Machine Learning for Finance Applications - Mortgage Data

Brian Clark

2019

Contents

Introduction

This vignette describes the mortgage data used throughout the workshop. The data is the Fannie Mae Single-Family Loan Performance data freely available **here**. The data covers only fixed rate conforming mortgages sold to Fannie Mae starting in Q1 of 2000. The data is updated on a regular basis. The raw data used in this workshop is a sample of loans covering Q1:2000 through Q4:2016.

The raw data has been processed and sampled. If you want to use the data for research purposes, Fannie mae provides R and SAS files to process the raw data once it is downloaded. There are two types of data: Aquisition and Performance. The acquisition data contains information availible at the time Fannie Mae acquired the loan (e.g., credit score, LTV, DTI, loan amount, etc.). The performance data tracks the performance of the loans on a monthly basis. Therefore, the dataset provides a rich set of variables over the life of the loan including and delinquency or loss information.

The initial dataset used for this workshop has been processed and sampled. I used a combination of Python and R scripts to sample the data. The basic structure of the data is as follows. There are 1,000 randomly sampled loans for each acquisition quarter from Q1:2000 to Q4:2016. Each loan is tracked over time using the performance data to compute a delinquency variable, which is defined as the first time the loan became 180+ days past due. Each loan has a single observation so the response variable is coded as 1 or 0 (delinquent or not). The remainder of the variables are defined based on the Fannie Mae supplied definitions (see the presentations **here** and **here**).

The remiander of this document shows how the data is processed for use in various ML algorithms. Note that there are several simplifications that should be addressed before using this data for research purposes. That is, the goal of this vignette is to provide new R-users with some sample code for basic data management functions (creating variables, sorting data, describing data, plotting data, etc.). It can be easily ammended to your own application.

Data Description

First, initialize the R session and load the data.

```r
rm(list=ls()) # clear the memory
setwd("C:/Users/CLARKB2/Documents/Classes/ML Course")
library("ggplot2")
library("reshape")
library("plm")
library("rpart")
library("zoo")
library("plyr")
library("dplyr")
library("stringr")
library("reshape2")
library("ggplot2")
library("pander")
library("DataCombine")
library("plm")
library("quantmod")

# Import the mortgage data:
load("Mortgage_Annual.Rda")
```

The next step is to process the data. Within the .Rda file is a data frame called `p.mort.dat.annual`. As a matter of preference, rename `p.mort.dat.annual` as `df`. Set the data as a panel dataset (`pdata.frame()`) based on `LOAN_ID` and `year`.

```r
# Rename the data (matter of preference):
df <- p.mort.dat.annual
rm(p.mort.dat.annual)

df <- pdata.frame(df, index=c("LOAN_ID","year"),
                  stringsAsFactors = F)

# Print the class of the variable:
class(df)
```

```
## [1] "pdata.frame" "data.frame"
```

Next, we can generate variables that we need. First, define default as the first instance of 180+ days delinquent, which is given in the data by `F180_DTE` (to refer to the variable in `df`, use `df$F180_DTE`). `F180_DTE` is the date in which a loan first becomes 180+ delinquent. If the loan never becomes delinquent, it is missing (i.e., `NA`). To make the default variable, first find the indices where `df$F180_DTE == df$date` and save it as a vector, `tmp`. The line `df$def[tmp] <- 1` sets the new variable `def = 1` for the year in which the loan defaults.

```r
# Generate Variables we want:
# 1. Default 1/0 indicator (180+ DPD):
df$def <- 0
# Save the indices (rows) of
tmp <- which(df$F180_DTE == df$date)
df$def[tmp] <- 1
```

We may want to generate some other variables. For example, the variable `NUM_UNIT` gives the number of units is a house. Use `table(df$NUM_UNIT)` to print a frequency table of values of the number of units per house. Then, define a new variable to be a dummy if the number of units is greater than one (`MULTI_UN`).

```r
# 2. Replace NUM_UNIT with MULTI_UNIT dummy:
table(df$NUM_UNIT)
```

```
##
##      1      2      3      4
## 305028   6452   1051   1085
```

```
df$MULTI_UN <- 0
tmp <- which(df$NUM_UNIT > 1)
df$MULTI_UN[tmp] <- 1
```

Finally, we can compress the data down to a single observation per loan. If you wanted to conduct a time-series or panal data analysis, you would skip this step. First, print the number of unique loans.

```
# 3. Count the number of loans:
print(length(unique(df$LOAN_ID)))
```

```
## [1] 66704
```

Next, compress the data down to a single observation per loan. First, we need to generate a variable equal to one if the loan ever defaulted. We can do this using the `plm` package and grouping the data by `LOAN_ID`. Make a new dataset `df.annual` with a few additional variables: i) `def.max` and ii) `n` which is the row number to be used for keeping observaitons.

```
# Compress the data to single loans:
df.annual <-df %>%
  group_by(LOAN_ID) %>%
  mutate(def.max = max(def)) %>%
  mutate(n = row_number()) %>%
  ungroup()
```

```
## Warning: `as_dictionary()` is soft-deprecated as of rlang 0.3.0.
## Please use `as_data_pronoun()` instead
## This warning is displayed once per session.

## Warning: `new_overscope()` is soft-deprecated as of rlang 0.2.0.
## Please use `new_data_mask()` instead
## This warning is displayed once per session.

## Warning: The `parent` argument of `new_data_mask()` is deprecated.
## The parent of the data mask is determined from either:
##
##   * The `env` argument of `eval_tidy()`
##   * Quosure environments when applicable
## This warning is displayed once per session.

## Warning: `overscope_clean()` is soft-deprecated as of rlang 0.2.0.
## This warning is displayed once per session.
```

```
# Print the variable names in df.annual
names(df.annual)
```

```
##  [1] "year"              "ZIP_3"             "V1"
##  [4] "LOAN_ID"           "ORIG_CHN"          "Seller.Name"
##  [7] "ORIG_RT"           "ORIG_AMT"          "ORIG_TRM"
## [10] "ORIG_DTE"          "FRST_DTE"          "OLTV"
## [13] "OCLTV"             "NUM_BO"            "DTI"
## [16] "CSCORE_B"          "FTHB_FLG"          "PURPOSE"
## [19] "PROP_TYP"          "NUM_UNIT"          "OCC_STAT"
## [22] "STATE"             "MI_PCT"            "Product.Type"
## [25] "CSCORE_C"          "MI_TYPE"           "RELOCATION_FLG"
## [28] "Monthly.Rpt.Prd"   "Servicer.Name"     "LAST_RT"
## [31] "LAST_UPB"          "Loan.Age"          "Months.To.Legal.Mat"
## [34] "Adj.Month.To.Mat"  "Maturity.Date"     "MSA"
## [37] "Delq.Status"       "MOD_FLAG"          "Zero.Bal.Code"
## [40] "ZB_DTE"             "LPI_DTE"            "FCC_DTE"
## [43] "DISP_DT"            "FCC_COST"           "PP_COST"
## [46] "AR_COST"            "IE_COST"            "TAX_COST"
## [49] "NS_PROCS"           "CE_PROCS"           "RMW_PROCS"
## [52] "O_PROCS"            "NON_INT_UPB"        "REPCH_FLAG"
## [55] "TRANSFER_FLAG"      "CSCORE_MN"          "ORIG_VAL"
## [58] "PRIN_FORG_UPB"      "MODTRM_CHNG"        "MODUPB_CHNG"
## [61] "Fin_UPB"            "modfg_cost"         "C_modir_cost"
## [64] "C_modfb_cost"       "Count"              "LAST_STAT"
## [67] "lpi2disp"           "zb2disp"            "INT_COST"
## [70] "total_expense"      "total_proceeds"     "NET_LOSS"
## [73] "NET_SEV"            "Total_Cost"         "Tot_Procs"
## [76] "Tot_Liq_Ex"         "LAST_DTE"           "FMOD_DTE"
## [79] "FMOD_UPB"           "FCE_DTE"            "FCE_UPB"
## [82] "F180_DTE"           "F180_UPB"           "VinYr"
## [85] "ActYr"              "DispYr"             "MODIR_COST"
## [88] "MODFB_COST"         "MODTOT_COST"        "d.HPI"
## [91] "date"               "n"                  "n.obs"
## [94] "n.year"             "n.year.max"         "def"
## [97] "MULTI_UN"           "def.max"
```

Finally, we can save only one observation per loan.

```
# keep one obs per loan:
tmp <- which(df.annual$n == 1)
df.annual <- df.annual[tmp,]
dim(df.annual)
```

```
## [1] 66704    98
```

Notice that the number of rows is equal to the number of unique loans as shown above. Now, retain only the variables needed for the analysis.

```
# Keep only relevant variables for default analysis:
my.vars <- c("ORIG_CHN","ORIG_RT",
             "ORIG_AMT","ORIG_TRM","OLTV",
             "DTI","OCC_STAT",
             "MULTI_UN",
             "CSCORE_MN",
             "ORIG_VAL",
             "VinYr","def.max")
df.model <- subset(df.annual,select=my.vars)
names(df.model)
```

```
##  [1] "ORIG_CHN"  "ORIG_RT"   "ORIG_AMT"  "ORIG_TRM"  "OLTV"
##  [6] "DTI"       "OCC_STAT"  "MULTI_UN"  "CSCORE_MN" "ORIG_VAL"
## [11] "VinYr"     "def.max"
```

```
# Print the number of defaults/non-defaults
table(df.model$def.max)
```

```
##
##     0     1
## 64397  2307
```

```
tmp <- table(df.model$def.max)
df.rate <- tmp[2]/sum(tmp)*100
message(sprintf("The default rate is: %4.2f%%",df.rate))
```

```
## The default rate is: 3.46%
```

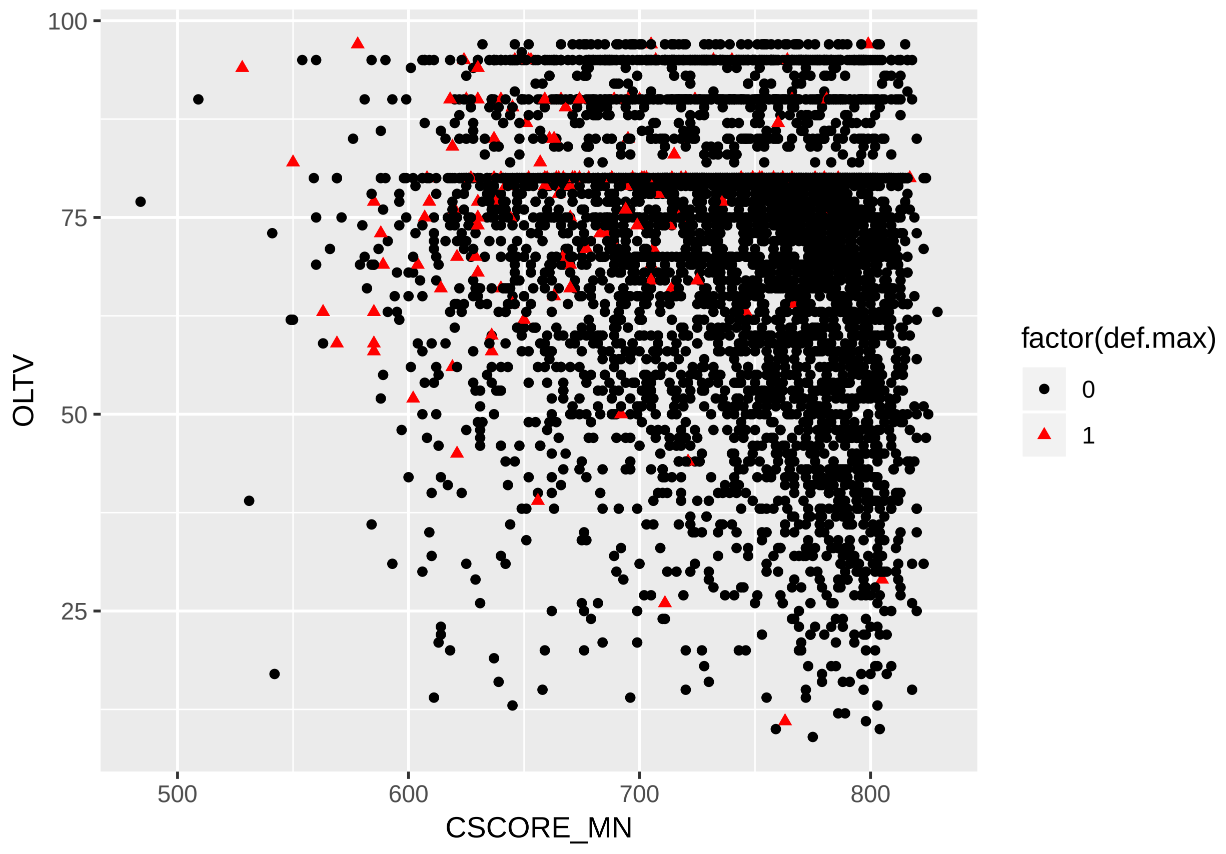
The last line prints the default rate. You can control the formatting just as you would in Matlab. The next step is to plot the data. We will use the `ggplot2()` package.

```r
# --------------------
# Plot the data:
# Set the colors for the points:
mycolor <- c("black","red")

# Generate a small sample for plotting:
df.model.small <- df.model[sample(dim(df.model)[1],5000),]

# Plot Defaults vs. CSCORE_MN and OLTV:
sp <- ggplot(df.model.small, aes(x=CSCORE_MN, y=OLTV,
                                 color=factor(def.max))) +
  geom_point(aes(shape=factor(def.max))) +
  scale_fill_manual(values=mycolor) +
  scale_colour_manual(values=mycolor)
```

The above code saves the plot object as `sp`. To show the plot, simply print it. `ggplot()` has a nice feature that we can simply add to plots (similar to `hold on` in Matlab) as follows. As an example, we can make the size of the plotted points larger for defaulted loans (i.e., make the red dots bigger).

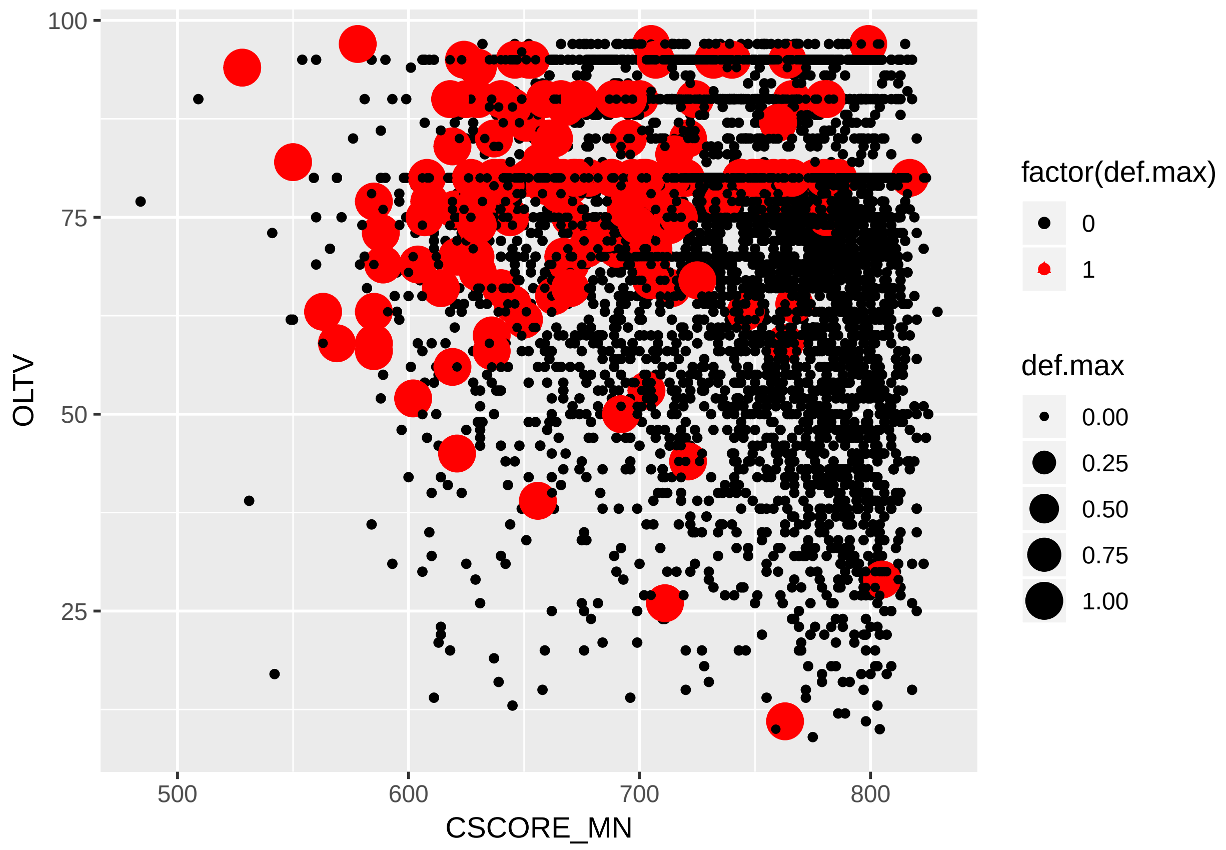
```r
print(sp)
```

```
## Warning: Removed 16 rows containing missing values (geom_point).
```



```r
# Add some new formatting:
sp <- sp + geom_point(aes(size=def.max))
print(sp)
```

```
## Warning: Removed 16 rows containing missing values (geom_point).

## Warning: Removed 16 rows containing missing values (geom_point).
```



Notice that the defaults tend to cluster more hevily for higher original LTV's and lower credit scores (note that `CSCORE_MN` is the minimum credit score of the borrowers). Finally, we can remove all unnecessary ojects.

```r
# Print the objects in memory:
ls()
```

```
## [1] "df"               "df.annual"       "df.model"        "df.model.small"
## [5] "df.rate"          "my.vars"         "mycolor"         "sp"
## [9] "tmp"
```

```r
# Remove all but df.model
rm(list=setdiff(ls(), "df.model"))
ls()
```

```
## [1] "df.model"
```

Now we only have a single object in memory - `df.model`. Note that in many ML applicaitons, memory becomes crucial. For this example, the data is pretty small so memory isn't a concern but in "big data" applications, the above commands could prove useful.

References