Machine Learning: XGBoost and Caret: Cheat Sheet

Installation

CRAN:

Install.packages("xgboost")
Install.packages("caret")

Github (recommended):

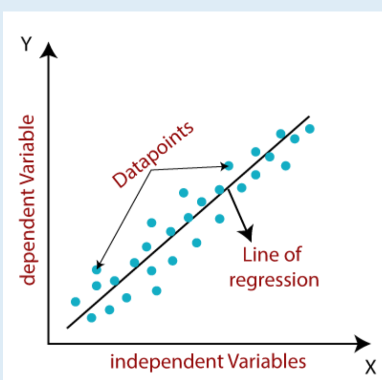
```
install.packages("drat", repos="https://cran.rstudio.com")
drat:::addRepo("dmlc")
install.packages("xgboost", repos="http://dmlc.ml/drat/", type = "source")
```

Basics

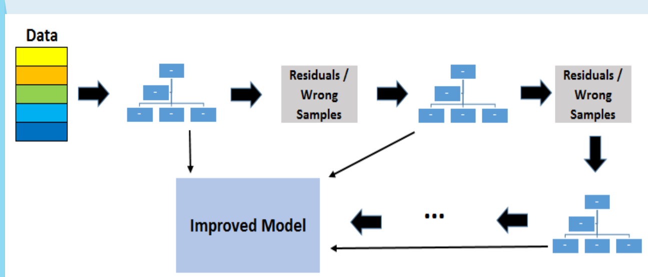
Machine Leaning allows for a program to make predictions based off a test data set. The packages allow for model training and application

Example Models

Linear Regression Model – booster = "gblinear"



Gradient Boosted Decision Tree



Datasets

Included Training and Testing dataset

```
data(agaricus.train, package='xgboost')
data(agaricus.test, package='xgboost')
```

Partitioning Data

In most cases you will have to split your own data into a test and train set which can be done using the caret package (iris data used for example)

```
set.seed(101)
Data(iris)

parts = createDataPartition( y = data$Species, p = 0.8, list = F)

Make Two New Data Sets (test and train)

train = data[parts, ]
test = data[-parts, ]
X_train = data.matrix(train[,-5])
y_train = train[,5]
X_test = data.matrix(test[,-5])
y_test = test[,5]
```

Data Structure

Xgboost provides a data storage method that allows for increased memory efficiency and decreased learning time
Xgb.DMatrix

```
xgboost_train = xgb.DMatrix(data = X_train, label = y_train)

xgboost_test = xgb.DMatrix(data= X_test, label= y_test)
```

Prediction Models

xgb.train

data =	Insert the DMatrix
nrounds =	max number of boosting iterations
max.depth =	Max depth of the boosting trees
Verbose =	0: No output 1: print evaluation metric 2: print information about tree
booster =	Allows for different models of regression "gblinear" "tree"
objective =	Specify the learning task and the corresponding learning objective

```
model <- xgb.train(data = xgboost_train,
max.depth=3,
nrounds=50,
Verbose = 0,
objective =  binary:logistic)
```

There are many option for customizing your test for maximum efficiency: https://xgboost.readthedocs.io/en/latest/parameter.html#parameters-for-tree-booster

Predictions

Use the model to predict the outcomes of the test data set

```
pred_test = predict(object = model, newdata = xgboost_test)
```

However, this will output the prediction in numeric form. We want the data to match the output in the test matrix which can be achieved using base code!

```
pred_test[(pred_test>3)] = 3
pred_y = as.factor
((levels(y_test))[round(pred_test)])
print(pred_y)
```

Now we can find the accuracy of our model!

Model Accuracy

The caret packages provides a simple and easy to understand way to view model accuracy in the form of a confusion matrix

```
conf_mat = confusionMatrix(y_test,
pred_y)print(conf_mat)
print(conf_mat(
```

```
Confusion Matrix and Statistics

            Reference
Prediction   setosa versicolor virginica
  setosa         10          0         0
  versicolor      0          9         1
  virginica       0          0        10

Overall Statistics

               Accuracy : 0.9667
                 95% CI : (0.8278, 0.9992)
    No Information Rate : 0.3667
    P-Value [Acc > NIR] : 4.476e-12

                  Kappa : 0.95
```

The matrix outputs useful statistics such as outputs, accuracy, and P-values. All of which can be used to measure model accuracy

Application

Let's apply the model to a new dataset!

```
new_data <- data.frame(Sepal.Length = c(5.1, 5.9, 7.3, 0.6, 4.3), Sepal.Width = c(3.5, 3.0, 5.6, 2.5, 6.7), Petal.Length = c(1.4, 4.2, 5.6, 2.2, 1.2), Petal.Width = c(0.2, 1.5, 0.5, 01.2, 0.7))
```

Now we can make a prediction about the new data set using the new_data matrix we just created, following the previous steps

```
new_data_pred <- predict(object = model, newdata = as.matrix(new_data))

new_data_pred[(new_data_pred>3)] = 3

pred_y =
as.factor((levels(y_test))[round(new_data_pred)])print(pred_y)
```

Credits

XGBoost - The XGBoost Contributors
Caret - Max Kuhn
Linear Regression Model Image - https://www.javatpoint.com/linear-regression-in-machine-learning
Gradient Boosting Tree Model - https://towardsdatascience.com/gradient-boosted-decision-trees-explained-9259bd8205af