Big Data Workshop: A Big Data CaseStudy with Banking Data

Miles McBain

2 December 2016

Table of Contents

# Intro

In this practical we're going to use a range of methods in a predictive modelling with big data scenario. We will use R to connect to h2o which is a big data statistics and machine learning platform built in Java.

## Background

The banking, insurance, and telecommunications industries frequently engage in a practice called 'below the line marketing'. That is, they try to upsell existing customers new products based on knowledge held about them. For example: Your bank may call you and try to sell you an income protection of life insurance product. This call is usually not random. It is likely based on a predictive model that has assigned you some kind of likelihood of accepting the deal.

For large companies, calling all of their customers to make offers of new products is fincancially infeasible. They are also sensitive to customer fatigue in terms of making many offers that customers do not find attractive. Predictive models are used to make targetted campaigns that maximise the probability of a deal being made. The data that feed these models can be very big indeed. Consider the pitcure of you that could be formed by your timestamped and geotagged bank transaction records.

## The Data

The data for this analysis are real records from a telemarketing campaign run by a Spanish bank. The data contain the records for 32000 calls to banking customers. A bank may collect data like this from a 'pilot' campaign based on customers selected by stratified random sample. The data contain a 0/1 indicator variable which represents the failure/success of the customer accepting the proposed deal.

## Questions

1. What are the important variables in the data that relate to likelihood of success or failure?
2. Can we identify groups or clusters of cusomters that have a higher likelihood of sucess or failure?
3. Can we use some predictive modelling techniques to construct a useful model of success or failure?

# Learning objectives

The exercise will introduce you to a big data processing platform and some typical activities in the context of big data and predictive modelling.

# Requirements

To complete this exercise you will need a computer with R Studio installed and the following packages:

* dplyr
* ggplot2
* readr
* h2o

# Instructions

## Setting up

1. Install any missing R packages in RStudio: install.packages("dplyr",readr","ggplot2","h2o")
2. Esure the latest version of the Java runtime is installed: <http://www.oracle.com/technetwork/java/javase/downloads/jre8-downloads-2133155.html>
3. Change the and comment the setwd() command below to the folder where you downloaded the workshop datasets.
4. Consider the discussion points in each question and if necessary, write R code to resolve them.

#setwd("~/") #Will need to set this to the right path.

## Load Data

Let's load the data and see we're dealing with:

library(readr)  
library(dplyr)  
bank\_data <- read\_csv("./datasets/bank\_customer\_data.csv")

bank\_data

## # A tibble: 32,127 × 21  
## housing job day qualification month location  
## <chr> <chr> <chr> <chr> <chr> <chr>  
## 1 no Lawyer tue other aug Inner.city  
## 2 no Driver thu Higher.school.certificate may Inner.city  
## 3 no Gardener wed Bachelour.degree may Inner.city  
## 4 yes Driver mon other may Inner.city  
## 5 yes Lawyer tue other jun Inner.city  
## 6 no Lawyer fri Bachelour.degree may Inner.city  
## 7 yes Academic thu doctorate aug Inner.city  
## 8 no Electrician thu other aug Inner.city  
## 9 no Gardener tue Bachelour.degree may Inner.city  
## 10 no Driver fri doctorate nov Inner.city  
## # ... with 32,117 more rows, and 15 more variables: online.banking <chr>,  
## # checking.account <chr>, prev.sales <int>, year.contact <int>,  
## # length <int>, central.interest.rate <dbl>, employment.rate <dbl>,  
## # customer.satisfaction <chr>, price.index <dbl>, credit.rating <chr>,  
## # age <int>, staff\_total <int>, confidence.index <dbl>,  
## # emails.month <int>, y <chr>

The dataset contains a mix of customer demographic variables and financial indicies from the times the calls were made. The binary response varibale is y.

A variable worh mentioning is length, the duration of the call in seconds. This information is not know about a call before it is made, so should not be used in later modelling.

## Start H2O

h2o is going to do the heavy lifting so you'll need to fire up a h2o instance if you have not already:

library(h2o)

h2o\_inst <-h2o.init(ip = "127.0.0.1", port = 54321)

## Connection successful!  
##   
## R is connected to the H2O cluster:   
## H2O cluster uptime: 5 hours 49 minutes   
## H2O cluster version: 3.10.0.8   
## H2O cluster version age: 1 month and 23 days   
## H2O cluster name: H2O\_started\_from\_R\_mcbainm\_otv783   
## H2O cluster total nodes: 1   
## H2O cluster total memory: 3.36 GB   
## H2O cluster total cores: 4   
## H2O cluster allowed cores: 2   
## H2O cluster healthy: TRUE   
## H2O Connection ip: 127.0.0.1   
## H2O Connection port: 54321   
## H2O Connection proxy: NA   
## R Version: R version 3.3.2 (2016-10-31)

In practice, rather than on your local machine, the H2O instance could be running on a more powerful cluster of servers, which you connect to through the API provided in the h2o library.

Next we need to push our data to the server. It's fastest to use h2o to read the csv file. This also ensures columns have types h2o can use.

h2o\_bank\_data <- h2o.uploadFile(path = "./datasets/bank\_customer\_data.csv")

##   
 |   
 | | 0%  
 |   
 |=================================================================| 100%

h2o\_bank\_data can be treated much like a regular R data frame, but operations on it are executed by the h2o instnace, often using paralell processing.

# Question 1

There are many ways to determine varibale importance. Principle Component Analysis (PCA) can be used as a tool to see what variables or themes of variables are accounting for most of the variation in the data. We are going to explore the usage of PCA for this purpose here.

First we remove the columns y and length from the data, since we're interested in the covariates:

covs <- !grepl("^y$|^length$", names(h2o\_bank\_data)) #find the colums that are not 'y' or 'duration'  
h2o\_bank\_data\_covs <- h2o\_bank\_data[, covs]

Next we run PCA on the data. To see the importnance of comparing variables on the same scale, you could rerun the analysis with transform = "NONE".

pca\_model <-h2o.prcomp(training\_frame = h2o\_bank\_data\_covs,  
 k = ncol(h2o\_bank\_data\_covs),  
 transform = "STANDARDIZE")

summary(pca\_model)

## Model Details:  
## ==============  
##   
## H2ODimReductionModel: pca  
## Model Key: PCA\_model\_R\_1480731054437\_95   
## Importance of components:   
## pc1 pc2 pc3 pc4 pc5  
## Standard deviation 1.851898 1.804361 1.135920 1.036605 0.997114  
## Proportion of Variance 0.207721 0.197193 0.078152 0.065084 0.060219  
## Cumulative Proportion 0.207721 0.404914 0.483066 0.548150 0.608369  
## pc6 pc7 pc8 pc9 pc10  
## Standard deviation 0.983074 0.926088 0.797095 0.645005 0.549693  
## Proportion of Variance 0.058535 0.051946 0.038483 0.025198 0.018301  
## Cumulative Proportion 0.666905 0.718851 0.757333 0.782532 0.800833  
## pc11 pc12 pc13 pc14 pc15  
## Standard deviation 0.530234 0.481687 0.470882 0.456416 0.450014  
## Proportion of Variance 0.017029 0.014053 0.013430 0.012617 0.012266  
## Cumulative Proportion 0.817862 0.831915 0.845345 0.857962 0.870228  
## pc16 pc17 pc18 pc19  
## Standard deviation 0.442833 0.424823 0.396616 0.386205  
## Proportion of Variance 0.011878 0.010931 0.009528 0.009034  
## Cumulative Proportion 0.882106 0.893037 0.902564 0.911598  
##   
## H2ODimReductionMetrics: pca  
##   
## No model metrics available for PCA  
##   
##   
##   
## NULL

It looks like 8 components explain most of the variation in the dataset.

The rotation describes the direction of a unit vector along each principle component on the space defined by the original columns. Looking at the magnitudes of each variable in the rotation can give some insigt into the 'flavours' of the principle components and which variables are making the strongest contributions. Be sure to look at strong negative and postive contributions.

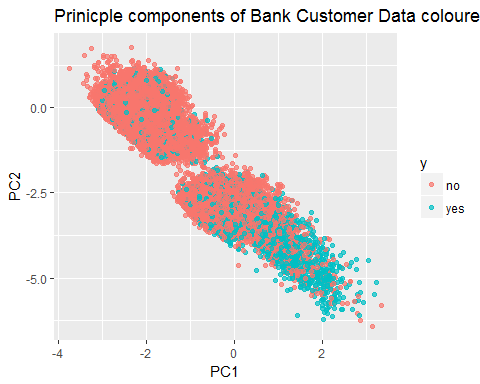
View(pca\_model@model$eigenvectors)

* Can you come up with some 'flavours' for the first two or three principle components?
* Do they make intuitive sense in terms of things you would expect to be important to a customer accepting a deal?

Finally to aid interpretation of the relationship we can plot the response over the first two PCs:

library(ggplot2)  
pca\_data <- as.data.frame(h2o.predict(object = pca\_model,  
 newdata = h2o\_bank\_data\_covs))  
  
plot\_data <- cbind(pca\_data,  
 as.data.frame(h2o\_bank\_data[,c("y")]))  
#as.data.frame() when called on a h2o data.frame will pull the data back into the R session. This is handy for plotting etc. But be careful not to do this with very large datasets without downsampling them first.

plot\_data %>%  
ggplot(aes(x = PC1, y = PC2, colour=y)) +  
 geom\_point(alpha = 0.7) +  
 ggtitle("Prinicple components of Bank Customer Data coloured by deal success")



* What is your interpretation of customers the bank should target in light of this plot?
* Try plotting other combinations of PCs.

## Question 2

In this question we consider a clustering approach. We try to find groupings of customers that are 'close' in terms of their representation in the data we have. A bank may call this task 'customer segmentation'.

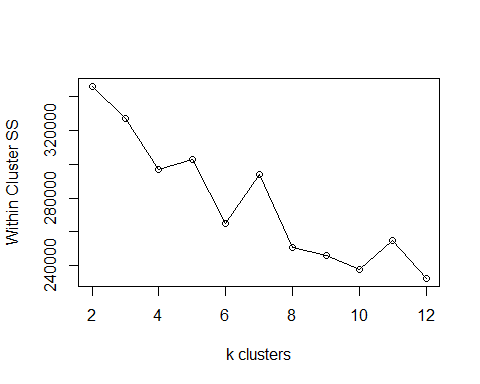
The algorithm used here is k-means. The main consideration in k-means clustering is how many clusters the algorithm should form. Clustering practicioners talk about this issue in terms of 'choosing k'. In reality common approaches are to keep k small and consider differences in the *within cluster sum of squares* as k increases.

To do k-means clustering on big data using h2o:

results = list()  
  
for(k in 2:12){  
 results[k] <- h2o.kmeans(training\_frame = h2o\_bank\_data\_covs,  
 k = k,  
 standardize = TRUE,  
 init = "Random"  
 )  
}

## Warning in `[<-`(`\*tmp\*`, k, value = <S4 object of class  
## structure("H2OClusteringModel", package = "h2o")>): implicit list embedding  
## of S4 objects is deprecated  
  
## Warning in `[<-`(`\*tmp\*`, k, value = <S4 object of class  
## structure("H2OClusteringModel", package = "h2o")>): implicit list embedding  
## of S4 objects is deprecated  
  
## Warning in `[<-`(`\*tmp\*`, k, value = <S4 object of class  
## structure("H2OClusteringModel", package = "h2o")>): implicit list embedding  
## of S4 objects is deprecated  
  
## Warning in `[<-`(`\*tmp\*`, k, value = <S4 object of class  
## structure("H2OClusteringModel", package = "h2o")>): implicit list embedding  
## of S4 objects is deprecated  
  
## Warning in `[<-`(`\*tmp\*`, k, value = <S4 object of class  
## structure("H2OClusteringModel", package = "h2o")>): implicit list embedding  
## of S4 objects is deprecated  
  
## Warning in `[<-`(`\*tmp\*`, k, value = <S4 object of class  
## structure("H2OClusteringModel", package = "h2o")>): implicit list embedding  
## of S4 objects is deprecated  
  
## Warning in `[<-`(`\*tmp\*`, k, value = <S4 object of class  
## structure("H2OClusteringModel", package = "h2o")>): implicit list embedding  
## of S4 objects is deprecated  
  
## Warning in `[<-`(`\*tmp\*`, k, value = <S4 object of class  
## structure("H2OClusteringModel", package = "h2o")>): implicit list embedding  
## of S4 objects is deprecated  
  
## Warning in `[<-`(`\*tmp\*`, k, value = <S4 object of class  
## structure("H2OClusteringModel", package = "h2o")>): implicit list embedding  
## of S4 objects is deprecated  
  
## Warning in `[<-`(`\*tmp\*`, k, value = <S4 object of class  
## structure("H2OClusteringModel", package = "h2o")>): implicit list embedding  
## of S4 objects is deprecated  
  
## Warning in `[<-`(`\*tmp\*`, k, value = <S4 object of class  
## structure("H2OClusteringModel", package = "h2o")>): implicit list embedding  
## of S4 objects is deprecated

sapply(2:12, function(x){results[[x]]@model$model\_summary$within\_cluster\_sum\_of\_squares}) %>%  
 plot(2:12, .,  
 typ='o',  
 ylab = "Within Cluster SS",  
 xlab="k clusters")



Since k-means is a non-deterministic algorithm results may vary each time the algorithm is run. This can be mitigated by using crossvalidation at the cost of additonal run-time. It does look that the Within cluster SS reaches a local minimum around 4/5/6, which is a nice low number of clusters to consider. We will consider 5 as a good choice for example. Like PCA, we can interpret the 'flavour' of the clusters by looking at the cluster centres:

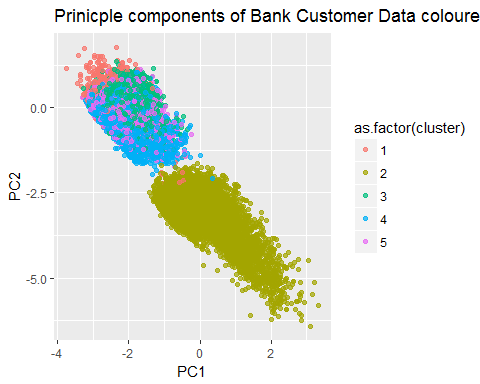
results[[5]]@model$centers

## Cluster Means:   
## centroid housing job day qualification month location  
## 1 1 yes Lawyer thu other jul Inner.city  
## 2 2 yes Lawyer mon other may Inner.city  
## 3 3 no Self-employed thu Bachelour.degree may Inner.city  
## 4 4 yes Lawyer wed other jul Inner.city  
## 5 5 yes Driver tue other may Inner.city  
## onlinebanking checkingaccount prevsales yearcontact centralinterestrate  
## 1 inactive no 0.000000 14.419492 5.833015  
## 2 inactive yes 0.211371 2.061693 2.175837  
## 3 inactive no -0.000000 2.245732 5.869076  
## 4 inactive yes 0.002934 2.284265 5.750131  
## 5 inactive no 0.000364 2.277060 5.839464  
## employmentrate customersatisfaction priceindex creditrating age  
## 1 3.394386 Satisfactory 107.848664 satisfactory 40.924788  
## 2 -1.553265 Satisfactory 107.202753 satisfactory 39.647779  
## 3 3.260364 Satisfactory 109.003955 satisfactory 36.202984  
## 4 3.041111 Satisfactory 106.994902 satisfactory 36.474997  
## 5 3.202147 Satisfactory 107.695915 satisfactory 50.672003  
## staff\_total confidenceindex emailsmonth  
## 1 15654.599576 91.026441 0.507415  
## 2 15215.908219 85.584415 0.977126  
## 3 15624.391766 93.994810 0.476689  
## 4 15656.511921 89.273214 0.580022  
## 5 15630.538657 94.554444 0.521557

What can be even more interesting is to combine the results of PCA and clustering and look at the labels of the clusters overlaid on the PCs:

cluster\_data <- as.data.frame(h2o.predict(object = results[[5]],  
 newdata = h2o\_bank\_data\_covs))  
  
clust\_plot\_data <- cbind(plot\_data,  
 cluster\_data) %>%   
 rename(cluster = predict) %>%  
 mutate(cluster = cluster + 1) #correcting so clusters match up with centroids above that start from 1.

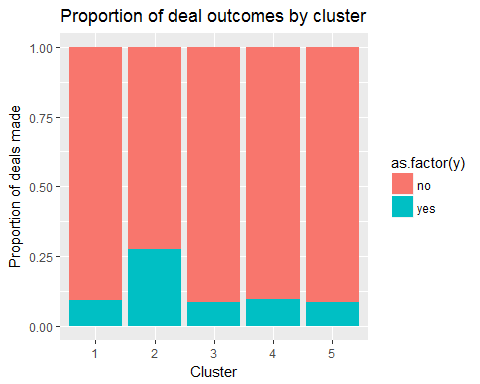
clust\_plot\_data %>%  
ggplot(aes(x = PC1, y = PC2, colour=as.factor(cluster))) +  
 geom\_point(alpha = 0.7) +  
 ggtitle("Prinicple components of Bank Customer Data coloured by cluster")



We know from the PCA plot lower half of the plot had a greater proportion of deals made. Try making this plot with different combinations of PCAs and numbers of clusters.

Summarising the proportion of deals made by cluster is also easy:

clust\_plot\_data %>%  
 ggplot(aes(x = as.factor(cluster), fill = as.factor(y))) +  
 geom\_bar(position = "fill") +   
 ggtitle("Proportion of deal outcomes by cluster") +  
 ylab("Proportion of deals made") +  
 xlab("Cluster")



* How would you summarise the deomgraphic of customers mosty likely to accept the bank's marketing offer, based on the clustering approach?

## Question 3

As we discussed in the introduction, quantitative models are frequently used to assign customers a probability of accepting deals based on data like this. A typical choice is Logistic Regression because it has well known interpretations and scales well. Inceasingly though organisations are exploring machine learning methods like Gradient Boosted Trees, Random Forests and Deep Learning (Neural Networks) in applied settings.

Platforms like h2o and spark have implmentations of these popular algorithms. Rather than go a longwinded demontration of these methods in R code, we will introduce the h2o GUI, 'H2O Flow' which can be used to experiment with these methods on big data.

To access the GUI on your local machine navigate your webbrowser to http:127.0.0.1:54321. If it is not accessible, run:

h2o\_inst <-h2o.init(ip = "127.0.0.1", port = 54321)

### H2O Flow insructions

#### Set up the Data Frame

1. *Import File* :
   * From Data Select upload the file.
   * Select Choose File
   * Navigate to chosen folder and select 'bank\_customer\_data.csv' ./data
   * Click Upload. This should automatically load Setup Parse block
2. *Parse Data* :
   * H20 Flow will automatically guess an appropriate data type for each column such as numeric or enum (enum, for enumerable, is the data type for categorical data). These can be manually changed if needed but in this case we will leave the default column types and parse configuration.
   * Select Parse to parse this file with these settings.
   * Once the Job is complete select View
3. *Split frame into Training and Validation sets*: -From Actions Select Split -Change the keys values to bank\_validation for the 0.75 split and bank\_training for the 0.25 split. -Select create. The data is now ready to be modelled

#### Set up the models

##### Neural Network

1. *Create Neural Net Model*
   * From Model select Deep learning
   * Select training data to be bank\_train and the validation data to be bank\_validation
   * Choose response\_column to be 'y' (the final variable on the list)
   * Observe that the main parameters of importance are
   * *activation* : The type of activation function used in the neural network
   * *hidden* : The number of hidden nodes arranges in there columns
   * *epoch* : The number of full iterations of the training data used for training
   * For these main variables choose activation = rectifier, hidden = 100, 100 (this means 2 hidden layers of 100 nodes each), epoch = 10
   * We wish to see how different variables impacted the overall results so tick the box next to variable importance
   * At the end of the parameters list, the remainder of which we will leave to the default for now, select Build Model

Observe the extensive 'Advanced' and 'Expert' parameters available. These are beyond the scope of this tutorial however they should give you a feeling for the complexity available in deep learning. In the H2O framework it is easy to experiment with these parameters and other data sets as an extension to this workshop.

1. *Results*
   * Once the job is finished select the action view The first two graphs are the ROC curve for the training data and Validation data effectively. The area under the curve AUC is the main indicator of performance.
   * The next graph shows the overall error for different epochs for both the training set and the validation set. We see the error has decreased from 1 to 10 epochs.
   * From the "variable importance" bar graph we get an idea of which variables were most important to the overall result. Due to the weights being initialised randomly as well as inherent noise in the data this can only be taken as an indicator rather than an exact measure. Most likely the duration will appear near the top of this list. Why would this be such an important variable? Why should we exclude it from our model?
2. *Ignoring Call Length*
   * Repeat the model fit. This can be done either by again choosing Deep learning from Model or by simply scrolling up and adjusting the previous model block.
   * Leave all parameters the same except this time choose to ignore the length column by selecting the appropriate box under ignored\_columns.
   * Observe the degraded performance on both the training and validation set. This can be seen in both the changed shape of the curve as well as the AUC value.
3. *Overfitting*
   * Repeat the model fit, leaving all parameters the same and still choosing to ignore the call duration, however now choose hidden = 100, epochs = 80. What do you expect to occur?
   * Observe in the scoring history graph that the validation score, in yellow, hits a minimum before beginning to climb again. This is a typical example of overfitting. By focusing the predictions too heavily on the training data the model does not generalise well to the validation data.
   * Notice the final reported score of MSE for training and validation is a discontinuous jump. This occurs because H2o automatically selects the model which minimises the error on the validation set. This can be avoided by unchecking the box overwrite with best mode which is the first expert parameter.

##### Gradient Boosting Machine ( or Gradient Boosted Tree)

We can compare the results of a neural Network trained on this data to a Gradient Boosted Tree model trained on this data.

1. *Create Gradient Boosted Model*
   * From Models select Gradient Boosted Machine
   * As before choose training data to be bank\_train and the validation data to be bank\_validation, choose the response variable to be y and ignore the duration column.
   * Note the following parameters of importance for a GBM
   * *Ntrees* : The number of trees created
   * *Max\_depth* : Maximum depth of each tree
   * *min\_rows*: The minimum number of observations allowed in a leaf node.
   * *Learn\_rate*: Add only a fraction of the value given by each new tree to the accumulative model.
   * Leave these parameters to the default values and build the model.
   * How does this model compare to a neural network?
2. *Overfitting with Gradient Boosting Machines* GBM models can be regularised in a number of ways. This includes limiting the allowed depth of each tree created, ensuring leaf nodes don't become too localised by requiring a minimum number of observations in each leaf, and adding only a small fraction of each new tree to the accumulative model by using a small learning rate.

* Rebuild the GBM with:
  + Max\_depth = 10
  + Min\_rows = 4
  + Learn\_rate = 0.5
  + Observe the overfitting that occurs. AUC is now very near one for the training set, degraded for the validation set and the divergence should be illustrated in the scoring history.