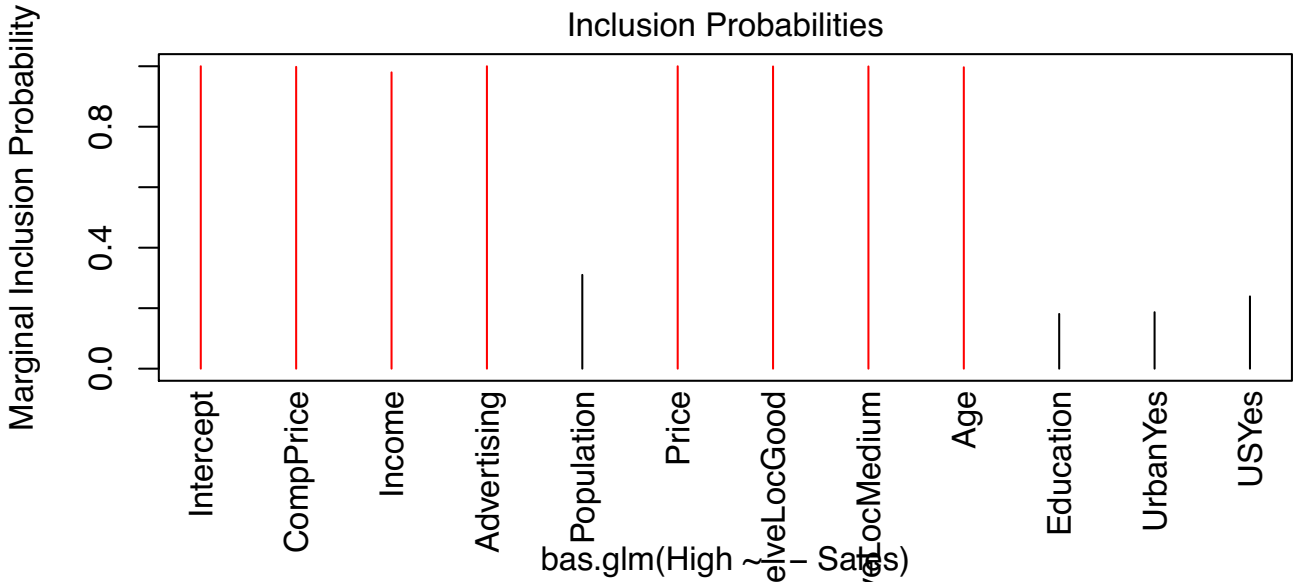
```
(tab[1,1] + tab[2,2])/sum(tab)

## [1] 0.88
```

GLMs and Bayesian Variable Selection

```
set.seed(42); library(BAS)
bas.carseats = bas.glm(High ~ . - Sales, data=Carseats,
                       subset=train, family=binomial(),
                       method="MCMC",  n.models=10000,
                       betaprior=bic.prior(n=200))
yhat = predict(bas.carseats, newdata=Carseats[-train,])
tab = table(ifelse(yhat$fit > .5, 1, 0), Carseats.test$High
CE.tab["BMA", 1]= (tab[1,1] + tab[2,2])/sum(tab)

CE.tab["BMA",]

## [1] 0.925
```

Inclusion Probabilities

```
plot(bas.carseats, which=4)
```



LASSO Variable Selection

```
set.seed(42); suppressMessages(library(glmnet))
glmnet.carseats = cv.glmnet(
  x=model.matrix(High ~ . - Sales, data=Carseats[train,]),
  y=(as.numeric(Carseats$High[train]) - 1), family="binomia
yhat = predict(glmnet.carseats,
                newx=model.matrix(High ~ . - Sales, data=Car
tab = table(ifelse(yhat > .0, 1, 0), Carseats.test$High)
CE.tab["LASSO", 1]= (tab[1,1] + tab[2,2])/sum(tab)

CE.tab["LASSO",]

## [1] 0.9
```

Summary

- Trees are simple to understand, but have high variability

Summary

- Trees are simple to understand, but have high variability
- Bagging (Averaging) reduces variability

Summary

- Trees are simple to understand, but have high variability
- Bagging (Averaging) reduces variability
- Random Forests adds additional constraints to average trees that are different (reduced correlation leads to reduction in variance).

Summary

- Trees are simple to understand, but have high variability
- Bagging (Averaging) reduces variability
- Random Forests adds additional constraints to average trees that are different (reduced correlation leads to reduction in variance).
- Boosting builds a mean function that uses multiple trees where the growth takes into account the previous trees. Smaller trees can be used (larger trees can overfit)

Summary

- Trees are simple to understand, but have high variability
- Bagging (Averaging) reduces variability
- Random Forests adds additional constraints to average trees that are different (reduced correlation leads to reduction in variance).
- Boosting builds a mean function that uses multiple trees where the growth takes into account the previous trees. Smaller trees can be used (larger trees can overfit)
- BART is similar to Boosting in that the mean function is a sum of trees, but uses a Bayesian approach to control complexity. Trees can be of different sizes and the number of trees can be large without overfitting

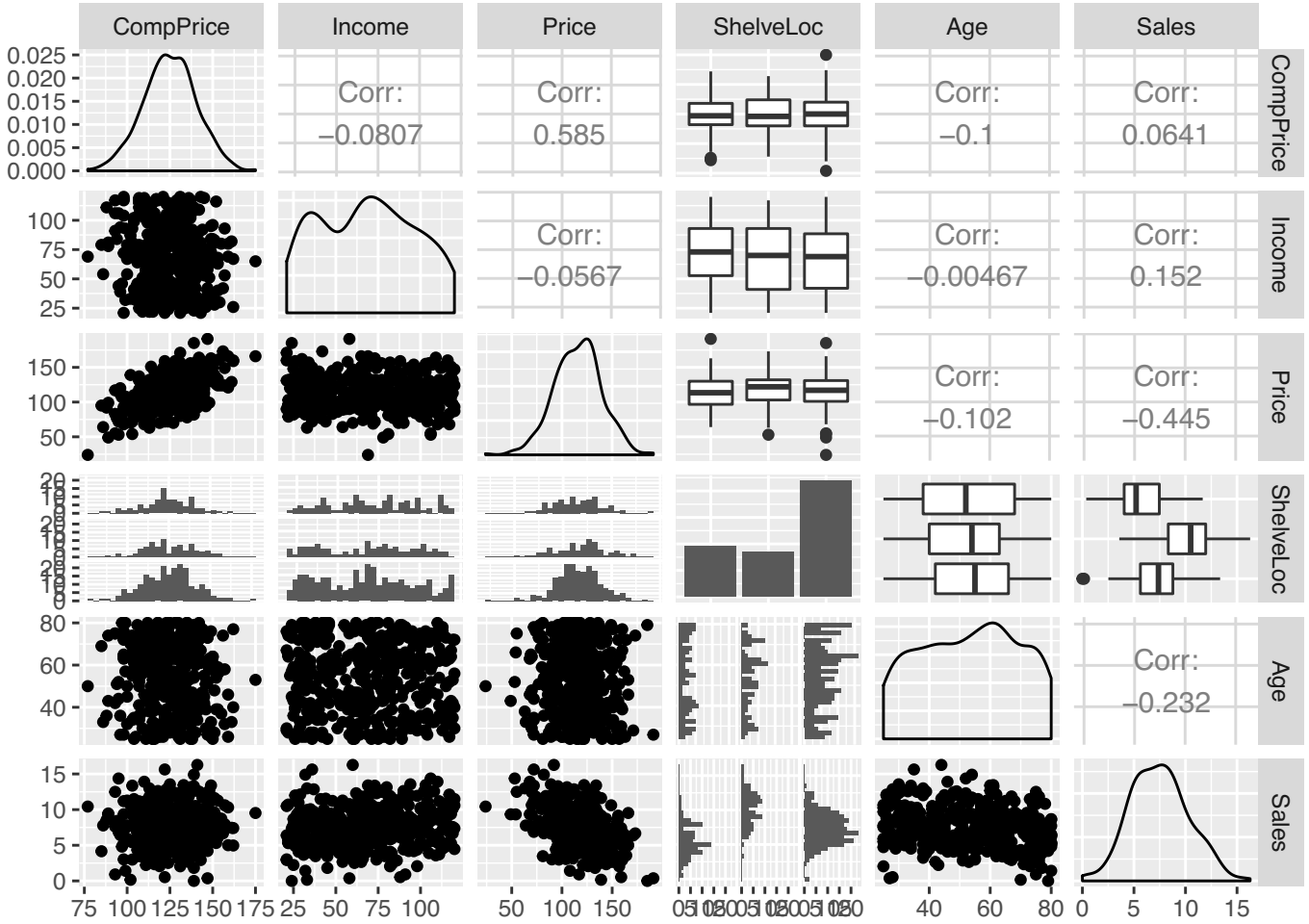
Summary

- Trees are simple to understand, but have high variability
- Bagging (Averaging) reduces variability
- Random Forests adds additional constraints to average trees that are different (reduced correlation leads to reduction in variance).
- Boosting builds a mean function that uses multiple trees where the growth takes into account the previous trees. Smaller trees can be used (larger trees can overfit)
- BART is similar to Boosting in that the mean function is a sum of trees, but uses a Bayesian approach to control complexity. Trees can be of different sizes and the number of trees can be large without overfitting
- Plots may suggest that a (generalized) linear model may be appropriate!

Summary

- Trees are simple to understand, but have high variability
- Bagging (Averaging) reduces variability
- Random Forests adds additional constraints to average trees that are different (reduced correlation leads to reduction in variance).
- Boosting builds a mean function that uses multiple trees where the growth takes into account the previous trees. Smaller trees can be used (larger trees can overfit)
- BART is similar to Boosting in that the mean function is a sum of trees, but uses a Bayesian approach to control complexity. Trees can be of different sizes and the number of trees can be large without overfitting
- Plots may suggest that a (generalized) linear model may be appropriate! BMA and lasso are better here
- Interpretability? For which methods can you explain how changes in a predictor change the response?

Data


CompPrice
Income
Price
ShelveLoc
Age
Sales
Corr: −0.0807
Corr: 0.585
Corr: −0.1
Corr: 0.0641
Corr: −0.0567
Corr: −0.00467
Corr: 0.152
Corr: −0.102
Corr: −0.445
Corr: −0.232

LECTURE 11
The boosting hypothesis and Adaboost

Voting algorithms or algorithms where the final classification or regression function is a weighted combination of "simpler" or "weaker" classifiers have been used extensively in a variety of applications.

We will study two examples of voting algorithms in greater depth: Bootstrap AGGregatING (BAGGING) and boosting.

11.1. Boosting

Boosting algorithms especially AdaBoost (adaptive boosting) have had a significant impact on a variety of practical algorithms and also have been the focus of theoretical investigation for a variety of fields. The formal term boosting and the first boosting algorithm came out of the field of computational complexity in theoretical Computer Science. In particular, learning as formulated by boosting came from the concept of Probably Approximatley Correct (PAC) learning.

11.2. PAC learning

The idea of Probably Approximatley Correct (PAC) learning was formulated in 1984 by Leslie Valiant as an attempt to characterize what is learnable. Let $\mathcal{X}$ be a set. This set contains encodings of all objects of interest in the learning problem. The goal of the learning algorithm is to infer some unknown subset of $\mathcal{X}$, called a concept, from a known class of concepts, $\mathcal{C}$. Unlike the previous statistical formulations of the learning problem, the issue of representation arises in this formulation due to computational issues.

- Concept classes A representation class over $\mathcal{X}$ is a pair $(\sigma, \mathcal{C})$, where $\mathcal{C} \subseteq \{0,1\}^*$ and $\sigma : \mathcal{C} \to 2^{\mathcal{X}}$. For $c \in \mathcal{C}$, $\sigma(c)$ is a concept over $\mathcal{X}$ and the image space $\sigma(\mathcal{C})$ is the concept class represented by $(\sigma, \mathcal{C})$. For $c \in \mathcal{C}$ the positive examples are $\text{pos}(c) = \sigma(c)$ and the negative examples are $\text{neg}(c) = \mathcal{X} - \sigma(c)$. The notations $c(x) = 1$ is equivalent to $x \in \text{pos}(c)$ and $c(x) = 0$ is equivalent to $x \in \text{neg}(c)$. We assume that domain points $x \in \mathcal{X}$ and representations $c \in \mathcal{C}$ are efficiently encoded with by codings of length $|x|$ and $|c|$ respectively.

- Parameterized representation We will study representation classes parameterized by an index n resulting in the domain $\mathcal{X} = \cup_{n\geq 1}\mathcal{X}_n$ and representation class $\mathcal{C} = \cup_{n\geq 1}\mathcal{C}_n$. The index n serves as a measure of the complexity of concepts in $\mathcal{C}$. For example, $\mathcal{X}$ may be the set $\{0,1\}^n$ and $\mathcal{C}$ the set of all Boolean formulae over n variables.
- Efficient evaluation of representations If $\mathcal{C}$ is a representation class over $\mathcal{X}$, then $\mathcal{C}$ is polynomially evaluatable if there is a (probabilistic) polynomial-time evaluation algorithm $\mathcal{A}$ that given a representation $c \in \mathcal{C}$ and domain point $x \in \mathcal{X}$ outputs $c(x)$.
- Samples A labeled example from a domain $\mathcal{X}$ is a pair $< x, b >$ where $x \in \mathcal{X}$ and $b \in \{0,1\}$. A sample $S = (< x_1, b_1 >, ..., < x_m, b_m >)$ is a finite sequence of labeled examples. A labeled example of $c \in \mathcal{C}$ has the form $< x, c(x) >$. A representation h and an example $< x, b >$ agree if $h(x) = b$. A representation h and a sample S are consistent if h agrees with each example in S.
- Distributions on examplesA learning algorithm for a representation class $\mathcal{C}$ will receive examples from a single representation $c \in \mathcal{C}$ whichw e call the target representation. Examples of the target representation are generated probabilistically: D_c is a fixed but arbitrary distribution over $\text{pos}(c)$ and $\text{neg}(c)$. This is the target distributions. The learning algorithm will be give access to an oracle EX which returns in unit time an example of the target representation drawn according to the target distribution D_c.
- Measure of error Given a target representation $c \in \mathcal{C}$ and a target distribution D the error of a representation $h \in \mathcal{H}$ is

$$e_c(h) = D(h(x) \neq c(x)).$$

In the above formulation $\mathcal{C}$ is the target class. In the above formulation $\mathcal{H}$ is the hypothesis class. The algorithm $\mathcal{A}$ is a learning algorithm for $\mathcal{C}$ and the output $h_{\mathcal{A}} \in \mathcal{H}$ is the hypothesis of $\mathcal{A}$.

We can now define learnability:

Definition (Strong learning)**.** *Let $\mathcal{C}$ and $\mathcal{H}$ be representation classes over $\mathcal{X}$ that are polynomially evaluatable. Then $\mathcal{C}$ is polynomially learnable by $\mathcal{H}$ if there is a (probabilistic) algorithm $\mathcal{A}$ with access to EX, taking inputs ε, δ with the property that for any target representation $c \in \mathcal{C}$, for any target distribution D, and for any input values $0 < \varepsilon, \delta < 1$, algorithm $\mathcal{A}$ halts in time polynomial in $\frac{1}{\varepsilon}, \frac{1}{\delta}, |c|, n$ and outputs a representation $h_{\mathcal{A}} \in \mathcal{H}$ that with probability greater than $1 - \delta$ satsifies $e_c(h) < \varepsilon$.*

The parameter ε is the accuracy parameter and the parameter δ is the confidence parameter. These two parameters characterize the name Probably (δ) Approximatley (ε) Correct ($e_c(h)$). The above definition is sometimes called distribution free learning since the property holds ofver an target represenation and target distribution.

Considerable research in PAC learning has focused on which representation classes $\mathcal{C}$ are polynomially learnable.

So far we have defined learning as approximating aribirarily close the target concept. Another model of learning called weak learning considers the case where the learning algorithm is required to perform slightly better than chance.

Definition (Weak learning). *Let $\mathcal{C}$ and $\mathcal{H}$ be representation classes over $\mathcal{X}$ that are polynomially evaluatable. Then $\mathcal{C}$ is polynomially weak learnable by $\mathcal{H}$ if there is a (probabilistic) algorithm $\mathcal{A}$ with access to EX, taking inputs ε, δ with the property that for any target representation $c \in \mathcal{C}$, for any target distribution D, and for any input values $0 < \varepsilon, \delta < 1$, algorithm $\mathcal{A}$ halts in time polynomial in $\frac{1}{\varepsilon}, \frac{1}{\delta}, |c|, n$ and outputs a representation $h_{\mathcal{A}} \in \mathcal{H}$ that with probability greater than $1 - \delta$ satsifies $e_c(h) < \frac{1}{2} - \frac{1}{p(|c|)}$, where p is a polynomial.*

Definition (Sample complexity). *Given a learning algorithm $\mathcal{A}$ for a representation class $\mathcal{C}$. The number of calls $s_{\mathcal{A}}(\varepsilon, \delta)$ to the oracle EX made by $\mathcal{A}$ on inputs ε, δ for the worst-case measure over all target representations $c \in \mathcal{C}$ and target distributions D is the sample complexity of the algorithm $\mathcal{A}$.*

We now state some Boolean classes whose learnability we will state as positive or negative examples of learnability.

- The class M_n consist of monomials over the Boolean variables $x_i, ..., x_n$.
- For a constant k, the class $kCNF_n$ (conjunctive normal forms) consists of Boolean formulae of the form $C_1 \wedge \cdots \wedge C_l$ where each C_i is a disjunction of at most k monomials over the Boolean variables $x_i, ..., x_n$.
- For a constant k, the class $kDNF_n$ (disjunctive normal forms) consists of Boolean formulae of the form $T_1 \vee \cdots \vee T_l$ where each T_i is a conjunction of at most k monomials over the Boolean variables $x_i, ..., x_n$.
- Boolean threshold functions $I(\sum_{i=1}^{n} w_i x_i > t)$ where $w_i \in \{0, 1\}$ and I is the indicator function.

Definition (Empirical risk minimization, ERM). *A consistent algorithm $\mathcal{A}$ is one that outputs hypotheses h that are consistent with the sample S and the range over possible hypotheses for $\mathcal{A}$ is $h \in \mathcal{C}$.*

The above algorithm is ERM in the case of zero error with the target concept in the hypothesis space.

Theorem. *If the hypothesis class is finite then $\mathcal{C}$ is learnable by the consistent algorithm $\mathcal{A}$.*

Theorem. *Boolean threshold functions are not learnable.*

Theorem. *$\{f \vee g : f \in kCNF, g \in kDNF\}$ is learnable.*

Theorem. *$\{f \wedge g : f \in kDNF, g \in kCNF\}$ is learnable.*

Theorem. *Let $\mathcal{C}$ be a concept class with finite VC dimension $VC(\mathcal{C}) = d < \infty$. Then $\mathcal{C}$ is learnable by the consistent algorithm $\mathcal{A}$.*

11.3. The hypothesis boosting problem

An important question both theoretically and practically in the late 1980's was whether strong learnablity and weak learnability were equivalent. This was the hypothesis boosting problem:

Conjecture. *A concept class $\mathcal{C}$ is weakly learnable if and only if it is strongly learnable.*

The above conjecture was proven true in 1990 by Robert Schapire.

Theorem. *A concept class $\mathcal{C}$ is weakly learnable if and only if it is strongly learnable.*

The proof of the above theorem was based upon a particular algorithm. The following algorithm takes as input a weaklearner, an error paramter ε, a confidence parameter δ, an oracle EX, and outputs a strong learner. At each iteration of the algorithm a weaklearner with error rate ε gets boosted so that its error rate decreases to $3\varepsilon^2 - 2\varepsilon^3$.

Algorithm 1: Learn$(\varepsilon, \delta, EX)$

input : error parameter ε, confidence parameter δ, examples oracle EX

return: h that is ε close to the target concept c with probability $\geq 1 - \delta$

if $\varepsilon \geq 1/2 - 1/p(n, s)$ **then** return WeakLearn(δ, EX);
$\alpha \leftarrow g^{-1}(\varepsilon)$: where $g(x) = 3x^2 - 2x^3$;

$EX_1 \leftarrow EX$;
$h_1 \leftarrow$ Learn$(\alpha, \delta/5, EX_1)$;
$\tau_1 \leftarrow \varepsilon/3$;
let $\hat{a}_1$ be an estimate of $a_1 = \Pr_{x\sim D}[h_1(x) \neq c(x)]$
  choose a sample sufficiently large that
  $|a_1 - \hat{a}_1| \leq \tau_1$ with probability $\geq 1 - \delta/5$
if $\hat{a}_1 \leq \varepsilon - \tau_1$ **then** return h_1;

defun $EX_2()$
  {flip coin
  if *heads* **then** return first $x : h_1(x) = c(x)$;
  else tails return first $x : h_1(x) \neq c(x)$}
$h_2 \leftarrow$ Learn$(\alpha, \delta/5, EX_2)$;
$\tau_2 \leftarrow (1 - 2\alpha)\varepsilon/9$;
let $\hat{e}$ be an estimate of $e = \Pr_{x\sim D}[h_2(x) \neq c(x)]$
  choose a sample sufficiently large that
  $|e - \hat{e}| \leq \tau_2$ with probability $\geq 1 - \delta/5$
if $\hat{e} \leq \varepsilon - \tau_2$ **then** return h_2;

defun $EX_3()$
  {return first $x : h_1(x) \neq h_2(x)$ };
$h_3 \leftarrow$ Learn$(\alpha, \delta/5, EX_3)$;

defun $h(x)$
  { $b_1 \leftarrow h_1(x)$, $b_2 \leftarrow h_2(x)$
  if $b_1 = b_2$ **then** return b_1
  else return $h_3(x)$}
return h

The above algorithm can be summarized as follows:

(1) Learn an initial classifier h_1 on the first N training points

(2) Learn h_2 on a new sample of N points, half of which are misclassifief by h_1
(3) Learn h_3 on N points for which h_1 and h_2 disagree
(4) The boosted classifier $h = \text{Majority vote}(h_1, h_2, h_3)$.

The basic result is that if the individual classifiers h_1, h_2, and h_3 have error ε the boosted classifier has error $2\varepsilon^2 - 3\varepsilon^3$.

To prove the theorem one needs to show that the algorithm is correct in the sense following sense.

Theorem. *For $0 < \varepsilon < 1/2$ and for $0 < \delta <\leq 1$, the hypothesis returned by calling $Learn(\varepsilon, \delta, EX)$ is ε close to the target concept with probability at least $1 - \delta$.*

We first define a few quantities

$$\begin{aligned} p_i &= \Pr_{x\sim D}[h_i(x) = c(x)] \\ q &= \Pr_{x\sim D}[h_1(x) \neq h_2(x)] \\ w &= \Pr_{x\sim D}[h_2(x) \neq h_1(x) = c(x)] \\ v &= \Pr_{x\sim D}[h_1(x) = h_2(x) = c(x)] \\ y &= \Pr_{x\sim D}[h_1(x) \neq h_2(x) = c(x)] \\ z &= \Pr_{x\sim D}[h_1(x) \neq h_2(x) \neq c(x)]. \end{aligned}$$

Given the above quantities

$$w + v = \Pr_{x\sim D}[h_1(x) = c(x)] = 1 - a_1 \tag{11.1}$$

$$y + z = \Pr_{x\sim D}[h_1(x) \neq c(x)] = a_1. \tag{11.2}$$

We can explicitly express the chance that EX_i returns an instance x in terms of the above variable:

$$\begin{aligned} D_1(x) &= D(x) \\ D_2(x) &= \frac{D(x)}{2}\left(\frac{p_1(x)}{a_1} + \frac{1 - p_(x)}{1 - a_1}\right) \\ D_3(x) &= \frac{D(x)q(x)}{w + y}. \end{aligned} \tag{11.3}$$

From equation (11.3) we have

$$\begin{aligned} 1 - a_2 &= \sum_{x\in\mathcal{X}_n} D_2(x)(1 - p_2(x)) \\ &= \frac{1}{2a_1}\sum_{x\in\mathcal{X}_n} D(x)p_1(x)(1 - p_2(x)) + \frac{1}{2(1 - a_1)}\sum_{x\in\mathcal{X}_n} D(x)(1 - p_1(x))(1 - p_2(x)) \\ &= \frac{y}{2a_1} + \frac{z}{2(1 - a_1)}. \end{aligned}$$

Combining the above equation with equations (11.1) and (11.2) we can solve for w and z in terms of y, a_1, a_2.

$$\begin{aligned} w &= (2a_2 - 1)(1 - a_1) + \frac{y(1 - a_1)}{a_1} \\ z &= a_1 - y. \end{aligned}$$

We now control the quantity

$$\begin{aligned}
\Pr_{x\sim D}[h(x) \neq c(x)] &= \Pr_{x\sim D}[(h_1(x) = h_2(x)) \vee (h_1(x) \neq h_2(x) \wedge h_3(x) \neq c(x))] \\
&= z + \sum_{x\in\mathcal{X}_n} D(x)q(x)p_3(x) \\
&= z + \sum_{x\in\mathcal{X}_n} (w+y)D_3(x)p_3(x) \\
&= z + a_3(w+y) \\
&\leq z + \alpha(w+y) \\
&= \alpha(2a_2 - 1)(1-a_1) + a_1 + \frac{y(\alpha - a_1)}{a_1} \\
&\leq \alpha(2a_2 - 1)(1-a_1) + \alpha \\
&\leq \alpha(2\alpha - 1)(1-\alpha) + \alpha = 3\alpha^2 - 2\alpha^3 = \varepsilon,
\end{aligned}$$

the inequalities follow from the fact that $a_i \leq \alpha < 1/2$ and $y \leq a_1$. □

One also needs to show that the algorithm runs in polynomial time. The following lemma implies this. The proof of the lemma is beyond the scope of the lecture notes.

Lemma. *On a good run the expected execution time of* $Learn(\varepsilon, \delta/2, EX)$ *is polynomial in* $m, 1/\delta, 1/\epsilon$.

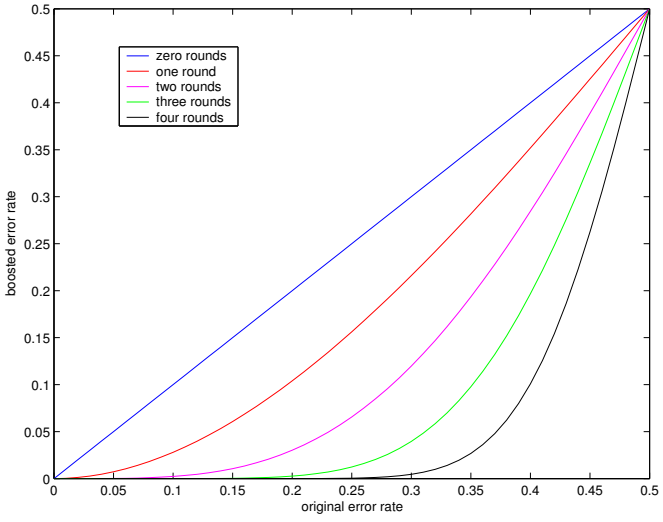


Figure 1. A plot of the boosted error rate as a function of the initial error for different numbers of boosting rounds.

11.4. ADAptive BOOSTing (AdaBoost)

We will call the above formulation of boosting the boost-by-majority algorithm. The formulation of boosting by majority by Schapire involved boosting by filtering since one weak learner served as a filter for the other. Another formulation of boost by majority was developed by Yoav Fruend also based upon filtering. Both

of these algorithms were later adjusted so that sampling weights could be instead of filtering. However, all of these algorithms had the problem that the strength $1/2 - \gamma$ of the weak learner had to be known a priori.

Freund and Schapire developed the following adaptive boosting algorithm, AdaBoost, to address these issues.

Algorithm 2: AdaBoost

input : samples $S = (x_i, y_i)_{i=1}^N$, weak learner, number of iterations T

return: $h(x) = \text{sign}\left[\sum_{i=1}^T \alpha_i h_i(x)\right]$

for *i=1* **to** N **do** $w_i^0 = 1/N$;

for *t=1* **to** T **do**
- $h_t \leftarrow$ Call WeakLearn with weights w^t;
- $\varepsilon_t = \sum_{j=1}^N w_j^t I_{\{y_j \neq h_t(x_j)\}}$;
- $\alpha_t = \log((1-\varepsilon_t)/\varepsilon_t)$;
- **for** *j=1* **to** N **do** $w_j^{t+1} = w_j^t \exp\left(-\alpha_t y_j h_t(x_j)\right)$;
- $Z_t = \sum_{j=1}^N w_j^{t+1}$;
- **for** *j=1* **to** N **do** $w_j^{t+1} = w_j^{t+1}/Z_t$;

For the above algorithm we can prove that the training error will decrease over boosting iterations. The advantage of the above algorithm is we don't need a uniform γ over all rounds. All we need is for each boosting round there exists a $\gamma_t > 0$.

Theorem. *Suppose WeakLearn when called by AdaBoost generates hypotheses with errors $\varepsilon_1, ..., \varepsilon_T$. Assume each $\varepsilon_i \leq 1/2$ and let $\gamma_i = 1/2 - \varepsilon_i$ then the following upper bound holds on the hypothesis h*

$$\frac{|j : h(x_j) \neq y_j|}{N} \leq \prod_{i=1}^T \sqrt{1 - 4\gamma_i^2} \leq \exp\left(-2\sum_{i=1}^T \gamma_i^2\right).$$

Proof.

If $y_i \neq h(x_i)$ then $y_i h(x_i) \leq 0$ and $e^{-y_i h(x_i)} \geq 1$. Therefore

$$\begin{aligned} \frac{|j : h(x_j) \neq y_j|}{N} &\leq \frac{1}{N}\sum_{i=1}^N e^{-y_i h(x_i)}, \\ &= \sum_{i=1}^N w_i^{T+1} \prod_{t=1}^T Z_t = \prod_{t=1}^T Z_t. \end{aligned}$$

In addition, since $\alpha_t = \log((1-\varepsilon_t)/\varepsilon_t)$ and $1 + x \leq e^x$

$$Z_t = 2\sqrt{\varepsilon_t(1-\varepsilon_t)} = \sqrt{1 - 4\gamma_t^2} \leq e^{-2\gamma_t}. \quad \square$$

11.5. A statistical interpretation of Adaboost

In this section we will reinterpret boosting as a greedy algorithm to fit an additive model. We first define our weak learners as a paramterized class of functions $h_\theta(x) = h(x;\theta)$ where $\theta \in \theta$. If we think of each weak learner as a basis function then the boosted hypothesis $h(x)$ can be thought of as a linear combination the weak learners

$$h(x) = \sum_{i=1}^{T} \alpha_i h_{\theta_i}(x),$$

where the $h_{\theta_i}(x)$ is the ith weak learner parameterized by θ_i. One approach to setting the parameters θ_i and weights α_i is called forward stagewise modelling. In this approach we sequentially add new basis functions or weak learners without adjusting the paramters and coefficients of the current solution. The following agorithm implements forward stagewise additive modeling.

Algorithm 3: Forward stagewise additive modeling

input : samples $S = (x_i, y_i)_{i=1}^N$, weak learner, number of iterations T, loss function L

return: $h(x) = \left[\sum_{i=1}^{T} \alpha_i h_{\theta_i}(x)\right]$

$h_0(x) = 0$;
for *i=1* **to** T **do**
  $(\alpha_t, \theta_t) = \arg\min_{\alpha \in \mathbb{R}^+, \theta \in \Theta} \sum_{i=1}^{N} L(y_i, h_{t-1}(x_i) + \alpha h_\theta(x))$;
  $h_t(x) = h_{t-1}(x) + \alpha_t h_{\theta_i}(x)$;

We will now show that the above algorithm with exponential loss

$$L(y, f(x)) = e^{-yf(x)}$$

is equivalent to AdaBoost.

At each iteration the following minimization is performed

$$\begin{aligned}
(\alpha_t, \theta_t) &= \arg \min_{\alpha \in \mathbb{R}^+, \theta \in \Theta} \sum_{i=1}^{N} \exp[-y_i(h_{t-1}(x_i) + \alpha h_\theta(x))], \\
(\alpha_t, \theta_t) &= \arg \min_{\alpha \in \mathbb{R}^+, \theta \in \Theta} \sum_{i=1}^{N} \exp[-y_i h_{t-1}(x_i)] \exp[-y_i \alpha h_\theta(x)], \\
(\alpha_t, \theta_t) &= \arg \min_{\alpha \in \mathbb{R}^+, \theta \in \Theta} \sum_{i=1}^{N} w_i^t \exp[-y_i \alpha h_\theta(x)], \qquad (11.4)
\end{aligned}$$

where $w_i^t = \exp[-y_i h_{t-1}(x_i)]$ does not effect the optimization functional. For any $\alpha > 0$ the objective function in equation (11.4) can be rewritten as

$$\begin{aligned}
\theta_t &= \arg\min_{\theta\in\Theta} \left[e^{-\alpha} \sum_{y_i = h_\theta(x_i)} w_i^t + e^{\alpha} \sum_{y_i \neq h_\theta(x_i)} w_i^t \right], \\
\theta_t &= \arg\min_{\theta\in\Theta} \left[(e^{-\alpha} - e^{\alpha}) \sum_{i=1}^N w_i^t I_{\{y_i \neq h_\theta(x_i)\}} + e^{\alpha} \sum_{i=1}^N w_i^t \right], \\
\theta_t &= \arg\min_{\theta\in\Theta} \sum_{i=1}^N w_i^t I_{\{y_i \neq h_\theta(x_i)\}}.
\end{aligned}$$

Therefore the weak learner that minimizes equation (11.4) will minimize the weighted error rate which if we plug back into equation (11.4) we can solve for α_t which is

$$\alpha_t = \frac{1}{2} \log \frac{1 - \varepsilon_t}{\varepsilon_t},$$

where

$$\varepsilon_t = \sum_{i=1}^N w_i^t I_{\{y_i \neq h_t(x_i)\}}.$$

The last thing to show is the updating of the linear model

$$h_t(x) = h_{t-1}(x) + \alpha_t h_t(x),$$

is equivalent to the reweighting used in AdaBoost. Due to the exponential loss function and the additive updating at each iteration the above sum can be rewritten as

$$w_i^{t+1} = w_i^t e^{-\alpha y_i h_t(x_i)}.$$

So AdaBoost can be interpreted as an algorithm that minimizes the exponential loss criterion via forward stagewise additive modeling.

We now give some motivation for why the exponential loss is a reasonable loss function in the classification problem. The first argument is that like the hinge loss for SVM classification the exponential loss serves as an aupper bound on the missclassification loss (see figure 2).

Another simple motivation for using the exponential loss is the minimizer of the expected loss with respect to some function class $\mathcal{H}$

$$f^*(x) = \arg\min_{f\in\mathcal{H}} \mathbb{E}_{Y|x} \left[e^{-Yf(x)} \right] = \frac{1}{2} \log \frac{\Pr(Y=1|x)}{\Pr(Y=-1|x)},$$

estimates one-half the log-odds ratio

$$\Pr(Y=1|x) = \frac{1}{1 + e^{-2f^*(x)}}.$$

11.6. A margin interpretation of Adaboost

We developed a geometric formulation of support vector machines in the seperable case via maximizing the margin. We will formulate AdaBoost as a margin maximization problem.

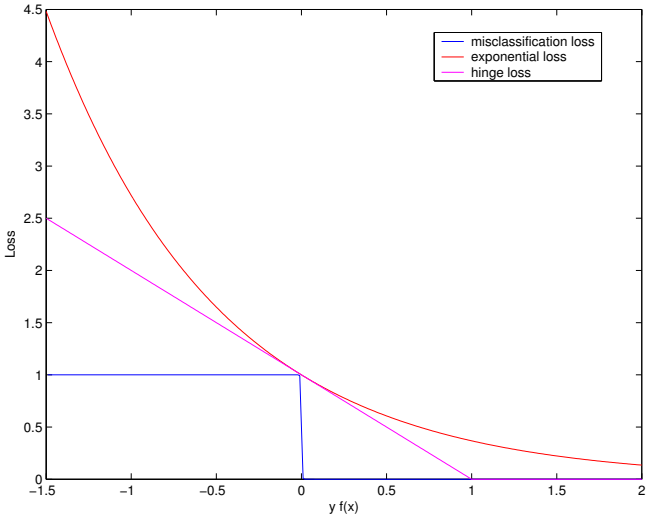


Figure 2. A comparison of loss functions for classification.

Recall that for the linear seperable SVM with points in $\mathbb{R}^d$ given a dataset S the following optimization problem characterizes the maximal margin classifier

$$\hat{w} = \arg \max_{w \in \mathbb{R}^d} \min_{x_i \in S} \frac{y_i \langle w, x_i \rangle}{||w||_{L_2}}.$$

In the case of AdaBoost we can construct a coordinate space with as many dimensions as weak classifiers, T, $u \in \mathbb{R}^T$, where the elements of $\{u_1, ..., u_T\}$ correspond to the outputs of the weak classifiers $\{u_1 = f_1(x), ..., u_T = f_T(x)\}$. We can show that AdaBoost is an iterative way to solve the following mini-max problem

$$\hat{w} = \arg \max_{w \in \mathbb{R}^T} \min_{u_i \in S} \frac{y_i \langle w, u_i \rangle}{||w||_{L_1}},$$

where $u_i = \{f_1(x_i), ..., f_T(x_i)\}$ and the final classifier has the form

$$h_T(x) = \sum_{i=1}^{T} \hat{w}_i^T f_i(x).$$

This follows immediately from the forward additive stagewise modeling interpretation since under seperability the addition at each iteration of a weak classifier to the linear expansion will result in a boosted hypothesis h_t that as a function of t will be nondecreasing in $y_i h_t(x_i)$ $\forall i$ with the L_1 norm on w^t constrained, $||w||_{L_1} = 1$, following from the fact that the weights at each iteration must satisfy the distribution requirement.

An interesting geometry arises from the two different norms on the weights w in the two different optimization problems. The main idea is that we want to relate the norm on w to properties of norms on points in either $\mathbb{R}^d$ in the SVM case or $\mathbb{R}^T$ in the boosting case. By Hölder's inequality for the dual norms $||x||_{L_q}$ and $||w||_{L_p}$ with $\frac{1}{p} + \frac{1}{q} = 1$ and $p, q \in [1, \infty]$ the following holds

$$|\langle x, w \rangle| \leq ||x||_{L_q} ||w||_{L_p}.$$

The above inequality implies that minimizing the L_2 norm on w is equivalent to maximizing the L_2 distance between the hyperplane and the data. Similarly, minimizing the L_1 norm on w is equivalent to maximizing the L_∞ norm between the hyperplane and the data.