GRADIENT BOOSTING

1 Introduction

The idea behind gradient boosting is to build an ensemble of trees (like a random forests) that is additive in nature (UNlike random forests). Unlike the grossly overfit decision trees that are used as the building blocks of random forests, gradient boosting uses *weak learners* to incrementally make its way towards the target variable. This is done by recursively fitting a tree to the residual of the previous ensemble. Stochastic gradient boosting chooses a subset of *observations* to use as the training set for each tree.

2 PenDigits Dataset

For this tutorial, we'll return to the optical character recognition problem and the handwritten digits (PenDigits) dataset. A description of this dataset, quoted from the UCI Machine Learning Repository website, is as follows:

> The samples written by 30 writers are used for training, cross-validation and writer dependent testing, and the digits written by the other 14 are used for writer independent testing. This database is also available in the UNIPEN format.
>
> We use a WACOM PL-100V pressure sensitive tablet with an integrated LCD display and a cordless stylus. The input and display areas are located in the same place. Attached to the serial port of an Intel 486 based PC, it allows us to collect handwriting samples. The tablet sends x and y tablet coordinates and pressure level values of the pen at fixed time intervals (sampling rate) of 100 miliseconds.
>
> These writers are asked to write 250 digits in random order inside boxes of 500 by 500 tablet pixel resolution. Subject are monitored only during the first entry screens. Each screen contains five boxes with the digits to be written displayed above. Subjects are told to write only inside these boxes. If they make a mistake or are unhappy with their writing, they are instructed to clear the content of a box by using an on-screen button. The first ten digits are ignored because most writers are not familiar with this type of input devices, but subjects are not aware of this.
>
> In our study, we use only (x, y) coordinate information. The stylus pressure level values are ignored. First we apply normalization to make our representation invariant to translations and scale distortions. The raw data that we capture from the tablet consist of integer values

> between 0 and 500 (tablet input box resolution). The new coordinates are such that the coordinate which has the maximum range varies between 0 and 100. Usually x stays in this range, since most characters are taller than they are wide.
>
> In order to train and test our classifiers, we need to represent digits as constant length feature vectors. A commonly used technique leading to good results is resampling the (x_t, y_t) points. Temporal resampling (points regularly spaced in time) or spatial resampling (points regularly spaced in arc length) can be used here. Raw point data are already regularly spaced in time but the distance between them is variable. Previous research showed that spatial resampling to obtain a constant number of regularly spaced points on the trajectory yields much better performance, because it provides a better alignment between points. Our resampling algorithm uses simple linear interpolation between pairs of points. The resampled digits are represented as a sequence of T points $(x_t, y_t)_{t=1}^{T}$, regularly spaced in arc length, as opposed to the input sequence, which is regularly spaced in time.
>
> So, the input vector size is $2T$, two times the number of points resampled. We considered spatial resampling to $T = 8, 12, 16$ points in our experiments and found that $T = 8$ gave the best trade-off between accuracy and complexity.

Let's see how well we can predict the handwritten digit using the 16 variables of coordinate information.

Preparing the data

Let's first discuss some of the inputs to this model. To begin with, we can't use a data frame, we have to use the Matrix package to create a design matrix. That input matrix should be of the class dgCMatrix. For data sets with many categorical variables, this first step creates all the dummy variables going into the model. The design matrix will be a sparse data matrix, meaning that zero entries are not stored. For the PenDigits data, this will not look much different than a regular matrix.

The **label** or target variable should be numeric containing class numbers as 0, ..., num_classes. The digit variable has 10 levels (the digits 0,...,9). However, if you type as.numeric(train$digit) the resulting vector will number the levels 1,..,10. In order to use the true numbers 0,..,9 we'll have to use the following command: as.numeric(levels(train$digit))[train$digit].

```
> # first prepare the data
> load("PenDigits.Rdata")
> library('Matrix')
> xtrain = model.matrix(digit ~ . , data=train)
> xtest = model.matrix(digit ~ . , data=test)
> ytrain = as.numeric(levels(train$digit))[train$digit]
> ytest = as.numeric(levels(test$digit))[test$digit]
>
```

Other model parameters

Now let's discuss the input parameters and run the model. First of all, we have to tell the model what type of classification problem we're doing - is it binary or multiclass or regression? Depending on which type of problem, we'll choose the objective function (loss function).

- For binary problems, use objective = "binary:logistic".
- For multiclass problems, use objective = "multi:softprob" if you'd like to output the probabilities for *every* class or (more likely) use objective = "multi:softmax" to output the decision class based on the max probability.
- For regression problems, use the default objective = reg:linear.

Several other parameters need to be specified and can be tuned to optimize your model. Below the default values of the parameters are given, along with a brief description of their purpose

- eta = 0.03. The **step-size shrinkage used to prevent over-fitting**, it ranges from 0 to 1 with smaller values creating models more robust to overfitting (although slower to compute). The gradient boosting model is an additive one where (essentially) we try to model the residual values from the previous round in every iteration. To prevent overfitting, we do not use the residual prediction outright but we shrink it by some penalty. This prevents us from fitting the training set perfectly and just inches us closer to the correct decision boundary at every step.
- gamma = 0. The minimum loss reduction required to make a further partition on a leaf node of the tree. The larger the value of gamma, the more conservative (read: 'less overfit on training') the model will be. Main idea behind increasing gamma is to **stop algorithm from overfitting the training set** by creating trees that just barely reduce the training error.
- max_depth = 6. The maximum depth of a tree. Deeper trees have more leaves (terminal nodes). **Deep trees mean that predictions are largely decided by the first few trees in the ensemble** which is undesirable even though the method may converge faster because it requires fewer trees.
- subsample = 1. The proportion of training observations to use at each iteration. For example, setting to 0.5 means that xgboost will randomly collect half of the data to grow trees. Makes computation time shorter. Anything less than 1 is considered *stochastic* gradient boosting. *Stochastic* boosting is particularly useful to **tame the influence of outliers** because they are not impacting every tree in the ensemble.
- colsample_bytree = 1. The proportion of variables to sample and use when constructing each *tree* (unlike random forests, which use the mtry parameter to sample at each node split.)
- colsample_bylevel = 1. The proportion of variables to sample and use when constructing each *node split* (like random forest's mtry parameter).
- nrounds. The max number of iterations taken - **the number of trees in the ensemble**.
- num_class. Specifies the number of levels of the target variable (classification).
- verbose = 1. If 0, method will stay silent and output answer. If 1 will print information of performance (eval_metric) for each iteration.
- eval_metric = . Set automatically by objective but can be over-ridden for another metric.
  - rmse. Root mean square error.
  - logloss. Negative log-likelihood.
  - error. Binary misclassification rate using cutoff probability of 0.5.
  - merror. Multiclass misclassification rate using maximum class probability.

Creating the model

Now that we understand the inputs, let's go ahead and run the model on the PenDigits dataset and check the misclassification rates on the training and validation data sets.

```
> #install.packages('xgboost')
> library(xgboost)
> set.seed(7515)
> # run the model (assuming parameters are already tuned)
> xgb <- xgboost(data = xtrain,
+               label = ytrain,
+               eta = 0.05,
+               max_depth = 15,
+               gamma = 0,
+               nround=100,
+               subsample = 0.75,
+               colsample_bylevel = 0.75,
+               num_class= 10,
+               objective = "multi:softmax",
+               nthread = 3,
+               eval_metric = 'merror',
+               verbose =0)
```

Use the created model and the predict() function to check the model's performance on validation data. Output the confusion matrix, and the overall misclassification rate:

```
> ptrain = predict(xgb, xtrain)
> pvalid = predict(xgb, xtest)
> cat('Confusion Matrix:')
```

```
Confusion Matrix:
```

```
> table(pvalid,test$digit)
```

```
pvalid   0   1   2   3   4   5   6   7   8   9
     0 343   0   0   0   0   0   0   0   2   0
     1   0 335   3   4   0   1   1  23   0   2
     2   0  26 358   0   0   0   0   2   0   0
     3   0   0   1 330   0   5   0   1   0   5
     4   0   1   0   0 363   0   0  10   0   0
     5   0   0   0   0   0 312   0   0   0   0
     6   0   0   0   0   0   0 335   0   0   0
     7   0   2   2   1   1   0   0 328   0   2
     8  20   0   0   0   0   3   0   0 334   1
     9   0   0   0   1   0  14   0   0   0 326
```

```
> XGBmrt = sum(ptrain!=train$digit)/length(train$digit)
> XGBmrv = sum(pvalid!=test$digit)/length(test$digit)
> cat('XGB Training Misclassification Rate:', XGBmrt)
```

```
XGB Training Misclassification Rate: 0.0002668802
```

```
> cat('XGB Validation Misclassification Rate:', XGBmrv)
```

```
XGB Validation Misclassification Rate: 0.0383076
```

We can also get information about variable importance:

```
> importance <- xgb.importance(feature_names = colnames(xtrain), model = xgb)
> head(importance)
```

```
   Feature       Gain      Cover  Frequency
1:     V16 0.16128930 0.11009869 0.06703725
2:     V14 0.13590450 0.11088325 0.07566581
3:      V2 0.08511651 0.07297172 0.06774247
4:     V10 0.08139058 0.08133519 0.07483614
5:      V1 0.07769683 0.08278055 0.06828176
6:     V11 0.06736463 0.05700984 0.05148096
```

3 Tuning the Model

While a grid search is a nice idea in theory, in practice it can often be too slow and provide too little improvement to argue it's use over a simpler approach that tunes parameters one at a time, keeping all others constant in an attempt to isolate their effects. In the following experiment, we will use the following parameter values for the default model (i.e. when the parameters are held constant):

- eta = 0.1
- colsample_bylevel = 0.67
- max_depth = 5
- sub_sample = 0.75

3.1 Tuning Eta

```
> #install.packages('ggplot2')
> #install.packages('reshape2')
> library(reshape2)
> library(ggplot2)
> #############################################
> ######### PARAMETERS TO BE SEARCHED #########
> #############################################
> #  eta candidates
> eta=c(0.01,0.1,0.25,0.5,.6,0.7)
> #  colsample_bylevel candidates
> cs=c(.33,.67,1)
> #  max_depth candidates
> md=c(3,5,7,9)
> #  sub_sample candidates
> ss=c(0.25,0.5,0.75,1)
> #  fixed number of rounds
> num_rounds=100
> #############################################
> # coordinates of default model in terms of
> # the entries in the vectors above:
> default=c(2,2,2,3)
```

3.2 Tuning Eta

We'll start with eta:

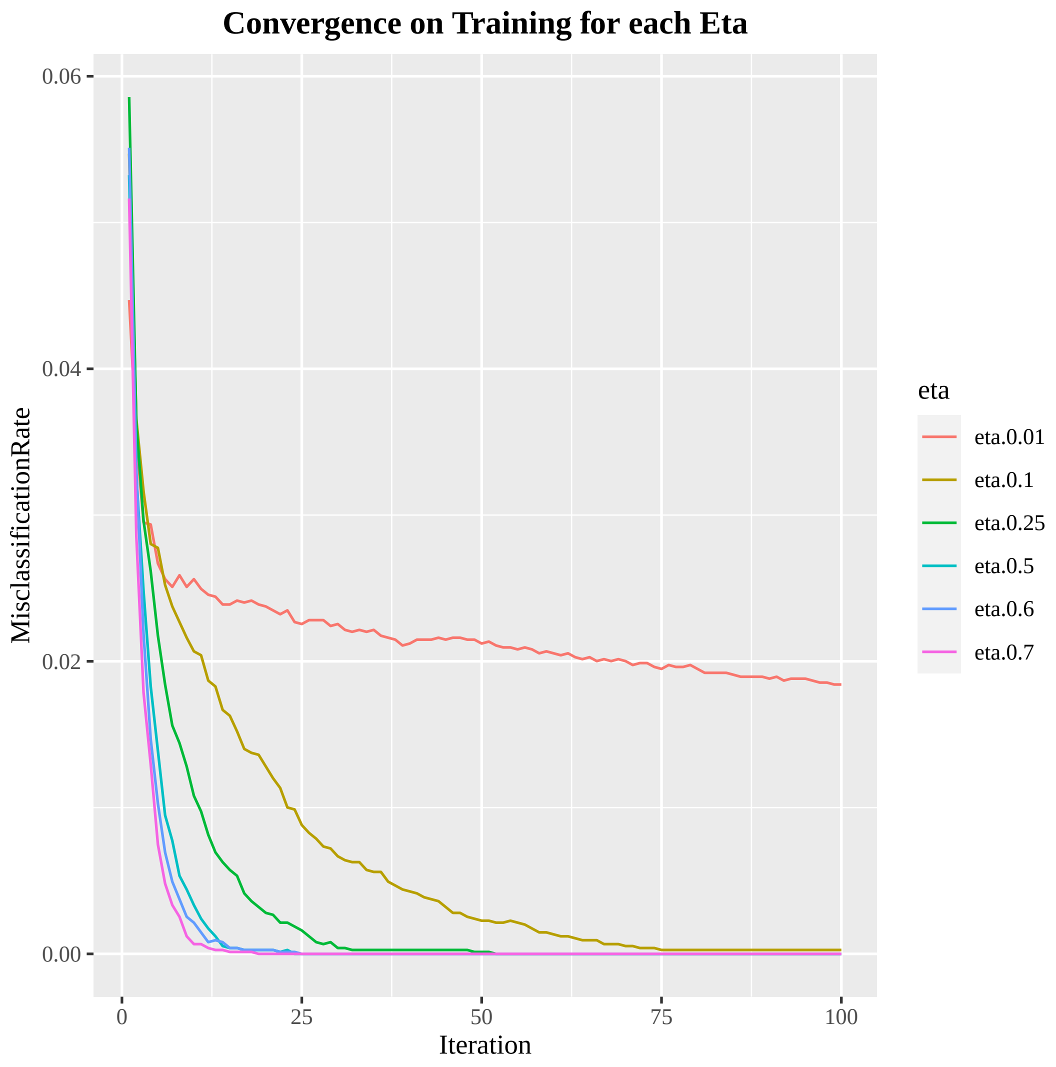
```
> # create empty matrices to hold the convergence
> # and prediction results for our search over eta:
> conv_eta = matrix(NA,num_rounds,length(eta))
> pred_eta = matrix(NA,dim(test)[1], length(eta))
> word = rep('eta',length(eta))
> colnames(conv_eta) = colnames(pred_eta) = paste(word,eta)
> for(i in 1:length(eta)){
+   params=list(eta = eta[i], colsample_bylevel=cs[default[2]],
+               subsample = ss[default[4]], max_depth = md[default[3]],
+               min_child_weight = 1)
+   xgb=xgboost(xtrain, label = ytrain,
+               nrounds = num_rounds,
+               params = params,
+               verbose=0,
+               num_class= 10,
+               objective = "multi:softmax")
+   conv_eta[,i] = xgb$evaluation_log$train_merror
+   pred_eta[,i] = predict(xgb, xtest)
+ }
> cat('Validation Misclassification Error for Each eta:')
```

```
Validation Misclassification Error for Each eta:
```

```
> (1-colMeans(ytest==pred_eta))
```

```
  eta 0.01    eta 0.1   eta 0.25    eta 0.5    eta 0.6    eta 0.7
0.07004002 0.03602058 0.03373356 0.03430532 0.03344768 0.03573471
```

```
> # Reshape the data frame so that the eta value is a variable
> # rather than having a column for each eta value:
> conv_eta = data.frame(iter=1:num_rounds, conv_eta)
> conv_eta2 = melt(conv_eta, id.vars = "iter",
+                  value.name = 'MisclassificationRate', variable.name = 'eta')
> ggplot(data = conv_eta2) + geom_line(aes(x = iter, y = MisclassificationRate, color = eta))+
+   labs(title = "Convergence on Training for each Eta",x='Iteration')+
+   theme(plot.title = element_text(family='Times',hjust='0.5', face='bold'))+
+   theme(text=element_text(family='Times'))
```



The misclassification rate seems to be improving (along with the convergence rate) as we increase eta toward the upper range of our sample. Eta at 0.6 produces the best results on validation.

3.3 Tuning colsample_bylevel

```
> # create empty matrices to hold the convergence
> # and prediction results for our search over
> # colsample_bylevel:
> conv_cs = matrix(NA,num_rounds,length(cs))
> pred_cs = matrix(NA,dim(test)[1], length(cs))
> word = rep('cs',length(cs))
> colnames(conv_cs) = colnames(pred_cs) = paste(word,cs)
> for(i in 1:length(cs)){
+   params=list(cs = cs[i], eta=eta[default[1]],
+               subsample = ss[default[4]], max_depth = md[default[3]],
+               min_child_weight = 1)
+   xgb=xgboost(xtrain, label = ytrain,
+               nrounds = num_rounds,
+               params = params,
+               verbose=0,
+               num_class= 10,
+               objective = "multi:softmax")
+   conv_cs[,i] = xgb$evaluation_log$train_merror
+   pred_cs[,i] = predict(xgb, xtest)
+ }
```

```
[11:01:28] WARNING: amalgamation/../src/learner.cc:516:
Parameters: { cs } might not be used.

  This may not be accurate due to some parameters are only used in language bindings but
  passed down to XGBoost core.  Or some parameters are not used but slip through this
  verification. Please open an issue if you find above cases.


[11:01:37] WARNING: amalgamation/../src/learner.cc:516:
Parameters: { cs } might not be used.

  This may not be accurate due to some parameters are only used in language bindings but
  passed down to XGBoost core.  Or some parameters are not used but slip through this
  verification. Please open an issue if you find above cases.


[11:01:44] WARNING: amalgamation/../src/learner.cc:516:
Parameters: { cs } might not be used.

  This may not be accurate due to some parameters are only used in language bindings but
  passed down to XGBoost core.  Or some parameters are not used but slip through this
  verification. Please open an issue if you find above cases.
```

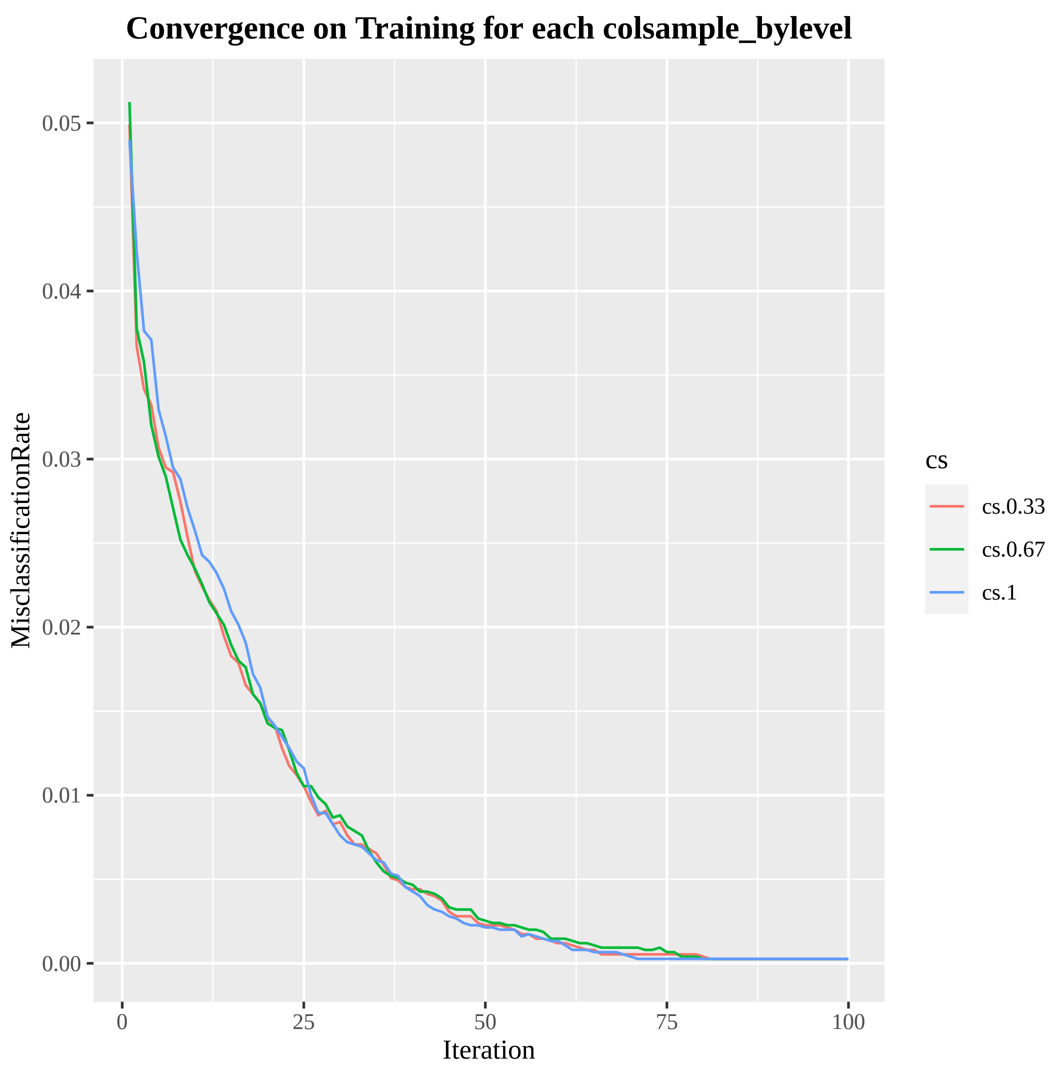
```
> cat('Validation Misclassification Error for Each colsample_bylevel:')
```

```
Validation Misclassification Error for Each colsample_bylevel:
```

```
> (Misclass_cs = 1-colMeans(ytest==pred_cs))
```

```
   cs 0.33    cs 0.67       cs 1
0.03573471 0.03659234 0.03859348
```

```
> # Reshape the data frame so that the eta value is a variable
> # rather than having a column for each eta value:
> conv_cs = data.frame(iter=1:num_rounds, conv_cs)
> conv_cs2 = melt(conv_cs, id.vars = "iter",
+                 value.name = 'MisclassificationRate', variable.name = 'cs')
> ggplot(data = conv_cs2) + geom_line(aes(x = iter, y = MisclassificationRate, color = cs))+
+   labs(title = "Convergence on Training for each colsample_bylevel",x='Iteration') +
+   theme(plot.title = element_text(family='Times',hjust='0.5', face='bold'))+
+   theme(text=element_text(family='Times'))
```



Looks like column sample does not affect the error convergence much at all in this example. Column sample of 0.33 had the lowest validation error.

3.4 Tuning subsample

```
> # create empty matrices to hold the convergence
> # and prediction results for our search over
> # colsample_bylevel:
> conv_ss = matrix(NA,num_rounds,length(ss))
> pred_ss = matrix(NA,dim(test)[1], length(ss))
> word = rep('ss',length(ss))
> colnames(conv_ss) = colnames(pred_ss) = paste(word,ss)
> for(i in 1:length(ss)){
+   params=list(cs = cs[default[2]], eta=eta[default[1]],
+               subsample = ss[i], max_depth = md[default[3]],
+               min_child_weight = 1)
+   xgb=xgboost(xtrain, label = ytrain, nrounds = num_rounds,
+               params = params,
+               verbose=0,
+               num_class= 10,
+               objective = "multi:softmax")
+   conv_ss[,i] = xgb$evaluation_log$train_merror
+   pred_ss[,i] = predict(xgb, xtest)
+ }
```

```
[11:01:52] WARNING: amalgamation/../src/learner.cc:516:
Parameters: { cs } might not be used.

  This may not be accurate due to some parameters are only used in language bindings but
  passed down to XGBoost core.  Or some parameters are not used but slip through this
  verification. Please open an issue if you find above cases.


[11:01:58] WARNING: amalgamation/../src/learner.cc:516:
Parameters: { cs } might not be used.

  This may not be accurate due to some parameters are only used in language bindings but
  passed down to XGBoost core.  Or some parameters are not used but slip through this
  verification. Please open an issue if you find above cases.


[11:02:07] WARNING: amalgamation/../src/learner.cc:516:
Parameters: { cs } might not be used.

  This may not be accurate due to some parameters are only used in language bindings but
  passed down to XGBoost core.  Or some parameters are not used but slip through this
  verification. Please open an issue if you find above cases.


[11:02:14] WARNING: amalgamation/../src/learner.cc:516:
Parameters: { cs } might not be used.

  This may not be accurate due to some parameters are only used in language bindings but
  passed down to XGBoost core.  Or some parameters are not used but slip through this
  verification. Please open an issue if you find above cases.
```

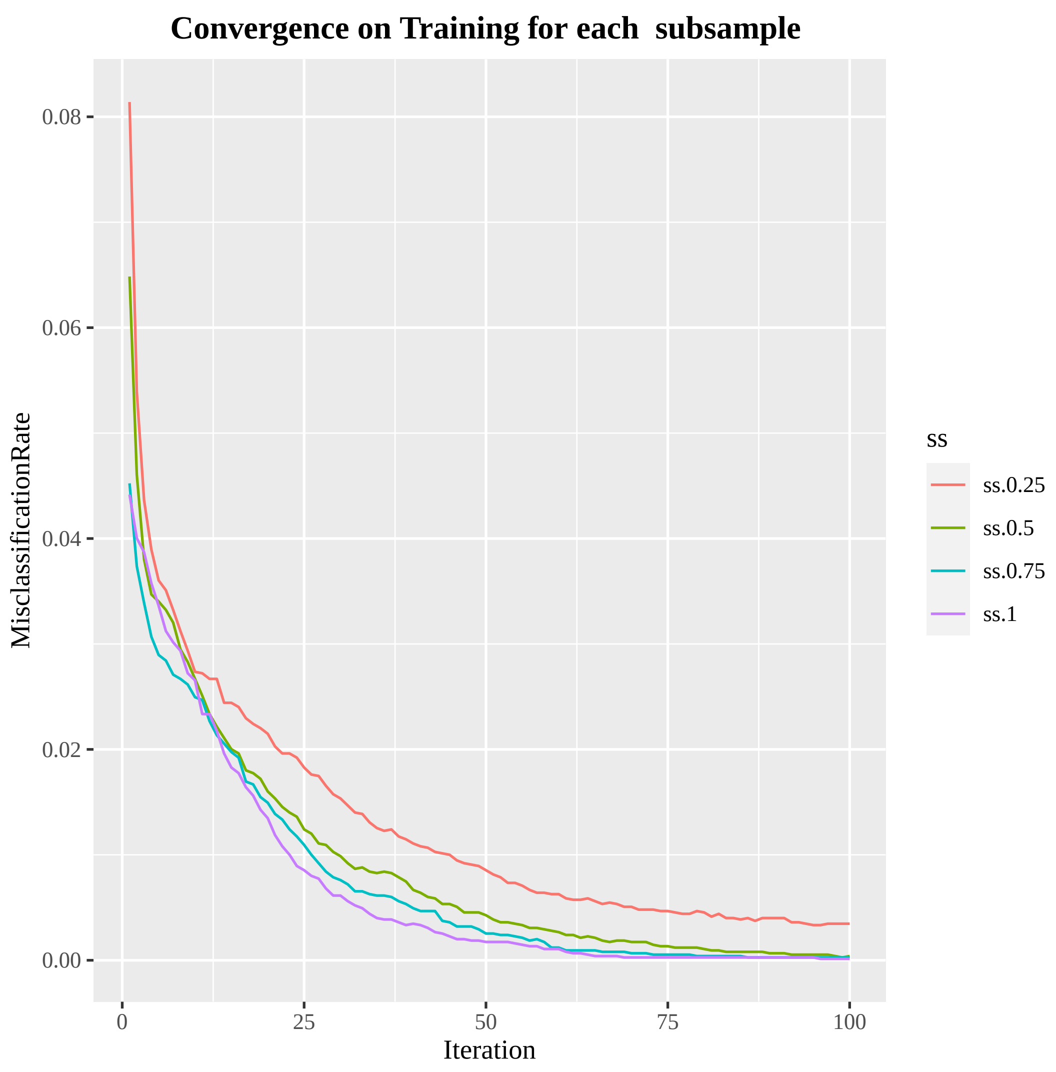
```
> cat('Validation Misclassification Error for Each subsample:')
```

```
Validation Misclassification Error for Each subsample:
```

```
> (Misclass_ss = 1-colMeans(ytest==pred_ss))
```

```
   ss 0.25     ss 0.5    ss 0.75       ss 1
0.03773585 0.03659234 0.03945111 0.03973699
```

```
> # Reshape the data frame so that the eta value is a variable
> # rather than having a column for each eta value:
> conv_ss = data.frame(iter=1:num_rounds, conv_ss)
> conv_ss2 = melt(conv_ss, id.vars = "iter",
+                 value.name = 'MisclassificationRate', variable.name = 'ss')
> ggplot(data = conv_ss2) + geom_line(aes(x = iter, y = MisclassificationRate, color = ss))+
+   labs(title = "Convergence on Training for each  subsample ",x='Iteration') +
+   theme(plot.title = element_text(family='Times',hjust='0.5', face='bold'))+
+   theme(text=element_text(family='Times'))
```



We have faster convergence for sampling rates of 50-100% and it appears 5̃0% subsample may optimize our performance on validation data.

3.5 Tuning max_depth

```
> # create empty matrices to hold the convergence
> # and prediction results for our search over
> # colsample_bylevel:
> conv_md = matrix(NA,num_rounds,length(md))
> pred_md = matrix(NA,dim(test)[1], length(md))
> word = rep('md',length(md))
> colnames(conv_md) = colnames(pred_md) = paste(word,md)
> for(i in 1:length(md)){
+   params=list(cs = cs[default[2]], eta=eta[default[1]],
+               subsample = ss[default[3]], max_depth = md[i],
+               min_child_weight = 1)
+   xgb=xgboost(xtrain, label = ytrain,
+               nrounds = num_rounds,
+               params = params,
+               verbose=0,
+               num_class= 10,
+               objective = "multi:softmax")
+   conv_md[,i] = xgb$evaluation_log$train_merror
+   pred_md[,i] = predict(xgb, xtest)
+ }
```

```
[11:02:22] WARNING: amalgamation/../src/learner.cc:516:
Parameters: { cs } might not be used.

  This may not be accurate due to some parameters are only used in language bindings but
  passed down to XGBoost core.  Or some parameters are not used but slip through this
  verification. Please open an issue if you find above cases.


[11:02:28] WARNING: amalgamation/../src/learner.cc:516:
Parameters: { cs } might not be used.

  This may not be accurate due to some parameters are only used in language bindings but
  passed down to XGBoost core.  Or some parameters are not used but slip through this
  verification. Please open an issue if you find above cases.


[11:02:36] WARNING: amalgamation/../src/learner.cc:516:
Parameters: { cs } might not be used.

  This may not be accurate due to some parameters are only used in language bindings but
  passed down to XGBoost core.  Or some parameters are not used but slip through this
  verification. Please open an issue if you find above cases.


[11:02:45] WARNING: amalgamation/../src/learner.cc:516:
Parameters: { cs } might not be used.

  This may not be accurate due to some parameters are only used in language bindings but
  passed down to XGBoost core.  Or some parameters are not used but slip through this
  verification. Please open an issue if you find above cases.
```

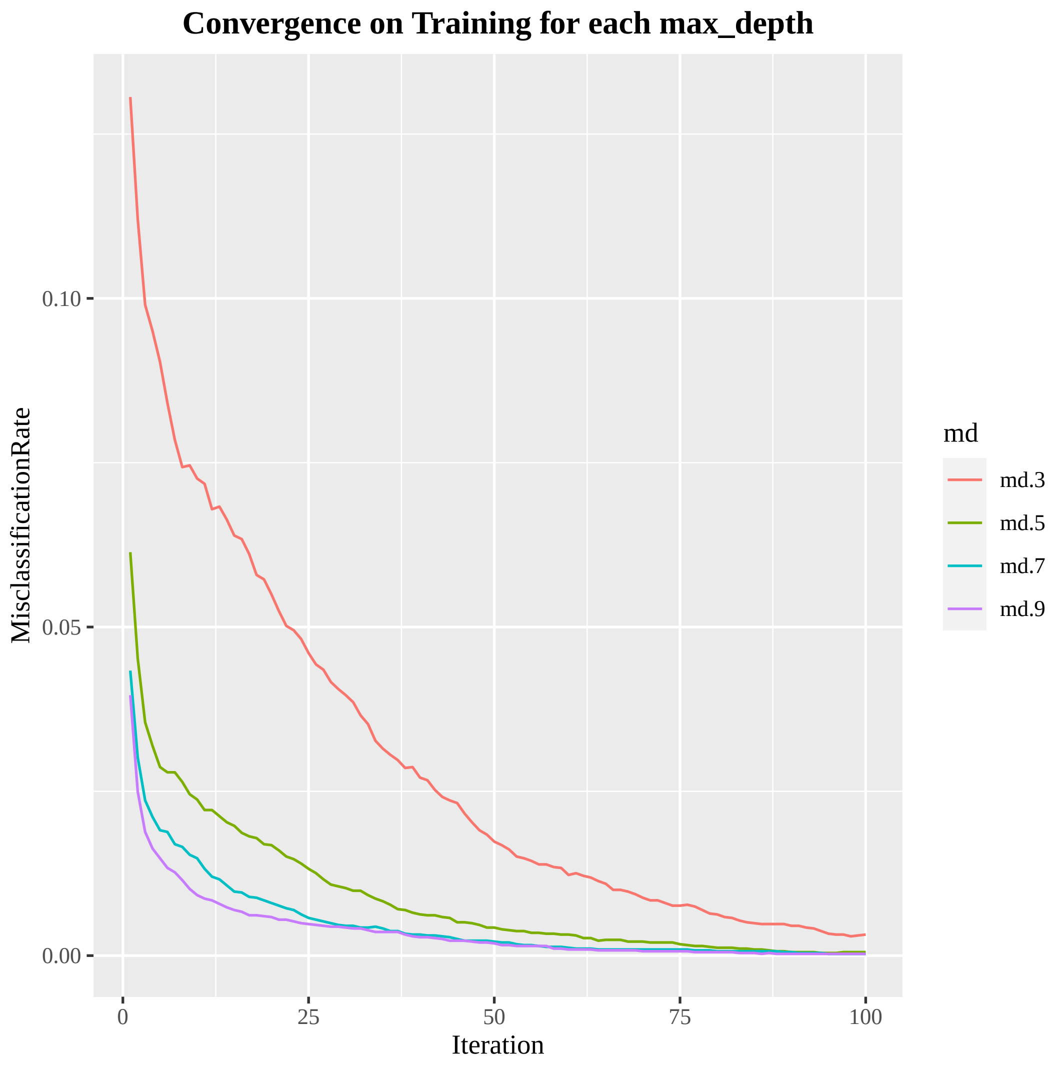
```
> cat('Validation Misclassification Error for Each max_depth:')
```

```
Validation Misclassification Error for Each max_depth:
```

```
> (Misclass_md = 1-colMeans(ytest==pred_md))
```

```
      md 3       md 5       md 7       md 9 
0.04230989 0.03487707 0.03573471 0.03401944 
```

```
> # Reshape the data frame so that the eta value is a variable
> # rather than having a column for each eta value:
> conv_md = data.frame(iter=1:num_rounds, conv_md)
> conv_md2 = melt(conv_md, id.vars = "iter",
+                 value.name = 'MisclassificationRate', variable.name = 'md')
> ggplot(data = conv_md2) + geom_line(aes(x = iter, y = MisclassificationRate, color = md))+
+   labs(title = "Convergence on Training for each max_depth",x='Iteration') +
+   theme(plot.title = element_text(family='Times',hjust='0.5', face='bold'))+
+   theme(text=element_text(family='Times'))
```



We have much faster convergence as max_depth increases to 9 and the validation error is also minimized for max_depth of 9, so we would either settle on 9 for this parameter or increase the range to check the performance of max_depth up to 12 or 15.