Gradient Boosting Machine (GBM) Tutorial

A fundamental understanding of GBM models is assumed, please seek resources to improve understanding and use this tutorial as a computational example.

Read data

More information can be found in the Data Import and Export tutorial notebook.

In [1]:
```
train = read.csv("train.csv")
test = read.csv("test.csv")
head(train)
```

Out[1]:

	RowID	CalendarYear	ModelYear	Make	Model	Cat1	Cat2	Cat3	Cat4	Cat
1	418079	2005	2004	AU	AU.14	B	C	A	A	A
2	232625	2006	2003	R	R.30	B	C	B	A	A
3	379029	2006	2006	AU	AU.14	B	A	A	A	A
4	181458	2007	2000	BU	BU.38	F	C	A	C	A
5	192434	2005	1999	BU	BU.38	F	A	A	C	A
6	443321	2007	2005	AU	AU.11	B	C	B	A	A

Set Random Seed

Set a random seed to allow reproducability of these results.

In [2]:
```
randomSeed = 1337
set.seed(randomSeed)
```

Load Package

R has a wide variety of open source packages for machine learning. This tutorial will use the standard 'gbm' package to make a GBM.

In [3]:
```
library(gbm)
```

```
Loading required package: survival
Loading required package: lattice
Loading required package: splines
Loading required package: parallel
Loaded gbm 2.1.1
```

Error Metric

R-Bloggers has a nice article on explaining the Log Loss function which can be read here (http://www.r-bloggers.com/making-sense-of-logarithmic-loss/).

In [4]:
```
LogLossBinary = function(actual, predicted, eps = 1e-15) {
    predicted = pmin(pmax(predicted, eps), 1-eps)
    - (sum(actual * log(predicted) + (1 - actual) * log(1 - predicted)))
/ length(actual)
}
```

Fit a simple GBM

The simple GBM below is fit using only 4 predictors. View the GBM package's references for more information on choosing appropriate hyperparameters and more sophisticated methods. These examples do not necessarily reflect best practices and should be viewed for illustration only. All analyses will also be done without any preprocessing of data.

In [5]:
```
gbmModel = gbm(formula = Response ~ Var1 + Var2 + Cat5 + Cat6,
               distribution = "bernoulli",
               data = train,
               n.trees = 2500,
               shrinkage = .01,
               n.minobsinnode = 20)
```

Find what the model predicts on the training data set.

In [6]:
```
gbmTrainPredictions = predict(object = gbmModel,
                              newdata = train,
                              n.trees = 1500,
                              type = "response")
```

In [7]:
```
head(data.frame("Actual" = train$Response,
                "PredictedProbability" = gbmTrainPredictions))
```

Out[7]:

	Actual	PredictedProbability
1	0	0.2827823
2	0	0.2780227
3	1	0.282297
4	0	0.2780227
5	1	0.2780227
6	0	0.238471

Log Loss calculated on the training set. We can calculate this value because we have the response for our training set.

In [8]:
```
LogLossBinary(train$Response, gbmTrainPredictions)
```

Out[8]: 0.590021043705372

Fit a GBM and test with a holdout subset

Create a subset of the training data to hold out and test the model's predictive power.

In [9]:
```
dataSubsetProportion = .2
randomRows = sample(1:nrow(train), floor(nrow(train) * dataSubsetProport
ion))
trainingHoldoutSet = train[randomRows, ]
trainingNonHoldoutSet = train[!(1:nrow(train) %in% randomRows), ]
```

Drop ID columns in the data sets used for modeling. Also drop the Model covariate for this illustration.

In [10]:
```
trainingHoldoutSet$RowID = NULL
trainingNonHoldoutSet$RowID = NULL
trainingHoldoutSet$Model = NULL
trainingNonHoldoutSet$Model = NULL
```

Fit a GBM on 5 predictors using the training data set that is not being held out.

In [11]:
```
gbmForTesting = gbm(formula = Response ~ Var1 + Var2 + Var3 + NVCat + NV
Var1,
                    distribution = "bernoulli",
                    data = trainingNonHoldoutSet,
                    n.trees = 1500,
                    shrinkage = .01,
                    n.minobsinnode = 50)
```

In [12]:
```
summary(gbmForTesting, plot = FALSE)
```

Out[12]:

	var	rel.inf
NVCat	NVCat	67.72105
Var2	Var2	22.97079
Var1	Var1	4.897743
Var3	Var3	3.416278
NVVar1	NVVar1	0.9941413

Make predictions using the model on the holdout and non-holdout data sets.

In [13]:
```
gbmHoldoutPredictions = predict(object = gbmForTesting,
                                newdata = trainingHoldoutSet,
                                n.trees = 100,
                                type = "response")

gbmNonHoldoutPredictions = predict(object = gbmForTesting,
                                   newdata = trainingNonHoldoutSet,
                                   n.trees = 100,
                                   type = "response")
```

Calculate Log Loss on the holdout and non-holdout sets.

In [14]:
```
print(paste(LogLossBinary(train$Response[randomRows], gbmHoldoutPredicti
ons),
            "Holdout Log Loss"))
print(paste(LogLossBinary(train$Response[!(1:nrow(train) %in% randomRow
s)], gbmNonHoldoutPredictions),
            "Non-Holdout Log Loss"))
```

```
[1] "0.592384640817789 Holdout Log Loss"
[1] "0.586523687486559 Non-Holdout Log Loss"
```

Fit a GBM with cross validation

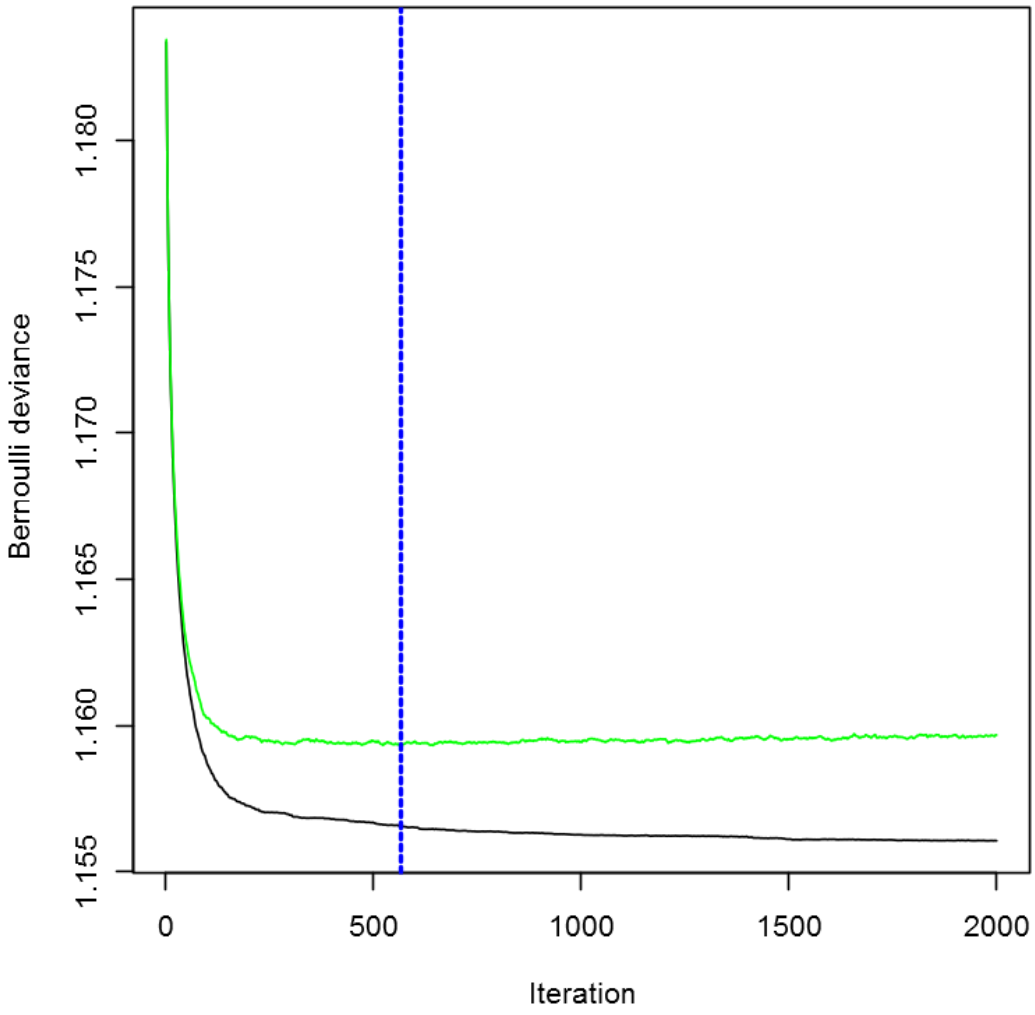
Let's fit a GBM with 5 fold cross validation and use the cross validation procedure to find the best number of trees for prediction. A random seed has to be set within the gbm function to ensure reproducibility for the distributed cross validation folds.

In [15]:
```
gbmWithCrossValidation = gbm(formula = Response ~ .,
                             distribution = "bernoulli",
                             data = trainingNonHoldoutSet,
                             n.trees = 2000,
                             shrinkage = .1,
                             n.minobsinnode = 200,
                             cv.folds = 5,
                             n.cores = 1)
```

Find the best tree for prediction. The black line is the training bernoulli deviance and the green line is the testing bernoulli deviance. The tree selected for prediction, indicated by the vertical blue line, is the tree that minimizes the testing error on the cross-validation folds.

In [16]:
```
bestTreeForPrediction = gbm.perf(gbmWithCrossValidation)
```

```
Using cv method...
```



Make predictions on the holdout and non-holdout data sets, as before, and view their Log Loss values.

In [17]:
```
gbmHoldoutPredictions = predict(object = gbmWithCrossValidation,
                                newdata = trainingHoldoutSet,
                                n.trees = bestTreeForPrediction,
                                type = "response")

gbmNonHoldoutPredictions = predict(object = gbmWithCrossValidation,
                                   newdata = trainingNonHoldoutSet,
                                   n.trees = bestTreeForPrediction,
                                   type = "response")
```

Calculate Log Loss for holdout and non-holdout sets.

In [18]:
```
print(paste(LogLossBinary(train$Response[randomRows], gbmHoldoutPredictions),
            "Holdout Log Loss"))
print(paste(LogLossBinary(train$Response[!(1:nrow(train) %in% randomRows)], gbmNonHoldoutPredictions),
            "Non-Holdout Log Loss"))
```

```
[1] "0.584214215280609 Holdout Log Loss"
[1] "0.578276452403044 Non-Holdout Log Loss"
```

Baseline GBM Benchmark

Fit one more GBM model on 6 predictors and create predictions on the testing data set. These predictions are scored as the 'GBM Benchmark' on the competition leaderboard.

In [19]:
```
gbmForTesting = gbm(formula = Response ~ Var1 + Var2 + Var3 + NVCat + NVVar1 + NVVar2,
                    distribution = "bernoulli",
                    data = trainingNonHoldoutSet,
                    n.trees = 1500,
                    shrinkage = .1,
                    n.minobsinnode = 50)

gbmTestPredictions = predict(object = gbmWithCrossValidation,
                             newdata = test,
                             n.trees = 1000,
                             type = "response")
```

In [20]:
```
outputDataSet = data.frame("RowID" = test$RowID,
                           "ProbabilityOfResponse" = gbmTestPredictions)
```

In [21]:
```
write.csv(outputDataSet, "gbmBenchmarkSubmission.csv", row.names = FALSE)
```