Part 1

Florin Andrei

10/15/2020

Part 1

Introduction

We are going to use logistic regression to predict which applicants are likely to default on their bank loans. The dataset is a sample of 50k loans, with 30 variables. The dataset file, along with a description of the variables, can be found here:

https://datascienceuwl.github.io/Project2018/TheData.html

The response variable ("fate") is constructed from the "status" variable in the dataset. "fate" has only two possible values: Good or Bad. Good loans are those with status = "Fully Paid". Bad loans are those with status = c("Charged Off", "Default"). All other status values are irrelevant and will be removed.

Preparing and cleaning the data

Let's load the data:

```
dsfile_url <- 'https://datascienceuwl.github.io/Project2018/loans50k.csv'
dsfile <- 'loans50k.csv'
if (!file.exists(dsfile)) {
  download.file(dsfile_url, destfile = dsfile)
}
loans <- read_csv(dsfile)
cat('Number of loans at this point:', nrow(loans))

## Number of loans at this point: 50000
```

Preparing the response variable

Some categories in the "status" variable are irrelevant, and we're going to remove them. The remainder will be collapsed into two main categories: Good and Bad, contained in the new "fate" column. The "status" column is not needed after this, so we'll drop it.

```
## Number of loans at this point: 34655
```

Eliminating useless predictors

The sample is large enough (N = tens of thousands) that correlations between the response variable ("fate") and the various other variables should be visible already, if there is any

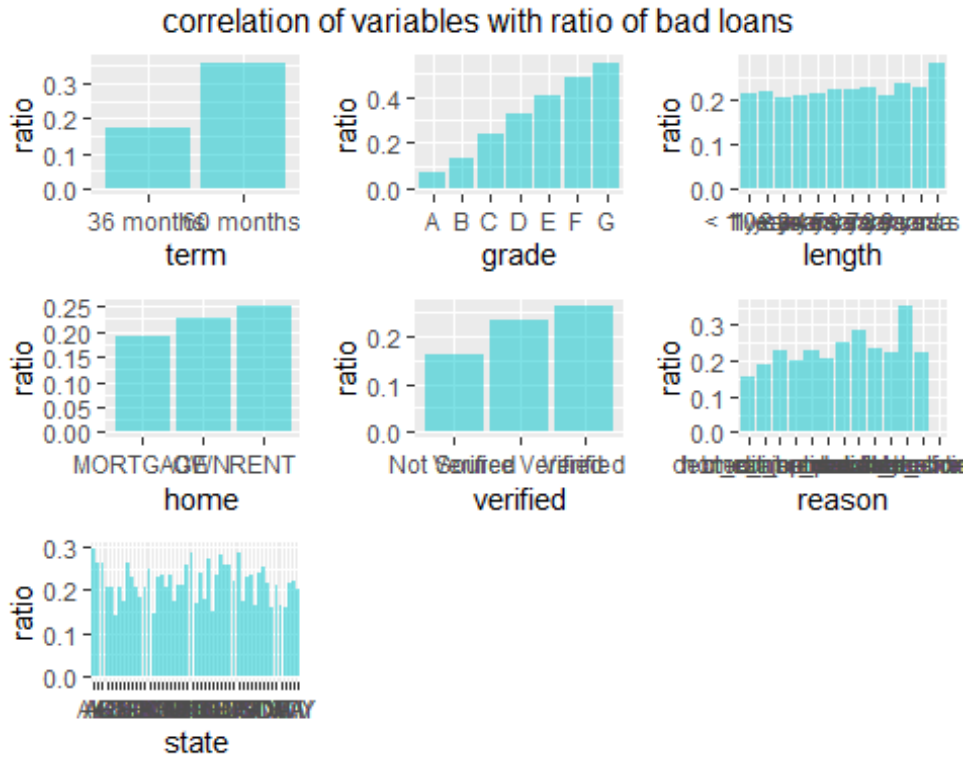
correlation. We're going to try and find out if the ratio Bad / (Good + Bad) loans is correlated with any categories (for categorical variables) or range buckets (for numeric variables).

Variables that show no correlation with "fate" are going to be dropped.

Categorical variables

employment has far too many little categories that seem to be quite ad-hoc and only have a few cases each. I think it's just noise, it doesn't seem to contribute useful data. I am going to drop it.

Let's visualize the ratio Bad / (Good + Bad) loans for all categories:



- length doesn't seem strongly correlated with good-vs-bad loans.
- term, grade, and state are strongly correlated. grade is **very** strongly correlated (to me it looks close to the solution to the prediction problem, right there).
- home, verified, and reason are mildly correlated.

Useless vars to remove so far:

```
# we cannot remove totalPaid because it will be used for the model
# we'll just ignore it for now
vars2remove <- c('employment', 'length')
```

Numeric variables

Here are the ratios of Good / Bad (blue / pink) loans as a function of each numeric variable:

Some histograms indicate clear correlation, but there are some cases where it's not clear if that column contributes to the outcome. So let's run cor.test() on those:

```
## column    p-val          conf int

## amount    1.348705e-22    -0.06300456 -0.04200555
## payment   1.842251e-07    -0.03852424 -0.01748369
## delinq2yr     0.001422763     -0.0276588 -0.006607913
## inq6mth   4.866885e-38    -0.07965343 -0.05869713
## log10_totalAcc    0.02825796      0.001255607 0.02230975
## log10_totalLim    8.997562e-45    0.06483269 0.08577034
## revolRatio    1.078567e-28    -0.0701477 -0.04916104
## totalBal      1.731052e-40    0.0609895 0.08193901
## avgBal    1.065168e-52    0.07147006 0.09238577
## bcRatio   9.664947e-33    -0.07484448 -0.05375733
## totalRevBal   0.6061615   -0.007759196 0.01329771
```

Clearly can be removed:

- totalRevBal

Perhaps could be removed:

- totalAcc
- delinq2yr

I choose to be conservative at this point, I will only eliminate totalRevBal.

```
vars2remove <- c(vars2remove, 'totalRevBal')
```

Feature engineering

The reason variable has a couple categories that have only a few cases each, not enough to draw conclusions at any reasonable confidence level. We will lump those together as "other".

Missing values

summary(loans) reveals that several columns have 360 NA entries:

- revolRatio
- bcOpen
- log10_bcOpen
- bcRatio

There is no way to infer these from other data:

- each row in the table is independent

- those columns does not seem like they could be deduced from other columns - if they could, then we ought to remove them altogether anyway

No other columns contain NAs at this point. So we will drop the NAs.

```
## Number of loans at this point: 34271
```

Exploring and Transforming the data

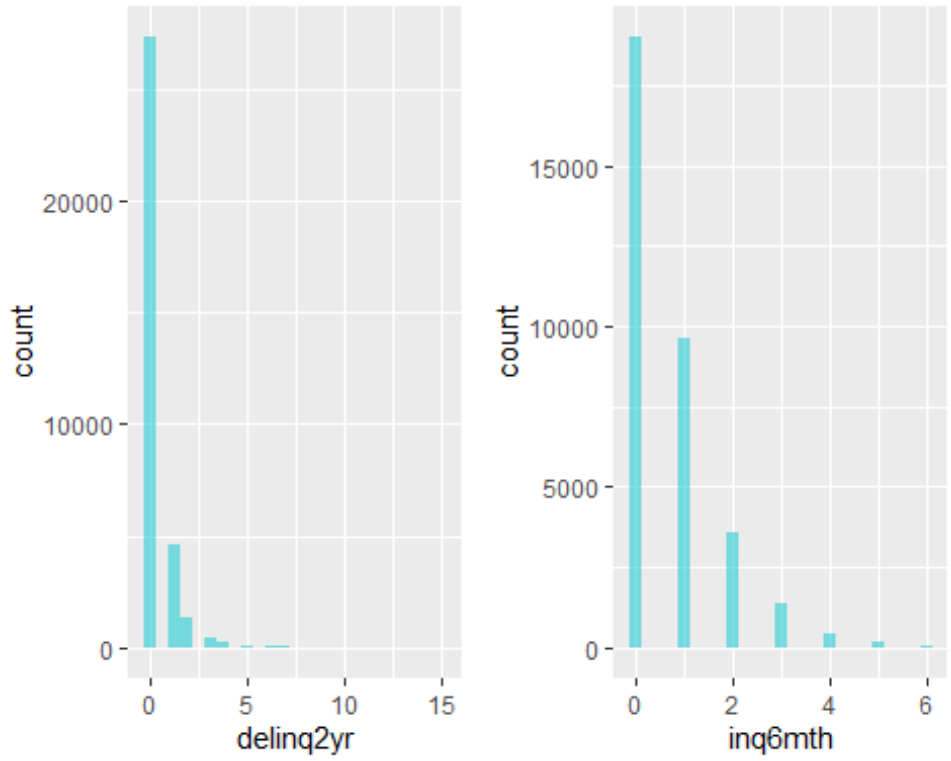
Fixing skewed variables

Some quantitative variables have strong right skew. We're going to remove the skew with sqrt() (mild skew) or log10() (strong skew).

For log10() transforms, if 0 is an outlier in the original data, then 0 will be replaced with 1 to avoid log10(0). If 0 is not an outlier (it's a common value), then sqrt() will be used instead (less efficient but safer).

```
loans <- loans %>%
  mutate(rate = sqrt(rate)) %>%
  mutate(payment = sqrt(payment)) %>%
  mutate(debtIncRat = sqrt(debtIncRat)) %>%
  mutate(pubRec = sqrt(pubRec)) %>%
  mutate(accOpen24 = sqrt(accOpen24)) %>%
  mutate(totalIlLim = sqrt(totalIlLim))

loans <- loans %>%
  mutate(income = log10(income)) %>%
  mutate(openAcc = log10(openAcc)) %>%
  mutate(totalAcc = log10(totalAcc)) %>%
  mutate(totalRevLim = log10(totalRevLim)) %>%
  mutate(totalLim = log10(totalLim))

loans <- loans %>%
  mutate(avgBal = replace(avgBal, avgBal == 0, 1)) %>% mutate(avgBal = log10(avgBal)) %>%
  mutate(totalBal = replace(totalBal, totalBal == 0, 1)) %>% mutate(totalBal = log10(totalBal)) %>%
  mutate(bcOpen = replace(bcOpen, bcOpen == 0, 1)) %>% mutate(bcOpen = log10(bcOpen)) %>%
  mutate(totalBcLim = replace(totalBcLim, totalBcLim == 0, 1)) %>% mutate(totalBcLim =
log10(totalBcLim))
```

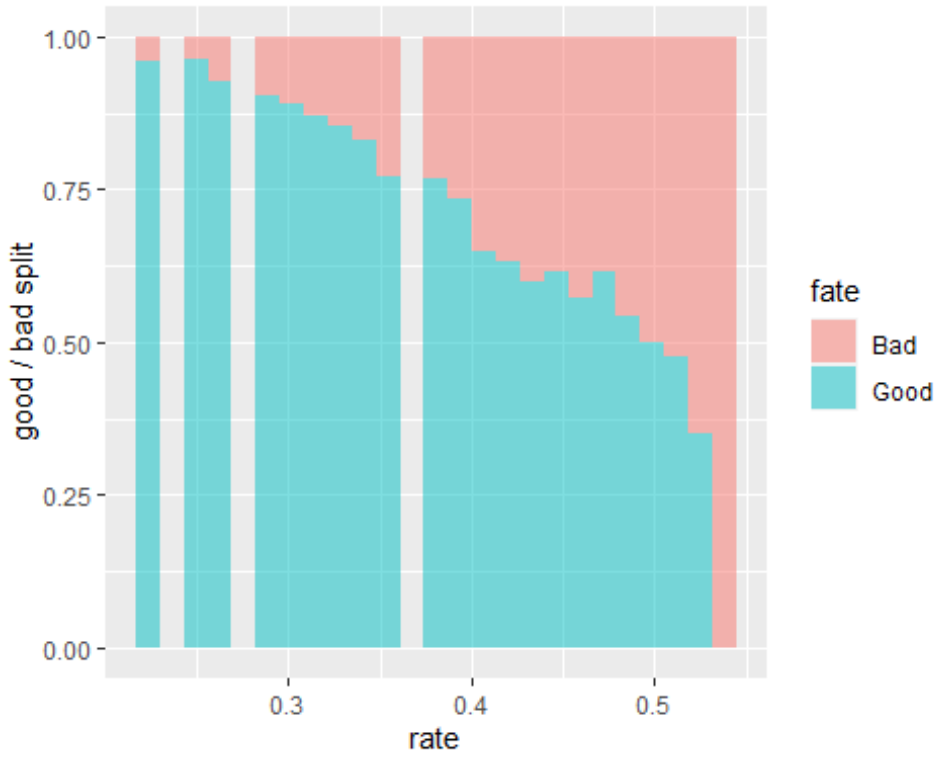
A few variables still show very strong skew even after log10(). Perhaps these should be dropped. (?)



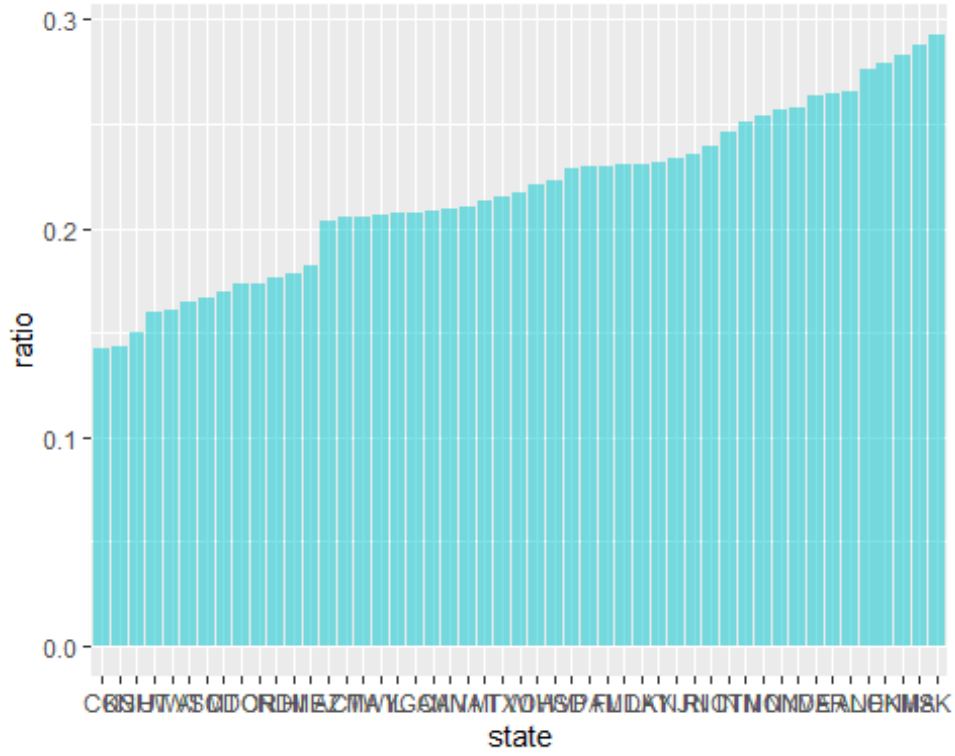
Data exploration

I've done a bit of this before, in the section **Eliminating useless predictors**. In fact, in that section we've eliminated all variables that did not seem to show different behavior for Good vs Bad loans. But let's sample a few variables again, after all these changes.

As an example of numeric variable, "rate" shows marked differences between Good and Bad loans. At low "rate" values, there are far more Good loans. At high "rate" values, the Bad loans rise to the point where they become equal to Good (this is intuitive behavior - it's harder to pay off loans at higher rates).



Among categorical variables, “state” shows some correlation as well. In some states (such as CO), the Bad loans are only about 14% of the total, whereas other states (such as AK) show as much as 28% Bad loans (twice as much as the other end of the scale).



(the figure ought to be stretched sideways a bit to show the state names more clearly)

All remaining variables actually show some correlation with the proportion of Bad loans, but it's hard to display all of them without taking too many pages in this document. The samples shown above are typical.

At this point the data frame should have no NA, no -Inf, the variables should not show strong skew (usually).

Part 2

The Logistic Model

Let's split the dataset into subsets, training (to create the model) and validation (to test the model), with an 80/20 split.

I tried to use step() to optimize the model, but it doesn't change my conclusions, so I abandoned it. I will just use the full model (all variables).

Build the full model:

```
model.full = glm(fate_num ~ ., data = loans.train, family = 'binomial')
```

Predict from test data:

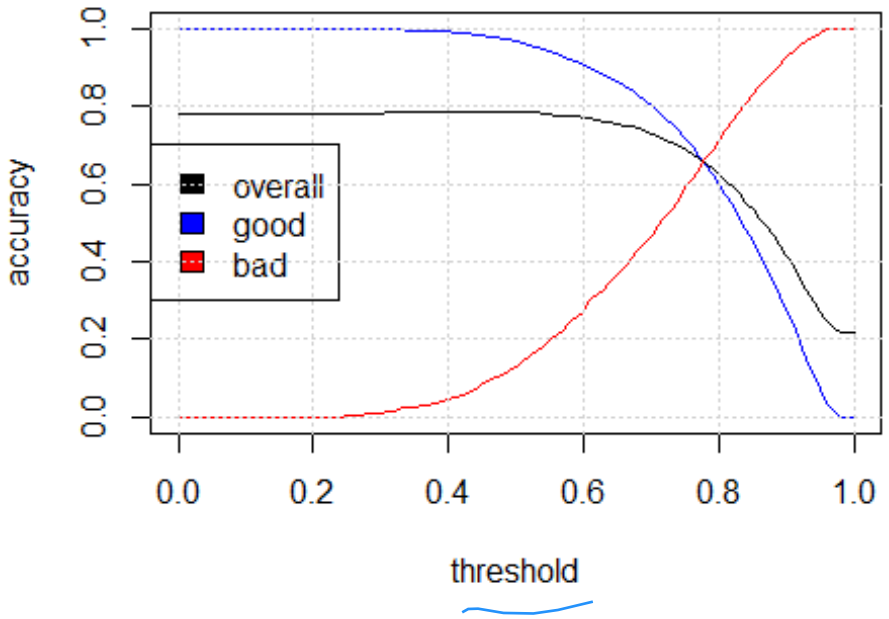
```
fitted <- predict(model.full, loans.test, type = 'response')

threshold <- 0.5

accuracy.test <- function(threshold, real, predict.output) {
  predicted <- cut(predict.output, breaks = c(-Inf, threshold, Inf), labels = c('Bad', 'Good'))
  contable <- table(real, predicted)
  accuracy.overall <- sum(diag(contable)) / sum(contable)
  accuracy.bad <- contable[1, 1] / sum(contable[1,])
  accuracy.good <-  contable[2, 2] / sum(contable[2,])
  acc <- list(overall = accuracy.overall, good = accuracy.good, bad = accuracy.bad, contable = contable)
  return(acc)
}

m.a <- accuracy.test(threshold, loans.test$fate, fitted)

## Contingency table:

##       predicted
## real    Bad Good  Sum
##   Bad   194 1301 1495
##   Good  180 5180 5360
##   Sum   374 6481 6855

##
## Model accuracy:

## overall accuracy   : 0.7839533

## good loans accuracy: 0.9664179

## bad loans accuracy : 0.1297659
```

78% overall accuracy is definitely better than a coin flip, but seems underwhelming given the wealth of data we have on the applicants. The main reason is - the model fails with bad loans, a lot. It does very well with good loans. That's a strong indication that the model is

too optimistic - it will give the thumbs up to loans that should never have passed careful scrutiny.

Optimizing the Threshold for Accuracy

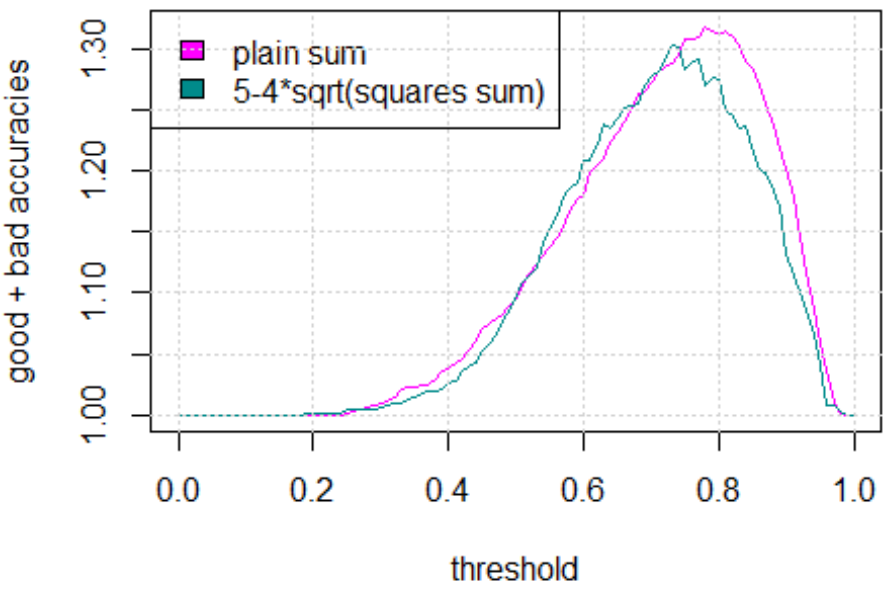
Let's sweep the (0, 1) range of thresholds to see how the model behaves at different cutoff points.



Something fascinating happens around threshold = 0.78 - all accuracy graphs converge. I'm sure there's a theoretical justification for it, I just draw a blank when I try to explain it.

The definition I have here for "overall" accuracy (maximize the total number of correct predictions) is naive and cannot be used in practice. It would be achieved by a naive model that approves all loans (all loans are "Good"). This is probably due to an imbalance between loan categories (more "good" than "bad" loans in the data) - well, that plus the fact that the model is just not very accurate, period. Perhaps filtering the data for a more balanced population ("good" = "bad") would fix it? I'd love to try it but I really do not have the time.

A better metric would consider good loans accuracy and bad loans accuracy, together, and ignore my naive "overall" metric. Perhaps try to maximize the sum of good + bad accuracies, or minimize the sqrt() of the sum of squares.



Very creative approach to consider other metrics of composite accuracy ✓

All sorts of interesting things happen - like, there are ripples in the graphs, local extremes which may question the legitimacy of the overall extreme.

Anyway, the best "overall" accuracy happens in the 0.75 ... 0.8 range. The initial 0.78 estimate is not bad. An exact value with many decimals could be obtained via binary search or something, but I think the model is not precise enough to justify it.

I choose 0.78 as the threshold for the best model accuracy.

I dislike the notion that a simple "overall" accuracy (true positives + true negatives / total loans) is a good indicator for the performance of the model. But the project instructions requires us to compute it at the chosen best threshold, so here it is: ✓

```
accuracy.test(0.78, loans.test$fate, fitted)$overall

## [1] 0.655434
```

The overall accuracy at the chosen threshold is 65.5%. The "good" and "bad" accuracies are close to that value, within 1 or 2%. The reason to choose this as the "best" accuracy is that at any other value either "good" or "bad" loan accuracies drop off very quickly.

This way we keep a balance between correctly estimating "good" loans and "bad" loans. But I feel something is fundamentally wrong with the model, as long as the simple overall accuracy does not have a global maximum away from the extremes of the threshold values. The graph of overall accuracy should not be a sigmoid, it should be a bell curve. Something happened in training that lead to this compromise threshold (imbalance between "good" vs "bad" loans in training data?).

Optimizing the Threshold for Profit

Let's get a baseline by computing the profit in the absence of any model - just subtract amount from totalPaid for all loans in the test data, paying no attention to the ultimate fate of the loan ("good" or "bad"):

```
profit.no.model <- sum(loans.test$totalPaid - loans.test$amount)
cat('Total profit when no model is applied: ', profit.no.model, '\n')

## Total profit when no model is applied:  1807687
```

...which is about $1.8M

Now let's get the ceiling for profit by computing it using the perfect "model" (we deny all loans that actually turned "bad"):

```
loans.test.perfect.model <- loans.test[which(loans.test$fate == 'Good'),]
profit.perfect.model <- sum(loans.test.perfect.model$totalPaid - loans.test.perfect.model$amount)
cat('Total profit when a perfect model is applied: ', profit.perfect.model, '\n')

## Total profit when a perfect model is applied:  12660986
```

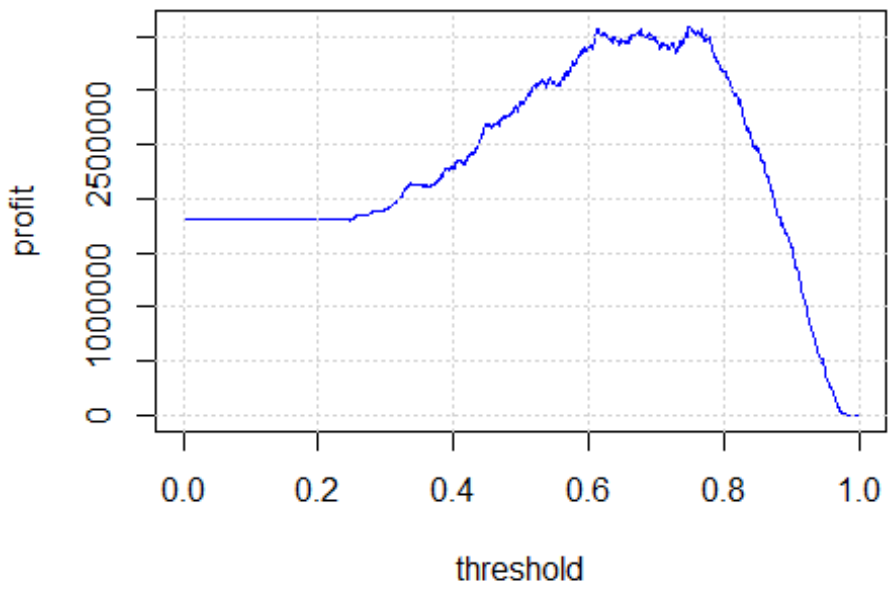
...which is about $12.7M

And now let's compare all that with the predictions of the model created in the previous section, at the "best" threshold estimate for accuracy. We will reject the loans that the model predicts to be "bad" and compute the profit that way.

```
loans.test.my.model <- loans.test
loans.test.my.model$modelPredictNum <- predict(model.full, loans.test.my.model, type = 'response')
loans.test.my.model$modelPredict <- cut(loans.test.my.model$modelPredictNum, breaks = c(-Inf, 0.78,
Inf), labels = c('Bad', 'Good'))
loans.test.my.model <- loans.test.my.model %>% select(-c('modelPredictNum'))
loans.test.my.model <- loans.test.my.model[which(loans.test.my.model$modelPredict == 'Good'),]
profit.my.model <- sum(loans.test.my.model$totalPaid - loans.test.my.model$amount)
cat('Total profit when our estimated model is applied: ', profit.my.model, '\n')

## Total profit when our estimated model is applied:  3441175
```

...which is about $3.4M. It's not a big increase over the baseline.

Let's scan the (0, 1) interval to find the threshold that maximizes profit:




```
## Maximum profit, and the threshold for it:

##     threshold  profit
## 750     0.749 3587831
```

The maximum profit of about $3.59M is achieved at filtering threshold = 0.749

Compared to not using any model at all, our model almost exactly doubles the profit (factor of 1.98x, or 98% increase). However, this is far behind a perfect model, which would increase profit (compared to the "flying blind" baseline) by a factor of 7x (600% increase).

The threshold optimized for profit (0.749) is quite close to the threshold optimized for accuracy (0.78), which should not be surprising (assuming a decent-enough model). At the max-profit threshold, the prediction accuracies are:

```
## $overall
## [1] 0.6907367
##
## $good
## [1] 0.7190299
##
## $bad
## [1] 0.5892977
##
## $contable
##       predicted
## real    Bad Good
```

```
##   Bad   881  614
##   Good 1506 3854
```

Results Summary

The classification model uses almost all variables in the initial dataset. The response variable fate is based on the status variable in the dataset: status = 'Fully Paid' becomes fate = 'Good', status = c('Charged Off', 'Default') becomes fate = 'Bad', all other status lines are removed from data. fate_num is generated from fate, with 1 meaning 'Good' and 0 meaning 'Bad' loans.

Several independent variables are removed: employment, length, totalRevBal. The reason variable has a few rarely occurring values which are lumped together as 'other'. drop_na() is used on the remaining data, after which the number of loans in the data is about 34k. Skewed variables are treated with sqrt() or log10().

The remaining dataset is randomly split 80/20 into train/test sets. A generalized logistic model is then run on all variables in the train dataset.

```
model.full = glm(fate_num ~ ., data = loans.train, family = 'binomial')
```

When running prediction with this model on the test data, the output can be fine tuned by changing the threshold that separates 'Good' from 'Bad' predicted loans. **The threshold value that maximizes profit is 0.749**. At this threshold value, **the profit is $3.59M**, which is a 98% increase over the baseline (no model) profit of $1.8M.

The overall model accuracy for best profit is 69%, with true positives at 72% and true negatives at 59%.

This is slightly different from the threshold for best accuracy, which is at 0.78 - and then the overall accuracy is around 65%, which is also equal to the true positive and true negative rates. The increase in profit when moving the threshold down to 0.749 is likely due to a better true positives rate, which more than compensates for the decrease in the true negatives rate.