Download data, load data into memory, add column names

**library**(caret)

## Loading required package: lattice

## Loading required package: ggplot2

**library**(nnet)

**library**(NeuralNetTools)

## Warning: package 'NeuralNetTools' was built under R version 3.2.3

url.train <- "http://archive.ics.uci.edu/ml/machine-learning-databases/adult/adult.data"

url.test <- "http://archive.ics.uci.edu/ml/machine-learning-databases/adult/adult.test"

url.names <- "http://archive.ics.uci.edu/ml/machine-learning-databases/adult/adult.names"

download.file(url.train, destfile = "adult\_train.csv")

download.file(url.test, destfile = "adult\_test.csv")

download.file(url.names, destfile = "adult\_names.txt")

*# Read the training and test data into memory*

train <- read.csv("adult\_train.csv", header = FALSE)

*# The test data has an unnecessary first line that messes stuff up, this fixes that problem*

all\_content <- readLines("adult\_test.csv")

skip\_first <- all\_content[-1]

test <- read.csv(textConnection(skip\_first), header = FALSE)

*# The data file doesn't have the column names in its header, add those in manually...*

varNames <- c("Age",

"WorkClass",

"fnlwgt",

"Education",

"EducationNum",

"MaritalStatus",

"Occupation",

"Relationship",

"Race",

"Sex",

"CapitalGain",

"CapitalLoss",

"HoursPerWeek",

"NativeCountry",

"IncomeLevel")

names(train) <- varNames

names(test) <- varNames

levels(test$IncomeLevel) <- levels(train$IncomeLevel)

file.remove("adult\_train.csv")

## [1] TRUE

file.remove("adult\_test.csv")

## [1] TRUE

Neural Network package nnet

Use caret package to train a model using neural net

set.seed(1414)

start <- proc.time()[3]

model.nn <- train(IncomeLevel ~ .,

data = train,

method = "nnet")

print(model.nn)

predictions <- predict(model.nn, test[,1:14])

accuracy <- sum(predictions == test[,15])/length(test[,15])

print(accuracy)

## [1] 0.8255021

end <- proc.time()[3]

print(paste("This took ", round(end-start, digits = 1), " seconds", sep = ""))

## [1] "This took 1986.7 seconds"

Using the full data, this took quite a long time to finish! Let’s see if we can make this go any faster by applying lessons we learned from our feature selection discussion. But the methods we used in that example were for a *regression* problem and now we are doing *classification*, so we have to try something else!

One thing we can do is use linear discriminant analysis (LDA) which is similar to principal component analysis (PCA) in that it attempts to generate linear combinations of input variables that generate the most explanatory power over the output variable; it is actually likely not optimal for this purpose because it requires the assumption of normally distributed explanatory variables, which is unlikely the case for many of these inputs. This is just an example of utilizing varImp() for classification.

set.seed(1414)

model.lda <- train(IncomeLevel ~ .,

data = train,

method = "lda")

## Loading required package: MASS

plot(varImp(model.lda))

Looks like our top 5 according to an ROC curve are EducationNum, Relationship, Age, HoursPerWeek, and MaritalStatus. Let’s rerun the model using only those 5 explanatory variables and see if we have better run time without having sacrificed much accuracy.

keeps <- c("EducationNum",

"Relationship",

"Age",

"HoursPerWeek",

"MaritalStatus",

"IncomeLevel")

train.reduced <- train[,which(names(train) %in% keeps)]

test.reduced <- test[,which(names(test) %in% keeps)]

set.seed(1414)

start <- proc.time()[3]

model.nn <- train(IncomeLevel ~ .,

data = train.reduced,

method = "nnet")

print(model.nn)

predictions <- predict(model.nn, test.reduced[,1:5])

accuracy <- sum(predictions == test.reduced[,6])/length(test.reduced[,6])

print(accuracy)

## [1] 0.8274676

end <- proc.time()[3]

print(round(end-start, digits = 1))

## elapsed

## 704.8

And now we again see the importance of feature selection! We managed to decrease the runtime of our model by almost two thirds and actually managed a very slight **increase** in out-of-sample accuracy. Other ways we could continue trying to improve this model is using the tuneGrid argument in train() to have it scan over a larger set of different sizes and decay parameters.

Neural Network Visualization

Last thing, how can we visualize what our model is doing? Neural networks are pretty complicated, involving non-linear transformations of our inputs into a ‘hidden layer’ of nodes that are then translated into our output prediction with a potentially very large number of parameters involved. The package NeuralNetTools has some nice functions available to us to try to make sense of it, though!

But, since we want to make this a nice picture, let’s not use categorical input variables; they will get converted into a number of dummies equal to the number of levels of the variable minus one, which makes a potentially ugly picture. We again reduce the training set but this time to the top 5 variables that are continuous or binary categorical and then generate a new neural network model.

keeps <- c("EducationNum",

"Age",

"HoursPerWeek",

"Sex",

"CapitalGain",

"IncomeLevel")

train.reduced <- train[,which(names(train) %in% keeps)]

test.reduced <- test[,which(names(test) %in% keeps)]

set.seed(1414)

start <- proc.time()[3]

model.nn <- train(IncomeLevel ~ .,

data = train.reduced,

method = "nnet")

print(model.nn)

## Neural Network

##

## 32561 samples

## 5 predictor

## 2 classes: ' <=50K', ' >50K'

##

## No pre-processing

## Resampling: Bootstrapped (25 reps)

## Summary of sample sizes: 32561, 32561, 32561, 32561, 32561, 32561, ...

## Resampling results across tuning parameters:

##

## size decay Accuracy Kappa Accuracy SD Kappa SD

## 1 0e+00 0.7780259 0.1609816 0.020561823 0.16473064

## 1 1e-04 0.7816805 0.1702198 0.020256713 0.14286335

## 1 1e-01 0.8103144 0.3666197 0.022508098 0.16377497

## 3 0e+00 0.8003222 0.3009663 0.019640885 0.13367296

## 3 1e-04 0.8011753 0.3429569 0.019254923 0.11029043

## 3 1e-01 0.8231934 0.4415391 0.007841973 0.04768502

## 5 0e+00 0.8194666 0.4370152 0.010044515 0.04526004

## 5 1e-04 0.8146747 0.4083829 0.017703961 0.09149039

## 5 1e-01 0.8247930 0.4519080 0.003592250 0.01564480

##

## Accuracy was used to select the optimal model using the largest value.

## The final values used for the model were size = 5 and decay = 0.1.

predictions <- predict(model.nn, test.reduced[,1:5])

accuracy <- sum(predictions == test.reduced[,6])/length(test.reduced[,6])

print(accuracy)

## [1] 0.8200356

end <- proc.time()[3]

print(round(end-start, digits = 1))

## elapsed

## 731.8

Now for our nice picture

plotnet(model.nn$finalModel, y\_names = "IncomeLevel")

title("Graphical Representation of our Neural Network")

The width of the connecting lines represent the relative weights between nodes and the colors represent direction, black for positive and grey for negative. B1 and B2 represent bias nodes that function similarly to the constant term in your standard OLS regression.

You can see that the CapitalGain input has a large positive weight with the 4th node in the hidden layer, while the bias term for the hidden layer has a large positive weight with the 1st node in the hidden layer and a large negative weight with the 3rd node in the hidden layer.

The last important aspect of neural networks is that they are sensitive to initial conditions; notice that in the code above, we used the set.seed() function so that R will generate the same “random” numbers every time this code is executed and thus will give the same results; try changing the seed to a different number and you will see that the network diagram may have different weights.

Bonus: another methodology to estimate relative feature importance!

garson(model.nn$finalModel)