statinfer

Gradient Boosting Method

Venkat Reddy

statinfer

Statinfer.com

Data Science Training and R&D

Training on

Data Science

Bigdata Analytics

Machine Learning

Predictive Modelling

Deep Learning

Corporate Training

Classroom Training

Online Training

Contact us

info@statinfer.com

venkat@statinfer.com

Note

- This presentation is just my class notes. The course notes for data science training is written by me, as an aid for myself.
- The best way to treat this is as a high-level summary; the actual session went more in depth and contained detailed information and examples
- Most of this material was written as informal notes, not intended for publication
- Please send questions/comments/corrections to info@statinfer.com
- Please check our website statinfer.com for latest version of this document

-*Venkata Reddy Konasani*

(Cofounder statinfer.com)

statinfer

Contents

Contents

Boosting

- Boosting is one more famous ensemble method
- Boosting uses a slightly different techniques to that of bagging.
- Boosting is a well proven theory that works really well on many of the machine learning problems like speech recognition
- If bagging is wisdom of crowds then boosting is wisdom of crowds where each individual is given some weight based on their expertise

Boosting

- Boosting in general decreases the bias error and builds strong predictive models.
- Boosting is an iterative technique. We adjust the weight of the observation based on the previous classification.
- If an observation was classified incorrectly, it tries to increase the weight of this observation and vice versa.

Boosting Main idea

Take a random sample from population of size N

Each record has 1/N Chance of picking

Let 1/N be the weight w

Build a classifier
Note down the accuracy

The Classifier may misclassify some of the records. Note them down

Take a weighted sample

This time give more weight to misclassified records from previous model

Update the weight w accordingly to pick the misclassified records

Build a new classifier on the reweighted sample

Since we picked many previously misclassified records, we expect this model to build a better model for those records

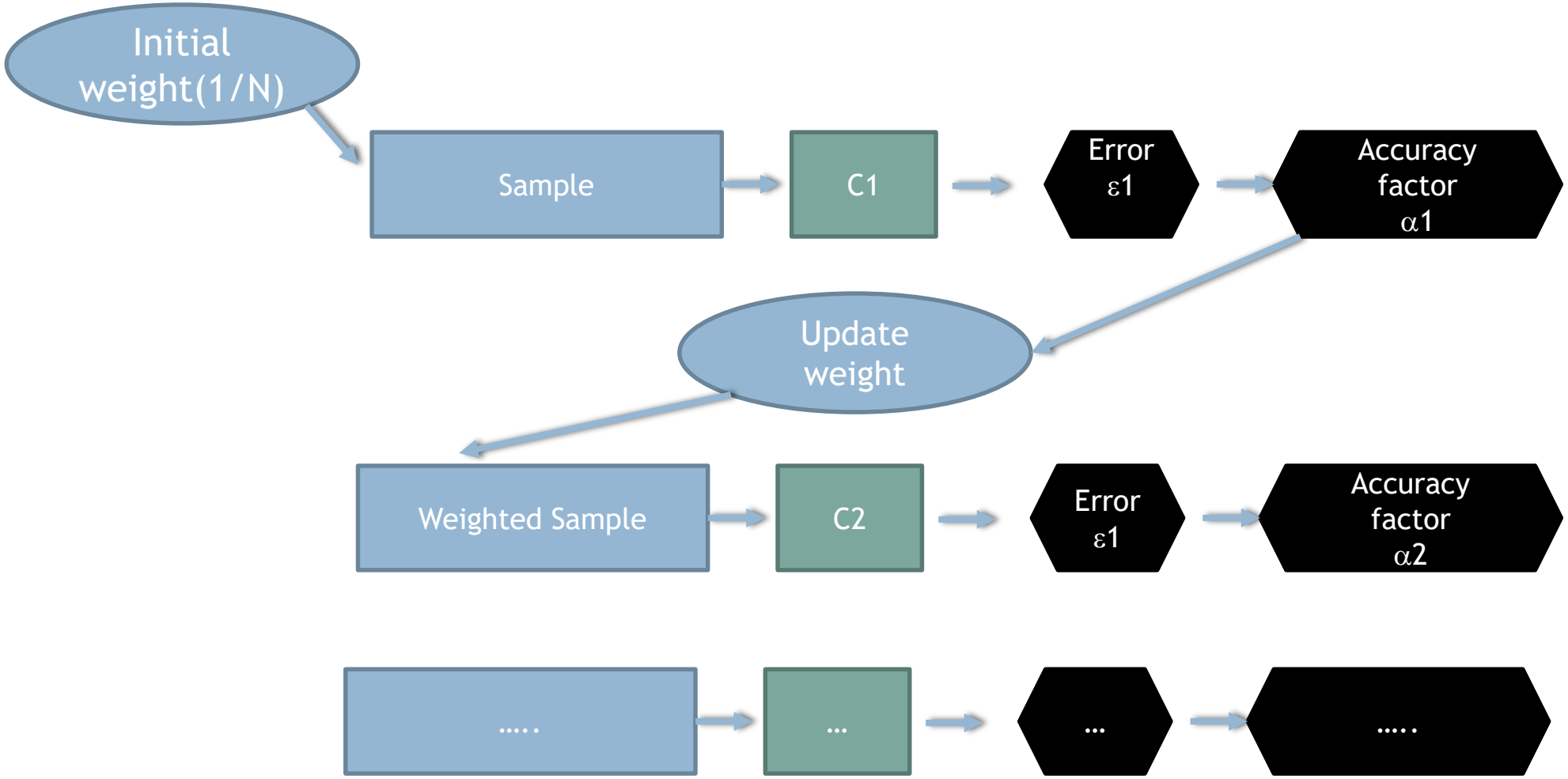
Check the error and resample

Does this classifier still has some misclassifications

If yes, then re-sample

Final Weighted Classifier $C = \sum \alpha_i c_i$

Boosting Main idea



statinfer

How weighted samples are taken

Data	1	2	3	4	5	6	7	8	9	10
Class	-	-	+	+	-	+	-	-	+	+
Predicted Class M1	-	-	-	-	-	-	-	-	+	+
M1 Result	✓	✓	✗	✗	✓	✗	✓	✓	✓	✓

Weighted Sample1	1	2	3	4	5	6	7	4	3	6
Class	-	-	+	+	-	+	-	+	+	+
Predicted Class M2	-	-	+	+	+	+	+	+	+	+
M2 Result	✓	✓	✓	✓	✗	✓	✗	✓	✓	✓

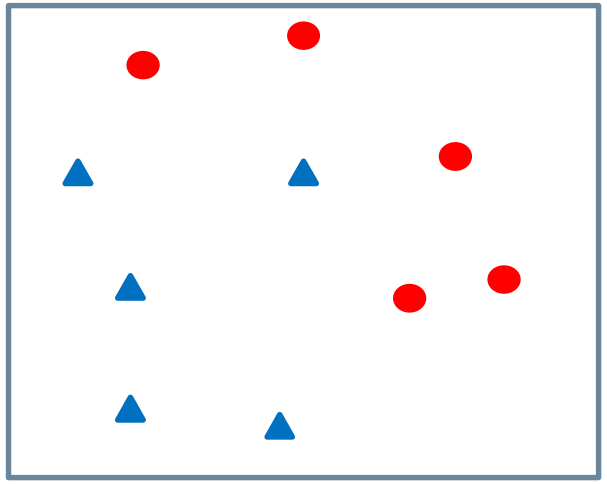
Weighted Sample2	6	5	3	4	5	6	7	7	5	7
Class	+	-	+	+	-	+	-	-	-	-
Predicted Class M3	+	-	+	+	-	+	-	-	-	-
M3 Result	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

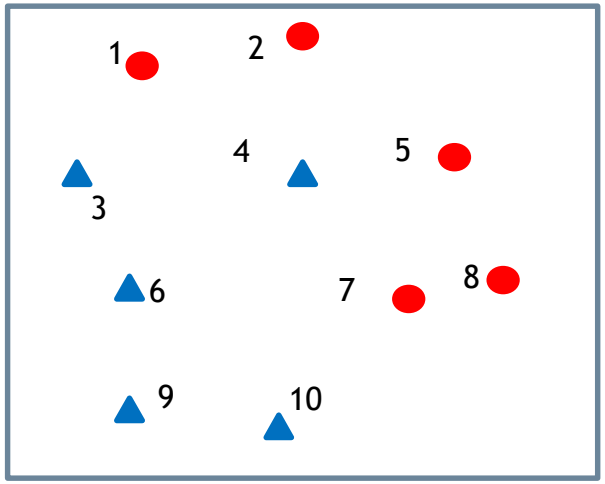
Boosting illustration

Boosting illustration

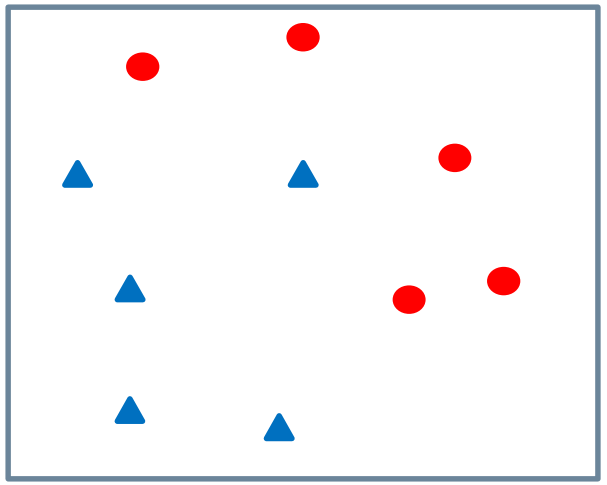
Below is the training data and their class
We need to take a note of record numbers, they will help us in weighted sampling later

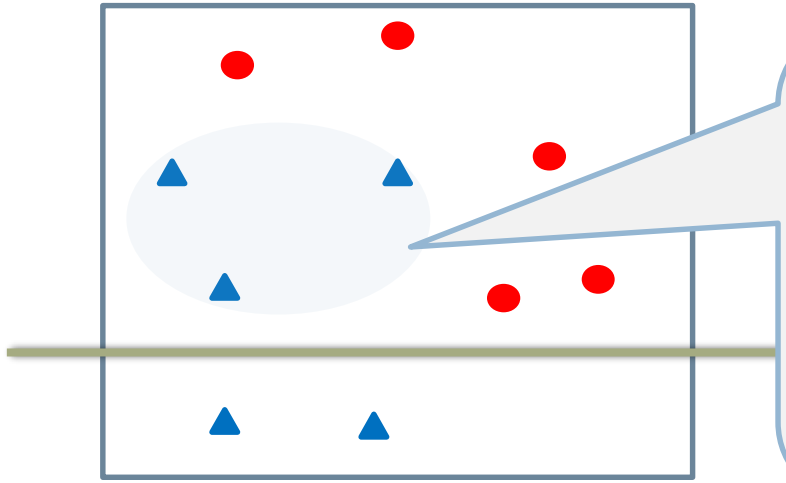
Data Points	1	2	3	4	5	6	7	8	9	10
Class	-	-	+	+	-	+	-	-	+	+





Boosting illustration

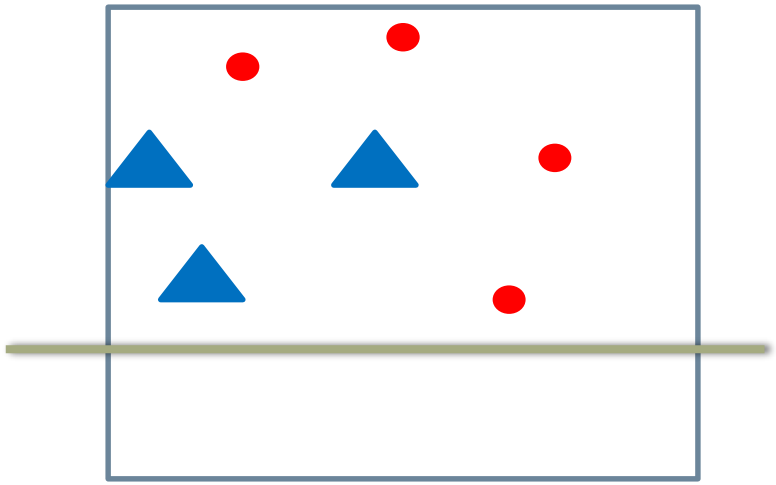


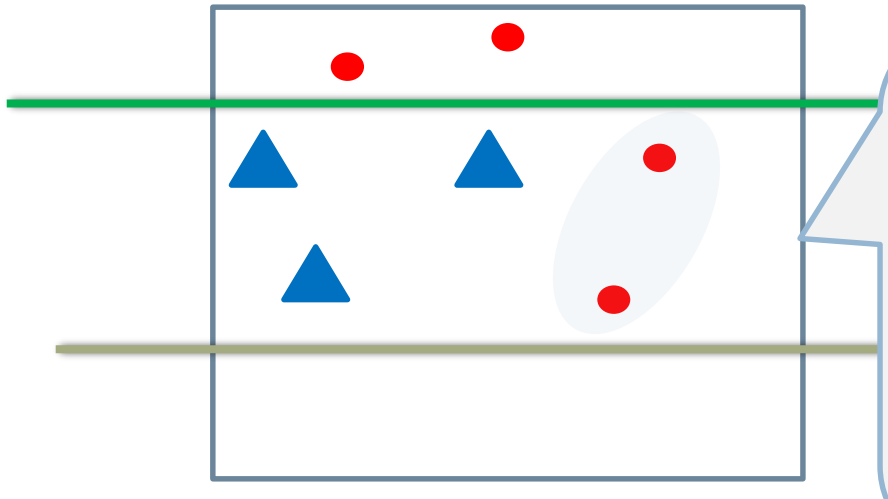


- Model M1 is built, anything above the line is - and below the line is +
- 3 out of 10 are misclassified by the model M1
- These data points will be given more weight in the re-sampling step
- We may miss out on some of the correctly classified records

Data	1	2	3	4	5	6	7	8	9	10
Class	-	-	+	+	-	+	-	-	+	+
Predicted Class M1	-	-	-	-	-	-	-	-	+	+
M1 Result	✓	✓	✗	✗	✓	✗	✓	✓	✓	✓

Boosting illustration

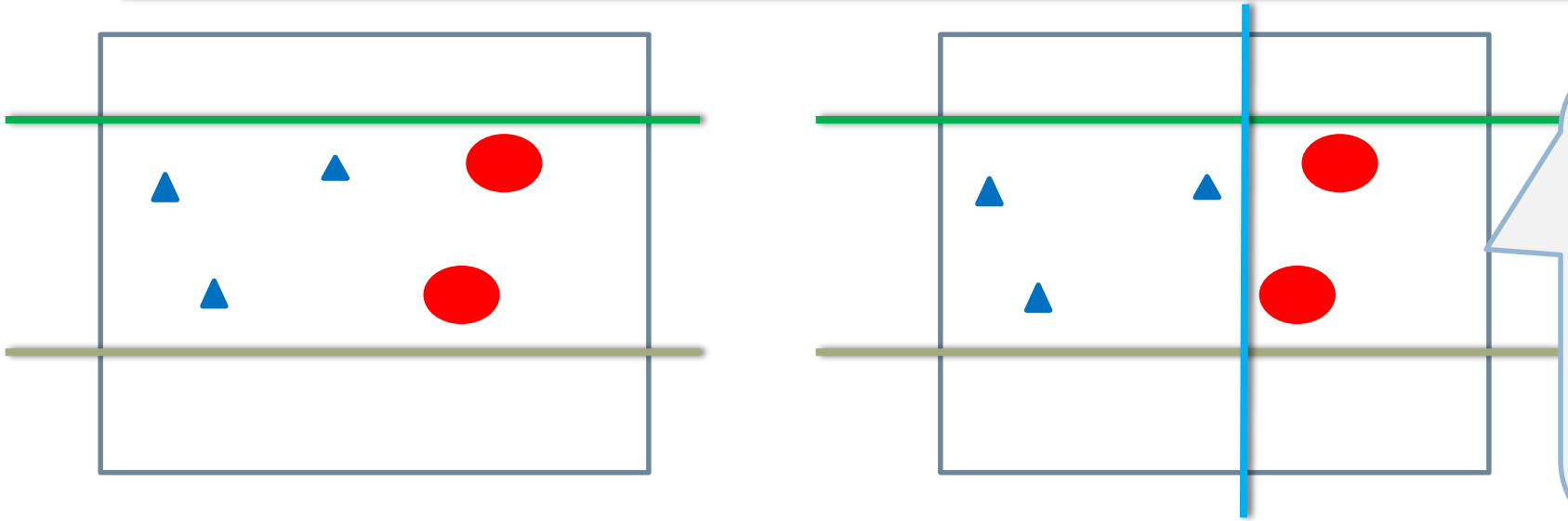




- The misclassified points 3,4,& 6 have appeared more often than others in this weighted sample.
- The sample points 9,10 didn't appear
- M2 is built on this data. Anything above the line is - and below the line is +
- M2 is classifying the points 5 & 7 incorrectly.
- They will be given more weight in the next sample

Weighted Sample1	1	2	3	4	5	6	7	4	3	6
Class	-	-	+	+	-	+	-	+	+	+
Predicted Class M2	-	-	+	+	+	+	+	+	+	+
M2 Result	✓	✓	✓	✓	✗	✓	✗	✓	✓	✓

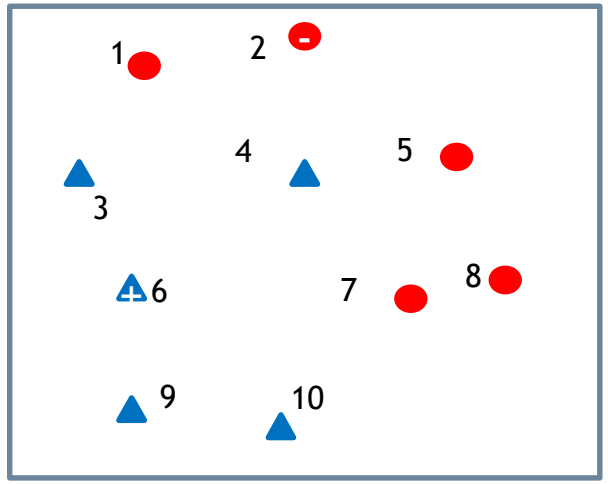
Boosting illustration

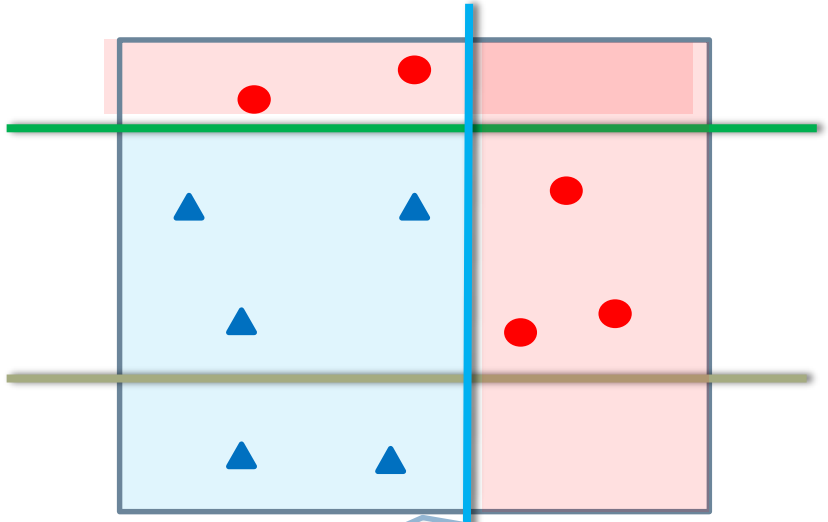


- The misclassified points 5 & 7 have appeared more often than others in this weighted sample.
- M3 is built on this data. Anything above the line is – and below the line is +
- M3 is now classifying everything correctly

Weighted Sample2	6	5	3	4	5	6	7	7	5	7
Class	+	-	+	+	-	+	-	-	-	-
Predicted Class M3	+	-	+	+	-	+	-	-	-	-
M3 Result	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔

Boosting illustration





- The final model now will be picked on weighted Votes.
- For a given data point more than 2 models seam to be indicating the right class.
- For example take point 6, it is classified as – by M1, + by M2 and + by M3, final result will be +
- Similarly take a point 2, it will be classified as –by M1, -by M2 and + by M3, final result will be -
- So the final weighted combination of three models predictions will yield in accurate perdition.

statinfer

Theory behind Boosting Algorithm

Theory behind Boosting Algorithm

- Take the dataset Build a classifier C_m and find the error
- Calculate error rate of the classifier
  - Error rate of ε_m
    - = $\sum w_i I(y_i \neq C_m(x))$ / $\sum w_i$
    - =Sum of misclassification weight / sum of sample weights
- Calculate an intermediate factor called α. It analogous to accuracy rate of the model. It will be later used in weight updating. It is derived from error
  - $\alpha_m = \log((1-\varepsilon_m)/\varepsilon_m)$

Theory behind Boosting Algorithm..contd

- Update weights of each record in the sample using the α factor. The indicator function will make sure that the misclassifications are given more weight
  - For i=1,2….N
    - $w_{i+1} = w_i e^{\alpha_m I(y_i \neq C_m(x))}$
    - Renormalize so that sum of weights is 1
- Repeat this model building and weight update process until we have no misclassification
- Final collation is done by voting from all the models. While taking the votes, each model is weighted by the accuracy factor α
  - $C = sign(\sum \alpha_i C_i(x))$

Gradient Boosting

- **Ada boosting**
  - Adaptive Boosting
  - Till now we discussed Ada boosting technique. Here we give high weight to misclassified records.
- **Gradient Boosting**
  - Similar to Ada boosting algorithm.
  - The approach is same but there are slight modifications during re-weighted sampling.
  - We update the weights based on misclassification rate and gradient
  - Gradient boosting serves better for some class of problems like regression.

GBM- Parameters

```
gbm(formula = formula(data),
    distribution = "bernoulli",
    data = list(),
    weights,
    var.monotone = NULL,
    n.trees = 100,
    interaction.depth = 1,
    n.minobsinnode = 10,
    shrinkage = 0.001,
    bag.fraction = 0.5,
    train.fraction = 1.0,
    cv.folds=0,
    keep.data = TRUE,
    verbose = "CV",
    class.stratify.cv=NULL,
    n.cores = NULL)
```

GBM- Parameters verbose

- If TRUE, gbm will print out progress and performance indicators.

GBM- Parameters distribution

- "multinomial" for classification when there are more than 2 classes
- "bernoulli" for logistic regression for 0-1 outcomes
- If not specified, gbm will try to guess:
  - if the response has only 2 unique values, bernoulli is assumed;
  - otherwise, if the response is a factor, multinomial is assumed;

GBM- Parameters n.trees = 100

- The number of steps
- the total number of trees to fit.
- This is equivalent to the number of iterations
- Default value is 100

Code n.trees

```
> #n.trees = 50
> gbm(y  ~ x1+x2, n.trees = 50 )
Distribution not specified, assuming bernoulli ...
gbm(formula = y ~ x1 + x2, n.trees = 50)
A gradient boosted model with bernoulli loss function.
50 iterations were performed.
There were 2 predictors of which 1 had non-zero influence.
>
> #n.trees = 30
> gbm(y  ~ x1+x2, n.trees = 30 )
Distribution not specified, assuming bernoulli ...
gbm(formula = y ~ x1 + x2, n.trees = 30)
A gradient boosted model with bernoulli loss function.
30 iterations were performed.
There were 2 predictors of which 1 had non-zero influence.
```

Number of trees/iterations

GBM- Parameters interaction.depth = 1

- Max_depth in $h_2$0.gbm()
- The depth of each tree, K (interaction.depth). The maximum depth of a tree.
- Interaction.depth equal k =1 means GBM will be building trees no deeper than 1
- A decision tree of depth-1 is just one rule that creates boundary in one variable. It captures one specific pattern effectively.
- Used to control over-fitting – We want to build many weak classifiers and aggregate them to make the ensemble learning effective
- Interaction depth is not equal to number of variables to use, it is just the depth of tree

GBM- Parameters interaction.depth = 1

- Tuning:
  - Have interaction depth as 1 to reduce overfitting. Simpler models are always less likely to overfit.
  - But if the depth of the tree is 1 then it takes of time to execute. You can try any value between 2-5 to reduce computation time

interaction.depth illustration

```
> gm<-gbm(Overall_Satisfaction~Region+Age+Order.Quantity+Customer_Type+Improvement.Area, data=train,
+          n.trees = 1 )
Distribution not specified, assuming bernoulli ...
> summary(gm)
                              var rel.inf
Region                     Region     100
Age                           Age       0
Order.Quantity     Order.Quantity       0
Customer_Type       Customer_Type       0
Improvement.Area Improvement.Area       0
```

Interaction depth=1 then first tree is considering only one split

```
> gm<-gbm(Overall_Satisfaction~Order.Quantity+Age+Customer_Type+Improvement.Area, data=train,
+         n.trees = 1 )
Distribution not specified, assuming bernoulli ...
> summary(gm)
                              var rel.inf
Improvement.Area Improvement.Area     100
Order.Quantity     Order.Quantity       0
Age                           Age       0
Customer_Type       Customer_Type       0
>
```

Interaction depth=1 then first tree is considering only one split

interaction.depth illustration

```
> #Lets increase interaction.depth = 2 with one tree. Keep back Order.Quantity
> gm<-gbm(Overall_Satisfaction~Region+Age+Order.Quantity+Customer_Type+Improvement.Area, data=train,
+          n.trees = 1,interaction.depth = 2)
Distribution not specified, assuming bernoulli ...
> summary(gm)
                              var  rel.inf
Region                     Region 73.28488
Improvement.Area Improvement.Area 26.71512
Age                           Age  0.00000
Order.Quantity     Order.Quantity  0.00000
Customer_Type       Customer_Type  0.00000
```

Interaction depth=2 then first tree is going up to two splits

interaction.depth illustration

```
>
> gm<-gbm(Overall_Satisfaction~Region+Age+Order.Quantity+Customer_Type+Improvement.Area, data=train,
+          n.trees = 1,interaction.depth = 3)
Distribution not specified, assuming bernoulli ...
> summary(gm)
                              var   rel.inf
Region                     Region 67.811804
Improvement.Area Improvement.Area 25.516421
Order.Quantity     Order.Quantity  6.671775
Age                           Age  0.000000
Customer_Type       Customer_Type  0.000000
>
```

Interaction depth is first tree is considering many splits

```
> gm<-gbm(Overall_Satisfaction~Region+Age+Order.Quantity+Customer_Type+Improvement.Area, data=train,
+          n.trees = 1,interaction.depth = 4)
Distribution not specified, assuming bernoulli ...
> summary(gm)
                              var   rel.inf
Region                     Region 63.884175
Improvement.Area Improvement.Area 25.176877
Order.Quantity     Order.Quantity  7.182624
Age                           Age  3.756325
Customer_Type       Customer_Type  0.000000
```

n.minobsinnode

- minimum number of observations in the trees terminal nodes.
- n.minobsinnode = 10 by default.
- Finetuning:
  - A really high value might lead to underfitting
  - A really low value might lead to overfitting
  - You can try 30-100 depending on dataset size
- Interaction depth and minimum number of samples per node are connected.
- If we take care of any one of them then the other one will adjust automatically

LAB: n.minobsinnode

```
> gm<-gbm(Bought~., data=train1,
+         distribution="bernoulli",
+         verbose=TRUE,
+         interaction.depth = 15,
+         n.trees = n,
+         n.minobsinnode=1,
+         bag.fraction=1,
+         shrinkage =1)
Iter   TrainDeviance   ValidDeviance   StepSize   Improve
     1        0.7155             nan     1.0000       nan
     2        0.5970             nan     1.0000       nan
     3        0.5231             nan     1.0000       nan
     4        0.5097             nan     1.0000       nan
     5           inf             nan     1.0000       nan
     6           inf             nan     1.0000       nan
     7           inf             nan     1.0000       nan
     8           inf             nan     1.0000       nan
     9           inf             nan     1.0000       nan
    10           inf             nan     1.0000       nan

>
> #Accuracy on Training and test data
> library(caret)
> conf_matrix<-confusionMatrix(ifelse(predict(gm,  n.trees =n, type="response")<0.5,0,1),train1$Bought)
> conf_matrix$overall[1]
 Accuracy
0.9175705
>
> conf_matrix<-confusionMatrix(ifelse(predict(gm,  n.trees = n, newdata=test1[,-4], type="response")<0.5,0,1)
,test1$Bought)
> conf_matrix$overall[1]
 Accuracy
0.7204301
```

Too few observations in terminal nodes can lead to overfitting

LAB: n.minobsinnode

```
> #Rebuild the model to reduce training accuracy, change min obs pernode to to avoid overfitting
> n=10
> gm<-gbm(Bought~., data=train1,
+         distribution="bernoulli",
+         verbose=TRUE,
+         interaction.depth = 15 ,
+         n.trees =n,
+         n.minobsinnode=25,
+         bag.fraction=1,
+         shrinkage =1)
Iter   TrainDeviance   ValidDeviance   StepSize   Improve
     1        0.8543             nan     1.0000       nan
     2        0.7434             nan     1.0000       nan
     3        0.6839             nan     1.0000       nan
     4        0.6440             nan     1.0000       nan
     5        0.6260             nan     1.0000       nan
     6        0.6097             nan     1.0000       nan
     7        0.6004             nan     1.0000       nan
     8        0.5929             nan     1.0000       nan
     9        0.5804             nan     1.0000       nan
    10        0.5688             nan     1.0000       nan

>
> #Accuracy on Training and test data
> library(caret)
> conf_matrix<-confusionMatrix(ifelse(predict(gm,  n.trees = n, type="response")<0.5,0,1),train1$Bought)
> conf_matrix$overall[1]
 Accuracy
0.8720174
```

Too few observations in terminal nodes can lead to overfitting. Have sufficient number of observations

Shrinkage

- Learn_rate in h20
- The learning rate is a multiplier fraction in gradient boosting algorithm before updating the learning function in each iteration
- This fraction restricts the algorithm's speed in reaching optimum values
- The direct empirical analogy of this parameter is not very obvious
- You can understand this as reduction in the weights of misclassification sample(before preparing for sampling in next step).
- It is also known as gradient descent step size
- a shrinkage parameter applied to each tree in the expansion.
- Also known as the learning rate or step-size reduction.
- Default value Shrinkage = 0.001

Shrinkage

Initialize $\hat{f}(\mathbf{x})$ to be a constant, $\hat{f}(\mathbf{x}) = \arg\min_{\rho} \sum_{i=1}^{N} \Psi(y_i, \rho)$.
For t in $1, \ldots, T$ do

1. Compute the negative gradient as the working response

$$z_i = -\frac{\partial}{\partial f(\mathbf{x}_i)} \Psi(y_i, f(\mathbf{x}_i)) \Big|_{f(\mathbf{x}_i)=\hat{f}(\mathbf{x}_i)} \tag{1}$$

2. Fit a regression model, $g(\mathbf{x})$, predicting z_i from the covariates $\mathbf{x}_i$.
3. Choose a gradient descent step size as

$$\rho = \arg\min_{\rho} \sum_{i=1}^{N} \Psi(y_i, \hat{f}(\mathbf{x}_i) + \rho g(\mathbf{x}_i)) \tag{2}$$

4. Update the estimate of $f(\mathbf{x})$ as

$$\hat{f}(\mathbf{x}) \leftarrow \hat{f}(\mathbf{x}) + \rho g(\mathbf{x}) \tag{3}$$

- By multiplying the gradient step by a fraction(shrinkage) we have control on the rate at which the boosting algorithm descends the error surface (or ascends the likelihood surface).
- When rate= 1 we return to performing full gradient steps.
- Friedman (2001) relates the learning rate to regularization through shrinkage.

Learning rate

Shrinkage - Illustration

Data	1	2	3	4	5	6	7	8	9	10
Class	-	-	+	-	-	-	-	-	-	+
Predicted Class M1	-	-	-	-	-	-	-	-	-	+
M1 Result	✔	✔	✘	✔	✔	✔	✔	✔	✔	✔

Weighted Sample1	3	3	3	3	3	3	3	3	9	10
Class	+	+	+	+	+	+	+	+	-	+
Predicted Class M2	+	+	+	+	+	+	+	+	-	+
M2 Result	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔

- Strictly picking wrongly classified examples might lead to overfitting. We introduce shrinkage as a regularization factor to reduce the weight of each model

Shrinkage

- Being conservative while picking up new sample, instead of strictly picking just the misclassified observations we would like to pickup a good portion of rightly classified observations as well to capture the right patterns
- A high shrinkage value gives high weight to each step/tree.
- A high shrinkage value makes the overall algorithm exit faster – Underfitting till a step and overfitting right after that
- There is a trade-off between learning rate and n.tree
- Even if the number of trees is set to a high number, if the shrinkage/learning rate is high then iterations after a limit will have no impact

Shrinkage - Illustration

```
> #Shrinkage Example
> n=1000
> gm<-gbm(Buy~., data=train1,
+           distribution="bernoulli",
+           verbose=T,
+           interaction.depth = 2 ,
+           n.trees = n,
+           n.minobsinnode=5,
+           bag.fraction=1,set.seed(125),
+           shrinkage = 1)
Iter   TrainDeviance   ValidDeviance   StepSize   Improve
     1        0.5465             nan     1.0000       nan
     2      263.5578             nan     1.0000       nan
     3           inf             nan     1.0000       nan
     4           inf             nan     1.0000       nan
     5           inf             nan     1.0000       nan
     6           inf             nan     1.0000       nan
     7           inf             nan     1.0000       nan
     8           inf             nan     1.0000       nan
     9           inf             nan     1.0000       nan
    10           inf             nan     1.0000       nan
    20           inf             nan     1.0000       nan
    40           inf             nan     1.0000       nan
    60           inf             nan     1.0000       nan
    80           inf             nan     1.0000       nan
   100           inf             nan     1.0000       nan
   120           inf             nan     1.0000       nan
   140           inf             nan     1.0000       nan
   160           inf             nan     1.0000       nan
   180           inf             nan     1.0000       nan
```

High shrinkage will lead to early exit of the algorithm. There is a risk of underfitting

The whole algorithm converged in just two trees

Shrinkage - Illustration

```
> #Accuracy on Training and test data
> conf_matrix<-confusionMatrix(ifelse(predict(gm,  n.trees = n, type="response")<0.2,0,1),train1$Buy)
> conf_matrix$overall[1]
Accuracy
0.934302
> conf_matrix$byClass[2]
Specificity
  0.4457831
>
> conf_matrix<-confusionMatrix(ifelse(predict(gm,  n.trees = n, newdata=test1["Age"], type="response")<0.2,0,1),test
1$Buy)
> conf_matrix$overall[1]
 Accuracy
0.9318182
> conf_matrix$byClass[2]
Specificity
  0.3478261
```

The early exit has lead to huge specificity error in both training and test data

Shrinkage tuning

- If shrinkage is high then algorithm exists early. There is a risk of underfitting
- If shrinkage is low then algorithm will need lot of iterations. If number of tree is less then it might also lead to underfitting
- If shrinkage is less then ntree should be high.
- Having a high ntree might take lot of execution time.
- What is the optimal shrinkage and ntree combination?

Shrinkage - Illustration

```
######### Shrinkage Finetuning

shrink_ntree <-data.frame()
shrink_ntree1 <-data.frame()

n=0
s=0
j=1

for(n in c(10,50, 100, 300, 500, 700, 1000)){
  for(s in c(0.001,0.002, 0.005, 0.01, 0.05, 0.07, 0.1, 0.3, 0.6, 0.8, 1)){
    gm<-gbm(Buy~., data=train1,
        distribution="bernoulli",
        verbose=T,
        interaction.depth = 2 ,
        n.trees = n,
        n.minobsinnode=5,
        bag.fraction=1,set.seed(125),
        shrinkage = s)

#Accuracy on Training and test data
conf_matrix<-confusionMatrix(ifelse(predict(gm,  n.trees = n, type="response")<0.2,0,1),train1$Buy)
shrink_ntree[i,j]=conf_matrix$byClass[2]

#conf_matrix<-confusionMatrix(ifelse(predict(gm,  n.trees = n, newdata=test1["Age"], type="response")<0.2,0,1
#shrink_ntree1[i,j]=conf_matrix$byClass[2]
i=i+1
  }
  i=1
  j=j+1
}
```

User defined function for shrinkage vs n-tree fine tuning

Shrinkage - Illustration

- Specificity Results of Above Function

```
for(n in c(10,50, 100, 300, 500, 700, 1000)){
  for(s in c(0.001,0.002, 0.005, 0.01, 0.05, 0.07, 0.1, 0.3, 0.6, 0.8, 1)){
```

n

s

The optimal combination of ntree and shrinkage

```
> shrink_ntree
              10         50        100        300        500        700       1000
0.001 0.00000000 0.00000000 0.00000000 0.13793103 0.31034483 0.57471264 0.57471264
0.002 0.00000000 0.00000000 0.00000000 0.54022989 0.57471264 0.57471264 0.57471264
0.005 0.00000000 0.13793103 0.31034483 0.57471264 0.57471264 0.57471264 0.57471264
0.01  0.00000000 0.31034483 0.57471264 0.57471264 0.57471264 0.57471264 0.59770115
0.05  0.54022989 0.57471264 0.57471264 0.59770115 0.59770115 0.60919540 0.65517241
0.07  0.57471264 0.57471264 0.57471264 0.59770115 0.59770115 0.66666667 0.72413793
0.1   0.57471264 0.57471264 0.59770115 0.59770115 0.67816092 0.72413793 0.78160920
0.3   0.59770115 0.68965517 0.73563218 0.77011494 0.88505747 0.90804598 0.95402299
0.6   0.57471264 0.72413793 0.77011494 0.77011494 0.77011494 0.77011494 0.77011494
0.8   0.04597701 0.04597701 0.04597701 0.04597701 0.04597701 0.04597701 0.04597701
1     0.43678161 0.43678161 0.43678161 0.43678161 0.43678161 0.43678161 0.43678161
```

bag.fraction

- The fraction of the training set observations randomly selected to propose the next tree in the expansion.
- According to algorithm we should pick complete sample based on the weights, but what if we pick a fraction of the data instead of complete sample
- This introduces randomness into the model fit. The errors will reduce by introducing randomness and averaging out(the spirit of ensemble method)
- Use set.seed to ensure that the model can be reconstructed as it is

bag.fraction

- The remaining fraction helps in validation
- The out of bag also used in internal cross validation
- It also helps identifying the number of iterations
- Default value is bag.fraction = 0.5

train.fraction

- The first train.fraction * nrows(data) observations are used to fit the gbm
- The remainder are used for computing out-of-sample estimates of the loss function.
- If you do-not choose train.fraction then **ValidDeviance** will be **missing** from output
- Note that if the data are sorted in a systematic way (such as cases for which y = 1 come first), then the data should be shuffled before running gbm.
- Do not confuse this with bag.fraction parameter. Bag fraction is for randomly picking up samples in each step.
- This parameter can also be used for deciding the right number of trees for a given learning rate

train.fraction- illustration

```
> train <- read.csv("./BParameters Data/Ecom_Cust_Relationship_Management/Ecom_Cust_Survey.csv")
>
> ###80% traiing and rest 20% testing
> sample_index<-sample(1:nrow(train),nrow(train)*0.8)
> train1 <- train[sample_index, ]
> test1<- train[-sample_index, ]
>
> n=1000
> gm<-gbm(Overall_Satisfaction~., data=train1,
+         distribution="bernoulli",
+         verbose=T,
+         interaction.depth = 2 ,
+         n.trees = n,
+         n.minobsinnode=5,
+         bag.fraction=1,set.seed(125),
+         shrinkage = 0.1,
+         train.fraction=0.5)
Iter   TrainDeviance   ValidDeviance   StepSize   Improve
     1        1.2569          1.2542     0.1000       nan
     2        1.1549          1.1522     0.1000       nan
     3        1.0707          1.0681     0.1000       nan
     4        1.0007          0.9979     0.1000       nan
     5        0.9402          0.9390     0.1000       nan
     6        0.8893          0.8904     0.1000       nan
     7        0.8449          0.8472     0.1000       nan
     8        0.8061          0.8083     0.1000       nan
     9        0.7731          0.7771     0.1000       nan
    10        0.7453          0.7504     0.1000       nan
    20        0.5848          0.5980     0.1000       nan
    40        0.5098          0.5328     0.1000       nan
    60        0.4983          0.5266     0.1000       nan
    80        0.4933          0.5258     0.1000       nan
   100        0.4888          0.5253     0.1000       nan
   120        0.4840          0.5249     0.1000       nan
   140        0.4806          0.5241     0.1000       nan
   160        0.4774          0.5240     0.1000       nan
   180        0.4748          0.5244     0.1000       nan
```

```
> #Accuracy on Training and test data
> conf_matrix<-confusionMatrix(ifelse(predict(gm,
n)
> conf_matrix$overall[1]
 Accuracy
0.9299026
```

Train fraction gives valid deviance

train.fraction- illustration

```
> train1<-train[order(train$Overall_Satisfaction),]
>
> n=1000
> gm<-gbm(Overall_Satisfaction~., data=train1,
+         distribution="bernoulli",
+         verbose=T,
+         interaction.depth = 2 ,
+         n.trees = n,
+         n.minobsinnode=5,
+         bag.fraction=1,set.seed(125),
+         shrinkage = 0.1,
+         train.fraction=0.5)
Iter   TrainDeviance   ValidDeviance   StepSize   Improve
     1           nan             nan     0.1000       nan
     2           nan             nan     0.1000       nan
     3           nan             nan     0.1000       nan
     4           nan             nan     0.1000       nan
     5           nan             nan     0.1000       nan
     6           nan             nan     0.1000       nan
     7           nan             nan     0.1000       nan
     8           nan             nan     0.1000       nan
     9           nan             nan     0.1000       nan
    10           nan             nan     0.1000       nan
    20           nan             nan     0.1000       nan
    40           nan             nan     0.1000       nan
    60           nan             nan     0.1000       nan
    80           nan             nan     0.1000       nan
   100           nan             nan     0.1000       nan
   120           nan             nan     0.1000       nan
   140           nan             nan     0.1000       nan
   160           nan             nan     0.1000       nan
```

- The same model on ordered data.

```
> conf_matrix$overall[1]
Accuracy
 0.53723
```

You need to shuffle the data. Train fraction considers strictly initial few records

Finding the right number of iterations.

- There are two ways
  - Using the bag fraction and calculating OOB. This doesn't work if bag fraction =1
  - Using test data and validation error. This doesn't work if train.fraction is not set

```
best.iter <- gbm.perf(gm,method="OOB")
print(best.iter)
```

```
> print(best.iter)
[1] 47
```

```
best.iter <- gbm.perf(gm,method="test")
print(best.iter)
```

```
> print(best.iter)
[1] 138
```

OOB generally underestimates the optimal number of iterations

statinfer

Variable Importance

VarImp in GBM

- VarImp in random forest uses permuting technique.
  - Shuffle the variable values and see the drop in the OOB sample. Average out on all trees
- VarImp in GBM uses a different technique.
  - VarImp in GBM is based on relative importance. The measures are based on the number of times a variable is selected for splitting in each tree.
  - The measure also considers the empirical squared improvement by the split - I^2. Below is the formula for influence
  - To obtain the overall influence we need to average out on all trees

$$\text{Influence}_j(T) = \sum_{i=1}^{L-1} I_i^2 1(S_i = j)$$

$$\text{Influence}_j = \frac{1}{M} \sum_{i=1}^{M} \text{Influence}_j(T_i)$$

VarImp in GBM

```
> summary(gm,n.trees=1)          # based on the first tree
                              var  rel.inf
Region                     Region 74.44318
Improvement.Area Improvement.Area 25.55682
Age                           Age  0.00000
Order.Quantity     Order.Quantity  0.00000
Customer_Type       Customer_Type  0.00000
> summary(gm,n.trees=best.iter) # based on the estimated best number of trees
                              var   rel.inf
Region                     Region 59.517546
Improvement.Area Improvement.Area 19.646381
Order.Quantity     Order.Quantity 10.721076
Age                           Age  9.570486
Customer_Type       Customer_Type  0.544511
```

Partial dependence plots

- Partial dependence plots show us the impact of a variable on the modelled response after marginalizing out(averaging out) all other explanatory variables.
- We marginalize the rest of the variables by substituting their average value(or integrating them).
- Partial plots tell us the exact impact of x variable and its impact on Y(positive or negative) at every point of x

LAB: Boosting

- Import Credit Risk Data
- Create a balanced sample
- Build GBM tree
- What are the important variables
- Draw the partial dependency plots.

Code: Boosting

```
#Response variable
table(Credit_risk_data$SeriousDlqin2yrs)
#There is class imbalance, lets create a balanced dataset.
Credit_risk_data_1<-Credit_risk_data[Credit_risk_data$SeriousDlqin2yrs==0,]
dim(Credit_risk_data_1)

########Take a small percentage of zeros and all one
risk_20pct_zero<-Credit_risk_data_1[sample(1:nrow(Credit_risk_data_1),15000), ]
dim(risk_20pct_zero)

risk_All_one<-Credit_risk_data[Credit_risk_data$SeriousDlqin2yrs==1,]
dim(risk_All_one)

#Final  Balanced data
Credit_risk_data_bal<-rbind(risk_20pct_zero,risk_All_one)
names(Credit_risk_data_bal)
#Shuffle the 1 and 0
Credit_risk_data_bal<-
Credit_risk_data_bal[sample(1:nrow(Credit_risk_data_bal),nrow(Credit_risk_data_bal)),]
```

Prepare data

Code: Boosting

```
> gm<-gbm(SeriousDlqin2yrs~., data=Credit_risk_data_bal,
+         distribution="bernoulli",
+         verbose=T,
+         interaction.depth = 3,
+         n.trees = n,
+         n.minobsinnode=5,
+         bag.fraction=0.5,set.seed(125),
+         shrinkage = 0.07,
+         train.fraction=0.7)
Iter   TrainDeviance   ValidDeviance   StepSize   Improve
     1        1.3093          1.3099     0.0700    0.0184
     2        1.2777          1.2783     0.0700    0.0161
     3        1.2497          1.2503     0.0700    0.0140
     4        1.2254          1.2261     0.0700    0.0121
     5        1.2041          1.2048     0.0700    0.0106
     6        1.1858          1.1866     0.0700    0.0092
     7        1.1705          1.1714     0.0700    0.0074
     8        1.1557          1.1567     0.0700    0.0073
     9        1.1428          1.1438     0.0700    0.0063
    10        1.1316          1.1327     0.0700    0.0053
    20        1.0579          1.0603     0.0700    0.0024
    40        1.0099          1.0143     0.0700    0.0009
```

Build GBM Model

Code: Boosting

```
> conf_matrix$overall[1]
 Accuracy
0.7764325
> # plot the performance # plot variable influence
> summary(gm,n.trees=n) # based on the estimated best number of trees
                                                        var    rel.inf
util                                                   util 55.1732188
NumberOfTime30_59DaysPastDue1 NumberOfTime30_59DaysPastDue1 24.8347868
age1                                                   age1  6.2202778
DebtRatio1                                       DebtRatio1  5.5686886
MonthlyIncome1                                 MonthlyIncome1  4.6460472
num_loans                                           num_loans  2.9715351
depend                                                 depend  0.5854457
```

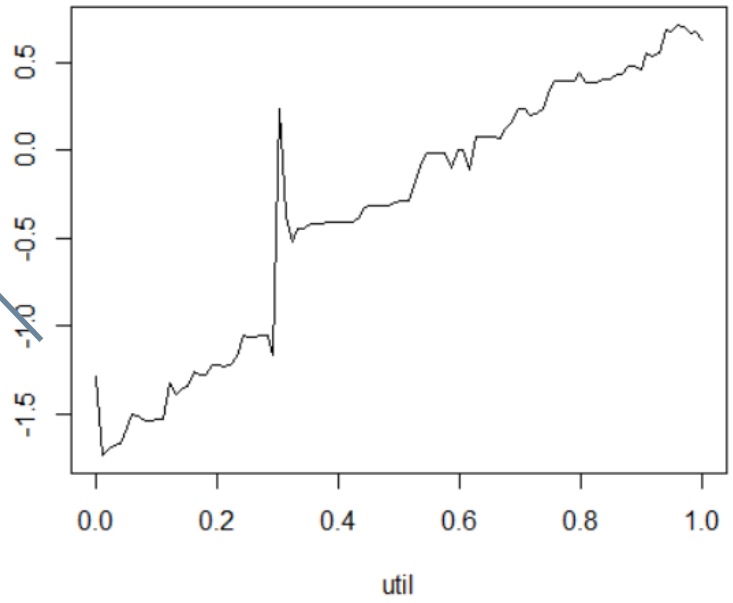
Accuracy and VarImp

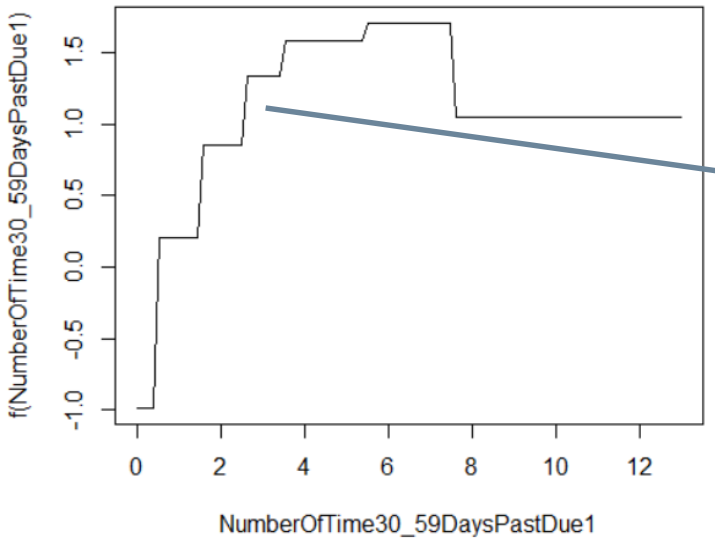
Code: Boosting

```
> ###Partial dependence plots
> names(Credit_risk_data_bal)
[1] "SeriousDlqin2yrs"                "NumberOfTime30_59DaysPastDue1" "util"
[4] "age1"                            "DebtRatio1"                    "MonthlyIncome1"
[7] "num_loans"                       "depend"
> plot.gbm(gm,i.var =2 , lwd = 2, col = "red")
> plot.gbm(gm,i.var =1 , lwd = 2, col = "red")
>
```

Partial Dependency plots

Negative influence on class 1





Influence on class 1 at different values of X

statinfer

Case Study: Direct Mail Marketing Response Model

LAB: Direct Mail Marketing Response Model

statinfer

- Large Marketing Response Data/train.csv
- How many variables are there in the dataset?
- Take a one third of the data as training data and one third as test data
- Look at the response rate from target variables
- Find out the overall missing values and missing values by variables
- Do the missing value and outlier treatment, prepare data for analysis
- Build a boosting model
- Find the training data accuracy
- Find the accuracy on test data

Code: Direct Mail Marketing Response Model

```
#####Boosting model
n=100
gbm_m1<-gbm(target~., data=train[,-1],
        distribution="bernoulli",
        verbose=T,
        interaction.depth = 2 ,
        n.trees = n,
        n.minobsinnode=5,
        bag.fraction=0.5,set.seed(125),
        shrinkage = 0.1,
        train.fraction = 0.5)

# plot the performance # plot variable influence
imp_var<-summary(gbm_m1,n.trees=1)   # based on the first tree
imp_var[1:10,]

imp_var<-summary(gbm_m1,n.trees=best.iter) # based on the estimated best number of trees
imp_var[imp_var$rel.inf>1,]
```

Code: Direct Mail Marketing Response Model

statinfer

```
###Boosting model-2 with scale_pos_weight For Class imbalance

#Create model weights first
table(train$target)/nrow(train)
model_weights <- ifelse(train$target ==0, 0.2,0.8)
table(model_weights)

n=300
gbm_m1<-gbm(target~., data=train[,-1],
          distribution="bernoulli",
          verbose=T,
          interaction.depth = 2 ,
          n.trees = n,
          n.minobsinnode=5,
          bag.fraction=0.5,set.seed(125),
          shrinkage = 0.1,
          weights = model_weights)
```

Conclusion

Conclusion

- GBM is one of the mot widely used machine learning technique in business
- GBM is less prone to overfitting when compared to other techniques like Neural nets
- We need to be patient, GBM might take a lot of time for execusion.

statinfer

Thank you

statinfer

Appendix

statinfer

Xgboosting

statinfer

Xgboost parameters

GBM vs Xgboost

1. Extreme Gradient Boosting – XGBoost
2. Both xgboost and gbm follows the principle of gradient boosting.
3. There are however, the difference in modelling & performance details.
4. xgboost used a more regularized model formalization to control over-fitting, which gives it better performance.
5. Improved convergence techniques, vector and matrix type data structures for faster results
6. Unlike GBM XGBoost package is available in C++, Python, R, Java, Scala, Julia with same parameters for tuning

Xgboost Advantages

1. Developers of xgboost have made a number of important performance enhancements.
2. Xgboost and GBM have big difference in speed and memory utilization
3. Code modified for better processor cache utilization which makes it faster.
4. Better support for multicore processing which reduces overall training time.

GBM vs Xgboost

- Most importantly, the memory error is somewhat resolved in xgboost
- For a given dataset, you are less likely to get memory error while using xgboost when compared to GBM

Xgboost parameters

```
XGBModel <- xgboost(param=param, data = trainMatrix, label = y, nrounds=40)
param <- list( )
```

Parameters

- booster
  - which booster to use, can be gbtree for tree based models or gblinear for linear models
  - Default value is gbtree
- nthread
  - This is used for parallel processing and number of cores in the system should be entered
  - Leave this field if you are executing on your PC or Laptop. If you are using a common server then decide the best value after few trail and errors
  - If you wish to run on all cores, value should not be entered and algorithm will detect automatically
  - default value is maximum number of threads available in your system

Xgboost parameters

- objective
  - default=reg:linear
  - This defines the type of problem that we are solving and the loss function to be minimized.
  - reg:logistic - logistic regression.
  - binary:logistic –logistic regression for binary classification, returns predicted probability (not class)
  - binary:logitraw logistic regression for binary classification, output score before logistic transformation.
  - multi:softmax –multiclass classification using the softmax objective
    - returns predicted class (not probabilities).
    - you also need to set an additional num_class (number of classes) parameter defining the number of unique classes. Class is represented by a number and should be from 0 to (num_class – 1).
  - multi:softprob –same as softmax, but returns predicted probability of each data point belonging to each class.
  - Note: num_class set the number of classes. To use only with multiclass objectives.

Xgboost parameters

- eval_metric
  - This depends on objective function that we set above
  - Default: metric will be assigned according to objective
    - rmse for regression
    - error for classification
    - mean average precision for ranking
  - The Options are
    - rmse – root mean square error
    - mae – mean absolute error
    - logloss – negative log-likelihood
    - error – Binary classification error rate (0.5 threshold)
    - merror – Multiclass classification error rate
    - mlogloss – Multiclass logloss
    - auc: Area under the curve

Xgboost parameters

- eta
  - eta controls the learning rate, Analogous to learning rate in GBM
  - As you know boosting uses ensemble algorithm. Larger number models will ensure more robust model.
  - Right after every boosting step you have each feature and their weights, eta shrinks the feature weights the boosting process more conservative.
  - Instead of considering each model as it is, we will give it less weight. We use eta to reduce the contribution of each tree
  - If eta is low then we will have larger value for nrounds.
  - low eta value means model more robust to overfitting but slower to compute
  - eta scales the contribution of each tree by a factor of 0 < eta < 1 when it is added to the current approximation.
  - Default value is 0.3. Typical final values to be used: 0.01-0.2

More on eta

- For example you need 10 rounds for a best model
- The learning rate is the shrinkage you do at every step you are making.
- If eta is 1.00, the step weight is 1.00. You will end up with 10 rounds only.
- If you make 1 step at eta = 0.25, the step weight is 0.25. you will end up in 40 steps
- If you decrease learning rate, you need to increase number of iterations in same proportion.

Xgboost parameters

- min_child_weight [default=1]
  - min_child_weight minimum sum of instance weight (hessian) needed in a child. If the tree partition step results in a leaf node with the sum of instance weight less than min_child_weight, then the building process will give up further partitioning.
  - In simple terms you can see this minimum number of instances needed to be in each node.
  - This is similar to min_child_leaf in GBM with a slght change. Here it refers to min "sum of weights" of observations while GBM has min "number of observations".
  - Used to control over-fitting. Higher values prevent a model from learning relations which might be highly specific to the particular sample selected for a tree.
  - Too high values can lead to under-fitting hence, it should be tuned using CV.
  - Default value is : 1
  - Try to tune it by looking at results of CV.

Xgboost parameters

- max_depth
  - The maximum depth of a tree, same as GBM.
  - Used to control over-fitting as higher depth will allow model to learn relations very specific to a particular sample.
  - Default value is 6
  - Try to tune it by looking at results of CV. Typical values:5-10
- gamma
  - A node is split only when the resulting split gives a positive reduction in the loss function. Gamma specifies the minimum loss reduction required to make a split.
  - Makes the algorithm conservative and reduces overfitting. The values can vary depending on the loss function and should be tuned.
  - A larger value of gamma reduces runtime.
  - Try to tune it by looking at results of CV. Start with very low values
  - Leave it as it is if you have no idea on the data
  - Default value is 0

Xgboost parameters

- subsample
  - Same as the subsample of GBM.
  - subsample ratio of the training instance. Denotes the fraction of observations to be randomly samples for each tree.
  - Setting it to 0.5 means that xgboost randomly collected half of the data instances to grow trees and this will prevent overfitting.
  - It makes computation shorter (because less data to analyse).
  - It is advised to use this parameter with eta and increase nround.
  - Lower values make the algorithm more conservative(less accurate) and prevents overfitting but too small values might lead to under-fitting.
  - Default value is 1 - gathers complete data
  - Typical values: 0.2-0.8 depending on size of data

Xgboost parameters

- scale_pos_weight, [default=1]
  - Control the balance of positive and negative weights
  - useful for unbalanced classes.
  - A typical value to consider: sum(negative cases) / sum(positive cases)

Xgboost parameters

- colsample_bytree
  - subsample ratio of columns when constructing each tree.
  - Similar to max_features in GBM. Denotes the fraction of columns to be randomly samples for each tree.
  - Typical values: 0.5-1
  - Default: 1
- num_parallel_tree
  - Experimental parameter, ignore it for now.
  - number of trees to grow per round.
  - Useful to test Random Forest through Xgboost (set colsample_bytree < 1, subsample < 1 and round = 1) accordingly.
  - Default: 1

LAB: Boosting

- Ecom products classification. Rightly categorizing the items based on their detailed feature specifications. More than 100 specifications have been collected.
- Data: Ecom_Products_Menu/train.csv
- Build a decision tree model and check the training and testing accuracy
- Build a boosted decision tree.
- Is there any improvement from the earlier decision tree

Code: Boosting

```
> train <- read.csv("D:/Google Drive/Training/Datasets/Ecom_Products_Menu/train.csv")
> test <- read.csv("D:/Google Drive/Training/Datasets/Ecom_Products_Menu/test.csv")
>
> #Dataset details
> dim(train)
[1] 50122   102
> dim(test)
[1] 11756   102
```

```
> ##Decison Tree
> library(rpart)
> ecom_products_ds<-rpart(Category ~ ., method="class", control=rpart.control(minsplit=30, cp=0.01),  data=train[,-1])
> #Training accuarcy
> library(caret)
> predicted_y<-predict(ecom_products_ds, type="class")
> table(predicted_y)
predicted_y
  Accessories    Appliances        Camara          Ipod       Laptops       Mobiles Personal_Care       Tablets
            0         10899          2733          2442             0             0         10288         23760
           TV
            0
```

Code: Boosting

```
> confusionMatrix(predicted_y,train$Category)
Confusion Matrix and Statistics

                Reference
Prediction       Accessories Appliances Camara  Ipod Laptops Mobiles Personal_Care Tablets    TV
  Accessories              0          0      0     0       0       0             0       0     0
  Appliances             825       5536   1086   130     506     709          1035     932   140
  Camara                  88        387   1456     4      55     388           252      84    19
  Ipod                    30         17     23  2032     144       5            13     159    19
  Laptops                  0          0      0     0       0       0             0       0     0
  Mobiles                  0          0      0     0       0       0             0       0     0
  Personal_Care          110        308    152     0      18      79          9545      19    57
  Tablets               1288        615   1247    51    5743     377           607   11885  1947
  TV                       0          0      0     0       0       0             0       0     0

Overall Statistics

               Accuracy : 0.6076
                 95% CI : (0.6033, 0.6119)
    No Information Rate : 0.2609
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.5053
 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: Accessories Class: Appliances Class: Camara Class: Ipod Class: Laptops Class: Mobiles Class: Personal_Care Class: Tablets Class: TV
Sensitivity                     0.00000            0.8066       0.36731     0.91655          0.000        0.00000               0.8335         0.9087   0.00000
Specificity                     1.00000            0.8760       0.97233     0.99144          1.000        1.00000               0.9808         0.6794   1.00000
Pos Pred Value                      NaN            0.5079       0.53275     0.83210            NaN            NaN               0.9278         0.5002       NaN
Neg Pred Value                  0.95329            0.9662       0.94708     0.99612          0.871        0.96892               0.9521         0.9547   0.95647
Prevalence                      0.04671            0.1369       0.07909     0.04423          0.129        0.03108               0.2285         0.2609   0.04353
Detection Rate                  0.00000            0.1105       0.02905     0.04054          0.000        0.00000               0.1904         0.2371   0.00000
Detection Prevalence            0.00000            0.2174       0.05453     0.04872          0.000        0.00000               0.2053         0.4740   0.00000
Balanced Accuracy               0.50000            0.8413       0.66982     0.95400          0.500        0.50000               0.9071         0.7941   0.50000
```

Code: Boosting

```
> #Accuarcy on Test data
> predicted_test_ds<-predict(ecom_products_ds, test[,-1], type="class")
> confusionMatrix(predicted_test_ds,test$Category)
Confusion Matrix and Statistics

                Reference
Prediction       Accessories Appliances Camara Ipod Laptops Mobiles Personal_Care Tablets   TV
  Accessories              0          0      0    0       0       0             0       0    0
  Appliances             172       1308    269   40      92     170           234     210   42
  Camara                  15         80    383    1      16      95            52      23    3
  Ipod                    14          4      3  469      28       0             3      49    5
  Laptops                  0          0      0    0       0       0             0       0    0
  Mobiles                  0          0      0    0       0       0             0       0    0
  Personal_Care           23         75     42    0       1      23          2242      10   17
  Tablets                274        134    294   12    1401      83           152    2751  442
  TV                       0          0      0    0       0       0             0       0    0

Overall Statistics

               Accuracy : 0.6085
                 95% CI : (0.5996, 0.6173)
    No Information Rate : 0.2588
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.5071
 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: Accessories Class: Appliances Class: Camara Class: Ipod Class: Laptops Class: Mobiles Class: Personal_Care Class: Tablets Class: TV
Sensitivity                     0.00000            0.8170       0.38648     0.89847         0.0000        0.00000               0.8356         0.9040    0.0000
Specificity                     1.00000            0.8790       0.97353     0.99056         1.0000        1.00000               0.9789         0.6796    1.0000
Pos Pred Value                      NaN            0.5156       0.57335     0.81565            NaN            NaN               0.9215         0.4963       NaN
Neg Pred Value                  0.95764            0.9682       0.94517     0.99526         0.8692        0.96844               0.9527         0.9530    0.9567
Prevalence                      0.04236            0.1362       0.08430     0.04440         0.1308        0.03156               0.2282         0.2588    0.0433
Detection Rate                  0.00000            0.1113       0.03258     0.03989         0.0000        0.00000               0.1907         0.2340    0.0000
Detection Prevalence            0.00000            0.2158       0.05682     0.04891         0.0000        0.00000               0.2070         0.4715    0.0000
Balanced Accuracy               0.50000            0.8480       0.68000     0.94452         0.5000        0.50000               0.9073         0.7918    0.5000
```

Code: Boosting

```
library(methods)
library(data.table)
library(magrittr)
```

```
> # converting datasets to Numeric format. xgboost needs at least one numeric column
> train[,c(-1,-102)] <- lapply( train[,c(-1,-102)], as.numeric)
> test[,c(-1,-102)] <- lapply( test[,c(-1,-102)], as.numeric)
> # converting datasets to Matrix format. Data frame is not supported by xgboost
> trainMatrix <- train[,c(-1,-102)] %>% as.matrix
> testMatrix <- test[,c(-1,-102)] %>% as.matrix
```

Code: Boosting

```
> #The label should be in numeric format and it should start from 0
> y<-as.integer(train$Category)-1
> table(y,train$Category)

y   Accessories Appliances Camara  Ipod Laptops Mobiles Personal_Care Tablets    TV
  0        2341          0      0     0       0       0             0       0     0
  1           0       6863      0     0       0       0             0       0     0
  2           0          0   3964     0       0       0             0       0     0
  3           0          0      0  2217       0       0             0       0     0
  4           0          0      0     0    6466       0             0       0     0
  5           0          0      0     0       0    1558             0       0     0
  6           0          0      0     0       0       0         11452       0     0
  7           0          0      0     0       0       0             0   13079     0
  8           0          0      0     0       0       0             0       0  2182
```

Code: Boosting

```
> test_y<-as.integer(test$Category)-1
> table(test_y,test$Category)

test_y Accessories Appliances Camara Ipod Laptops Mobiles Personal_Care Tablets   TV
     0         498          0      0    0       0       0             0       0    0
     1           0       1601      0    0       0       0             0       0    0
     2           0          0    991    0       0       0             0       0    0
     3           0          0      0  522       0       0             0       0    0
     4           0          0      0    0    1538       0             0       0    0
     5           0          0      0    0       0     371             0       0    0
     6           0          0      0    0       0       0          2683       0    0
     7           0          0      0    0       0       0             0    3043    0
     8           0          0      0    0       0       0             0       0  509
```

Code: Boosting

```
> #Setting the parameters for multiclass classification
> param <- list("objective" = "multi:softprob","eval.metric" = "merror",   "num_class" =9)
> #"multi:softmax" --set XGBoost to do multiclass classification using the softmax objective,
> #you also need to set num_class(number of classes)
> #"merror": Multiclass classification error rate. It is calculated as #(wrong cases)/#(all cases).
>
> library(xgboost)
> XGBModel <- xgboost(param=param, data = trainMatrix, label = y, nrounds=50)
[0]     train-merror:0.269223
[1]     train-merror:0.241750
[2]     train-merror:0.229500
[3]     train-merror:0.222776
[4]     train-merror:0.218966
[5]     train-merror:0.211923
[6]     train-merror:0.208312
[7]     train-merror:0.203703
[8]     train-merror:0.199553
[9]     train-merror:0.196481
[10]    train-merror:0.192969
[11]    train-merror:0.190695
[12]    train-merror:0.188241
[13]    train-merror:0.185487
[14]    train-merror:0.183193
[15]    train-merror:0.180400
[16]    train-merror:0.177886
[17]    train-merror:0.175552
[18]    train-merror:0.173217
```

Code: Boosting

```
> #Training accuarcy
> predicted_y<-predict(XGBModel, trainMatrix)
> probs <- data.frame(matrix(predicted_y, nrow=nrow(train), ncol=9,  byrow = TRUE))
> probs_final<-as.data.frame(cbind(row.names(probs),apply(probs,1, function(x) c(0:8)[which(x==max(x))])))
> table(probs_final$V2)

    0     1     2     3     4     5     6     7     8 
 2152  6949  3980  2225  5317  1298 11426 15326  1449 
```

Code: Boosting

```
> confusionMatrix(probs_final$V2,y)
Confusion Matrix and Statistics

          Reference
Prediction     0     1     2     3     4     5     6     7     8
         0  1870    28    13     2    68    26    77    52    16
         1    64  6517   114     1    13   113   109    14     4
         2     8    73  3625     2     3   175    87     7     0
         3     5     4     1  2197     0     1     0     8     9
         4    78    15     4     1  4052     3    12   942   210
         5    27    54    56     2     2  1115    35     7     0
         6    68   100    77     1     5    81 11047    19    28
         7   205    72    72    11  2282    43    79 11972   590
         8    16     0     2     0    41     1     6    58  1325

Overall Statistics

               Accuracy : 0.8723
                 95% CI : (0.8693, 0.8752)
    No Information Rate : 0.2609
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.8448
 Mcnemar's Test P-Value : < 2.2e-16

Statistics by Class:

                     Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6 Class: 7 Class: 8
Sensitivity           0.79880   0.9496  0.91448  0.99098  0.62666  0.71566   0.9646   0.9154  0.60724
Specificity           0.99410   0.9900  0.99231  0.99942  0.97102  0.99623   0.9902   0.9095  0.99741
Pos Pred Value        0.86896   0.9378  0.91080  0.98742  0.76208  0.85901   0.9668   0.7812  0.91442
Neg Pred Value        0.99018   0.9920  0.99265  0.99958  0.94612  0.99093   0.9895   0.9682  0.98239
Prevalence            0.04671   0.1369  0.07909  0.04423  0.12901  0.03108   0.2285   0.2609  0.04353
Detection Rate        0.03731   0.1300  0.07232  0.04383  0.08084  0.02225   0.2204   0.2389  0.02644
Detection Prevalence  0.04294   0.1386  0.07941  0.04439  0.10608  0.02590   0.2280   0.3058  0.02891
Balanced Accuracy     0.89645   0.9698  0.95339  0.99520  0.79884  0.85595   0.9774   0.9124  0.80233
```

Code: Boosting

```
> #Accuarcy on Test data
>
> predicted_test_boost<-predict(XGBModel, testMatrix)
> probs_test <- data.frame(matrix(predicted_test_boost, nrow=nrow(test), ncol=9,  byrow = TRUE))
>
> probs_final_test<-as.data.frame(cbind(row.names(probs_test),apply(probs_test,1, function(x) c(0:8)[which(x==max(x))])))
> table(probs_final_test$V2)

   0    1    2    3    4    5    6    7    8
 447 1645 1042  517 1217  238 2695 3672  283
```

Code: Boosting

```
> confusionMatrix(probs_final_test$V2,test_y)
Confusion Matrix and Statistics

          Reference
Prediction    0    1    2    3    4    5    6    7    8
         0  327   15    2    1   30    6   38   22    6
         1   27 1476   34    1    4   60   34    8    1
         2    1   29  885    0    4   80   34    9    0
         3    1    1    1  503    0    1    2    6    2
         4   29    6    2    1  751    4    2  348   74
         5   11   21   20    0    0  173   12    0    1
         6   39   35   31    0    1   32 2531    6   20
         7   55   18   16   14  719   15   25 2618  192
         8    8    0    0    2   29    0    5   26  213

Overall Statistics

               Accuracy : 0.8061
                 95% CI : (0.7989, 0.8133)
    No Information Rate : 0.2588
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.764
 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6 Class: 7 Class: 8
Sensitivity           0.65663   0.9219  0.89304  0.96360  0.48830  0.46631   0.9433   0.8603  0.41847
Specificity           0.98934   0.9834  0.98542  0.99875  0.95439  0.99429   0.9819   0.8790  0.99378
Pos Pred Value        0.73154   0.8973  0.84933  0.97292  0.61709  0.72689   0.9391   0.7130  0.75265
Neg Pred Value        0.98488   0.9876  0.99011  0.99831  0.92532  0.98281   0.9832   0.9474  0.97420
Prevalence            0.04236   0.1362  0.08430  0.04440  0.13083  0.03156   0.2282   0.2588  0.04330
Detection Rate        0.02782   0.1256  0.07528  0.04279  0.06388  0.01472   0.2153   0.2227  0.01812
Detection Prevalence  0.03802   0.1399  0.08864  0.04398  0.10352  0.02024   0.2292   0.3124  0.02407
Balanced Accuracy     0.82298   0.9526  0.93923  0.98118  0.72135  0.73030   0.9626   0.8697  0.70612
```

statinfer

Case Study: Direct Mail Marketing Response Model

LAB: Direct Mail Marketing Response Model

statinfer

- Large Marketing Response Data/train.csv
- How many variables are there in the dataset?
- Take a one third of the data as training data and one third as test data
- Look at the response rate from target variables
- Find out the overall missing values and missing values by variables
- Do the missing value and outlier treatment, prepare data for analysis
- Build a boosting model
- Find the training data accuracy
- Find the accuracy on test data

Code: Direct Mail Marketing Response Model

statinfer

Note: This code is to create initial benchmark model only. You need to spend some time and finetune it to create the final model

```
library(xgboost)

train_all <- read.csv("D:/3. Big Data D/2.BigDataSets/Springleaf Data/train.csv")
dim(train_all)

train <- train_all[sample(1:nrow(train_all),nrow(train_all)/3, replace=F, set.seed(55)), ]
dim(train)

test <- train_all[sample(1:nrow(train_all),nrow(train_all)/3, replace=F, set.seed(75)), ]
dim(test)
```

Code: Direct Mail Marketing Response Model

```
> #Response Variable freq and proportion
> table(train$target)

    0     1
37177 11233
> table(test$target)

    0     1
37286 11124
>
> table(train$target)/nrow(train)

        0         1
0.7679612 0.2320388
> table(test$target)/nrow(test)

        0         1
0.7702128 0.2297872
```

Code: Direct Mail Marketing Response Model

```
#Take all the variables in one vector
variable_names <- names(train)[2:(ncol(train)-1)]
variable_names

# If there are any charecter variables, we will convert them to factors
# For fatser computation  and easy interpratation

for (f in variable_names) {
  if (class(train[[f]])=="character") {
    levels <- unique(c(train[[f]], test[[f]]))
    train[[f]] <- as.integer(factor(train[[f]], levels=levels))
    test[[f]]  <- as.integer(factor(test[[f]],  levels=levels))
  }
}
```

Code: Direct Mail Marketing Response Model

```
> #Are there any missing values?
> sum(is.na(train))
[1] 531893
> sum(is.na(test))
[1] 535610
>
> #Missing values for each variable
> i=1
> Miss_val_by_var <- data.frame(var_name=0, miss_val_count=0)
>
> for (f in variable_names) {
+   Miss_val_by_var[i,1]= f
+   Miss_val_by_var[i,2]= sum(is.na(train[[f]]))
+   i=i+1
+ }
```

Code: Direct Mail Marketing Response Model

```
> head(Miss_val_by_var[order(-Miss_val_by_var$miss_val_count),], n=20)
    var_name miss_val_count
208 VAR_0207          48410
214 VAR_0213          48410
839 VAR_0840          48410
206 VAR_0205          47667
207 VAR_0206          47621
210 VAR_0209          45314
209 VAR_0208          41807
211 VAR_0210          41807
212 VAR_0211          41807
75  VAR_0074          33654
213 VAR_0212           4214
349 VAR_0350            288
241 VAR_0242            284
242 VAR_0243            284
243 VAR_0244            284
244 VAR_0245            284
245 VAR_0246            284
246 VAR_0247            284
247 VAR_0248            284
248 VAR_0249            284
```

```
> #Verification
> sum(is.na(train))
[1] 531893
> sum(Miss_val_by_var$miss_val_count)
[1] 531893
```

Code: Direct Mail Marketing Response Model

```
> #Percentiles of missing value
> quantile(Miss_val_by_var$miss_val_count, c(0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1))
   0%   10%   20%   30%   40%   50%   60%   70%   80%   90%  100%
    0     0     0     0     0     0     0     0    14   284 48410
> quantile(Miss_val_by_var$miss_val_count, c(0, 0.1, 0.25,  0.5, 0.75, 0.8, 0.9,0.93, 0.95, 0.97, 0.98, 0.99, 1))
   0%   10%   25%   50%   75%   80%   90%   93%   95%   97%   98%   99%  100%
    0     0     0     0     0    14   284   284   284   284   284   284 48410
>
> #Percentage missing vs divide by nrow to know the percentage missing
> quantile(Miss_val_by_var$miss_val_count, c(0, 0.1, 0.25,  0.5, 0.75, 0.8, 0.9,0.93, 0.95, 0.97, 0.98, 0.99, 1))/nrow(trai
n)
          0%          10%          25%          50%          75%          80%          90%          93%          95%
0.0000000000 0.0000000000 0.0000000000 0.0000000000 0.0000000000 0.0002891964 0.0058665565 0.0058665565 0.0058665565
         97%          98%          99%         100%
0.0058665565 0.0058665565 0.0058665565 1.0000000000
> #round off
> round(quantile(Miss_val_by_var$miss_val_count, c(0, 0.1, 0.25,  0.5, 0.75, 0.8, 0.9,0.93, 0.95, 0.97, 0.98, 0.99,0.993,0.
995,0.997,0.998, 0.999, 1))/nrow(train),2)
   0%   10%   25%   50%   75%   80%   90%   93%   95%   97%   98%   99% 99.3% 99.5% 99.7% 99.8% 99.9%  100%
 0.00  0.00  0.00  0.00  0.00  0.00  0.01  0.01  0.01  0.01  0.01  0.01  0.01  0.29  0.88  0.98  1.00  1.00
```

Code: Direct Mail Marketing Response Model

```
> # All missing values treatment in one go. Replace with -1
> # You should spend some more time on missing value treatment
> train[is.na(train)] <- -1
> test[is.na(test)] <- -1
> # All missing values treatment in one go. Replace with -1
> # You should spend  more time on missing value treatment
> train[is.na(train)] <- -1
> test[is.na(test)] <- -1
>
> #Are there any missing values?
> sum(is.na(train))
[1] 0
> sum(is.na(test))
[1] 0
```

Code: Direct Mail Marketing Response Model

```
> ##################################
> ###Boosting model
> xgb_m1 <- xgboost(data      = data.matrix(train[,variable_names]),
+                   label       = train$target,
+                   nrounds     = 20,
+                   objective   = "binary:logistic",
+                   eval_metric = "auc")
[1]     train-auc:0.733834
[2]     train-auc:0.755258
[3]     train-auc:0.766682
[4]     train-auc:0.775081
[5]     train-auc:0.782229
[6]     train-auc:0.788121
[7]     train-auc:0.795813
[8]     train-auc:0.802086
[9]     train-auc:0.806619
[10]    train-auc:0.812521
[11]    train-auc:0.817359
[12]    train-auc:0.821908
[13]    train-auc:0.827070
[14]    train-auc:0.832384
[15]    train-auc:0.836504
[16]    train-auc:0.841229
[17]    train-auc:0.845504
[18]    train-auc:0.848192
[19]    train-auc:0.851668
[20]    train-auc:0.855857
>
```

Code: Direct Mail Marketing Response Model

```r
##Confusion Matrix and Accuracy
###Training data
predicted_xgb_m1<-predict(xgb_m1, data.matrix(train[ ,variable_names]))
predicted_xgb_m1

predicted_xgb_m1_class<- ifelse(predicted_xgb_m1>0.5,1,0)
conf_matrix_xgb<-confusionMatrix(predicted_xgb_m1_class,train$target)
conf_matrix_xgb
```

```
> conf_matrix_xgb
Confusion Matrix and Statistics

          Reference
Prediction     0     1
         0 36135  6937
         1  1042  4296

               Accuracy : 0.8352
                 95% CI : (0.8318, 0.8385)
    No Information Rate : 0.768
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.4339
 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.9720
            Specificity : 0.3824
         Pos Pred Value : 0.8389
         Neg Pred Value : 0.8048
             Prevalence : 0.7680
         Detection Rate : 0.7464
   Detection Prevalence : 0.8897
      Balanced Accuracy : 0.6772

       'Positive' Class : 0
```

Code: Direct Mail Marketing Response Model

```
###############Testing results
predicted_xgb_m1_test<-predict(xgb_m1, data.matrix(test[ ,variable_names]))
predicted_xgb_m1_test

predicted_xgb_m1_test_class<- ifelse(predicted_xgb_m1_test>0.5,1,0)
conf_matrix_xgb_test<-confusionMatrix(predicted_xgb_m1_test_class,test$target)
conf_matrix_xgb_test
```

```
> conf_matrix_xgb_test
Confusion Matrix and Statistics

          Reference
Prediction     0     1
         0 35608  7819
         1  1678  3305

               Accuracy : 0.8038
                 95% CI : (0.8003, 0.8074)
    No Information Rate : 0.7702
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.3127
 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.9550
            Specificity : 0.2971
         Pos Pred Value : 0.8200
         Neg Pred Value : 0.6633
             Prevalence : 0.7702
         Detection Rate : 0.7356
   Detection Prevalence : 0.8971
      Balanced Accuracy : 0.6261

       'Positive' Class : 0
```

Code: Direct Mail Marketing Response Model

```
> library(xgboost)
> xgb_m2 <- xgboost(data        = data.matrix(train[,variable_names]),
+                   label       = train$target,
+                   nrounds     = 40,
+                   scale_pos_weight=(sum(train$target=='0')/sum(train$target=='1')),
+                   objective   = "binary:logistic",
+                   eval_metric = "auc")
[1]     train-auc:0.737944
[2]     train-auc:0.759104
[3]     train-auc:0.773007
[4]     train-auc:0.782394
[5]     train-auc:0.789552
[6]     train-auc:0.798137
[7]     train-auc:0.805344
[8]     train-auc:0.810235
[9]     train-auc:0.816750
[10]    train-auc:0.821868
[11]    train-auc:0.827503
[12]    train-auc:0.832045
[13]    train-auc:0.837106
[14]    train-auc:0.840396
[15]    train-auc:0.845075
[16]    train-auc:0.848911
[17]    train-auc:0.851845
[18]    train-auc:0.855318
[19]    train-auc:0.857864
[20]    train-auc:0.861492
[21]    train-auc:0.865062
[22]    train-auc:0.867910
[23]    train-auc:0.871091
[24]    train-auc:0.873334
[25]    train-auc:0.875429
[26]    train-auc:0.878475
```

Code: Direct Mail Marketing Response Model

```
> ##Confusion Matrix and Accuracy
> ###Training data
> predicted_xgb_m2<-predict(xgb_m2, data.matrix(train[ ,variable_names]))
> #predicted_xgb_m2
>
> predicted_xgb_m2_class<- ifelse(predicted_xgb_m2>0.5,1,0)
> library(caret)
> conf_matrix_xgb<-confusionMatrix(predicted_xgb_m2_class,train$target)
> conf_matrix_xgb
Confusion Matrix and Statistics

          Reference
Prediction     0     1
         0 30423  1987
         1  6754  9246

               Accuracy : 0.8194
                 95% CI : (0.816, 0.8229)
    No Information Rate : 0.768
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.5587
 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.8183
            Specificity : 0.8231
         Pos Pred Value : 0.9387
         Neg Pred Value : 0.5779
             Prevalence : 0.7680
         Detection Rate : 0.6284
   Detection Prevalence : 0.6695
      Balanced Accuracy : 0.8207

       'Positive' Class : 0
```

Code: Direct Mail Marketing Response Model

```
> ###############Testing results
> predicted_xgb_m2_test<-predict(xgb_m2, data.matrix(test[ ,variable_names]))
> #predicted_xgb_m2_test
>
> predicted_xgb_m2_test_class<- ifelse(predicted_xgb_m2_test>0.5,1,0)
> conf_matrix_xgb_test<-confusionMatrix(predicted_xgb_m2_test_class,test$target)
> conf_matrix_xgb_test
Confusion Matrix and Statistics

          Reference
Prediction     0     1
         0 29078  3496
         1  8208  7628

               Accuracy : 0.7582
                 95% CI : (0.7544, 0.762)
    No Information Rate : 0.7702
    P-Value [Acc > NIR] : 1

                  Kappa : 0.4054
 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.7799
            Specificity : 0.6857
         Pos Pred Value : 0.8927
         Neg Pred Value : 0.4817
             Prevalence : 0.7702
         Detection Rate : 0.6007
   Detection Prevalence : 0.6729
      Balanced Accuracy : 0.7328

       'Positive' Class : 0
```

GBM Reference

- Y. Freund and R.E. Schapire (1997) “A decision-theoretic generalization of on-line learning and an application to boosting,” Journal of Computer and System Sciences, 55(1):119-139.
- G. Ridgeway (1999). “The state of boosting,” Computing Science and Statistics 31:172-181.
- J.H. Friedman, T. Hastie, R. Tibshirani (2000). “Additive Logistic Regression: a Statistical View of Boosting,” Annals of Statistics 28(2):337-374.
- J.H. Friedman (2001). “Greedy Function Approximation: A Gradient Boosting Machine,” Annals of Statistics 29(5):1189-1232.
- J.H. Friedman (2002). “Stochastic Gradient Boosting,” Computational Statistics and Data Analysis 38(4):367-378.
- B. Kriegler (2007). Cost-Sensitive Stochastic Gradient Boosting Within a Quantitative Regression Framework. PhD dissertation, UCLA Statistics.
- C. Burges (2010). “From RankNet to LambdaRank to LambdaMART: An Overview,” Microsoft Research Technical Report MSR-TR-2010-82.

statinfer

Thank you

statinfer

Statinfer.com

Download the course videos and handouts from the below link
https://statinfer.com/course/machine-learning-with-r-2/curriculum/?c=b433a9be3189