Machine Learning Cheat Sheet - R

Regression

lm(y~x1+x2 , data) model building
summary(model) basic summary
vif(model) VIF is in car library

Other Options

lm(~0+x1, data) No intercept model
lm(y~.) y vs all variables
lm(y~.-x5-x6) y vs all except x5 & x6

Logistic Regression

glm(y ~ x1+x2, binomial(), data)
summary(model)
vif(model) library(car)

Prediction and Accuracy

Log_odds=predict(model) for log odd predictions
p=predict(model, type="response") probabilities
ifelse(p>threshold,1,0) classes based on threshold
conf_mat =table(data$y, predicted)
accuracy= sum(diag(conf_mat))/(sum(conf_mat))
pred_test=predict(model, test_data) on test data

Variable Importance

library(caret)
varImp(model_1, scale = FALSE) scale=FALSE for not scaling impact to percentages

Decision Trees

rpart(y~x1+x2, method="class", data, control=rpart.control(minsplit=30, cp))

Plotting the Tree

library(rpart.plot); library(rattle)
fancyRpartPlot(tree)
asRules(Ecom_Tree) gives the rules

Choosing Complexity Parameter

printcp(Sample_tree)
plotcp(Sample_tree) try values between (0.05-0.1)

Prediction and Accuracy

predict(tree) returns probability of classes
predict(tree, type="class") returns the class
predict(tree, newdata=test_data, type="class") on test data
conf_mat =table(data$y, predicted)
accuracy= sum(diag(conf_mat))/(sum(conf_mat))

Model Validation Metrics

accuracy= sum(diag(conf_mat))/(sum(conf_mat))
sensit= conf_mat [1,1]/(conf_mat [1,1]+ conf_mat [1,2])
specif= conf_mat [2,2]/(conf_mat [2,1]+ conf_mat [2,2])
F1_Score(data$y, predicted, 0) F1 score of "0" class
F1_Score(data$y, predicted, 1) F1 score of "1" class

ROC Curve

library(pROC)
roccurve <- roc(data$y, predicted_prob)
plot(roccurve)
auc(roccurve)

Train, Validation and Test data

shuffle the data before this
train<-data[1:80000,]
validation <-data[80001:90000,]
test <-data[90001:100000,]
train and test using caret package
library(caret)
seed <- createDataPartition(data$id, p=0.80, list=F)
id can be any vector with length = nrows
List=F to get seed as vector, non-list
train <- data[seed,]
test <- data[-seed,]

K-Fold Cross Validation

train_kf <- trainControl(method="cv", number=10)
K_fold_tree<-train(as.factor(y)~., method="rpart", trControl= train_kf, control=rpart.control(minsplit=1, cp=0.000001), data)
K_fold_tree$resample$Accuracy accuracy of the models
mean(K_fold_tree$resample$Accuracy)

SVM

library(e1071)
svm(as.factor(y)~., data, kernel ="radial", cost = C, gamma=G)
High Cost - Low Slack; Hard Margin Classifier; Overfitting
Low Cost - High Slack; Soft Margin Classifier; Underfitting
High Gamma; Very less support vectors
Low Gamma; Almost all are support vectors
Kernel function default value is radial

Prediction and Accuracy

pred <- predict(svm_mod, test_data)
pred <- predict(svm_mod, test_data,probability = TRUE)
predict probability code won't work if you don't mention “probability = TRUE" option
svm(as.factor(y)~., data, probability = TRUE)

Neural Networks

library(neuralnet)
nn_model=neuralnet(y~ . ,data, hidden=c(2,2), stepmax = 1000, learningrate=NULL, linear.output = FALSE)
hidden=c(n1,n2,..) a vector-hidden nodes in each layer
stepmax =1000 the number of epochs. epoch is complete run on training data
threshold=0.00001 stopping criteria connected to weight changes, stop if the weight change is less than threshold
learningrate=NULL parameter to control the weights movement, no learning rate by default (i.e. learning rate=1)
linear.output = FALSE classification or regression
err.fct="ce" cross entropy or square error
There is no weight decay in neuralnet package. Use nnet package. But you can't build a deep network using nnet

Prediction and Accuracy

pred=data.frame(compute(nn_model,test_data)$net.result) compute returns several values
conf_mat=confusionMatrix(data$y, pred)
accuracy= sum(diag(conf_mat))/(sum(conf_mat))

Random Forest

library(randomForest)
rf_model <- randomForest(train, factor(train$y), ntree, mtry,classwt=c(w1,w2), sampsize, replace=F)
ntree - number of trees; mtry - number of variables randomly sampled while splitting in each tree
classwt - weights of (0,1) for imbalanced data;
sampsize - take a random sample instead of boot strap;
replace - above sample without replacement.

Prediction and variable importance

varImp(rf_model)
partialPlot(rf_model, pred.data=train,x.var=x1) single variable graph by averaging out the impact of all variables

GBM

library(gbm)
gm<-gbm(y~., data, interaction.depth = 3, n.trees = n, bag.fraction=0.5,set.seed(2), shrinkage = 0.07, train.fraction=0.7,weights = model_weights)
interaction.depth - depth of each tree; n.tree =n - number of trees;
bag.fraction=0.5 - fraction of the data selected for next tree;
set.seed - use seed to regenerate results
shrinkage = 0.05 - use shrinkage*weights for creating weighted samples in each iteration
train.fraction=0.8 - before building the model choose initial 80% of the data
weights - model_weights=ifelse(train$target ==0, 0.2,0.8)