Outline

Bagging

Random Forests

Boosting

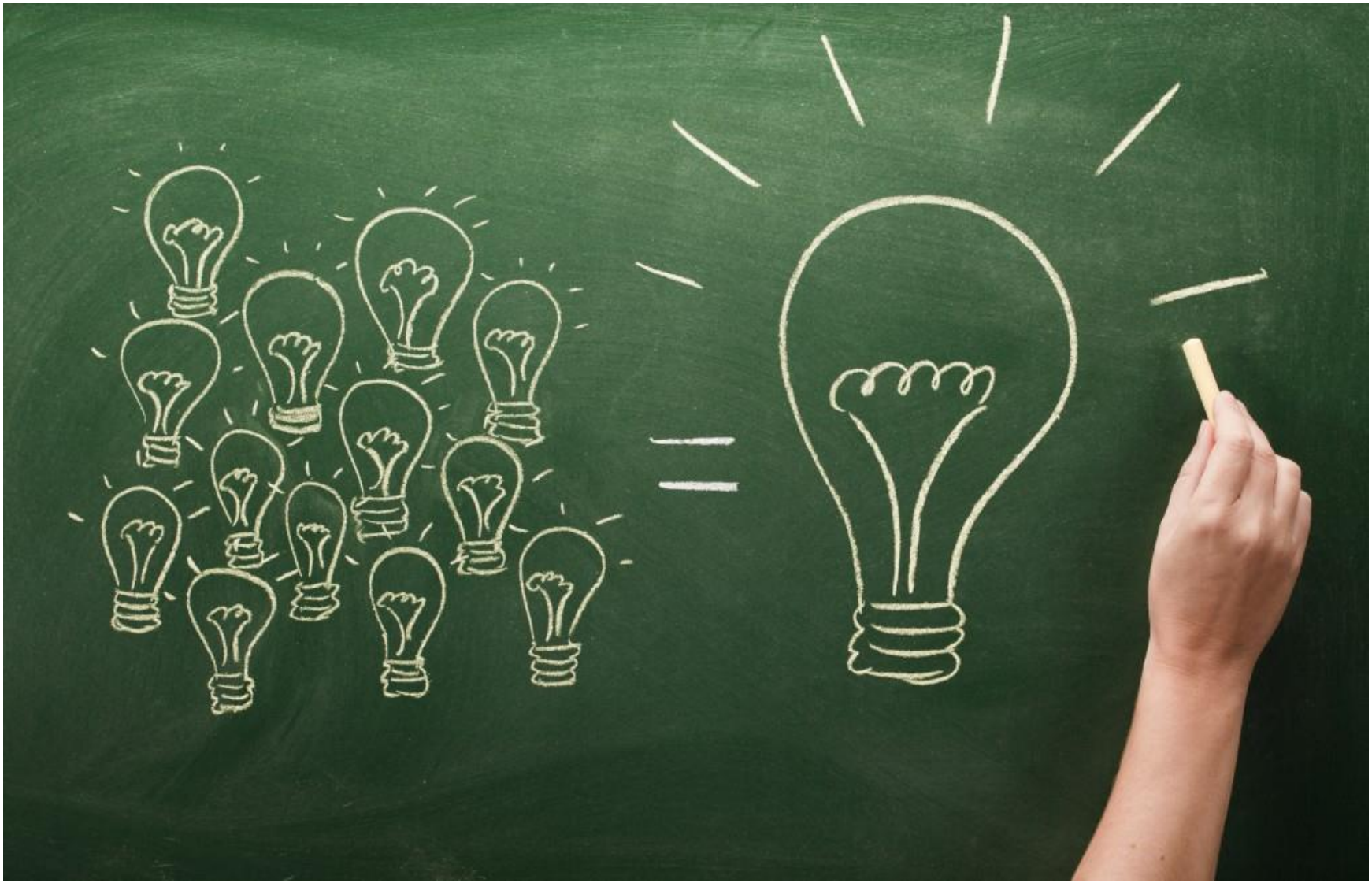
Outline

Bagging

Random Forests

Boosting

Power of the crowds



http://www.scaasymposium.org/portfolio/part-v-the-power-of-innovation-and-the-market/

Ensemble methods

- A single decision tree does not perform well
- But, it is super fast
- What if we learn multiple trees?

We need to make sure they do not all just learn the same

Bagging

If we split the data in random different ways, decision trees give different results, **high variance.**

Bagging: Bootstrap **agg**regat**ing** is a method that result in low variance.

If we had multiple realizations of the data (or multiple samples) we could calculate the predictions multiple times and take the average of the fact that averaging multiple onerous estimations produce less uncertain results

Bagging

Say for each sample *b*, we calculate $f^b(x)$, then:

$$\hat{f}_{avg}(x) = \frac{1}{B}\sum_{b=1}^{B}\hat{f}^b(x)$$

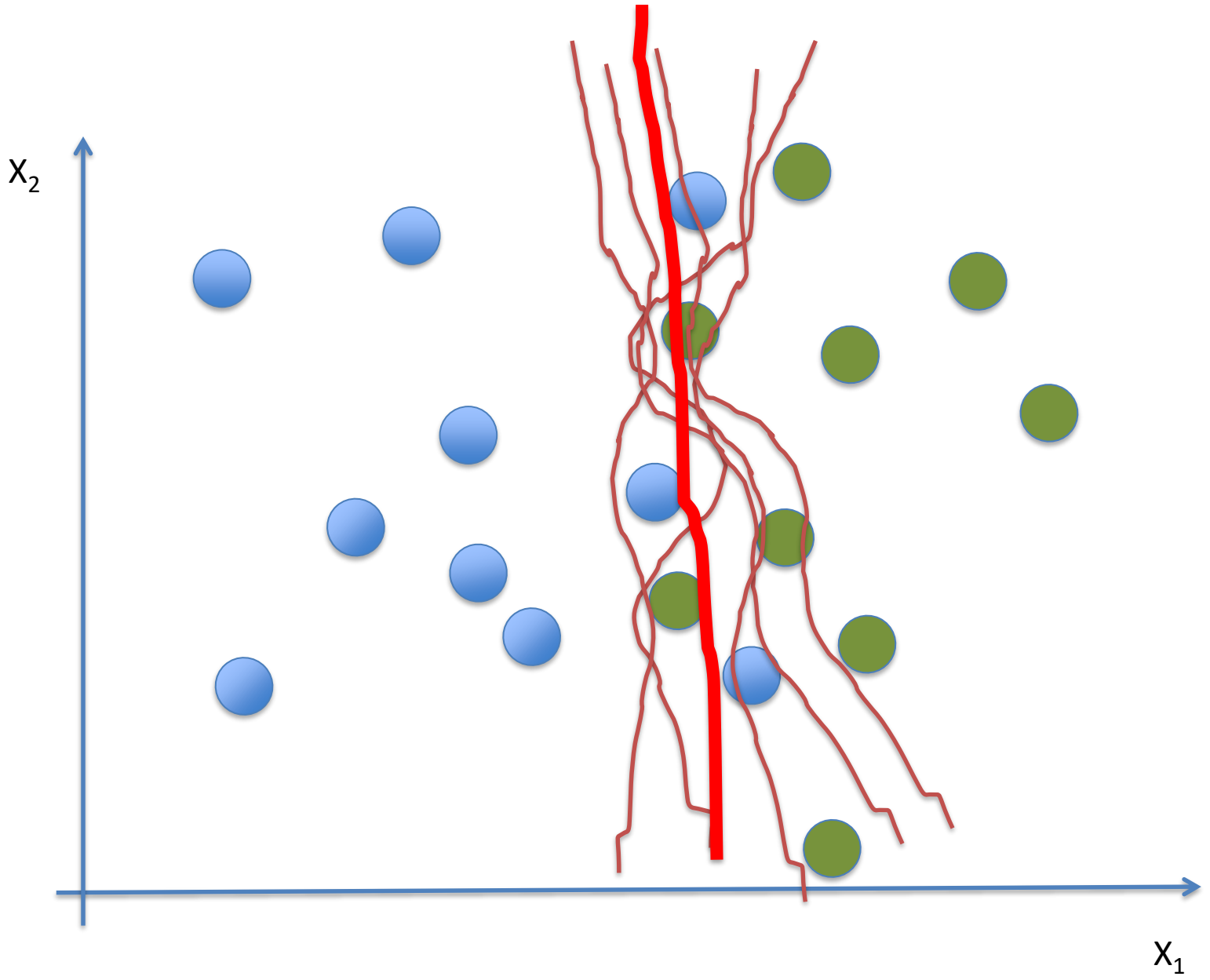
How?

Bootstrap

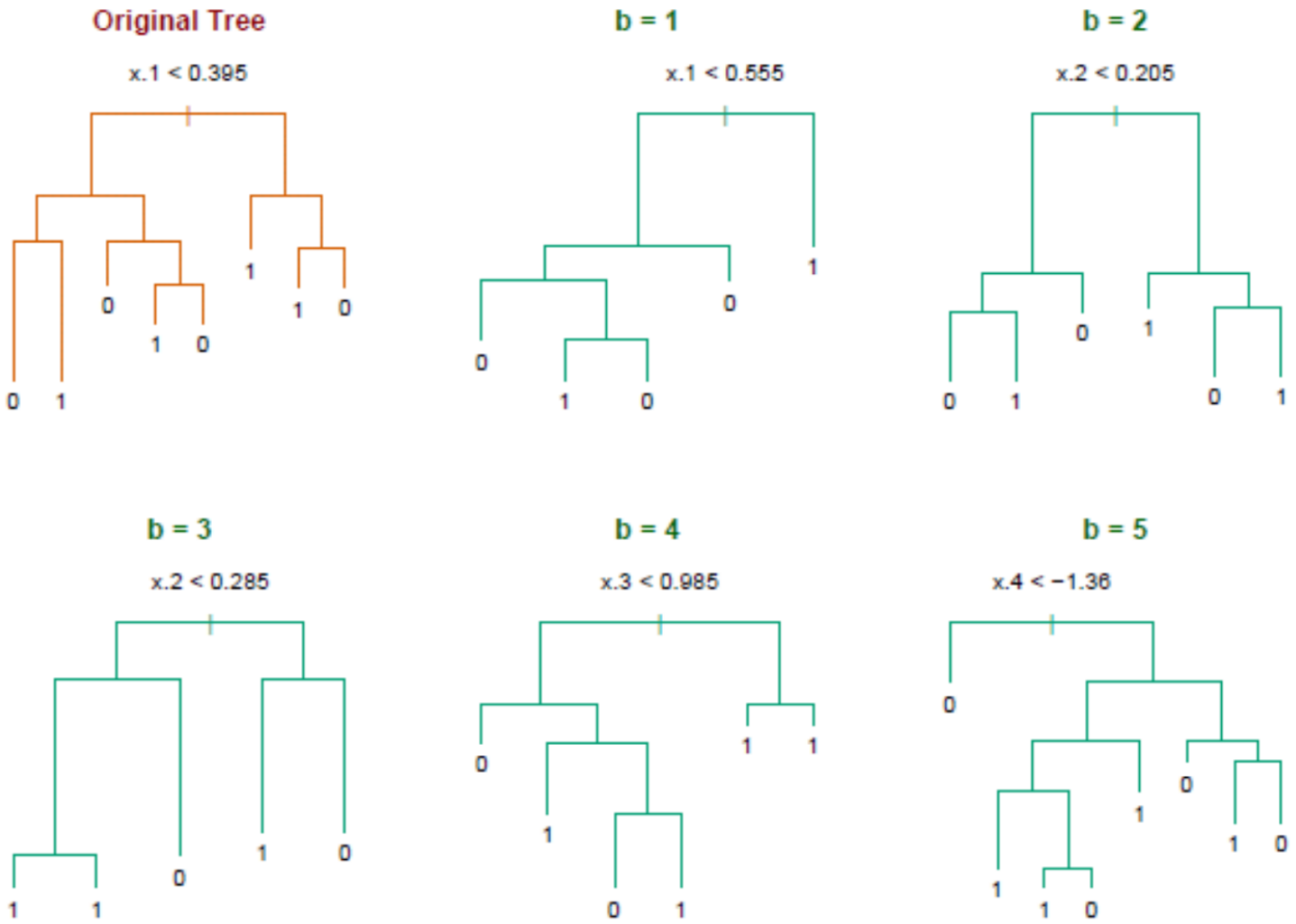
Construct B (hundreds) of trees (no pruning)

Learn a classifier for each bootstrap sample and average them

Very effective


X_2
X_1

Bagging decision trees



Hastie et al.,"The Elements of Statistical Learning: Data Mining, Inference, and Prediction", Springer (2009)

Out-of-Bag Error Estimation

- No cross validation?
- Remember, in bootstrapping we sample with replacement, and therefore **not all observations are used for each bootstrap sample**. On average 1/3 of them are not used!
- We call them out-of-bag samples (OOB)
- We can predict the response for the *i-th* observation using each of the trees in which that observation was OOB and do this for *n* observations
- Calculate overall OOB MSE or classification error

Bagging

- Reduces overfitting (variance)
- Normally uses one type of classifier
- Decision trees are popular
- Easy to parallelize

Bagging - Example

```
rm(list = ls())

set.seed(10)
y<-c(1:1000)
x1<-c(1:1000)*runif(1000,min=0,max=2)
x2<-c(1:1000)*runif(1000,min=0,max=2)
x3<-c(1:1000)*runif(1000,min=0,max=2)

all_data<-data.frame(y,x1,x2,x3)
index <- sample(nrow(all_data),size=floor((nrow(all_data)/4)*3))
training<- all_data[index,]
testing<- all_data[-index,]

################## Simple lm model ##############
lm_fit<-lm(y~x1+x2+x3,data=training)
predictions<-predict(lm_fit,newdata=testing)
error<-sqrt((sum((testing$y-predictions)^2))/nrow(testing))

error
```

Bagging - Example

```
################## Bagging ##############

predictions_bagging<-NULL

# We will build 1000 lm models each based on 66% of data recor
for (n in 1:1000){
  sub_index=sample(nrow(training),round(nrow(training)*.66))
  sub_model=lm(y~x1+x2+x3,data=training[sub_index,]) # build m
  predictions<-predict(sub_model,testing)# predict
  predictions_bagging=cbind(predictions_bagging,predictions) #
}

predictions_bagging_final=apply(predictions_bagging,1,mean) #
error2<-sqrt((sum((testing$y-predictions)^2))/nrow(testing)) #
error2
```

Bagging – Example II

```
rm(list = ls())
library(caret)
library(rpart)
set.seed(2018)

data(GermanCredit)

Index=sample(nrow(GermanCredit),nrow(GermanCredit)*.8)
Train=GermanCredit[Index,]
Test=GermanCredit[-Index,]

Model<-rpart(Class~.,data=Train, method='class')
Predictions=predict(Model,Test,type="class")
table(Predictions,Test$Class)
```

Bagging – Example II

```
> table(Predictions,Test$Class)

Predictions Bad Good
       Bad   24   20
       Good  28  128
```

Bagging – Example II

```
Pred_matrix<-NULL

for (n in 1:1000){ # Creat 1000 Tress
  Train=GermanCredit[Index,]
  Index_row=sample(nrow(Train),round(nrow(Train)*0.6)) # Each t
  Train<-Train[Index_row,]

  Model<-rpart(Class~.,data=Train, method='class')
  Predictions=predict(Model,Test)
  Pred_matrix=cbind(Pred_matrix,Predictions[,2])# add predictic
}

Pred_avg=apply(Pred_matrix,1,mean) # Average probobilities

Pred_avg_factor<-Pred_avg
Pred_avg_factor[Pred_avg<0.5]<-'Bad' # Determine good ans bad c
Pred_avg_factor[Pred_avg>0.5]<-'Good'

table(Pred_avg_factor,Test$Class)
```

Bagging – Example II

```
> table(Pred_avg_factor,Test$Class)

Pred_avg_factor Bad Good
           Bad   19   13
           Good  33  135
>
```

Bagging - issues

Suppose that there is one very strong predictor in the data set, along with a number of other moderately strong predictors.

Then all bagged trees will select the strong predictor at the top of the tree and therefore all trees will look similar.

How do we avoid this?

Bagging - issues

We can penalize the splitting (like in pruning) with a penalty term that depends on the number of times a predictor is selected at a given length

We can restrict how many times a predictor can be used

We only allow a certain number of predictors

Bagging - issues

Remember we want i.i.d such as the bias to be the same and variance to be less?

Other ideas?

What if we consider only a subset of the predictors at each split?

We will still get correlated trees unless ….

we **randomly** select the subset !


Random Forests

Outline

Bagging

Random Forests

Boosting

Random Forests

As in bagging, we build a number of decision trees on bootstrapped training samples each time a split in a tree is considered, a random sample of *m* predictors is chosen as split candidates from the full set of p predictors.

Note that if $m = p$, then this is bagging.

Random Forests

Random forests are popular. Leo Breiman's and Adele Cutler maintains a random forest website where the software is freely available, and of course it is included in every ML/STAT package

http://www.stat.berkeley.edu/~breiman/RandomForests/

Random Forests Algorithm

For b = 1 to B:

(a) Draw a bootstrap sample Z^* of size *N* from the training data.

(b) Grow a random-forest tree to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until the minimum node size $\boldsymbol{n_{min}}$ is reached.

i. Select ***m*** variables at random from the *p* variables.

ii. Pick the best variable/split-point among the *m*.

iii. Split the node into two daughter nodes.

Output the ensemble of trees.

To make a prediction at a new point *x* we do:

For regression: average the results

For classification: majority vote

Random Forests Tuning

The inventors make the following recommendations:

- For classification, the default value for *m* is $\sqrt{p}$ and the minimum node size is one.
- For regression, the default value for m is $p/3$ and the minimum node size is five.

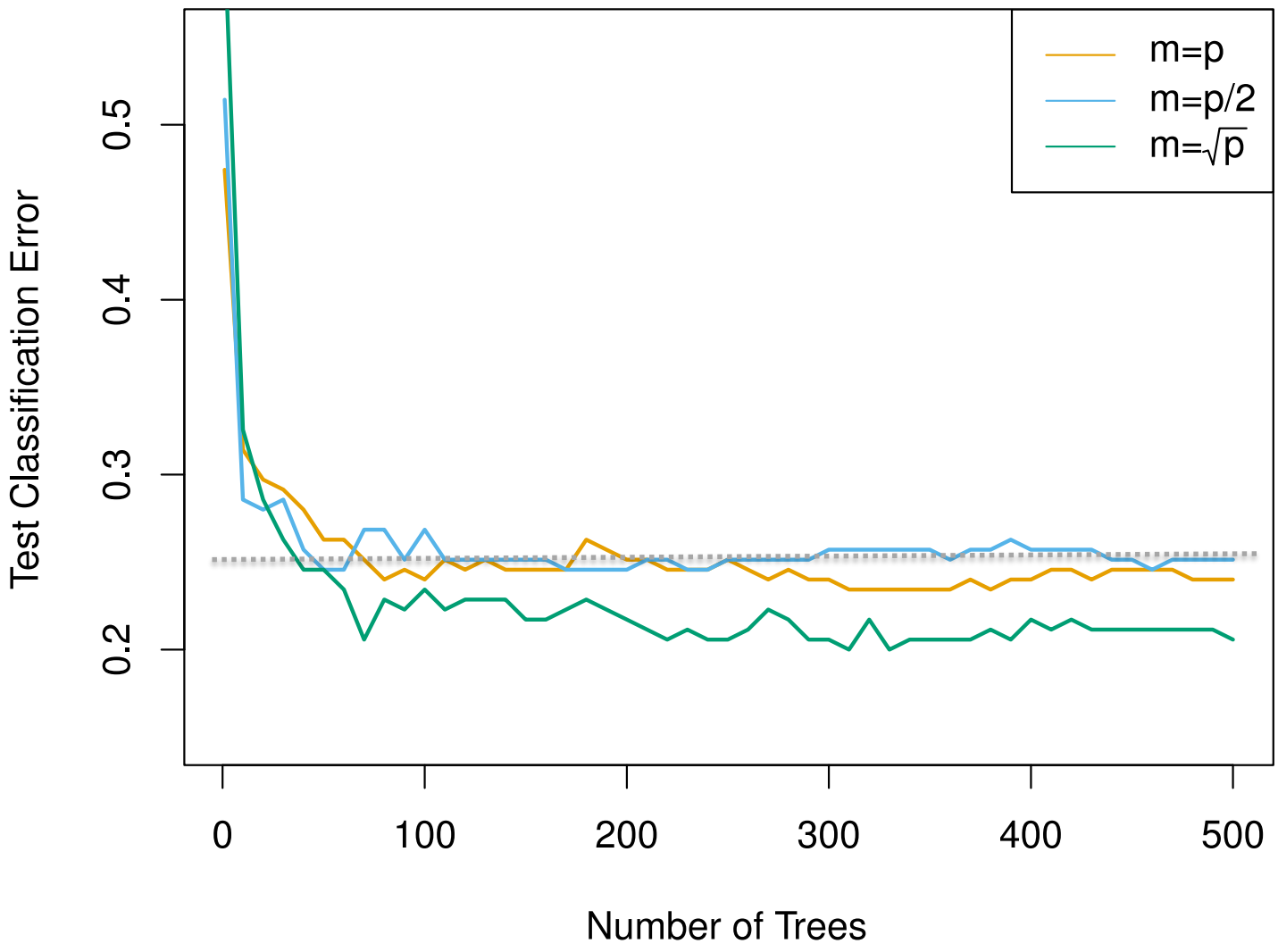
In practice the best values for these parameters will depend on the problem, and they should be treated as tuning parameters.

Like with Bagging, we can use OOB and therefore RF can be fit in one sequence, with cross-validation being performed along the way. Once the OOB error stabilizes, the training can be terminated.

Example

- 4,718 genes measured on tissue samples from 349 patients.
- Each gene has different expression
- Each of the patient samples has a qualitative label with 15 different levels: either normal or 1 of 14 different types of cancer.

Use random forests to predict cancer type based on the 500 genes that have the largest variance in the training set.


Test Classification Error
0.2
0.3
0.4
0.5
0
100
200
300
400
500
Number of Trees
m=p
m=p/2
m=√p

Random Forests Issues

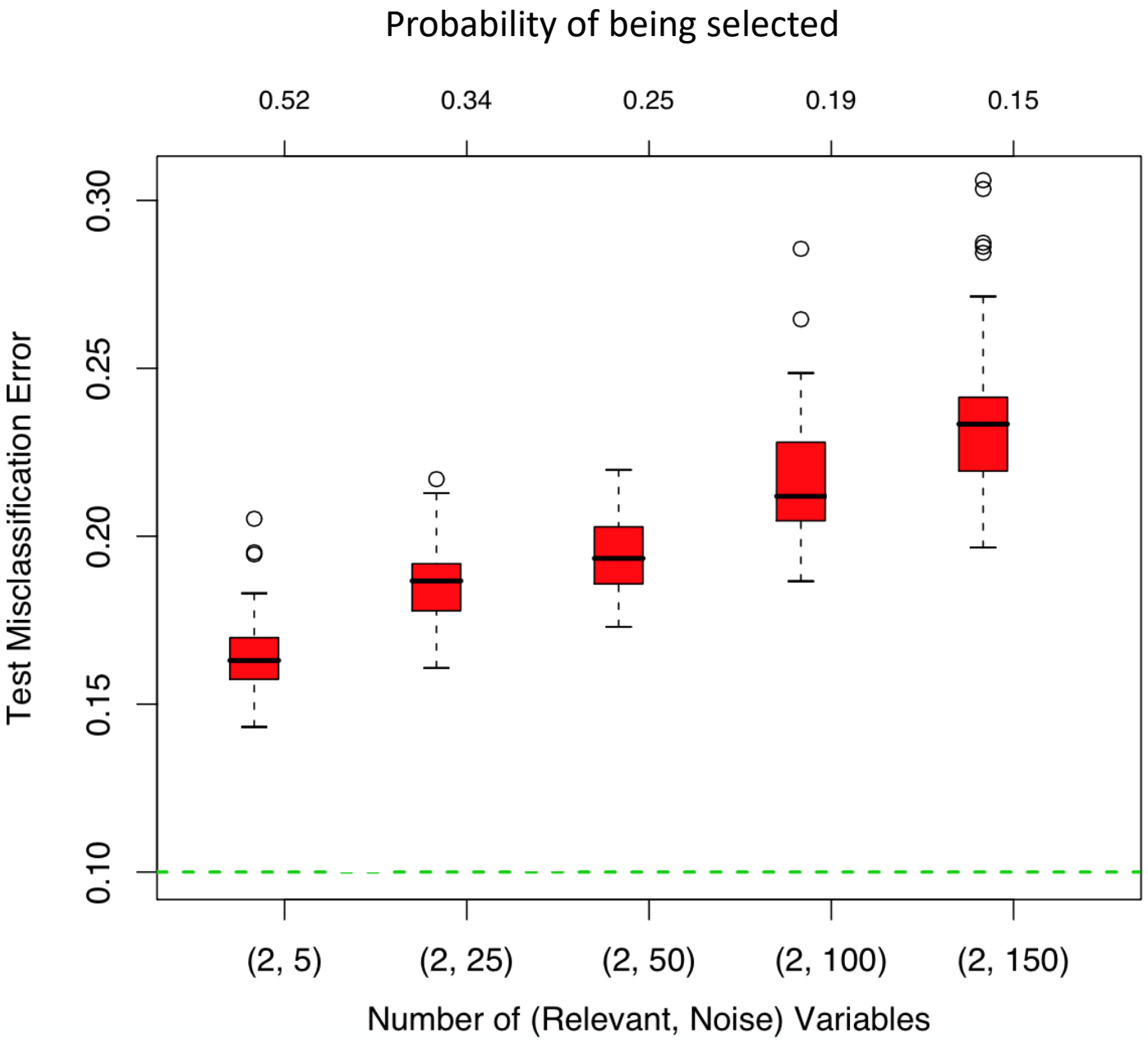
When the number of variables is large, but the fraction of relevant variables is small, random forests are likely to perform poorly when *m* is small

Why?

Because:

At each split the chance can be small that the relevant variables will be selected

For example, with 3 relevant and 100 not so relevant variables the probability of any of the relevant variables being selected at any split is ~0.25


Probability of being selected
0.52
0.34
0.25
0.19
0.15
Test Misclassification Error
0.10
0.15
0.20
0.25
0.30
(2, 5)
(2, 25)
(2, 50)
(2, 100)
(2, 150)
Number of (Relevant, Noise) Variables

Can RF overfit?

Random forests "cannot overfit" the data wrt to number of trees.

Why?

The number of trees, *B* does not mean increase in the flexibility of the model

Tree Depth

I have seen discussion about gains in performance by controlling the depths of the individual trees grown in random forests. I usually use full-grown trees and seldom it costs much (in the classification error) and results in one less tuning parameter.

Random Forest Example

```
"

library(randomForest)

Train=GermanCredit[Index,]
Model_forest <- randomForest(Class ~ ., data = Train,ntree=500)
pred<-predict(Model_forest,Test)
table(pred,Test$Class)
```

```
> table(pred,Test$Class)

pred    Bad Good
  Bad    22   11
  Good   30  137
> 
```

Random Forest Example

```
Train=GermanCredit[Index,]
Model_forest2 <- randomForest(Class ~ ., data = Train,ntree=40)
pred2<-predict(Model_forest2,Test)
table(pred2,Test$Class)
```

```
> table(pred2,Test$Class)

pred2   Bad Good
  Bad    23   15
  Good   29  133
```

Random Forest Example

```
#####################################
library(caret)
Model_forest_caret <- train(Class ~ ., data = Train,method='rf',
                            trControl=trainControl(method = "oob"),
                            tuneGrid=expand.grid(mtry = 10:50))
pred_caret<-predict(Model_forest_caret,Test)
table(pred_caret,Test$Class)
```

No of variables randomly selected at each split

```
43    0.75500   0.3864262459
44    0.76625   0.4166822634
45    0.76375   0.4090425865
46    0.75375   0.3840285160
47    0.76750   0.4163424125
48    0.75500   0.3878825734
49    0.75250   0.3786870842
50    0.76125   0.4042048787

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was mtry = 17.
```

Random Forest Example

```
> table(pred_caret,Test$Class)

pred_caret Bad Good
      Bad   21   15
      Good  31  133
```

Outline

Bagging

Random Forests

Boosting

Boosting

Boosting is a general approach that can be applied to many statistical learning methods for regression or classification.

Bagging: Generate multiple trees from bootstrapped data and average the trees.

RF produces more independent trees by randomly selecting a subset of predictors at each step

Boosting

Boosting works very differently.

1. Boosting does not involve bootstrap sampling
2. Trees are grown sequentially: each tree is grown using information from previously grown trees
3. Like bagging, boosting involves combining a large number of decision trees, $\hat{f}^1, \ldots, \hat{f}^B$

Boosting

- Definition of Boosting:

  Boosting refers to a general method of producing a very accurate prediction rule by combining rough and moderately inaccurate rules-of-thumb.

- Intuition:

  1) No learner is always the best;

  2) Construct a set of base-learners which when combined achieves higher accuracy

Boosting(cont'd)

- 3) Different learners may:

  --- Be trained by different algorithms

  --- Use different modalities(features)

  --- Focus on different subproblems

  --- ……

4) A week learner is "rough and moderately inaccurate" predictor but one that can predict better than chance.

background of Adaboost[2]

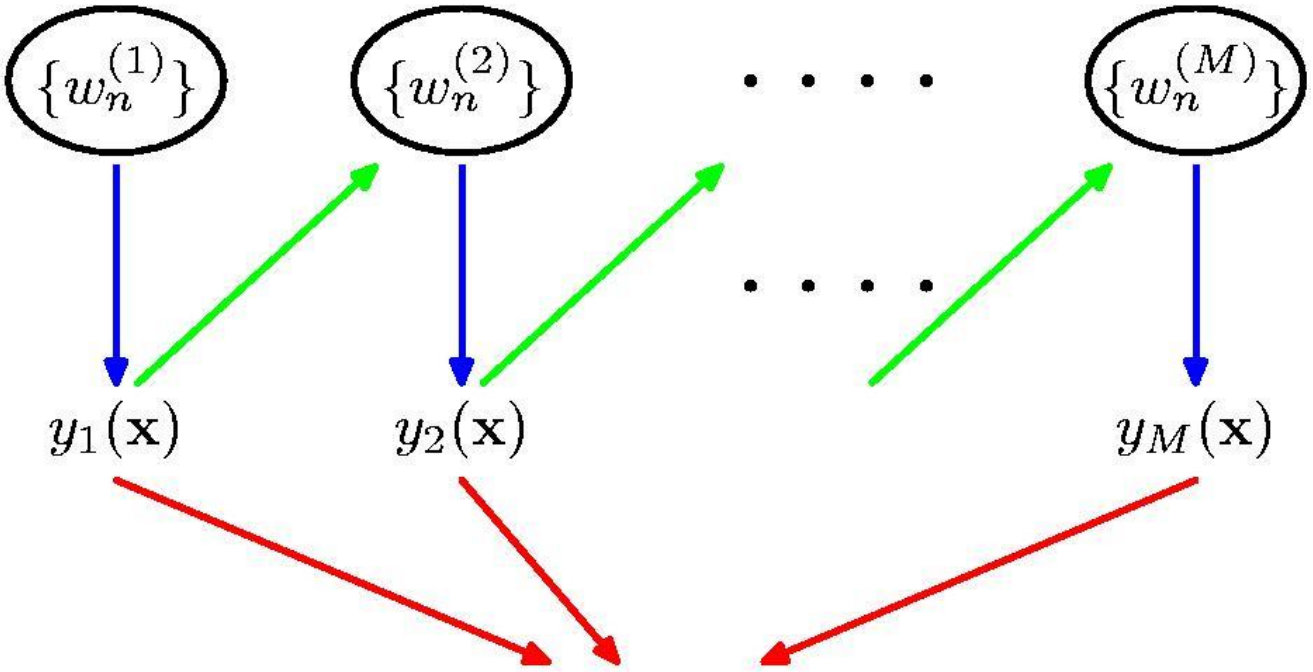
- [Freund & Schapire '95]:
  - introduced "AdaBoost" algorithm
  - strong practical advantages over previous boosting algorithms
- experiments and applications using AdaBoost:

[Drucker & Cortes '96]
[Jackson & Craven '96]
[Freund & Schapire '96]
[Quinlan '96]
[Breiman '96]
[Maclin & Opitz '97]
[Bauer & Kohavi '97]
[Schwenk & Bengio '98]
[Schapire, Singer & Singhal '98]
[Abney, Schapire & Singer '99]
[Haruno, Shirai & Ooyama '99]
[Cohen & Singer' 99]
[Dietterich '00]
[Schapire & Singer '00]
[Collins '00]
[Escudero, Màrquez & Rigau '00]
[Iyer, Lewis, Schapire, Singer & Singhal '00]
[Onoda, Rätsch & Müller '00]
[Tieu & Viola '00]
[Walker, Rambow & Rogati '01]
[Rochery, Schapire, Rahim & Gupta '01]
[Merler, Furlanello, Larcher & Sboner '01]
⋮

- continuing development of theory and algorithms:

[Breiman '98, '99]
[Schapire, Freund, Bartlett & Lee '98]
[Grove & Schuurmans '98]
[Mason, Bartlett & Baxter '98]
[Schapire & Singer '99]
[Cohen & Singer '99]
[Freund & Mason '99]
[Domingo & Watanabe '99]
[Mason, Baxter, Bartlett & Frean '99, '00]
[Duffy & Helmbold '99, '02]
[Freund & Mason '99]
[Ridgeway, Madigan & Richardson '99]
[Kivinen & Warmuth '99]
[Friedman, Hastie & Tibshirani '00]
[Rätsch, Onoda & Müller '00]
[Rätsch, Warmuth, Mika, Onoda, Lemm & Müller '00]
[Allwein, Schapire & Singer '00]
[Friedman '01]
[Koltchinskii, Panchenko & Lozano '01]
[Collins, Schapire & Singer '02]
[Demiriz, Bennett & Shawe-Taylor '02]
[Lebanon & Lafferty '02]
⋮

Schematic illustration of the boosting Classifier



$$Y_M(\mathbf{x}) = \text{sign}\left(\sum_m^M \alpha_m y_m(\mathbf{x})\right)$$

Adaboost

- 1. Initialize the data weighting coefficients $\{w_n\}$ by setting $w_n^{(1)} = 1/N$ for $n = 1,...,N$
- 2. For $m = 1,...,M$:
- (a) Fit a classifier $y_m(x)$ to the training data by minimizing the weighted error function

$$J_m = \sum_{n=1}^{N} w_n^{(m)} I(y_m(x_n) \neq t_n)$$

- Where $I(y_m(x_n) \neq t_n)$ is the indicator function and equals 1 when $y_m(x_n) \neq t_n$ and 0 otherwise.

Adaboost(cont'd)

- (b) Evaluate the quantities

$$\varepsilon_m = \frac{\sum_{n=1}^{N} w_n^{(m)} I(y_m(x_n) \neq t_n)}{\sum_{n=1}^{N} w_n^{(m)}}$$

and then use these to evaluate

$$\alpha_m = \ln\{\frac{1-\varepsilon_m}{\varepsilon_m}\}$$

Adaboost(cont'd)

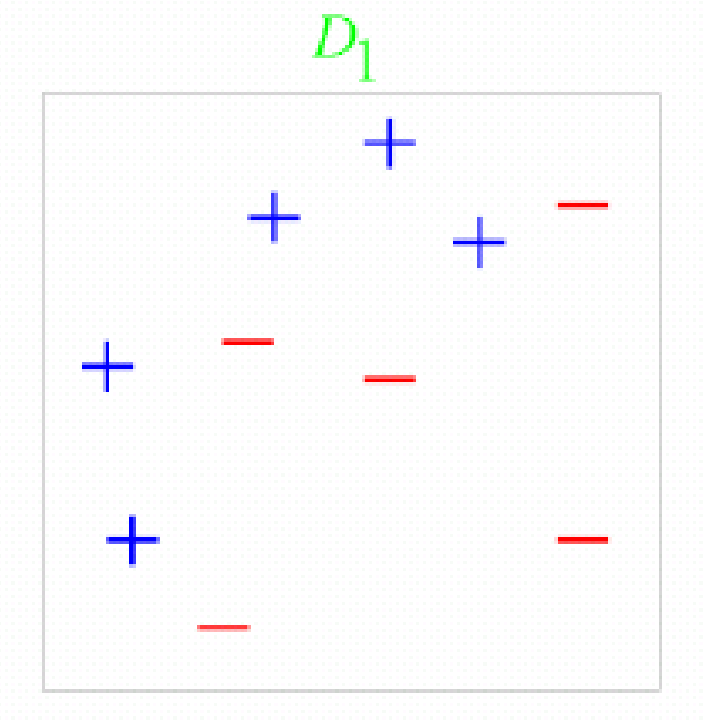
- (c) Update the data weighting coefficients

$$w_n^{(m+1)} = w_n^{(m)} \exp\{\alpha_m I(y_m(x_n) \neq t_n)\}$$

- 3. Make predictions using the final model, which is given by

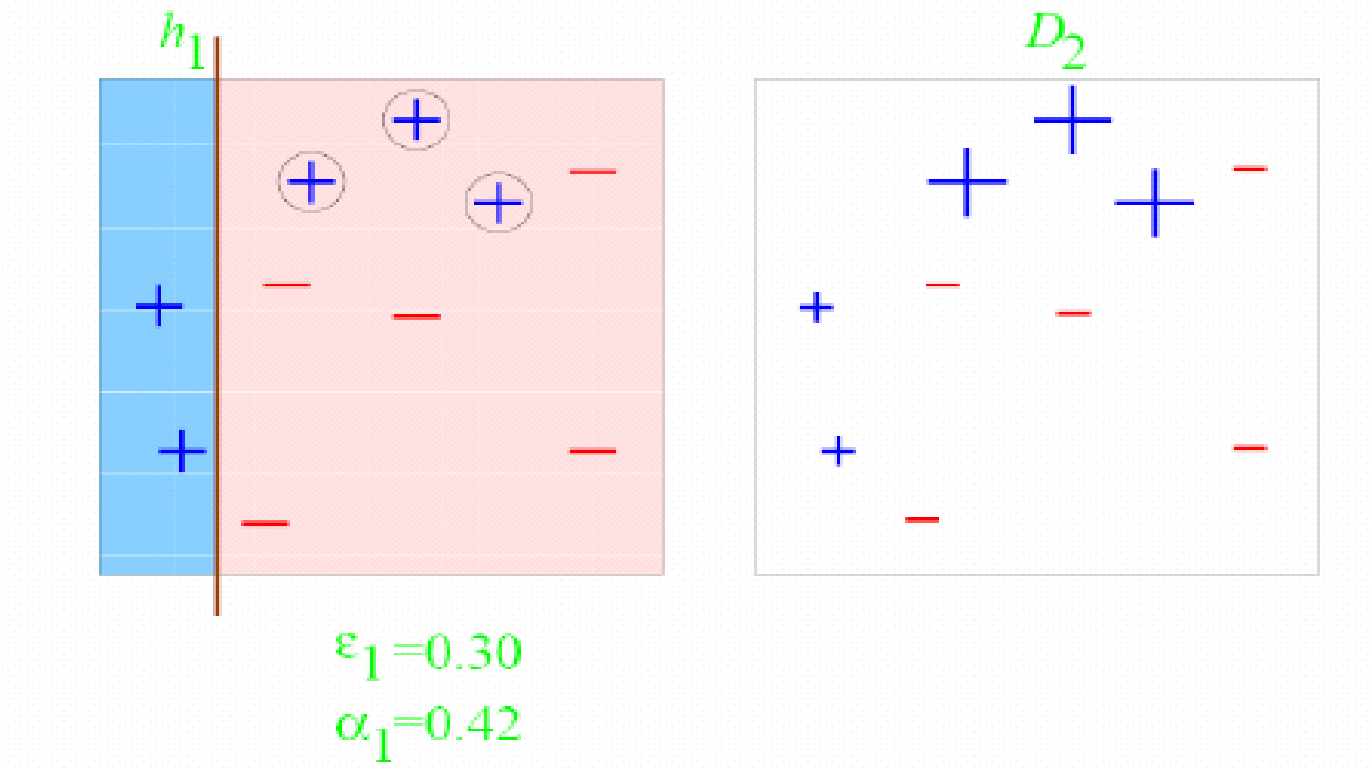
$$Y_M(x) = sign(\sum_{m=1}^{M} \alpha_m y_m(x))$$

A toy example[2]



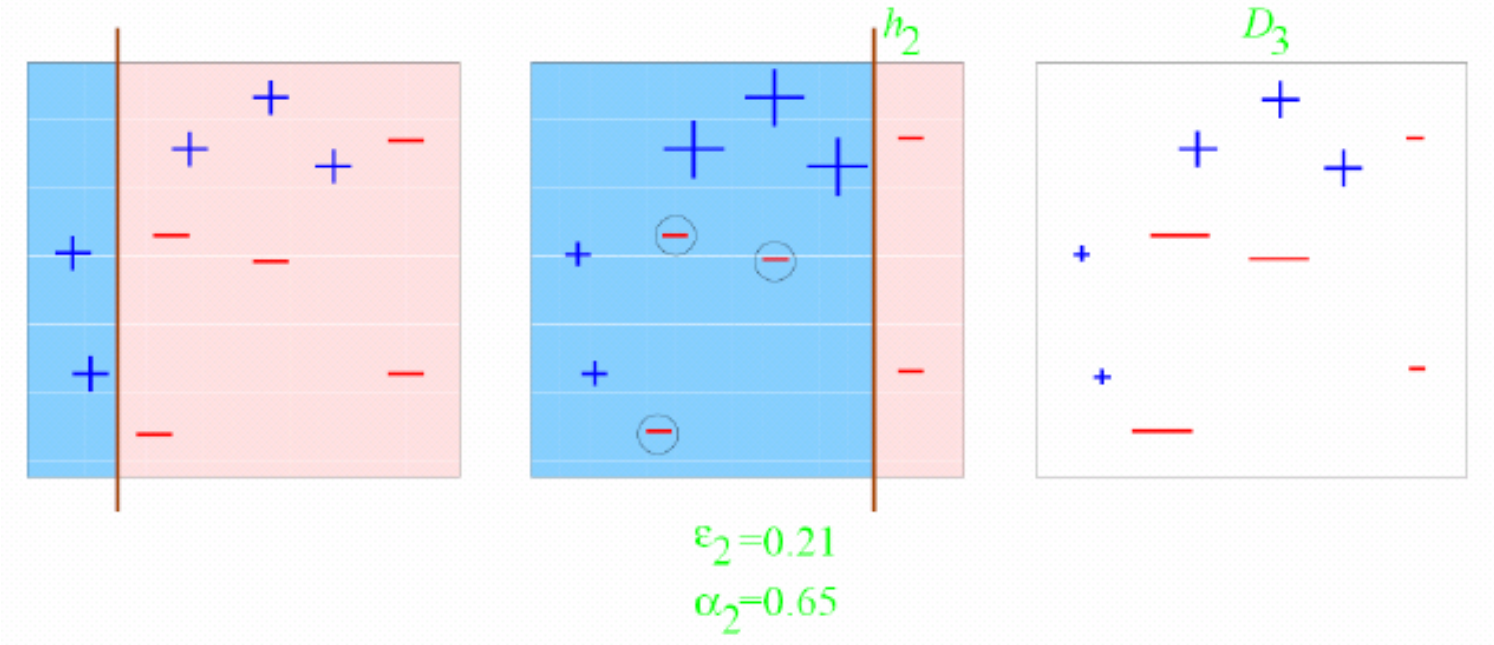
Training set: 10 points
(represented by plus or minus)
Original Status: Equal Weights
for all training samples

A toy example(cont'd)



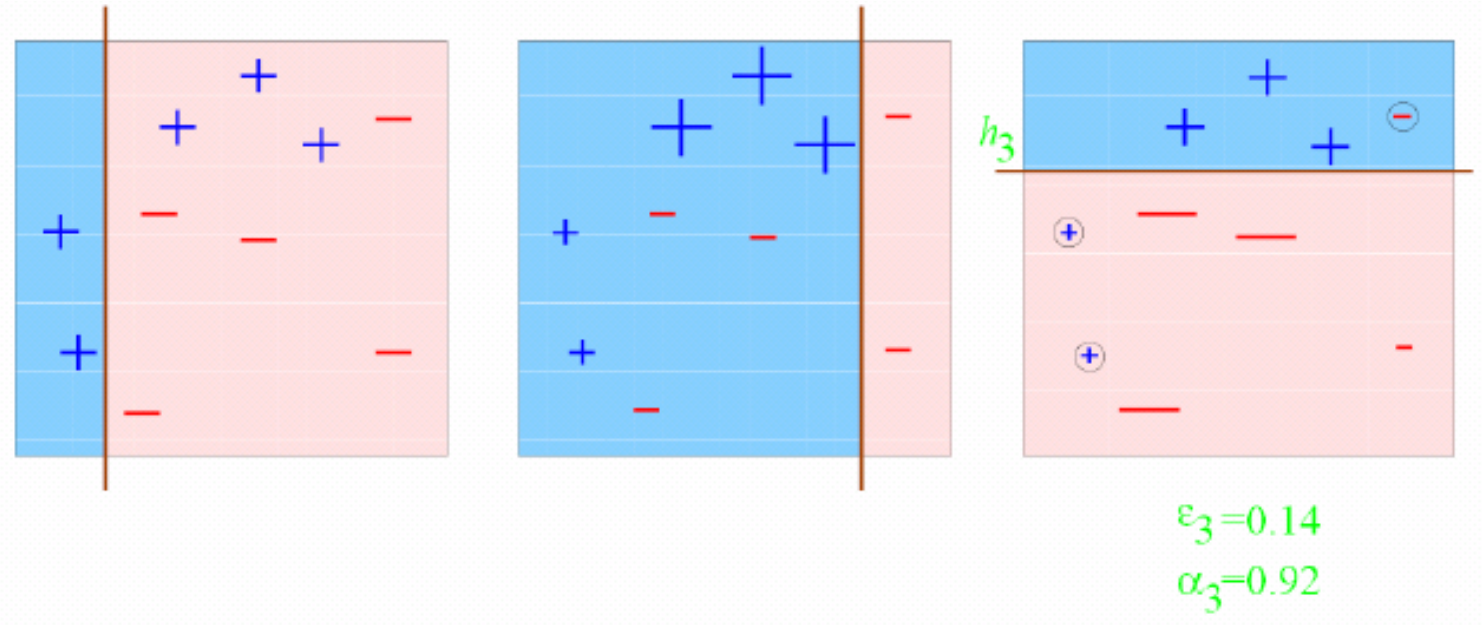
Round 1: Three "plus" points are not correctly classified;
They are given higher weights.

A toy example(cont'd)



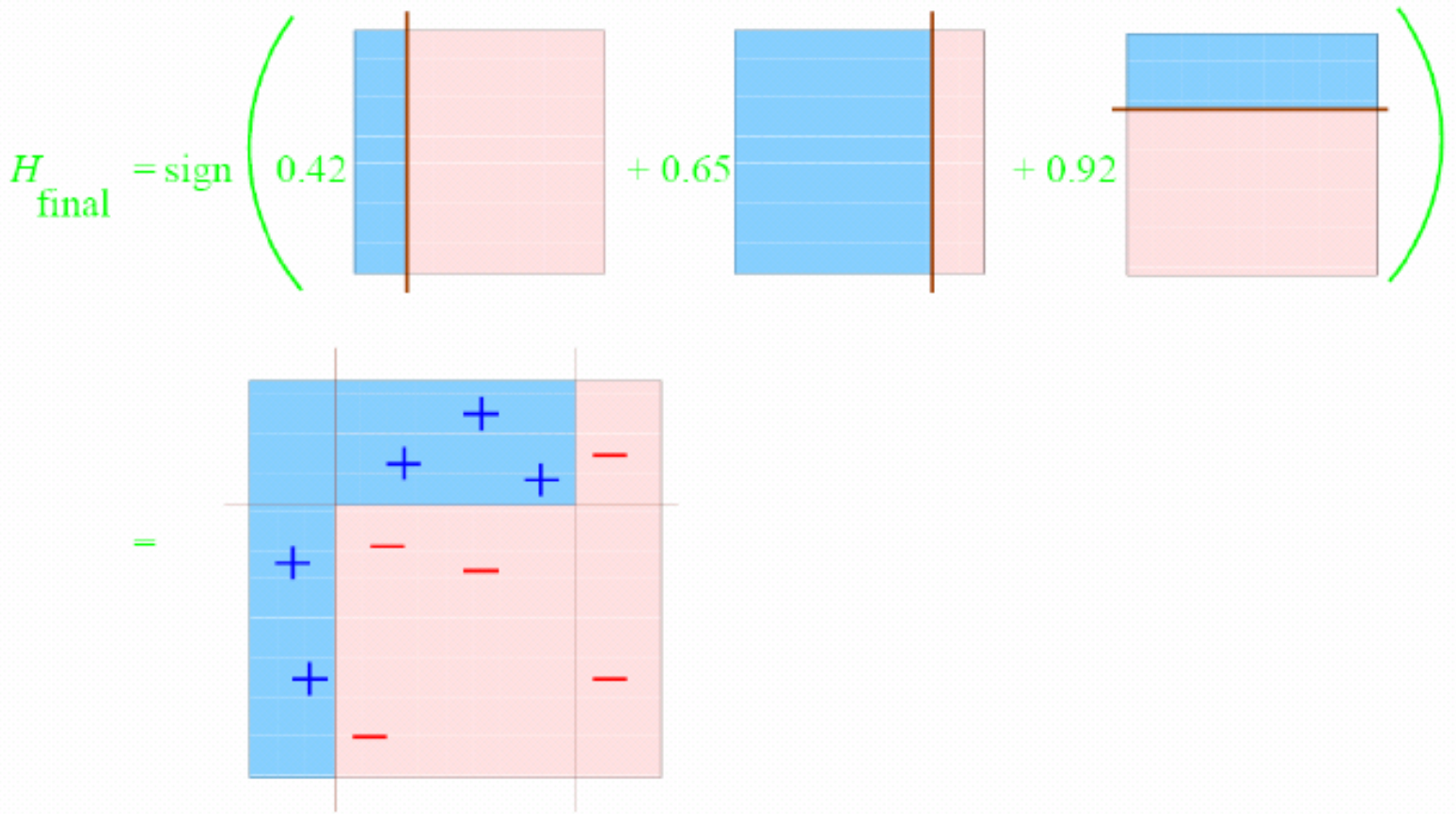
Round 2: Three "minuse" points are not correctly classified;
They are given higher weights.

A toy example(cont'd)



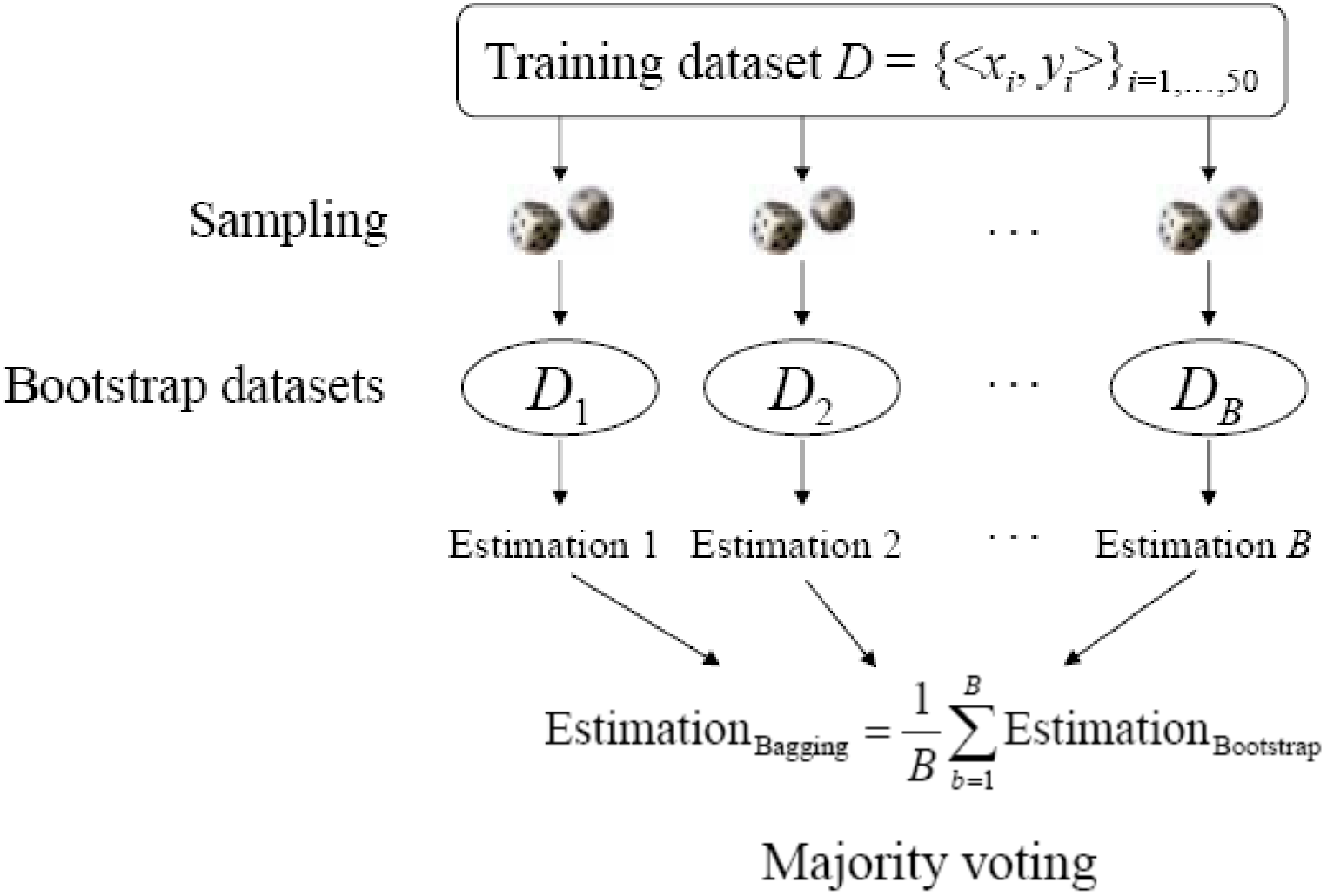
Round 3: One "minuse" and two "plus" points are not correctly classified;
They are given higher weights.

A toy example(cont'd)



Final Classifier: integrate the three "weak" classifiers and obtain a final strong classifier.

Revisit Bagging



Shrinkage

The predictions of each tree are added together sequentially.

The contribution of each tree to this sum can be weighted to slow down the learning by the algorithm. This weighting is called a shrinkage or a learning rate.

> *Each update is simply scaled by the value of the "learning rate parameter v"*

— Greedy Function Approximation: A Gradient Boosting Machine [PDF], 1999

The effect is that learning is slowed down, in turn require more trees to be added to the model, in turn taking longer to train, providing a configuration trade-off between the number trees and learning rate.

> *Decreasing the value of v [the learning rate] increases the best value for M [the number of trees].*

R Example

```
rm(list = ls())
library(caret)
library(ada)
set.seed(2018)

data(GermanCredit)

Index=sample(nrow(GermanCredit),nrow(GermanCredit)*.8)
Train=GermanCredit[Index,]
Test=GermanCredit[-Index,]

Model<-ada(Class~.,data=Train, iter=500)
Predictions=predict(Model,Test)
table(Predictions,Test$Class)
```

```
> table(Predictions,Test$Class)

Predictions Bad Good
       Bad   26   17
       Good  26  131
```

```
#-------------------------
```

R Example

AdaBoost.M1	AdaBoost.M1	Classification	adabag, plyr	coeflearn
Adaptive Mixture Discriminant Analysis	amdai	Classification	adaptDA	model
Adaptive-Network-Based Fuzzy Inference System	ANFIS	Regression	frbs	num.labels, max.it
Bagged AdaBoost	AdaBag	Classification	adabag, plyr	mfinal, maxdepth
Boosted Classification Trees	ada	Classification	ada, plyr	iter, maxdepth, nu

Controls the depth of trees,

number of boosting iterations to perform.
Default = 50.

shrinkage parameter for boosting,
default taken as 1.

Caret Example

```
#-------------------------------------------------------
Model_caret<-train(Class~.,data=Train, method='ada')
Predictions_caret=predict(Model_caret,Test)
table(Predictions_caret,Test$Class)
```

```
Resampling: Bootstrapped (25 reps)
Summary of sample sizes: 800, 800, 800, 800, 800, 800, ...
Resampling results across tuning parameters:

  maxdepth  iter  Accuracy      Kappa
  1          50   0.7080208537  0.1414830997
  1         100   0.7175607190  0.2114075284
  1         150   0.7238340183  0.2511969153
  2          50   0.7210698919  0.2495039034
  2         100   0.7358628982  0.3075408536
  2         150   0.7383237277  0.3268073602
  3          50   0.7369795310  0.3165055721
  3         100   0.7447870139  0.3496180726
  3         150   0.7457845397  0.3575366838

Tuning parameter 'nu' was held constant at a value of 0.1
Accuracy was used to select the optimal model using the largest value.
The final values used for the model were iter = 150, maxdepth = 3 and nu = 0.1.
```

Caret Example

```
> table(Predictions_caret,Test$Class)

Predictions_caret Bad Good
             Bad   24   11
             Good  28  137
> 
```