LendingClub Loans Pricing

HarvardX - PH125.9x Data Science Capstone

Emmanuel Rialland - https://github.com/Emmanuel_R8

December 08, 2019

Contents

List of Tables

List of Figures

TODO

0.1 Dataset

[TODO: DTI, amount... by grade]

0.2 Model preparation

[TODO: Bias/Complexity trade-off] [TODO: AoC? or AuC?]

Correlation matrix?

Introduction

Lending Club (LC) is an American company listed on the New York stock exchange that provides a platform for peer-to-peer lending. Unlike banks, it does not take deposits and invest them. It is purely a matching system. Each loan is split into $25 that multiple investors can invest in. LC is remunerated by fees received from both sides. LC states that they have intermediated more than $50bln since they started operations. Further description of the company is easily available online numerous sources.

In order for investors to make the best investment decisions, LC make a historical dataset publicly available to all registered investors. This dataset is the subject of this report. It was downloaded from the Kaggle data science website[1].

The size of the dataset is rich enough that it could be used to answer many different questions. We decided for a focused approach. Following Chapter 5 of (Peng, 2012), we will first formulate the question we want to answer to guide our analysis.

The business model of LC is to match borrowers and investors. Naturally, more people want to receive money than part with it. An important limiting factor to LC's growth is the ability to attract investors, build a trusting relationship where, as a minimum first step, investors trust LC to provide accurate, transparent and reliable information of the borrowers. For this purpose, LC decided not only to provide extensive information about potential borrowers' profile, but also historical information about past borrowers' performance. This is, as we understand, one of the key purposes of this dataset. We decided to use the dataset for this very purpose. Essentially, the questions are: **given a borrower profile, is his/her rating appropriate in terms of risk of default? And if a default occurs, what is the expected recovery? The summary question is: given a borrower profile, is the risk/reward balance appropriate to commit funds?** In answering this question, we understand that LC allows investment of very granular amounts. Therefore, even an individual investor can diversify his/her loan and risk portfolio. It is not necessary to 'gamble' funds on a single borrower. This is exactly what institutional investors achieve through syndication (although on a very different scale, typically $10-25mln for a medium-size bank).

For this exercise, we made two simplifying (hopefully not simplistic) assumptions:

- In determining the risk/return balance, we have not accounted for LC's cost of intermediation. By ignoring fees paid by both sides, we obviously overestimate the returns to the investors. But in first approximation, **we will assume that the risk/reward balance, from the investors' point of view, across ratings is independent from fees.** This is a simplification. Real-world fees are higher the lower the investment grade and push the investors to receive, and the borrowers to pay, higher interest margin.
- All-in interest rates paid by borrowers are fixed. This is highly desirable for borrowers to be able to manage their cashflow. However, an investor should always consider an investment

[1] https://www.kaggle.com/wendykan/lending-club-loan-data/data

return as a margin above a risk-free return. Banks would look at LIBOR; bond investors (e.g. life insurers) would look at government bonds. Those risk-free rates can change very quickly, whereas we understand that LC sets those rates on a less frequent basis. In other word, the risk premium will vary rapidly. **We assume that individual investors are 'in-elastic' to change in implied risk premia.** But we recognise this as a limitation of our work.

This report is organised as follows:

- [XXXX]

Chapter 1

Internal Rate of Return, Credit Margins and Net Present Values

In this section, we describe the response variables that we will generate for each loan and that will be used in the rest of this report.

As indicated in the introduction, our purpose is to test a model that predicts the financial risk of a loan.

1.1 Important warning

The calculations presented here are simplistic, although they bear some resemblance to what financial institutions (*FIs*) do. The literature on credit assessment and pricing is very rich and very complex. Finding the optimal capital allocation to particular risks while at the same time satisfying internal risk policies and regulatory requirements is a problem that financial institutions have yet to solve in full. Investing in a loan is not only a matter of assessing the risk of a particular borrower, but also assessing systemic risks (which exist across all borrowers), risks associated with funding the loan (interest, currency and liquidity markets), each requiring a risk assessment and pricing.

In other words, nobody would, let alone should, make any investment decision based on the calculations below.

1.2 Background

This subsection can be skipped by anybody with basic financial knowledge.

A bird in hand or two in the bush; a penny today or a pound tomorrow. What is the price of delaying obtaining and owning something? This is what pricing a loan is about. A lender could keep his/her cash in hand, or lend it and have it in hand later. He/she would accept this in exchange for receiving a bit more: this is the rate of interest. A lender wants to be compensated for delaying the possibility of using the cash, but also for taking the risk of not receiving it, partially or in full, when repayment is due.

There are borrowers that one can see as (almost) completely safe or risk-free such as central banks or governments of strong economies. A lender always has the possibility to lend to them instead of more risky borrowers. Therefore, a lender would require a higher interest rate than risk-free. The additional interest that a lender requires is commensurate with the risk of the borrower not repaying (called *credit worthiness*) and is called the *credit margin*.

For each individual borrower, an FI would assess information provided by the borrower and massive amounts of historical data to answer the question: considering historical borrowers with a profile similar to the applicant's, what is the probability of not getting principal and interest back (*Probability of Default* or *PD*)? And, in case the borrower stops paying and using additional courses of action (such as seizing and selling assets), what is the total loss that could be expected on average (*Loss given Default* or *LGD*)?

Making that assessment, the FI would require an interest rate which would roughly be the sum of:

- the risk-free rate;
- a margin to cover the average loss of similar individual borrowers[1];
- a margin to cover all the operational costs of running their operations; and,
- a margin to remunerate the capital allocated by the FI (banking regulations require all banks to allocate an amount of capital against any risk taken; those are stipulated in a number of complex rules).

Said crudely, this total is the amount for the FI to get out of bed and look at a loan. Although this sounds like an exact science (for some definition of the word), it is not. At the end of the day, the FI will also have to contend with the competition from other FIs or non banking lenders, market liquidity (if there is a lot of money available to be lent, it brings prices down) and, critically, whether the borrower would at all be interested in accepting that cost.

Note that the dataset is distorted by this additional survival effect: the application information of many loans does not appear merely because the rate of interest was considered too high (this is not dissimilar to *survival effects* where some data did not survive through the history of a dataset[2]).

1.3 Internal Rate of Return

For the purpose of this report, we will simplify things enormously: we will only consider the first two components of the interest rate. The risk-free rate and the credit margin that would cover the cost of default/losses of individual borrowers.

With respect to a given loan and its cash flow, two calculations are important here: the Net Present Value (NPV) and the Internal Rate of Return (IRR). If we remember that an FI is indifferent to holding a principal P today or receiving it with an annual interest a year later (i.e. $P \times (1+r)$ where r is the annual rate of interest), we can say that any amount CF_1 received in a year is equivalent to $CF_0 = \frac{CF_1}{1+r}$ today. More generally, a steam of future cash receipts is worth:

$$NPV(r) = \sum_{Yeari=1}^{Year=n} \frac{CF_i}{(1+r)^i}$$

The amount $NPV(r)$ is called the *Net Present Value* of the cash flow discounted at the rate r. Given that the LendingClub repayments are monthly, the formula becomes:

[1] It is important to realise that the average margin only brings the borrower back to having earned the risk-free rate average: the additional income from the credit margin will be spent to cover average losses. In addition, we present the credit margin as income against borrower-specific losses. It does not address a lot of other risks such as correlation risks: a borrower might default because the economy as a whole gets worse, in which case many borrowers will default. This is a cyclical *systemic* risk similar to the 2007 US real estate crisis.

[2] A well-known example is historical stock prices which disappear when companies are de-listed or go bankrupt.

$$NPV(r) = \sum_{Month_i=1}^{Month_i=12\times n} \frac{CF_i}{(1+\frac{r}{12})^i}$$

If we now have a day 1 cash flow CF_0, we can calculate:

$$CF_0 - \sum_{Yeari=1}^{Year=n} \frac{CF_i}{(1+r)^i}$$

However, for any given CF_0, there is no reason that it would equal the NPV of the future cash flow (i.e no reason why the difference would be equal to zero). But this is an equation depending on r. If we can find a value of r that zeroes this formula, it is called the internal rate of return of the cash flow:

$$CF_0 - \sum_{Yeari=1}^{Year=n} \frac{CF_i}{(1+IRR(CF))^i} = 0$$

or for monthly cash flow:

$$CF_0 - \sum_{Monthi=1}^{Month=12\times n} \frac{CF_i}{(1+\frac{IRR(r)}{12})^i} = 0$$

1.4 Dataset calculation

For each loan. we calculated the IRR, credit margin and NPV. The calculations were performed in Julia due to R's slow performance on such a large dataset. For this reason, the resulting datasets are available on the GitHub repository as gzipped CSV files.

1.4.1 IRR

We used the dataset to calculate the IRR of each loan. We used the following information for the dataset: `funded_amnt`(loan amount funded), `int_rate` (all-in interest rate), `term` (tenor of the loan in months), `total_pymnt` (total cumulative amount received from the borrower), `total_rec_prncp` (amount repaid allocated to principal repayment), `total_rec_int` (amount repaid allocated to interest payment), `recoveries` (any amount recovered later from the borrower) and `total_rec_late_fee` (any late payments fees paid by the borrower).

From that information, we recreated a cash flow for each loan. The R code is presented in appendix. Unfortunately, this code takes close to a full day to run on the entire dataset of completed loans (i.e. excluding all ongoing loans). This is just impractical for anybody to run to check this report and the resulting IRR results dataset is included in the Github repository. To make things practical, the dataset was actually created using code in Julia[3]. It is a direct translation of the R code, with a similar syntax (therefore very easy to follow). The Julia code runs about 500 times quicker (this is not a typo), or about 3 minutes. We appreciate that this is the departure from the assignment description.

Similarly, we calculate the credit margin required by each loan noting that:

$$\text{Risk-free} + \text{Credit Margin} = IRR(loan)$$

[3] https://julialang.org/

1.4.2 Credit Margins

As noted in the previous section, risk-free rates change over time. When solving for the credit margin, we use the relevant risk-free rate.

Again, this was coded in Julia. The Julia code here takes about 1h20min to run. On the assumptions that the equivalent R code would take therefore almost 30 days to run through the full dataset, we did not write any R code for this calculation. The code is in appendix, and we believe easy to follow.

1.4.3 NPV

We also calculated the NPV of each loan. Again, this was coded in Julia. The code is in appendix.

We calculate both the NPV as an absolute dollar amount and as a portion of the original

Chapter 2

Dataset

The data is sourced as a *SQLite* database that downloaded from (Kan, 2019) and imported as a `tibble dataframe` with the `RSQLite` package. The variables were reformatted according to their respective types.The full list of variable is given in appendix (see Table 6.1). This dataset will be reduced as we focused on the core intent of modeling the probability of default.

Note that the dataset was anonymised (all identifying ID numbers are deleted) and we therefore removed the corresponding empty columns from the dataset. Since the identification `ID`s have been removed to anonymise the dataset, we cannot see if a borrower borrowed several times.

2.1 Preamble

The LendingClub dataset, although rich, is difficult to interpret. The only explanation of what the variables mean comes from a spreadsheet attached to the dataset. The explanations are not precise and/or subject to conflicting interpretation. Despite serching the LendingClub website, no further original information was found. We collected a number of reasonable assumptions in Appendix (see **?** in Appendix).

The dataset has been used a number of times in the past by various people. One paper (Kim and Cho, 2019) mentions they used a dataset that included 110 variables, which is less than ours with 145 variables. It is therefore clear that the dataset has changed over time in ways we do not know. For example, have loans been excluded because the full 145 veriables were not available?

2.2 General presentation

The original dataset is large: it includes 2260668 loan samples, each containing 145 variables (after the identification variables filled with null values). The loans were issued from 2007-06-01 to 2018-12-01.

2.2.1 Business volume

The dataset represents a total of ca.$34bln in loan principals, which is a substantial share of the total amount stated to have been intermediated to date by LC (publicly reported to be $50bln+). About 55%/60% of the portfolio is fully repaid. See Table 2.1.

Figure 2.1 plots the number, volume (cumulative principal amount) and average principal per loan. It shows that the business grew exponentially (in the common sense of the word) from inception until 2016. At this point, according to Wikipedia [1]:

[1] source: https://en.wikipedia.org/wiki/LendingClub - Retrieval date 15 September 2019

Table 2.1: Number of loans per status

Loan status	Count	Proportion (%)
Charged Off	261655	11.574
Current	919695	40.682
Default	31	0.001
Does not meet the credit policy. Status:Charged Off	761	0.034
Does not meet the credit policy. Status:Fully Paid	1988	0.088
Fully Paid	1041952	46.090
In Grace Period	8952	0.396
Late (16-30 days)	3737	0.165
Late (31-120 days)	21897	0.969

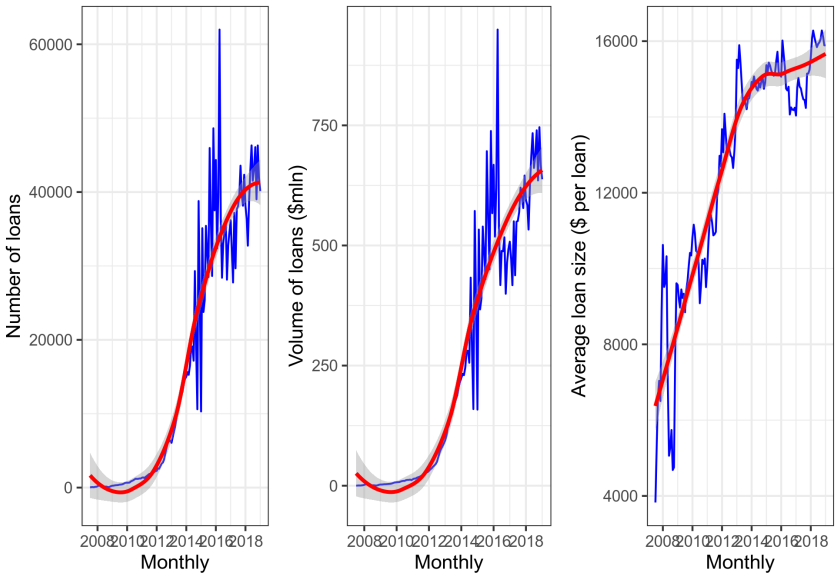


Figure 2.1: Business volume written per month

" *Like other peer-to-peer lenders including Prosper, Sofi and Khutzpa.com, LendingClub experienced increasing difficulty attracting investors during early 2016. This led the firm to increase the interest rate it charges borrowers on three occasions during the first months of the year. The increase in interest rates and concerns over the impact of the slowing United States economy caused a large drop in LendingClub's share price.*"

The number and volume of loans plotted have been aggregated by month. The growth is very smooth in the early years, and suddenly very volatile. As far as the first part of the dataset is concerned, a starting business could expect to be volatile and could witness a yearly cycle (expected from economic consumption figures) superimposed on the growth trend. This is not the case.

An interesting metric is that the average principal of loans has increased (see RHS Figure 2.1, on a sample of 100,000 loans). Partly, the increase in the early years could be interpreted success in improving marketing, distribution capabilities and confidence building. This metric plateau-ed in 2016 and decreased afterwards, but to a much lesser extent than the gross volume metrics. However, it is more volatile than the two previous metrics in the early years.

By the end of the dataset, those metrics have essentially recovered to their 2016 level.

2.2.2 Loan lifecyle and status

In the dataset, less loans are still outstanding than matured or "*charged off*" (term that LC use to mean partially or fully written off, i.e. there are no possibilty for LC and/or the investors to receive further payments). The share of outstanding loans is:

```
## Share of current loans =  42.214 %
```

The dataset describes the life cycle of a loan. In the typical (ideal) case, we understand it to be:

Loan is approved → Full amount funded by investors → Loan marked as Current → Fully Paid

In the worst case, it is:

Loan is approved → Full amount funded by investors → Loan marked as Current →

→ Grace period (missed payments under 2 weeks) → Late 15 to 31 days →

→ Late 31 to 120 days → Default → Charged Off

Note that *Default* precedes and is distinct from *Charged Off* [2]. A couple of things could happen to a loan in default:

- LC and the borrower restructure the loan with a new repayment schedule, where the borrower may repay a lesser amount over a longer period; or,
- the claim could be sold to a debt recovery company that would buy the claim from LC/investors. This would be the final payment (if any) received by LC and the investors.

[2] See LendingClub FAQ at [https://help.lendingclub.com/hc/en-us/articles/215488038] and help page [https://help.lendingclub.com/hc/en-us/articles/216127897-What-happens-when-a-loan-is-charged-off-]

The dataset also describes situations where a borrower negotiated a restructuring of the repayment schedule in case of unexpected hardship (e.g. disaster, sudden unemployment).

Note that this progression of distinguishing default (event in time) from actual financial loss mirrors what banks and rating agencies do. The former is called the *Probability of Default* (PD), the latter *Loss Given Default* (LGD). Ratings change over time (in a process resembling Markov Chains transitions). LGD show some correlations with ratings. The dataset, although detailed, does not include the full life of each loan to conduct this sort of analysis (change of loan quality over time). This is an important reason why we decided to focus on the loan approval and expected return.

2.2.3 Loan application

Before a loan is approved, the borrower undergoes a review process that assess his/her capacity to repay. This includes:

- employment situation and income, as well whether this income and possibly its source has been independently verified;
- whether the application is made jointly (likely with a partner or a spouse, but there are no details);
- housing situation (owner, owner with current mortgage, rental) and in which county he/she lives (that piece of information is partially anonymised by removing the last 2 digits of the borrower's zipcode);
- the amount sought, its tenor and the purpose of the loan; and,
- what seems to be previous credit history (number of previous deliquencies). The dataset is very confusing in that regard: in the case of the **joint** applicant, it is clear that such information relates to before the loan is approved . In the case of the **principal borrower** however, the variable descriptions could be read as being pre-approval information, or post-approval gathered during the life of the loan. We have assumed that the information related to the principal borrower is also pre-approval. We also used *Sales Supplements* from the LC website[3] that describe some of the information provided to investors. LendingClub also provides a summary description of its approval process in its regulatory filings with the Securities Exchange Commission (California, 2019).

2.3 Rates

2.3.1 IRR and required credit margins

Figure 2.2 shows the evolution of credit margins over time grouped by ratings. The plots are made with a random sample of 300,000 loans.

We notice long periods where certain margins remain very stable which indicate that *both* the initial pricing was constant *and* that the proportion of default remains very low.

The graphs show considerations that are relevant to the modeling":

- The margins clearly change over time. To the extent that they reflect a change in probability of default, the predictions will require to account for time (probably in a non-linear fashion).[4]

[3] See https://www.lendingclub.com/legal/prospectus

[4] Note that we will add the second and third power of time to create this non-linearity. This will be the only feature engineering that will be performed on the dataset.

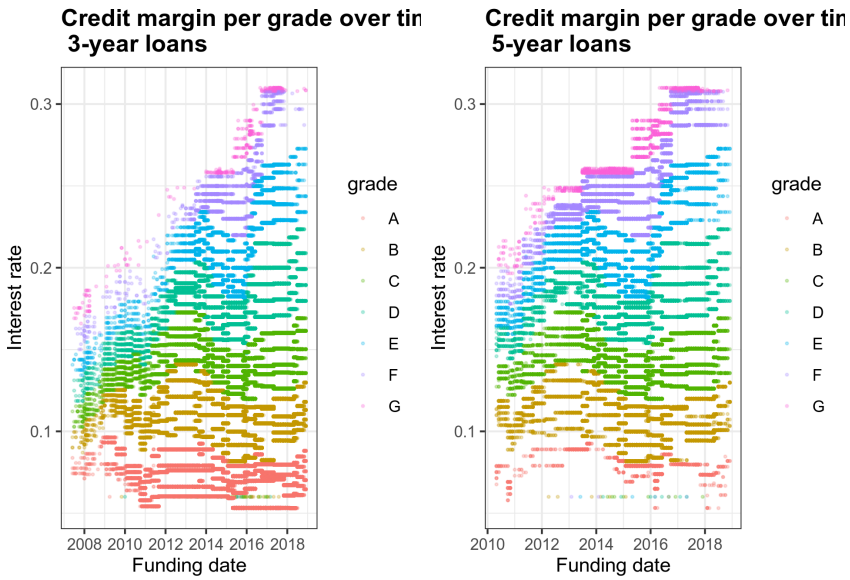


Figure 2.2: Credit margins per grade over time

- For a given rating, it widens and narrows over time. The changes happen in multiples that depends on the ratings:
  - For high quality / low margin loans: the changes are multiples of the margin, for example going from roughly 3% to 6/7%.
  - Although the range of change is wide, those changes do not happen very often, especially in the later years.
  - By comparison, for low quality / high margin loans, the range of change is proportionally smaller, but more frequent and volatile.
- In other words, the relation between loan quality (its rating) and its pricing (the credit margin) will significantly non-linear.

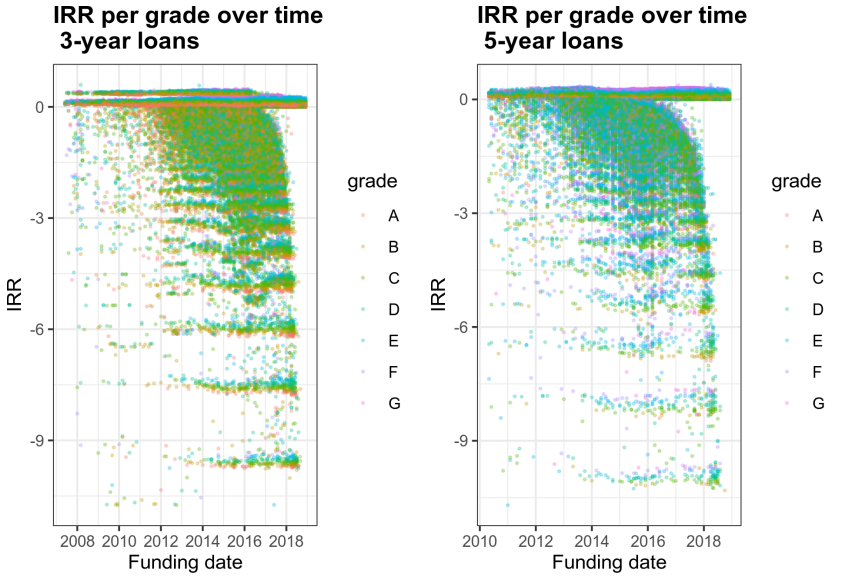


Figure 2.3: Credit margins per grade over time

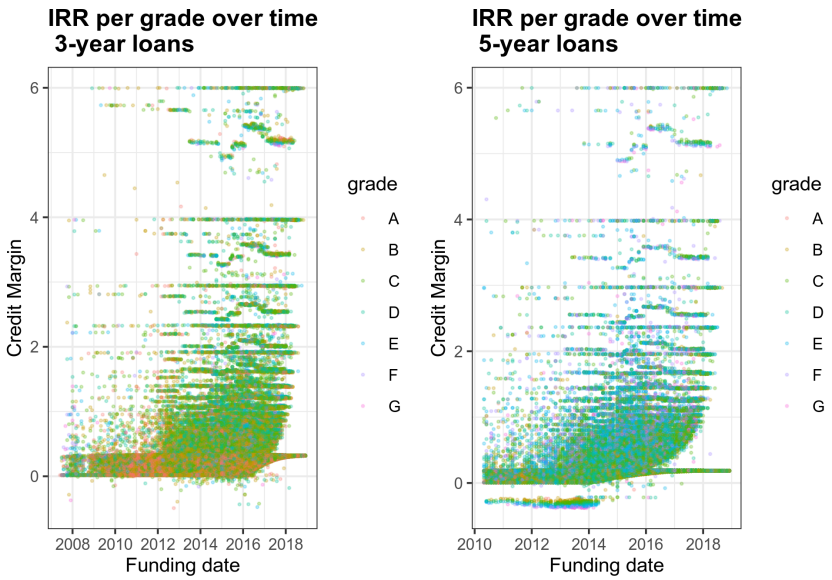


Figure 2.4: Credit margins per grade over time

2.3.2 Dataset

Because we are interested decisions made prior to invest, we will limit the predictors to those that are realistically available prior to funding.

2.3.3 Interest rates

Based on this information, the loan is approved or not. Approval includes the final amount (which could be lower than the amount requested), tenor (3 or 5 years) and a rating similar to those given to corporate borrowers. Unlike corporate borrowers however, the rating mechanically determines the rate of interest according to a grid known to the borrower in advance[5]. The rates have changed over time. Those changes where not as frequent as market conditions (e.g. changes in Federal Reserve Bank's rates)[6].

Figure 2.5 [7] shows the predetermined interest rate depending on the initial rating as of July 2019.

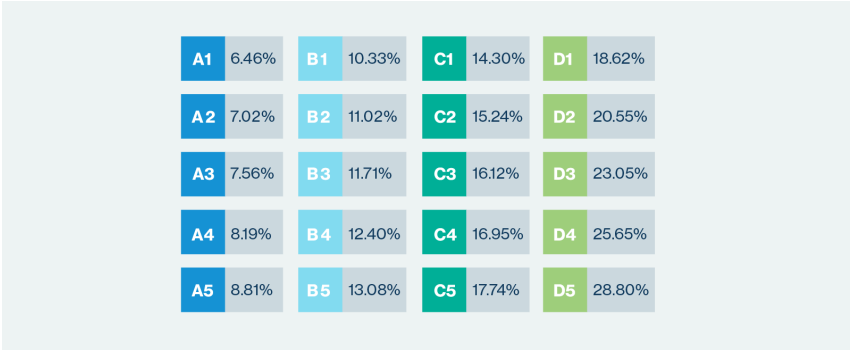


Figure 2.5: Interest rates given rating

At the date of this report, the ratings range from `A` (the best) down to `D`, each split in 5 sub-ratings.

[5] https://www.lendingclub.com/investing/investor-education/interest-rates-and-fees
[6] Corporate borrowers would negociate interest margins on a case-by-case basis despite similar risk profiles.
[7] source: https://www.lendingclub.com/investing/investor-education/interest-rates-and-fees

However, LC previously also intermediated loans rated F or G (until 6 November 2017) and E (until 30 June 2019) [8]. This explains that such ratings are in the dataset. We will assume that the ratings in the dataset are the rating at the time of approval and that, even if loans are re-rated by LC, the dataset does not reflect it.

Figures 2.6 shows the change in interest rate over time for different ratings and separated for each tenor. (Each figure is on a sample of 100,000 loans.) For each rating, we can see several parallel lines which correspond to the 5 sub-rating of each rating. We note that the range of interest rates has substantial widened over time. That is, the risk premium necessary to attract potential investors has had to substantially increase. In the most recent years, the highest rates exceed 30% which is higher than many credit cards.3-year loans are naturally considered safer (more A-rated, less G-rated). Identical ratings attract identical rates of interest.

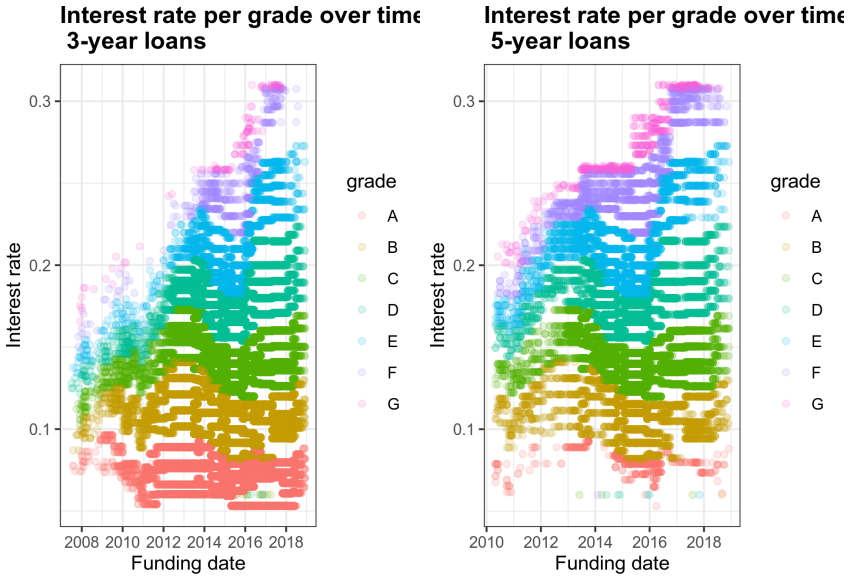


Figure 2.6: Interest rate per grade over time

By comparison, we plot the 3-year (in red) and 5-year (in blue) bank swap rates in Figure 2.7. We see that the swap curve has flattened in recent times (3-year and 5-y rates are almost identical). We also can see that in broad terms the interest rates charged reflect those underlying swap rates. It is therefore most relevant to examine the credit margins added to the swap rates.

Figures 2.8 shows the change in credit margin over time for different ratings and separated for each tenor. (Each figure is on a sample of 100,000 loans.) As above, for each rating, we can see several parallel lines which correspond to the 5 sub-rating of each rating. We note that the range of credit margins has widened over time but less than the interest rates. Identical ratings attract identical credit margins.

2.3.4 Purpose

When applying, a potential borrower must state the purpose of the loan. As shown in table 2.2, by far the main purpose is the consolidation of existing debts.

2.3.5 Payments

The loans are approved for only two tenors, 3 and 5 years, with monthly repayments. Installments are calculated easily with the standard formula:

[8] See https://www.lendingclub.com/info/demand-and-credit-profile.action

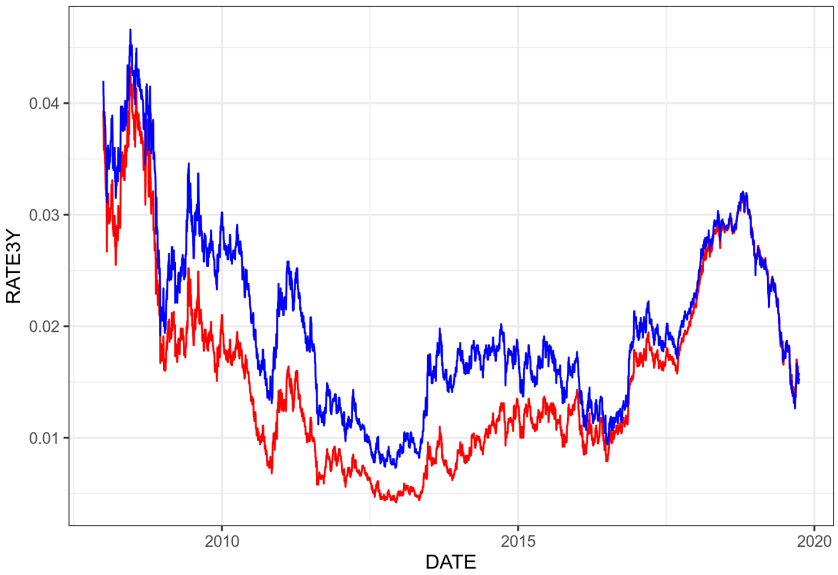


Figure 2.7: Historical Swap Rates

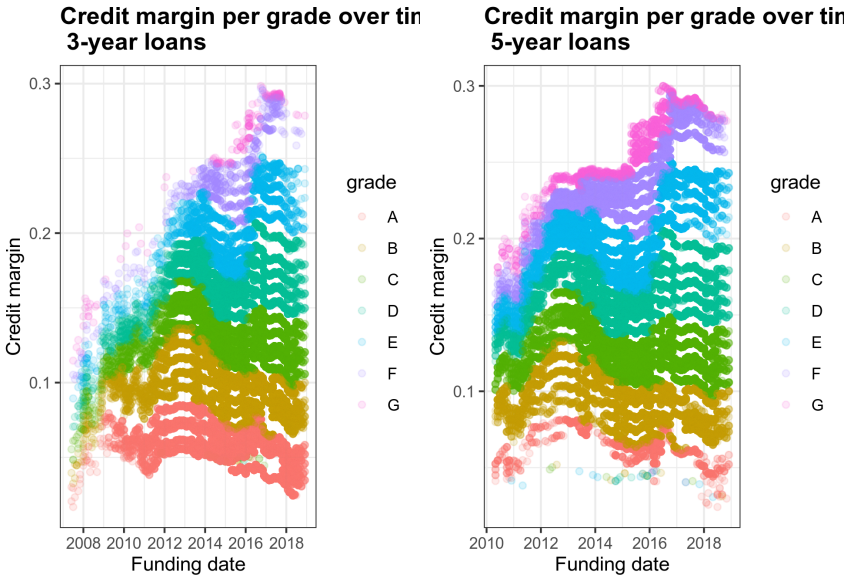


Figure 2.8: Credit margins per grade over time

Table 2.2: Number of loans per purpose

Borrowing purpose	Count
debt_consolidation	758691
credit_card	286044
home_improvement	84709
other	75358
major_purchase	28451
small_business	15171
medical	15081
car	14184
moving	9218
vacation	8751
house	7011
wedding	2350
renewable_energy	914
educational	423

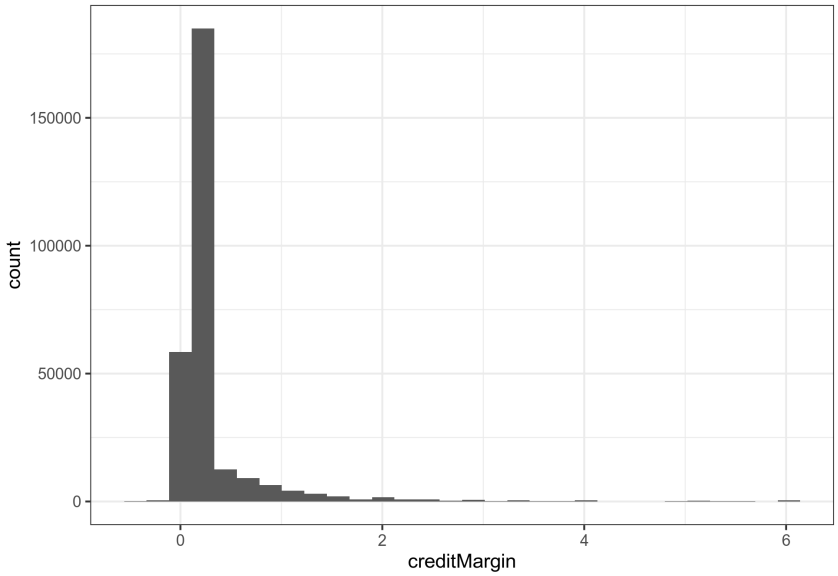


Figure 2.9: Histograms of credit margins per purpose

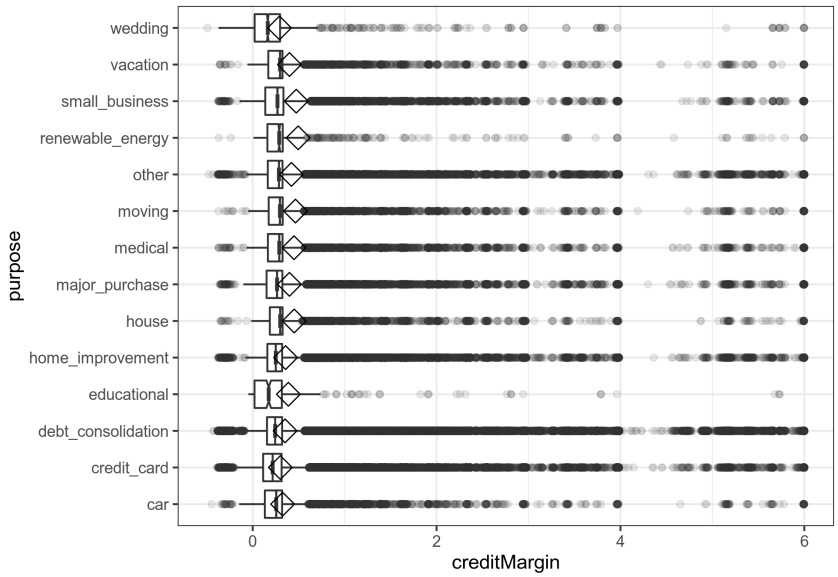


Figure 2.10: Boxplots of credit margins per purpose

$$Installment = Principal \times rate \times \frac{1}{1 - \frac{1}{(1+rate)^N}}$$

Where $Principal$ is the amount borrowed, $rate = \frac{\text{Quoted Interest Rate}}{12}$ is the monthly interest rate, and N is the number of installments (36 or 60 monthly payments). The following piece of code shows that the average error between this formula and the dataset value is about 2 cents. We therefore precisely understand this variable.

```
local({
  installmentError <- loans %>%
    mutate(
      PMT = round(funded_amnt * int_rate / 12 / (1 - 1 / (1 + int_rate / 12) ^
                                                   term), 2),
      PMT_delta = abs(installment - PMT)
    ) %>%
    select(PMT_delta)

  round(mean(100 * installmentError$PMT_delta), digits = 2)
})
```

2.4 Net present value

The behaviour of the NPV of loan losses is important.

2.4.1 Average NPV and credit margin by subgrade

Figure 2.11 shows that as ratings worsen, the average NPV[9] expressed as a portion of the funded amount decreases. For the best quality loans, we see that the NPV exceeds 1.00 = 100%: at a risk-free rate, investors receive more than what is necessary to compensate for credit loss and can use the excess to cover additional costs mentioned in the Preamble. As ratings worsen, the NPV drops down to about 50%.

[9]The averages are *not* weighted by loan amount since an investor can invest in $25 parcels. Weighting would have been appropriate if investors were instead forced to invest in the whole amount.

If loans were adequately priced, the excess returns (thanks to higher interest) should on average offset credit losses, that is an NPV average should be at least 100%. This seems to be the case down to ratings of about `D4`. Further down, credit losses become too frequent and/or too substantial to be covered on average. We posit that this justified rejecting loans applications rated `E1` and below.

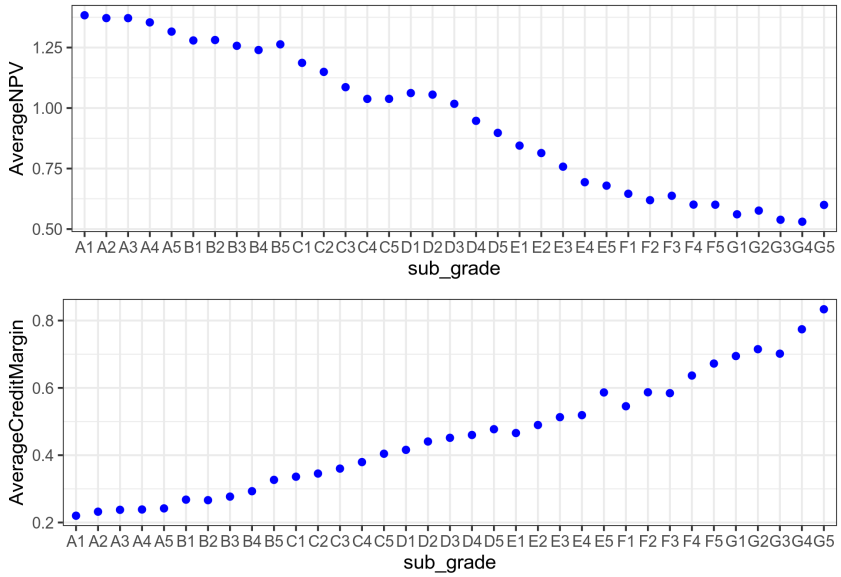


Figure 2.11: Average NPV et credit margin (%) depending on sub-rating

2.4.2 Principal losses

2.4.3 Distribution of principal losses by rating

Figure 2.12 shows that for a given grade, the losses are very widely spread. The loans are group by ratings and loans that have been fully repaid are removed.

Setting aside the loans rated “A” or “B”, the distributions seem log-normal. Unsurprisingly, the worse the rating the larger the principal loss.

2.4.4 NPV distribution by rating

Principal loss does not reflect the timing of that loss: a loss now is worse than a loss later. This subsection looks at the NPVs of actual loan cashflow (principal and interest) discounted the risk-free rate.

Figure 2.13 shows that for a given grade, the NPVs are very widely spread. From top to bottom, loans are group by ratings: from quality ratings of `A` and `B`, average ratings of `C` and `D`, to poor ratings of `E` and below. From left to right, we focus on different parts of how NPVs are distributed. Note that each graph is based on a random sample of 100,000 loans (about 1/12th of the original set) and therefore the NPV densities are comparable from graph to graph.

At the outset, column by column (where NPVs are on the same scale), the NPV distribution show several modes on the same location. The modes are made more apparent by zooming on where the modes are present: the leftmost column basically shows the entire range of the NPVs (as portion of the loan). The middle graph zooms on the -20% / 50% range. The rightmost column zooms on the -100% / -25% section. Looking at the left hand scale, we can see that the lower NPVs overall gain in importance as the loan rating worsen.

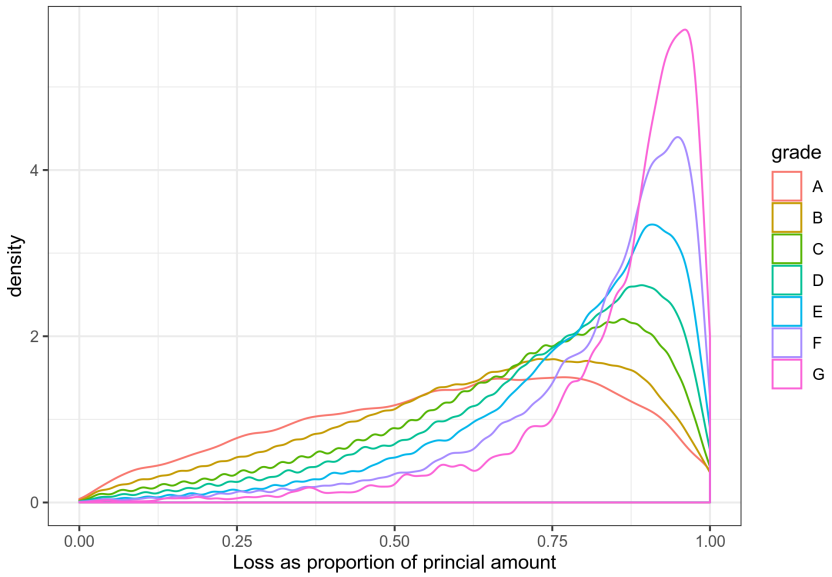


Figure 2.12: Distribution of the Principal Loss (%) depending on rating (y-axis square-root scaling)

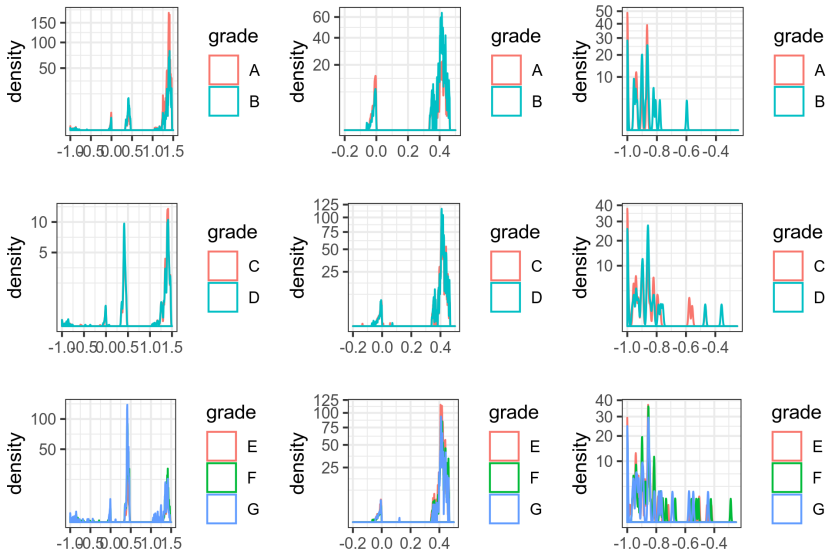


Figure 2.13: Distribution of NPV (%) depending on rating (y-axis square-root scaling)

Zooming without scaling the y-axis and grouping all the ratings available for investment on a single plot gives more details.

- Figure 2.14 shows a mode with a maximum around 1.25 / 1.5 being loans seemingly repaid in full (the mode is above 100% given the repayment of principal *and* interest);

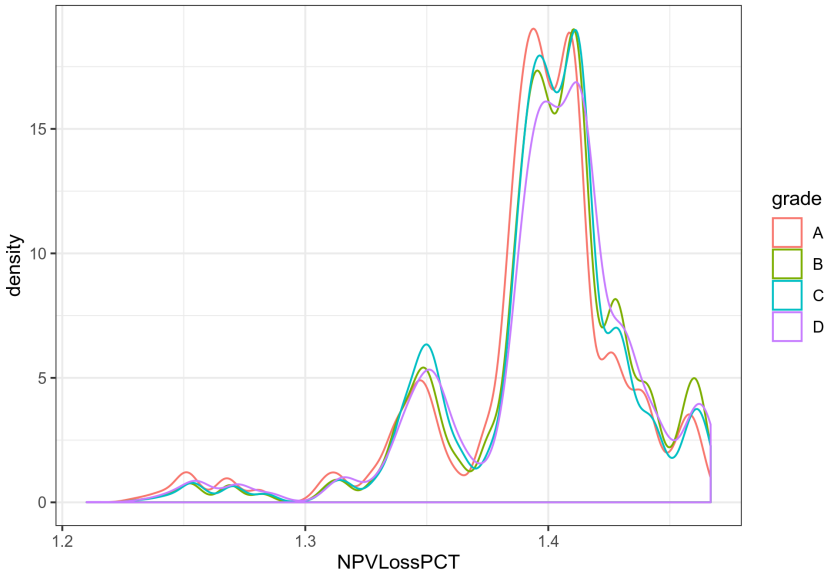


Figure 2.14: NPV % over 120% (no y-axis scaling)

- Figure 2.15 and figure 2.16 show a second and third mode around 41% and -1%;

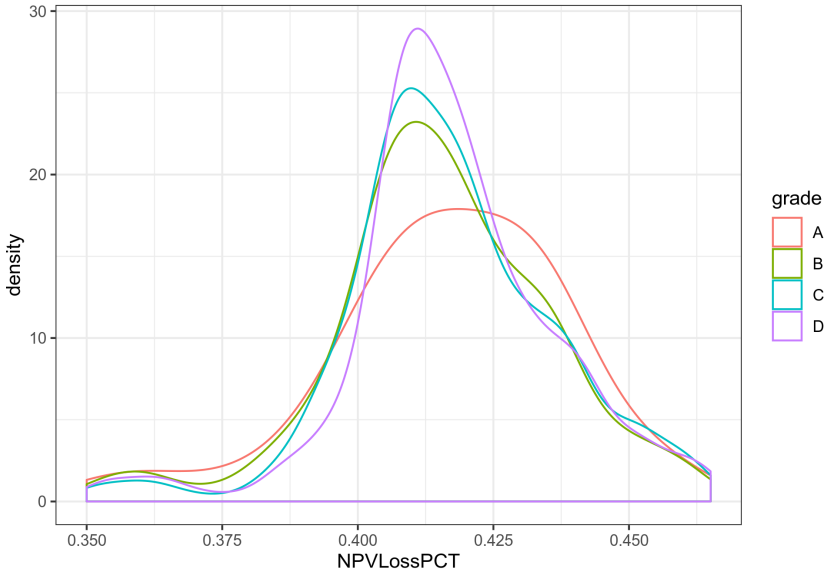


Figure 2.15: NPV % around 41% (no y-axis scaling)

- Finally, figure 2.17 one last very diffuse mode around -100%.

The overall trend is what we should expect. What is surprising is the existence of (1) very clearly defined modes which (2) are common to all types of borrowers. They roughly look log-normal, apart from the mode around 41% which look Gaussian.

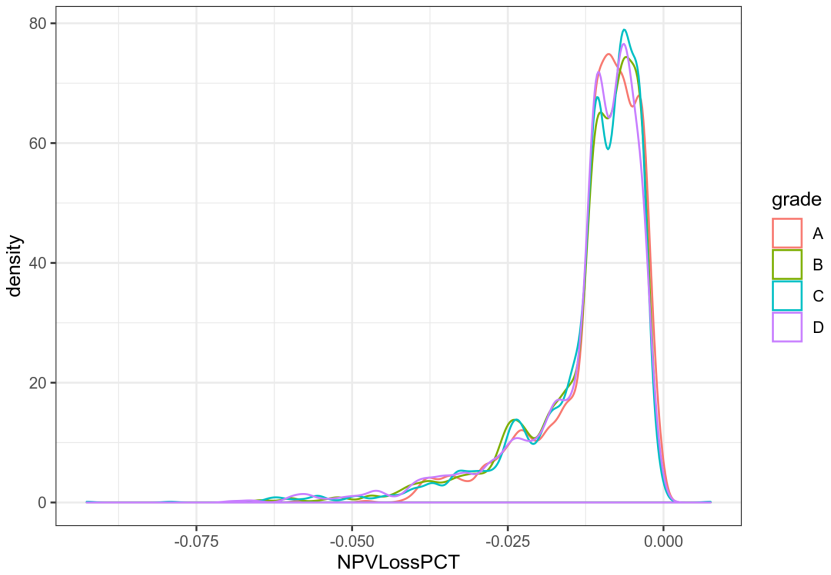


Figure 2.16: NPV % around -1% (no y-axis scaling)

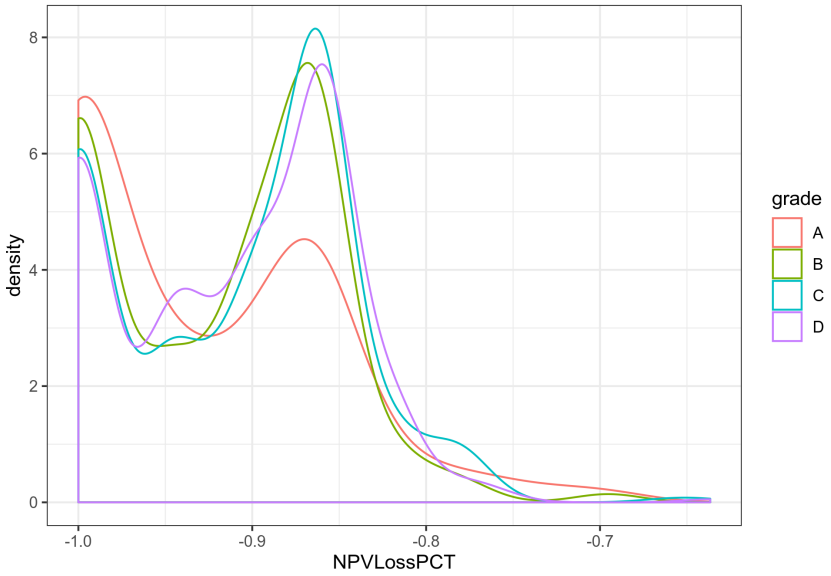


Figure 2.17: NPV % for close to total loss % (no y-axis scaling)

Table 2.3: Matured loans per status

Loan status	Count	Proportion (%)
Fully Paid	964057	24101425
Charged Off	207546	5188650
Does not meet the credit policy. Status:Fully Paid	1334	33350
Does not meet the credit policy. Status:Charged Off	438	10950

2.5 Loan decision

As indicated in the introduction, our focus is on loans that have gone through their entire life cycle to consider their respective pricing, risk and profitability. To that effect, we will remove all loans which are still current (either performing or not), and we will only retain loans which currently available (rated `A1` to `D5`). From here on, everything will be based on this reduced dataset.

In this reduced dataset, we focus on loans that have matured or been terminated. It contains 1306356 samples. Most of the loans (ca.80%) have been repaid in full (in other words **1 in 5 loans defaulted**). See Table 2.3.

When grouped by grade (Figure 2.18), we see a clear correlation between grade and default: the lower the grade the higher the portion defaults (note the limited scale with a minimum at about 50%). In addition, most of the business is written in the B- or C-rating range.

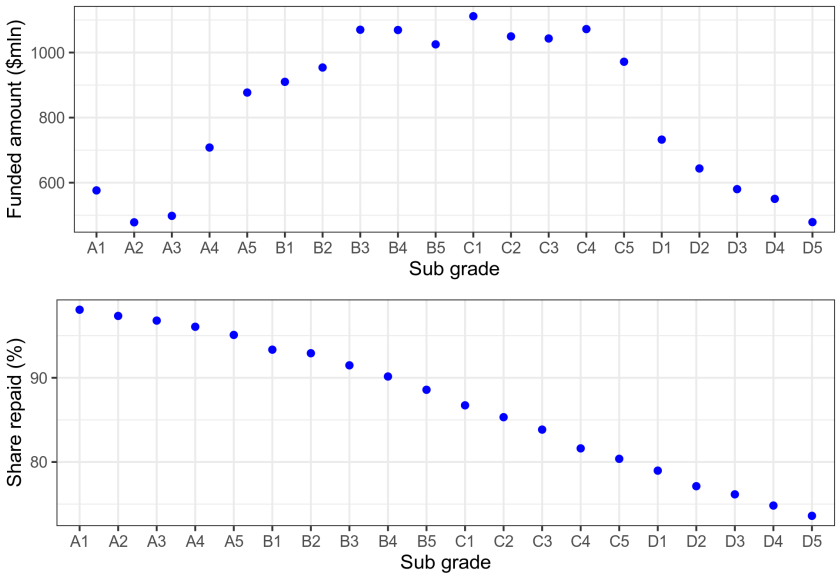


Figure 2.18: Funding and Write-offs by Sub-grades

Chapter 3

Logistic Regression Model and Credit Scorecard

At the outset, the dataset presents a number of challenges:

- There is a mix of continuous and categorical data.
- The number of observations is very large.
- The number of predictors is potential large, in particular if we perform one-hot encoding of categorical values.
- The dataset has no context or reference point to interpret dollar amounts. Everybody intuitively understands that owing $10 or $1,000,000 are very different (amounts matter). Owing $1,000 when living in New York is different from owing $1,000 if living on $2 per day in a developing country (surrounding economic environment matter). That intuition is shared economic knowledge, but that intuition is nowhere represented in the dataset. As readers, we automatically attribute that implicit knowledge to the data we read in the dataset. However, any model based on that data will never reflect that implicit knowledge if we do not supplement with external data. As shown in the previous section, credit margins have changed over time. This is clearly related to the wider US economic environment. Financial hardship is a key driver for some of the loans. Availability of disposable income is important to assess the ability to repay. Therefore, the cost of living, which varies from state to state, seems relevant. As noted in the post-mortem section, such information will not be used.

___ WARNING: IF YOU RUN THE MODELING SECTION, YOU WILL HAVE: MEMORY USAGE OF UP TO 32GB, ALL A LOT DISK SWAPPING WHEN DATASETS ARE JUGGLED IN AND OUT OF MEMORY TO DISK TO MINIMISE MEMORY USAGE, AND CALCULATION TIMES BY BATCHES OF UP TO 10 MINUTES (ON MY LAPTOP). THIS EXCLUDES TIME NECESSARY TO PREPARE THE DATASET AS EXPLAINED ON THE PREVIOUS SECTIONS (THAT IS HOURS).___

3.1 Introduction

We split the dataset randomly into training and validation sets (80/20 ratio).

3.2 Logistic Regression

Logistic models (also called *logit model*) are used to model binary events. Examples would be passing or failing an exam, a newborn being a boy or a girl, a voter choosing a particular political party, or – relevant to us – a borrower defaulting or not on a loan. If the binary variable is modeled as a 0/1 outcome, the model will yield a value between 0 and 1 which can be used as a probabilty.

We are interested in using a number of variables (being continuous and/or categorical) to model the binary response. A natural model is a linear combination of the variables. Since the predicted value would be continuous and not be bounded by 0/1, the outcome is transformed. A commonly used transformation of the logodds (logarithm of the odds given a particular probability) $\log\left(\frac{p}{1-p}\right)$. This expression has a few advantages: it converts any value (between $-\infty$ and $+\infty$ produced by the linear regression), and it is symmetrical around $x = 0$ and $y = 1/2$. That is, using the odds instead of the probability avoids infinity; it behaves identically when p approaches 0 or 1 (this would not be the case using p). The reciprocal of the logodds is $p = \frac{1}{1-e^{-x}}$.

For a number of X_i variables, the model to fit is then:

$$p = \frac{1}{1 - e^{-\sum_i \alpha_i X_i}}$$

The commonly used method to evaluate the creditworthiness of a borrower is to create scorecards whereby particular characteristics are segmented into intervals and attributed discrete scores. In plain English, a continuous variable (say the age of the applicant) is segmented into intervals (e.g. 0-18 year-olds, 18-26 year-olds,...), and then given a score. Those different segments become categorical variables. The task of the model is to:

- identify the best way to segment a continuous variable to maximise the information value of the different segments (called bins). Intuitively, empty or quasi-empty bins (either no or few applicants in the bin) are not very informative; bins for which the response is completely random are not informative, whereas a bin where the response always has the same response is informative (i.e. anybody with income of 0 and 10 dolars per year will default, anybody with a salary of $1 million per month will repay).
- use a generalised linear model using the new categorical variables;
- transforms the linear coefficients estimated for each category into numerical scores.

  WARNING: To run this model, you will need at least 32GB of memory (real plus virtual/swap space) and a lot of time!

3.2.1 Data preparation

The initial preparation of the dataset takes place in the `CleanLoan.Rmd` file where we bind:

- the raw LendingClub dataset (see **?** for how it was prepared);
- from that raw set, only retain the variables that we selected to be used in the model (see column "Used in model?" in subsection **??**(sec:list-model-variables));
- the NPV, IRR and credit margins calculated; and,
- the percentage of principal lost.

Some variables (such as relating to financial outcomes) are used for visualisations, not predictions.

```
# Quirk in bookdown?...
library(tidyverse)

#############################################################################################
##
## select the variables that might be used to create the training+test set
##

modelVarsIn <- c(LC_variable[LC_variable$inModel == TRUE, "variable_name"])$variable_name
modelVarsIn <- c(modelVarsIn,
                 "grade_num", "sub_grade_num",
                 "principal_loss_pct", "creditMargin", "monthDefault")

# Make sure that some variables are NOT in included in the final training set
modelVarsOut <- c("grade_num", "sub_grade_num",
                  "principal_loss_pct", "creditMargin", "monthDefault",
                  "zip_code")
```

We prepare a dataset with ONLY the predictors NOT removing NA's.

```
## #############################################################################################
##
## Prepare a dataset with ONLY the predictors NOT removing NA's
##

loansPredictors <-
  loansWorkingSet %>%

  # Keep the chosen predictors
  # Use tidyselect::one_of() to avoid errors if column does not exist
  select(one_of(modelVarsIn)) %>%

  ##
  ## Dates to numeric, in 'decimal' years since 2000
  ##
  mutate_at(c("issue_d", "earliest_cr_line"), function(d) {
    return(year(d) - 2000 + (month(d) - 1) / 12)
  }) %>%

  ## Add polynomials of the dates to model the time-trend shape
  mutate(
    issue_d2 = issue_d^2,
    issue_d3 = issue_d^3,
    earliest_cr_line2 = earliest_cr_line^2,
    earliest_cr_line3 = earliest_cr_line^3
  ) %>%

```

```
  ## Create a logical flag TRUE for non-defaulted (good) loans
  mutate(isGoodLoan = (principal_loss_pct < 0.001)) %>%

  select(-tidyselect::one_of(modelVarsOut))
```

We split the dataset into a training (80%) and test set (20%).

```
## ##############################################################################################
##
## Create training / test sets 80%/20%
##
proportionTraining <- 0.8
set.seed(42)

nSamples <- nrow(loansPredictors)

sampleTraining <- sample(1:nSamples, floor(nSamples * proportionTraining),
                         replace = FALSE)
loansTraining <- loansPredictors %>% slice(sampleTraining)
loansTest <- loansPredictors %>% slice(-sampleTraining)

# Subsets of the training set
set.seed(42)
nSamplesTraining <- nrow(loansTraining)
```

We also create subsamples of the training set (1% and 20% thereof) for when quick calculation need making.

```
# 1%
sample01 <- sample(1:nSamplesTraining, floor(nSamplesTraining * 0.01),
                   replace = FALSE)
loans01 <- loansTraining %>% slice(sample01)

# 20%
sample20 <- sample(1:nSamplesTraining, floor(nSamplesTraining * 0.20),
                   replace = FALSE)
loans20 <- loansTraining %>% slice(sample20)
```

```
##             used   (Mb) gc trigger    (Mb)   max used    (Mb)
## Ncells   1662162   88.8    4823303  257.6    4823303   257.6
## Vcells 248358750 1894.9 1269991127 9689.3 1586076402 12100.9
```

3.3 Binning and Weight of Evidence

This subsection owes a debt to the source code of the `smbinning` package from which we reimplemented some aspects using the `tidyverse` *style* (the original source code uses SQL statements to access dataframes), and the documentation vignette of the `Information` package [1]. Our code is provided and imported as a package located on github [2].

3.3.1 Background

Binning, Weight of Evidence (WoE) and Information Value (IV) has been widely used since the 1950's to convert continuous values to factors in a way that attempts to maximise the information content of the factors. This is achieved by adjusting the number and location of cut-off points to optimally partition the range of the continuous value.

After coninuous variables are binned, all variables are categorical. Therefore WoE and IV measures then can apply to both types of variables. They have some very important features:

WOE and IV enable one to:

- Consider each variable's independent contribution to the outcome;
- The WoE is a factor-related measure which assesses the relevance of each factor for a given variable;
- The IV is a variable-related measure which enables ranking variables between each other;
- The theoretical background to the measure lies in information theory. It cannot be over-emphasized that the measures **do not** use the values of the factors: it is measure only calculated using the binary GOOD/BAD values (see definitions below);
- Any NA values can be given their own factor: NAs are easily handle without filling values considerations. Given the previous point: the fact that NAs are present has no impact on the measures calculations! Sparse, incomplete or badly filled dataset are easily handled!
- The calculation of is simple and quick (size of the dataset is not an issue in our case);
- The interpretation and Visualisation of those measures is easy and intuitive;

3.3.2 WOE and IV definitions

(This subsections is essentially an extract for the Information R package vignette).

Let's say that we have a binary dependent variable Y and a set of predictive variable X taking a discrete set of values x_1 to x_p (the factors). Y captures the loan defaults. Basically, the X_i represent one-hot encoding of the variable X.

In this situation, Naive Bayes can be formulated in a logarithmic form as:

$$\log \frac{P(Y=1|x_1,\ldots,x_p)}{P(Y=0|x_1,\ldots,x_p)} = \log \frac{P(Y=1)}{P(Y=0)} + \sum_{j=1}^{p} \log \frac{f(x_j|Y=1)}{f(x_j|Y=0)}$$

The naive Bayes model essentially says that the conditional log odds is equal to the sum of the individual factors (which will be the WoE). The word "naive" comes from the fact that this model relies on the assumption that all predictors are conditionally independent given Y, which is a highly optimistic (i.e. unrealistic) assumption.

[1] https://cran.r-project.org/web/packages/Information/vignettes/Information-vignette.html
[2] https://github.com/Emmanuel-R8/SMBinning

3.3.2.1 Weight of Evidence

In an information theory context, this can be remormulated in terms with $P(Y = 1|X = x_j)$ replaced by $GOOD_j$ being the proportion of good loans when only looking at bin j (how many good loans in that bin divided by the total number of loans in that bin), and similarly $P(Y = 0|X = x_j)$ replaced by BAD_j. We also define $GOOD$ as the proportion of good loans for the *entire variable* X

$$\log \frac{GOOD_j}{BAD_j} = \underbrace{\log \frac{GOOD}{BAD}}_{\text{sample log-odds}} + \underbrace{\log \frac{f(x_j|GOOD)}{f(x_j|BAD)}}_{\text{WOE}}$$

This relationship says that the conditional logit of $GOOD_j$ (odds of a good loan in a bin j), given x_j, can be written as the overall log-odds (total odds, i.e., the *intercept*) plus the log-density ratio – also known as the *Weight of Evidence*.

Note that the WoE and the conditional log odds of $Y = 1$ are perfectly correlated since the *ntercept* is constant. Hence, the greater the value of WoE, the higher the chance of observing $Y = 1$. In fact, when WoE is positive the chance of of observing $Y = 1$ is above average (for the sample), and vice versa when WoE is negative. When WoE equals 0 the odds are simply equal to the sample average.

3.3.2.2 Ties to Naive Bayes and Logistic Regression

Notice that the left-hand-side of the equation above – i.e., the conditional log odds of the variable – is exactly what we are trying to predict in a logistic regression model. Hence, when building a logistic regression model – which is perhaps the most widely used technique for building binary classifiers – we are actually trying to estimate the weight of evidence.

In our credit scoring situation, a "semi-naive" version of this model is quite popular. The idea is to transform the data into WOE vectors and then use logistic regression to fit the model

$$\log \frac{GOOD_j}{BAD_j} = \log \frac{GOOD}{BAD} + \sum_{j=1}^{p} \beta_j \log \frac{f(x_j|Y = 1)}{f(x_j|Y = 0)}$$

thus partly relaxing the assumption that all predictors in the model are independent. It should be noted that the underlying WoE vectors are still estimated univariately and that the coefficients merely function as scalars. For a more general model, GAM is a great choice.

As mentioned above, this relationship *does not* depend on the actual values of the bins; only the number of good and bad loans is used. If a bin represents NAs, the NAs have no impact on the calculation.

3.3.2.3 The Information Value

We can leverage WoE to measure the predictive strength of X – i.e., how well it helps us separate cases when $Y = 1$ from cases when $Y = 0$. This is done through the information value (IV) which is defined like this for continuous variables:

$\text{IV}_j = \int \log \frac{f(X_j|Y=1)}{f(X_j|Y=0)} \left(f(X_j|Y = 1) - f(X_j|Y = 0)\right) dx$

Note that the IV is essentially a weighted "sum" of all the individual WoE values where the weights incorporate the absolute difference between the numerator and the denominator (WoE captures the relative difference). Generally, if IV < 0.05 the variable has very little predictive power and will not add any meaningful predictive power to your model.

3.3.2.4 Estimating WOE

In summary, now considering k variables, the most common approach to estimating the conditional densities needed to calculate WoE is to bin a variable X_k into individual bins $x_{k,j}$ and then use a histogram-type estimate.

$$\text{WoE}_{k,j} = \log \frac{GOOD_{k,j}}{BAD_{k,j}}$$

and the IV for variable X_k can be calculated as

$$\text{IV}_k = \sum_j (GOOD_{k,j} - BAD_{k,j}) \times \text{WOE}_{k,j}$$

3.3.3 Modeling

We took insipration of the `smbinning` R package to optimally partition continuous variables into factors/bins. We however could not directly use this package as it does not interact with the tidyverse functions and uses SQL statements to access the content of dataframes. This solution enables to easily access SQL databases. But we note the the tiyverse takes the opposite approach of converting R statements into SQL statements.

We decided to implement a few functions that we will require (not the entire `smbinning` package) as a new package called `binner` separately available on GitHub.

Let's install that package:

```
# Ensure that `binner` is here and available
if ("package:binner" %in% search()) {
  detach("package:binner", unload = TRUE, force = TRUE)
}

if (!("binner" %in% installed.packages()[,1])) {
  devtools::install_local("~/Development/R/DVPT-PACKAGES/Score_modeling_binning/binner",
                          force = TRUE,
                          quiet = TRUE)
  # devtools::install_github("Emmanuel-R8/SMBinning")
}
```

and attach it to the environment.

```
library(binner)
```

3.3.4 Loop through all variables

Before creating the bins, we create a new dataframe and transform some string values to factors.

```
loansBinning <- loansTraining %>%
  mutate(
    home_ownership = as_factor(home_ownership),
    emp_length = as_factor(emp_length),
    grade = as_factor(grade)
  )
```

For each variable we will create a new entry in a new tibble.

```
##############################################################################################
##
## New tibble to store the list of bins + Weight of Evidences factors
##
listBins <-
  tibble(
    variable = "",
    type = "",
    IV = 0.0,
    WoE = list(),
    .rows = 0
  )
```

We then loop through each variable to create factors (for continuous variables) and calculate a table with the Weights of Evidence. We then calculate the Information Value.

```
# About 500 sec wall-time
startTime <- proc.time()

for (n in names(loansBinning)) {

  # We don't test the response with itself
  if (n != "isGoodLoan") {

    # For categorical variable
    if (class(loansBinning[[1, n]]) == "factor") {
      # cat(" is a factor, ")
      result <- WoETableCategorical(
        df = loansBinning,
        x = n,
        y = "isGoodLoan",
```

Table 3.1: Variable relevance by Information Value

Information Value	Relevance
IV $<$ 2%	useless
2% $<$ IV $<$ 10%	weak
10% $<$ IV $<$ 30%	medium
30% $<$ IV $<$ 50%	strong
50% $<$ IV	warning: something is probably wrong!

```
        maxCategories = 100)

    } else {
    # For continuous variable
      result <- WoETableContinuous(df = loansBinning,
                                   x = n,
                                   y = "isGoodLoan",
                                   p = 0.05)
    }

    tryCatch({
      if (is.na(result)) {
        # In case no WoE table is create (of not enough bins)
        add_row(
          listBins,
          variable = n,
          type = NA,
          IV = NA
        )
      } else {

        listBins <- listBins %>%
          add_row(
            variable = n,
            type = result$type,
            IV = result$IV,
            WoE = list(result$table)
          )

        }
    },
    finally = {})
  }
}
proc.time() - startTime
```

```
##    user  system elapsed
## 411.415   1.058 414.064
```

3.3.5 Select relevant variables

Table 3.1 guidelines are recommended to the relevance of variables given their Information Value (Bokhari, 2019).

Table 3.2: 15 top variables by IV

variable	IV
sub_grade	49.39
int_rate	46.53
grade	46.02
term	17.33
dti	7.45
verification_status	5.68
avg_cur_bal	5.43
tot_hi_cred_lim	5.00
num_tl_op_past_12m	4.85
mort_acc	4.38
total_bc_limit	4.12
issue_d2	4.09
issue_d3	4.09
issue_d	4.08
loan_amnt	3.62

The 15 best variables (in terms of Information Value in excess of 2%) are in Table 3.2:

```
bestBins <- listBins %>% filter(IV >= 0.02)

bestBins %>%
  select(variable, IV) %>%
  mutate(IV = round(100 * IV, digits = 2)) %>%
  arrange(desc(IV)) %>%
  slice(1:15) %>%
  knitr::kable(caption = "15 top variables by IV")
```

However, we notice that this list includes variables that would not be available at the time the credit scoring is performed.

```
bestBins <-
  bestBins %>%
  filter(!(variable %in% c(
    "loanID", "term", "int_rate", "creditMargin", "loan_status",
    "grade", "sub_grade", "grade_num", "sub_grade_num",
    "emp_length", "home_ownership", "monthDefault",
    "principal_loss_pct", "creditMargin", "monthDefault",
    "isGoodLoan")))
```

Table 3.3: 15 top relevant variables by IV

variable	IV
dti	0.07445
verification_status	0.05681
avg_cur_bal	0.05430
tot_hi_cred_lim	0.05003
num_tl_op_past_12m	0.04852
mort_acc	0.04382
total_bc_limit	0.04118
issue_d2	0.04087
issue_d3	0.04087
issue_d	0.04083
loan_amnt	0.03616
mths_since_recent_inq	0.03488
mo_sin_rcnt_tl	0.03359
num_actv_rev_tl	0.03354
open_rv_24m	0.03338

The 15 best variables that will retain are in the following table. Interestingly, the square and cubic powers of the issue date are retained.

```
bestBins %>%
  select(variable, IV) %>%
  mutate(IV = round(IV, digits = 5)) %>%
  arrange(desc(IV)) %>%
  slice(1:15) %>%
  knitr::kable(caption = "15 top relevant variables by IV")
```

And the 10 worst (but retained) variables are:

```
bestBins %>%
  select(variable, IV) %>%
  mutate(IV = round(IV, digits = 5)) %>%
  arrange(IV) %>%
  slice(1:15) %>%
  knitr::kable(caption = "15 worst variables by IV")
```

3.3.6 Create data table with only binary variables (transform every bin to a 0/1 value)

For each variable, create new variables for each bin in the WoE table of that variable. Strictly speaking, this is not necessary for the generalised model algorithms. They are able to model

Table 3.4: 15 worst variables by IV

variable	IV
max_bal_bc	0.02120
revol_util	0.02549
open_rv_12m	0.02650
inq_last_6mths	0.02654
mo_sin_old_rev_tl_op	0.02705
mths_since_recent_bc	0.02727
mo_sin_rcnt_rev_tl_op	0.02959
bc_open_to_buy	0.03048
bc_util	0.03048
num_rev_tl_bal_gt_0	0.03221
percent_bc_gt_75	0.03257
open_rv_24m	0.03338
num_actv_rev_tl	0.03354
mo_sin_rcnt_tl	0.03359
mths_since_recent_inq	0.03488

using categories containing the factors. However, as we will see, the model will generate a number of NAs for variable (factors) which are co-linear. (This is the case for any linear model.) This will require removing those individual factors, which we found easier to do when each factor is given an individual variable (column).

```
# Variable will contain all the best characteristics. Every continuous variable is reformatted
# into factors reflecting the appropriate bins

allFactorsAsCharacteristics <- loansTraining[,"loanID"]
allFactorsAsBins <- loansTraining[,"loanID"]
```

```
for (index in 1:nrow(bestBins)) {
#for (index in 1:27) {
  name <- bestBins$variable[index][[1]]

  cat("--- Variable No. ",
      index,
      "-- Name: ",
      name, "\n")

  ltIndex <- which(names(loansTraining) == name)

  characteristic <- categoriseFromWoE(df = loansTraining[, ltIndex],
                                      varName =  name,
                                      woeTable = bestBins$WoE[index][[1]])

```

```
  bins <- categoriseFromWoE.Wide(df = loansTraining[, ltIndex],
                                 varName =  name,
                                 woeTable = bestBins$WoE[index][[1]])

  allFactorsAsCharacteristics <-
    allFactorsAsCharacteristics %>%
    cbind(characteristic)


  allFactorsAsBins <-
    allFactorsAsBins %>%
    cbind(bins)

}
```

```
## --- Variable No.  1 -- Name:  loan_amnt
## --- Variable No.  2 -- Name:  verification_status
## --- Variable No.  3 -- Name:  issue_d
## --- Variable No.  4 -- Name:  dti
## --- Variable No.  5 -- Name:  inq_last_6mths
## --- Variable No.  6 -- Name:  revol_util
## --- Variable No.  7 -- Name:  open_rv_12m
## --- Variable No.  8 -- Name:  open_rv_24m
## --- Variable No.  9 -- Name:  max_bal_bc
## --- Variable No.  10 -- Name:  avg_cur_bal
## --- Variable No.  11 -- Name:  bc_open_to_buy
## --- Variable No.  12 -- Name:  bc_util
## --- Variable No.  13 -- Name:  mo_sin_old_rev_tl_op
## --- Variable No.  14 -- Name:  mo_sin_rcnt_rev_tl_op
## --- Variable No.  15 -- Name:  mo_sin_rcnt_tl
## --- Variable No.  16 -- Name:  mort_acc
## --- Variable No.  17 -- Name:  mths_since_recent_bc
## --- Variable No.  18 -- Name:  mths_since_recent_inq
## --- Variable No.  19 -- Name:  num_actv_rev_tl
## --- Variable No.  20 -- Name:  num_rev_tl_bal_gt_0
## --- Variable No.  21 -- Name:  num_tl_op_past_12m
## --- Variable No.  22 -- Name:  percent_bc_gt_75
## --- Variable No.  23 -- Name:  tot_hi_cred_lim
## --- Variable No.  24 -- Name:  total_bc_limit
## --- Variable No.  25 -- Name:  issue_d2
## --- Variable No.  26 -- Name:  issue_d3
```

```
saveRDS(allFactorsAsCharacteristics, "datasets/allCharacteristics100.rds")
saveRDS(allFactorsAsBins, "datasets/allBins100.rds")
```

3.3.7 Comparison of individual characteristics

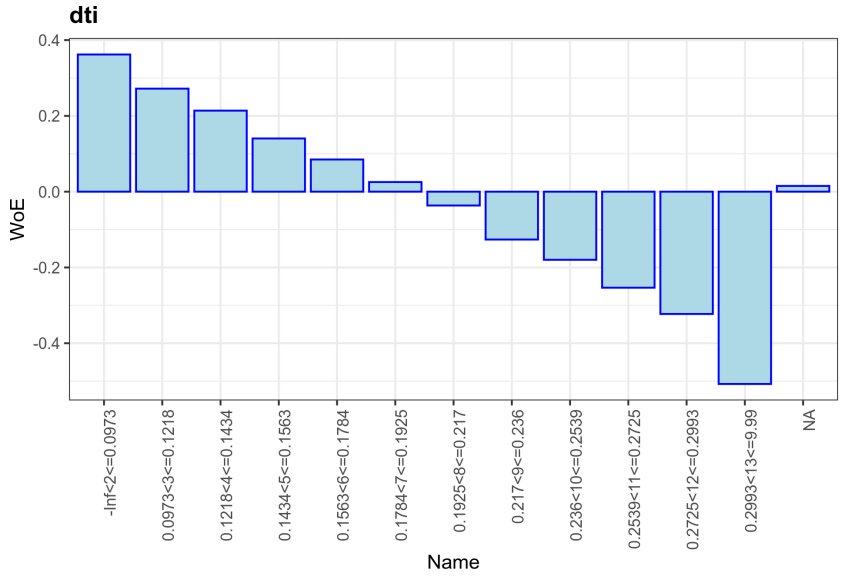
- re-train secures the lowest Akaike criterion and will be chosen.

- On this test, the best variables are: SORT BY IV. Then WoE for the first few ones.

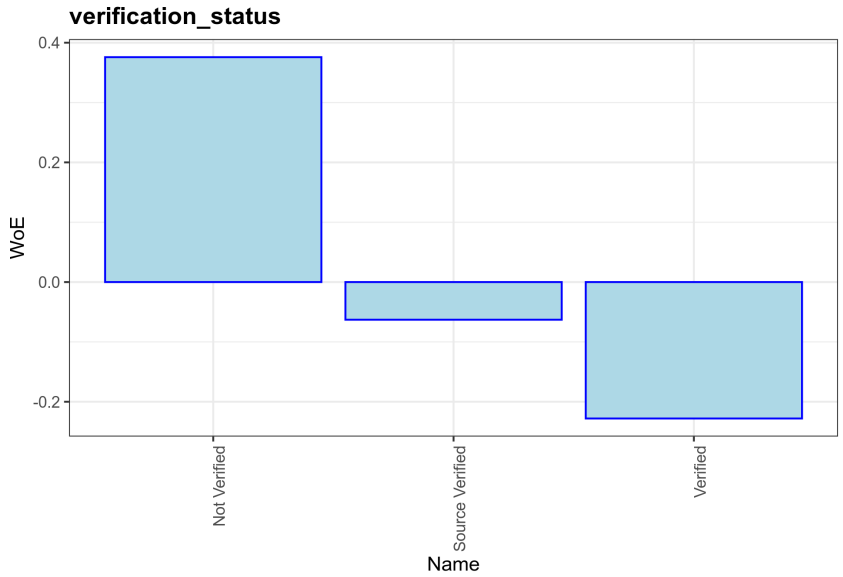
```r
loansBinning <- loans20 %>%
  mutate(
    home_ownership = as_factor(home_ownership),
    emp_length = as_factor(emp_length)
  )
```

The following plots the 10 top variables weight of evidence plots. They show for those variables the weght of evidence of each individual factor. A positive WoE shows that the factor is positive to explain a positive outcome (that is that a loan is good).

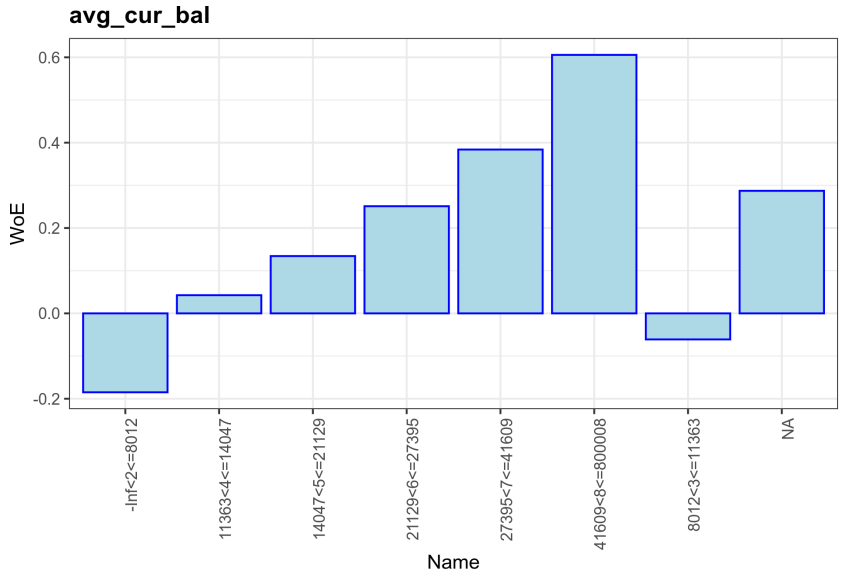
```r
best10 <- bestBins %>% arrange(desc(IV)) %>% slice(1:10)

plotBinWoE <- function(n = 1) {
  vName <- best10[n,]$variable

  if (best10[n,]$type == "numeric") {
    best10[n,]$WoE[[1]] %>%
      arrange(WoE) %>%
      ggplot(aes(Name, WoE)) +
      geom_col(col = "blue", fill = "lightblue") +
      theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
      ggtitle(vName)
  } else {
    best10[n,]$WoE[[1]] %>%
      arrange(WoE) %>%
      ggplot(aes(Name, WoE)) +
      geom_col(col = "blue", fill = "lightblue") +
      theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
      ggtitle(vName)

  }

}

listPlots <- lapply(1:10, function(n) { plotBinWoE(n) })

listPlots[[1]]
```



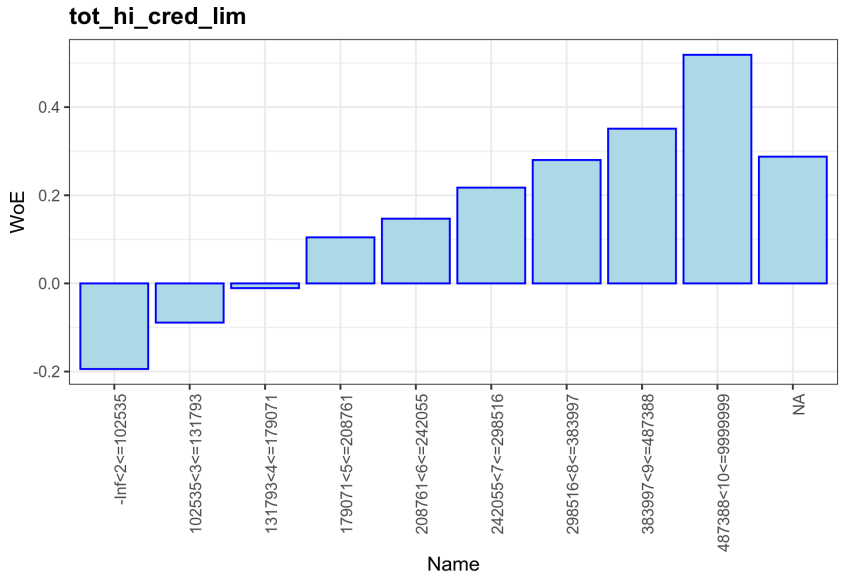

```
listPlots[[2]]
```



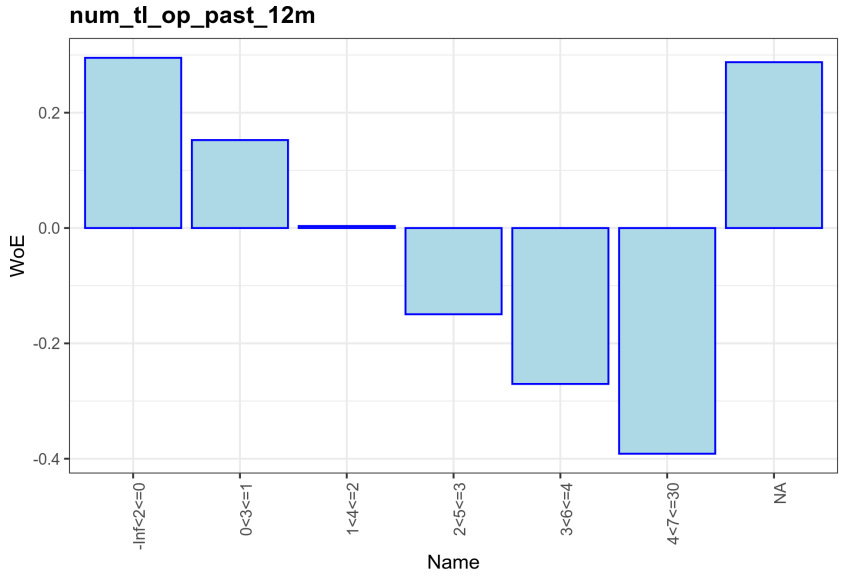

```
listPlots[[3]]
```



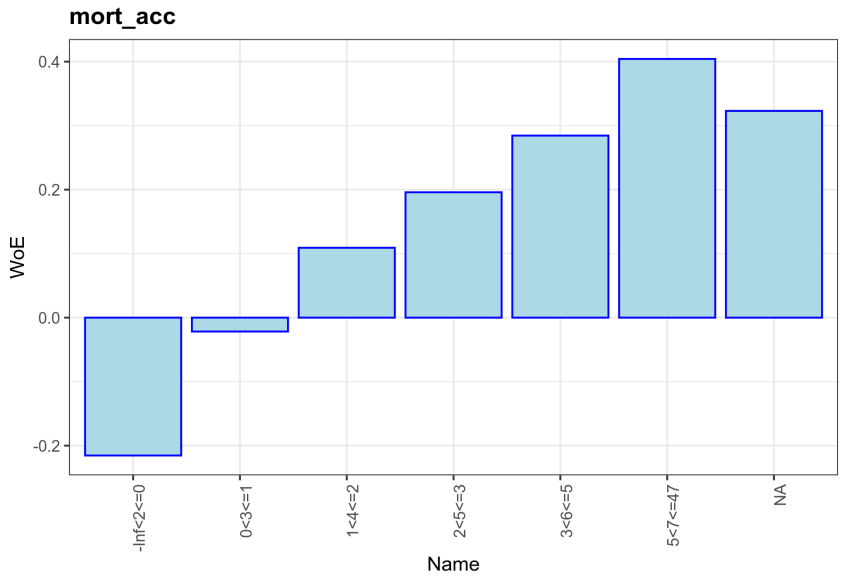

```
listPlots[[4]]
```



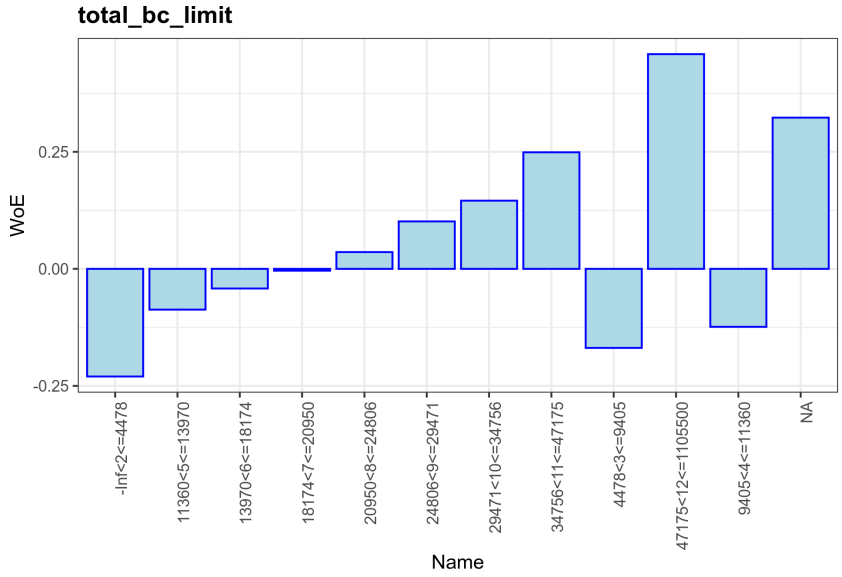

```
listPlots[[5]]
```



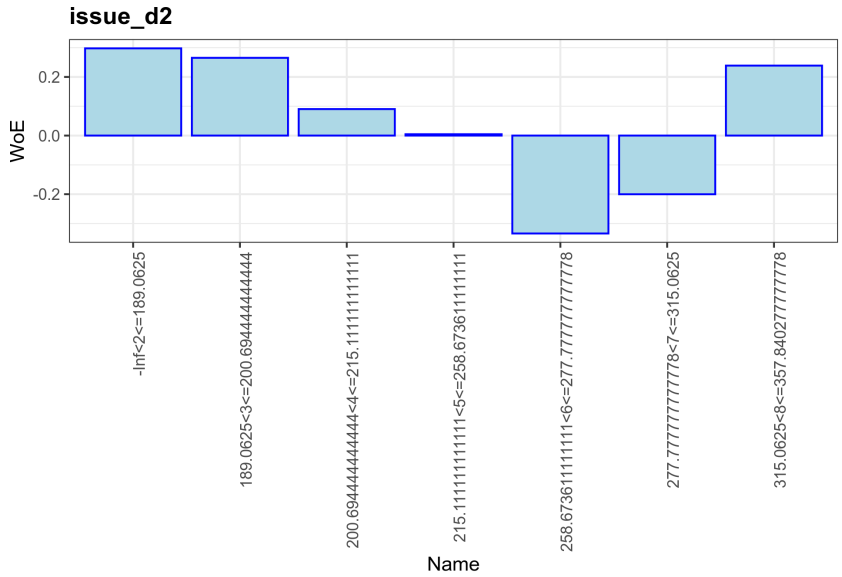

```
listPlots[[6]]
```



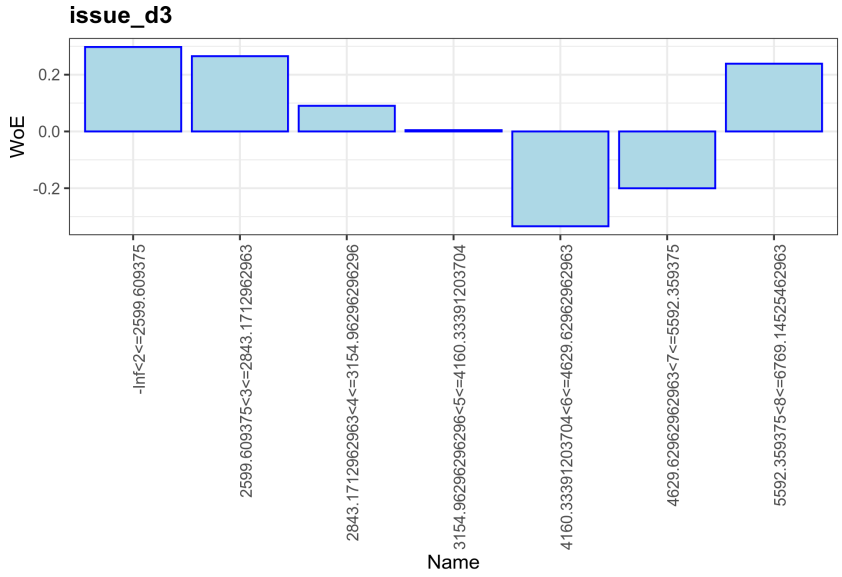

```
listPlots[[7]]
```



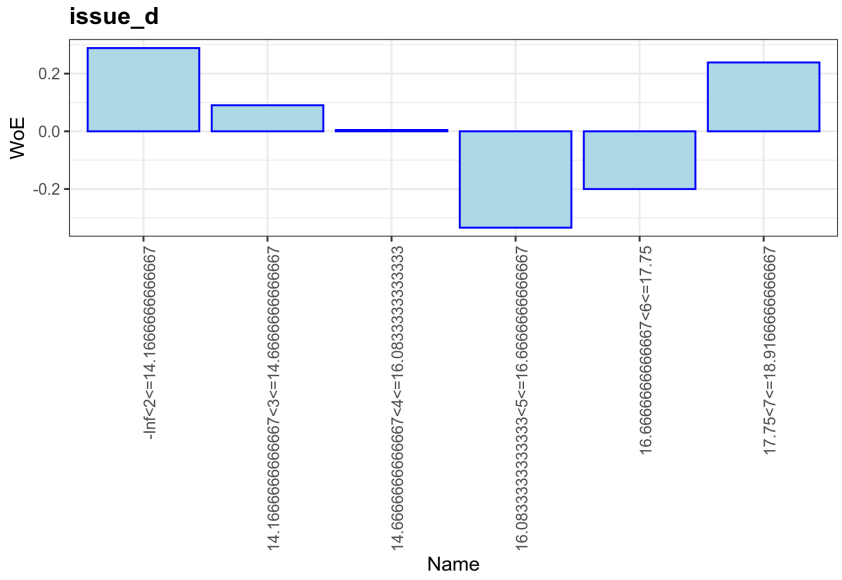

```
listPlots[[8]]
```




```
listPlots[[9]]
```




```
listPlots[[10]]
```



3.4 Logistic Regression

3.4.1 Logistic regression using the `SpeedGLM` package

NOTE: `speedglm` has a `select()` function which would shadow `dplyr::select()`, so it is only used fully qualified to avoid any collision.

```
##             used   (Mb) gc trigger   (Mb)   max used    (Mb)
## Ncells   2739840  146.4    5130328  274.0    5130328   274.0
## Vcells 433024202 3303.8 1016445594 7754.9 1586076402 12100.9
```

```
## [1] 222
```

```
## [1] 27
```

3.4.2 Remove identical bins

```
namesCharacteristics <- names(loanSampleCharacteristics)
namesBins <- names(loanSampleBins)


# Starting list of names
# > The following two variables are only useful if all the bins have been given their own column (if
# > characteristic has 7 factors, that creates 7 bins and column).


# Running GLM would give a number of NAs if some columns are linearly dependant (in our case
# actually equal). A "trick" to identify them is to run a hash digest on each column and spot
# identical hashes The original variables were not identical, but when split into categories, they
# can become identical. For example, if no income is provided (income=NA category), no
# debt-to-income ratio can be calculated (dti=NA).
duplicateNames <- loanSampleBins[duplicated(lapply(loanSampleBins, digest::digest))] %>% names()


# Remove any categories uniformaly constant (only 0)
#
# This is important when working on a small extract of the dataset for variables that are seldom
# used (e.g. customers from certain states).
zeroColumnsNames <- loanSampleBins[, colSums(loanSampleBins) == 0] %>% names()
```

3.4.3 First training on variables previously selected on their Information Value

We use `speedglm` being quick. Note that alternatives were also tried: `glm` crashed on even small extracts of the dataset. `glmnet` returns errors that were not understandable or documented on the internet.

3.4.3.1 GLM on characteristics

```
##             used   (Mb) gc trigger   (Mb)   max used    (Mb)
## Ncells   2739931  146.4    5130328  274.0    5130328   274.0
## Vcells 433024593 3303.8 1016445594 7754.9 1586076402 12100.9
```

```
# About 700 sec wall-time to complete training dataset
# The dataset is the sample less duplicate, less zero-ed columns
{
  doMC::registerDoMC(cores = 1) # Too many processes push over 32GB
  startTime <- proc.time()

  loansData <- loanSampleCharacteristics %>%
    select(-one_of( c(duplicateNames,
                      zeroColumnsNames)))

  SGLM_C_train <- speedglm::speedglm(isGoodLoan ~ .,
```

```
                                    data = loansData,
                                    family = binomial())

  doMC::registerDoMC(cores = NULL)
  cat(proc.time() - startTime, "\n")
}
```

```
## 146.743 6.152 162.84 0 0
```

```
#summary(SGLM_C_train)

# There are a number of NA estimates that need to be removed by removing indivual bins
length( which(is.na(summary(SGLM_C_train)$coefficient$Estimate)) )
```

```
## [1] 28
```

```
saveRDS(SGLM_C_train, "datasets/SGLM_C_train.rds")
```

3.4.3.2 GLM on bins

```
##             used   (Mb) gc trigger   (Mb)   max used    (Mb)
## Ncells   2765528  147.7    5130328  274.0    5130328   274.0
## Vcells 433096867 3304.3 1219814712 9306.5 1586076402 12100.9
```

```
# About 700 sec wall-time to complete training dataset
# The dataset is the sample less duplicate, less zero-ed columns
{
  doMC::registerDoMC(cores = 1) # Too many processes push over 32GB
  startTime <- proc.time()

  loansData <- loanSampleBins

  SGLM_B_train <- speedglm::speedglm(isGoodLoan ~ .,
                                     data = loansData,
                                     family = binomial())

  doMC::registerDoMC(cores = NULL)
  cat(proc.time() - startTime, "\n")
}
```

```
## 161.674 8.439 191.064 0 0
```

```
saveRDS(SGLM_B_train, "datasets/SGLM_B_train.rds")

# There are a number of NA estimates that need to be removed by removing indivual bins
```

Table 3.5: First training: Best bins

binName	Estimate	Std. Error	z	p
‘(loan_amnt) 3500<loan_amnt<=9000‘	0.9848386	0.0108316	90.9228	0.00e+00
‘(loan_amnt) -Inf<loan_amnt<=3500‘	1.1617352	0.0152225	76.3171	0.00e+00
‘(loan_amnt) 9000<loan_amnt<=10000‘	0.7380869	0.0124309	59.3751	0.00e+00
‘(loan_amnt) 10000<loan_amnt<=12000‘	0.6002188	0.0121587	49.3652	0.00e+00
‘(verification_status) verification_status=Not Verified‘	0.2840857	0.0070796	40.1275	0.00e+00
‘(loan_amnt) 14975<loan_amnt<=15000‘	0.5501235	0.0142603	38.5774	0.00e+00
‘(loan_amnt) 12000<loan_amnt<=14975‘	0.5037774	0.0131783	38.2277	0.00e+00
‘(num_tl_op_past_12m) 0<num_tl_op_past_12m<=1‘	0.3467503	0.0121796	28.4698	2.77e-178
‘(loan_amnt) 15000<loan_amnt<=17000‘	0.3762410	0.0135998	27.6652	1.83e-168
‘(mort_acc) 5<mort_acc<=47‘	1.0623545	0.0423491	25.0856	7.14e-139

```
speedglm:::summary.speedglm(SGLM_B_train)$coefficients %>%
  as.data.frame() %>%
  rownames_to_column(var = "binName") %>%
  filter(is.na(Estimate)) %>%
  slice(1:10)
```

binName	Estimate	Std. Error	z value	Pr(>|z|)
‘(loan_amnt) 28000<loan_amnt<=40000‘	NA	NA	NA	NA
‘(verification_status) verification_status=Verified‘	NA	NA	NA	NA
‘(issue_d) 17.75<issue_d<=18.917‘	NA	NA	NA	NA
‘(dti) dti=NA‘	NA	NA	NA	NA
‘(inq_last_6mths) inq_last_6mths=NA‘	NA	NA	NA	NA
‘(revol_util) revol_util=NA‘	NA	NA	NA	NA
‘(open_rv_12m) open_rv_12m=NA‘	NA	NA	NA	NA
‘(open_rv_24m) 5<open_rv_24m<=53‘	NA	NA	NA	NA
‘(open_rv_24m) open_rv_24m=NA‘	NA	NA	NA	NA
‘(max_bal_bc) 7905<max_bal_bc<=776843‘	NA	NA	NA	NA

```
speedglm:::summary.speedglm(SGLM_B_train)$coefficients %>%
  as.data.frame() %>%
  rownames_to_column(var = "binName") %>%
  rename(p = "Pr(>|z|)",
         z = "z value") %>%
  arrange(desc(z)) %>%
  slice(1:10) %>%
  knitr::kable(caption = "First training: Best bins")
```

```
# Does the model throw NAs? They are produced not only for identical variables, but also co-linear
# combinations. Let us select those variables:
```

```

NAsFirstTraining <-

  # Take the results (just the estimated coefficients) from the model
  tibble(
    name = rownames(summary(SGLM_B_train)$coefficient),
    estimate = summary(SGLM_B_train)$coefficient$Estimate
  ) %>%

  # Reformat the names to be the same as in the bins
  mutate(name = stringr::str_remove_all(name, "\`")) %>%

  # Make sure that repsonse is not deleted by mistake and list all NAs
  filter(name != "isGoodLoan" & is.na(estimate))

NAsFirstTraining <- NAsFirstTraining$name
```

3.4.4 Second training

```
##             used   (Mb) gc trigger   (Mb)   max used    (Mb)
## Ncells   2768419  147.9    6815958  364.1    5130328   274.0
## Vcells 433156465 3304.8 1216113575 9278.3 1586076402 12100.9
```

```
# Start the model training again
{
  startTime <- proc.time()
  doMC::registerDoMC(cores = 2) # Too many processes push over 32GB if more than 2

  loansData <- loanSampleBins %>%
    select(-one_of( c(duplicateNames,
                      zeroColumnsNames,
                      NAsFirstTraining)))

  SGLM_B_retrain <- speedglm::speedglm(isGoodLoan ~ .,
                                       data = loansData,
                                       family = binomial())

  doMC::registerDoMC(cores = NULL)
  cat(proc.time() - startTime)
}
```

```
## 147.596 7.585 181.135 0 0
```

```
speedglm:::summary.speedglm(SGLM_B_retrain)$coefficients %>%
  as.data.frame() %>%
  rownames_to_column(var = "binName") %>%
```

Table 3.6: Second training: Best bins

binName	Estimate	Std. Error	z	p
‘(loan_amnt) 3500<loan_amnt<=9000‘	0.9848386	0.0108316	90.9228	0.00e+00
‘(loan_amnt) -Inf<loan_amnt<=3500‘	1.1617352	0.0152225	76.3171	0.00e+00
‘(loan_amnt) 9000<loan_amnt<=10000‘	0.7380869	0.0124309	59.3751	0.00e+00
‘(loan_amnt) 10000<loan_amnt<=12000‘	0.6002188	0.0121587	49.3652	0.00e+00
‘(verification_status) verification_status=Not Verified‘	0.2840857	0.0070796	40.1275	0.00e+00
‘(loan_amnt) 14975<loan_amnt<=15000‘	0.5501235	0.0142603	38.5774	0.00e+00
‘(loan_amnt) 12000<loan_amnt<=14975‘	0.5037774	0.0131783	38.2277	0.00e+00
‘(num_tl_op_past_12m) 0<num_tl_op_past_12m<=1‘	0.3467503	0.0121796	28.4698	2.77e-178
‘(loan_amnt) 15000<loan_amnt<=17000‘	0.3762410	0.0135998	27.6652	1.83e-168
‘(mort_acc) 5<mort_acc<=47‘	1.0623545	0.0423491	25.0856	7.14e-139

```
  rename(p = "Pr(>|z|)",
         z = "z value") %>%
  arrange(desc(z)) %>%
  slice(1:10) %>%
  knitr::kable(caption = "Second training: Best bins")
```

```
# Just in case, new NAs popped up. Then select variables whose significance value is under 2 sigmas.

NAsSecondTraining <-
  tibble(
    name = rownames(summary(SGLM_B_retrain)$coefficient),
    estimate = summary(SGLM_B_retrain)$coefficient$Estimate,
    zValue = abs(summary(SGLM_B_retrain)$coefficient$"z value")
  ) %>%

  mutate(name = stringr::str_remove_all(name, "\`")) %>%
  filter(name != "isGoodLoan") %>%

  # Remove stray NAs or anything less than 2 sigmas
  filter(is.na(estimate) | zValue < 2)

NAsSecondTraining <- NAsSecondTraining$name
```

3.4.5 Third training

```
##             used   (Mb) gc trigger   (Mb)   max used    (Mb)
## Ncells   2769643  148.0    6815958  364.1    5130328   274.0
## Vcells 433201172 3305.1 1167533032 8907.6 1586076402 12100.9
```

Table 3.7: Third training: Best bins

binName	Estimate	Std. Error	z	p
‘(loan_amnt) 3500<loan_amnt<=9000‘	0.9383885	0.0107265	87.4833	0.00e+00
‘(loan_amnt) -Inf<loan_amnt<=3500‘	1.1079606	0.0151211	73.2723	0.00e+00
‘(loan_amnt) 9000<loan_amnt<=10000‘	0.7026804	0.0123379	56.9531	0.00e+00
‘(loan_amnt) 10000<loan_amnt<=12000‘	0.5616891	0.0120636	46.5606	0.00e+00
‘(verification_status) verification_status=Not Verified‘	0.3225105	0.0070171	45.9606	0.00e+00
‘(num_tl_op_past_12m) -Inf<num_tl_op_past_12m<=0‘	0.5700935	0.0133125	42.8238	0.00e+00
‘(num_tl_op_past_12m) 0<num_tl_op_past_12m<=1‘	0.4623390	0.0112317	41.1636	0.00e+00
‘(loan_amnt) 14975<loan_amnt<=15000‘	0.5296854	0.0141852	37.3408	3.57e-305
‘(num_rev_tl_bal_gt_0) -Inf<num_rev_tl_bal_gt_0<=2‘	0.4810422	0.0129686	37.0927	3.68e-301
‘(loan_amnt) 12000<loan_amnt<=14975‘	0.4583936	0.0130818	35.0405	5.44e-269

```
# Start the model training again. About 350 sec.
{

  startTime <- proc.time()
  doMC::registerDoMC(cores = 2)

  loansData <- loanSampleBins %>%
    select(-one_of( c(
      duplicateNames,
      zeroColumnsNames,
      NAsFirstTraining,
      NAsSecondTraining
    )))

  SGLM_B_reretrain <- speedglm::speedglm(isGoodLoan ~ .,
                                         data = loansData,
                                         family = binomial())

  doMC::registerDoMC(cores = NULL)
  cat(proc.time() - startTime)
}
```

```
## 48.314 1.681 50.297 0 0
```

```
speedglm:::summary.speedglm(SGLM_B_reretrain)$coefficients %>%
  as.data.frame() %>%
  rownames_to_column(var = "binName") %>%
  rename(p = "Pr(>|z|)",
         z = "z value") %>%
  arrange(desc(z)) %>%
  slice(1:10) %>%
  knitr::kable(caption = "Third training: Best bins")
```

Table 3.8: Training AIC and Log-likelihood

Criteria	First #1	Second #2	Third #3
AIC	977037.2	977037.2	982957.0
Log likelihood	-488350.6	-488350.6	-491370.5

```
NAsThirdTraining <-
  tibble(
    name     = rownames(summary(SGLM_B_reretrain)$coefficient),
    estimate = summary(SGLM_B_reretrain)$coefficient$Estimate,
    zValue   = abs(summary(SGLM_B_reretrain)$coefficient$"z value")
  ) %>%
  mutate(name = stringr::str_remove_all(name, "\`")) %>%
  filter(name != "isGoodLoan") %>%

  # Remove stray NAs (Shouldn't be any) or less than 2 sigmas
  filter(is.na(estimate) | zValue < 2)

NAsThirdTraining <- NAsThirdTraining$name
```

Table @reg(tab:0502-training-criteria) shows the Akaike Information Criterion and Log-likelihood for each of the 3 training results. Two points to note:

- The results did not change from first to second training. This is normal since we only removed non-informative categorical bins for which the regression could not determine a coefficient (i.e. colinearity with other variables);
- Removing seemingly non-significant bins makes the model worse (at least by those measures). This would suggest that individual bins should not be considered but only the entire variable to which they belong.

In the next section we will only work with the second model since it appears to be the best.

```
tibble("Criteria" = c("AIC", "Log likelihood"),
       "First #1" = c(speedglm:::summary.speedglm(SGLM_B_train)$aic,
                      speedglm:::summary.speedglm(SGLM_B_train)$logLik),

      "Second #2" = c(speedglm:::summary.speedglm(SGLM_B_retrain)$aic,
                      speedglm:::summary.speedglm(SGLM_B_retrain)$logLik),

      "Third #3" = c(speedglm:::summary.speedglm(SGLM_B_reretrain)$aic,
                     speedglm:::summary.speedglm(SGLM_B_reretrain)$logLik)) %>%

  knitr::kable(caption = "Training AIC and Log-likelihood")
```

3.5 Model result

3.5.1 Final list of variables

```
# Model to use
GLModel <- SGLM_B_retrain

# List of model variables
modelNames <-
  attr(GLModel$coefficients, "names") %>%
  enframe(x = .) %>%
  rename(variableName = "value") %>%
  select(variableName) %>%
  mutate(variableName = str_remove_all(variableName, "\`"))
```

```
# and their coefficients in the model (the summary function for speedglm objects is not exported and
# bookdown seems to have a problem with that).

sumSpeed <- speedglm:::summary.speedglm

GLMCoefficients <-
  sumSpeed(GLModel)$coefficients %>%
  as_tibble() %>%
  cbind(modelNames) %>%
  rename(
    zValue = "z value",
    pValue = "Pr(>|z|)",
    stdErr = "Std. Error"
  ) %>%

  # reorder columns to have names first
  select(variableName, everything())
```

3.5.2 Scoring

Scoring expresses the coefficients that were estimated during the logistic regression into points on a scale. The conversion is done using three parameters that are chosen somewhat arbitrarily.

- the number of points increase / decrease that would reflect halving / doubling the odds of defaulting;
- an *anchoring* score reflecting a particular odd.

For our purpose, we will choose 5,000 points (Score_{anchor}) being equivalent to 1 in a 20 to default (Odds_{anchor}), i.e. Score_{anchor} 5,000 points <=> $\text{Odds}_{Anchor} = \frac{1/20}{1-1/20}$. We will also choose 100 to reflect *times 2* change in odds ($DoubleOdds$). Those choices are completely arbitrary. Basically, the score is a linear representation of the odds. The score is defined by a point (the anchor) and the slope of the line going through that point.

Those values will reflect the total estimated score for a borrower (i.e. loan sample). The number of characteristics (information points gathered in a credit application), or number of bins, have to be irrelevant in calculating this score. In other words, if LendingClub were to gather 5 more information points, the score should, **mutatis mutandis**, be unchanged. (However, we would hope that the quality of the estimated score would improve.)

The score per variable needs to be adjusted using the number of information points, that is the number of characteristics. Here, the model has been trained on the number of bins. We first need to determine how many characteristics are used in the model.

```r
# The list of characteristics is extracted from the list of bins names.
numberOfCharacteristics <-

  modelNames %>%

  # List of all names excluding the intercept which is not part of the scoring calculations (it
  # would mean giving points for free as a base line)
  filter(variableName != "(Intercept)") %>%

  # Extract strings ("[a-zA-Z0-9-_]*") preceded by an opening bracket ("(?<=\\()") and followed by a
  # closing bracket ("(?=\\))"). (The column names were formatted this way for that purpose.) See
  # "https://github.com/rstudio/cheatsheets/blob/master/strings.pdf" for details on the regex.
  mutate(characteristic = str_match(variableName, "(?<=\\()[a-zA-Z0-9-_]*(?=\\))")) %>%

  # We are only interested in how many distinct characteristic there are.
  distinct(characteristic) %>%
  nrow()

numberOfCharacteristics
```

```
## [1] 24
```

We can now perform the scoring calculation.

```r
ProbDefaultAtAnchor <- 1/20

# The model is trained so that the 'Good' outcome is that a loan does not default. The odds that
# therefore calculated on the probability of no default.
ProbAtAnchor <- 1 - ProbDefaultAtAnchor
OddsAnchor <- ProbAtAnchor / (1 - ProbAtAnchor)
ScoreAnchor <- 2000

# Doubling odds = 100 points
DoubleOdds <- 100

ScoreFactor <- OddsAnchor / log(2)

# 5,000 points is 20:1 odds of default
ScoreOffset <- ScoreAnchor - ScoreFactor * log(OddsAnchor)

# Score at the intercept
Intercept <- summary(GLModel)$coefficients["(Intercept)", "Estimate"]
```

Then we apportion across characteristics.

```r
ScorePerVariable <- (ScoreFactor * Intercept + ScoreOffset) / numberOfCharacteristics
InterceptPerVariable <- Intercept * ScoreFactor / numberOfCharacteristics

GLMScores <-
  GLMCoefficients %>%
  mutate(
    weight = Estimate * ScoreFactor,
    weightScaled = weight + ScorePerVariable,
    points = round(weightScaled)
  )
```

VERY IMPORTANT:

The intercept points have been allocated across all characteristics. Therefore the regression coefficient estimated for the intercept becomes redundant and needs removing.

```r
GLMScores[1, "points"] <-  0
```

The `speedglm` package does not have a `predict` function once models have been trained. However, using the model is a simple matrix multiplication: Loan matrix × Scorecard weights

3.6 Training set

```r
##             used   (Mb) gc trigger   (Mb)   max used    (Mb)
## Ncells   2775162  148.3    6815958  364.1    5130328   274.0
## Vcells 479346147 3657.2 1167993288 8911.1 1586076402 12100.9
```

```r
# Remove every variable that is not in the list of variables in the model then convert into a matrix
allMatrix <-
  allFactorsAsBins[, !is.na(match(
    names(allFactorsAsBins),
    str_remove_all(GLMCoefficients$variableName, "\`")
  ))] %>%
  as.matrix()
```

```r
# Add a column of 1s for the intercept
allMatrix <-
  cbind(as.vector(rep.int(
    x = 1, times = dim(allMatrix)[1]
  )), allMatrix)
dim(allMatrix)
```

```r
## [1] 1045084     168
```

```r
##             used   (Mb) gc trigger   (Mb)   max used    (Mb)
## Ncells   2775007  148.3    6815958  364.1    5130328   274.0
## Vcells 423434271 3230.6 1167993288 8911.1 1586076402 12100.9
```

```r
CoefficientsVector <- GLMCoefficients$Estimate %>% as.matrix()

# Score per variable
TrainingScorecard <- allMatrix %*% ( GLMScores$points %>% as.matrix() )
dim(allMatrix)
```

```r
## [1] 1045084     168
```

```r
dim(GLMScores$points)
```

```r
## NULL
```

```r
# CHECK: Any variable with NAs?
# GLMCoefficients$modelName[which(is.na(CoefficientsVector))]
# CoefficientsVector[is.na(CoefficientsVector)] <- 0
# dim(CoefficientsVector)

TrainingLogit <- allMatrix %*% CoefficientsVector
TrainingLogit <-
  enframe(TrainingLogit[, 1]) %>%
  mutate(oddsGood = exp(value),
         p = 1 / (1 + oddsGood)) %>%
  cbind(TrainingScorecard)
```

3.6.1 Densities of the training results

We plot the results of the training model and group the results by rating ("A" to "G") in Figure 3.1.

```
gridExtra::grid.arrange(
  loansTraining %>%
    cbind(TrainingLogit) %>%
    filter(between(value, -2, 5)) %>%
    ggplot(aes(value, col = grade)) +
    geom_density(adjust = 0.5) +
    ggtitle("Logit value"),

  loansTraining %>%
    cbind(TrainingLogit) %>%
    ggplot(aes(p, col = grade)) +
    geom_density(adjust = 0.5) +
    scale_x_log10() +
    ggtitle("Probability of being GOOD"),

  loansTraining %>%
    cbind(TrainingLogit) %>%
    filter(between(oddsGood, 0.1, 100)) %>%
    ggplot(aes(oddsGood, col = grade)) +
    geom_density(adjust = 0.5) +
    scale_x_log10() +
    ggtitle("Odds of being GOOD"),

  loansTraining %>%
    cbind(TrainingLogit) %>%
    ggplot(aes(TrainingScorecard, col = grade)) +
    geom_density(adjust = 1) +
    ggtitle("Scorecards"),

  ncol = 1
)
```

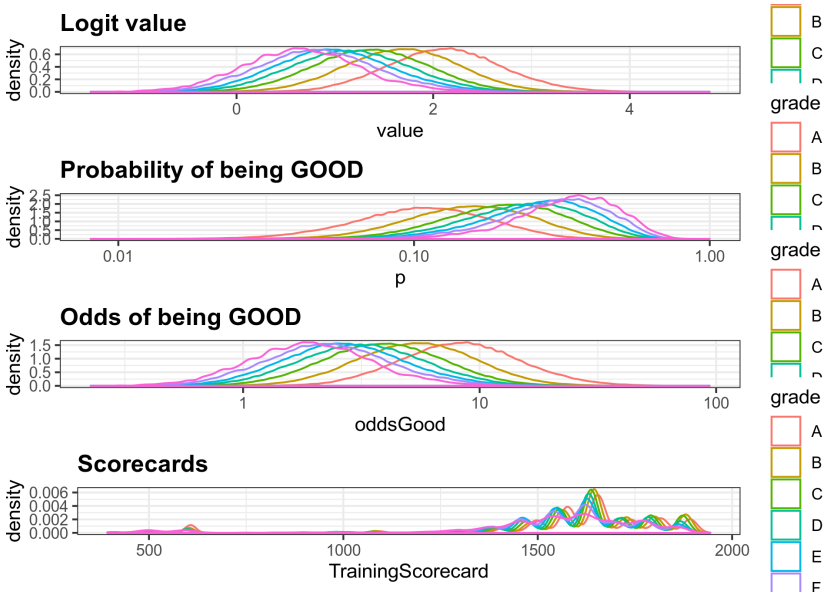


Figure 3.1: Model results on the training set

3.7 Test set

```
# Prepare the full test set
bestBins <- readRDS("datasets/bestBins100.rds")
predictionCategories <- loansTest[, "loanID"]

for (index in 1:length(bestBins$variable)) {
  binned <-
    binner::categoriseFromWoE.Wide(
      df = loansTest,
      varName = bestBins$variable[index],
      woeTable = bestBins$WoE[[index]]
    )

  predictionCategories <- cbind(predictionCategories, binned)
}

dim(predictionCategories)
```

```
## [1] 261272    222
```

```
# Retain only the relevant scorecard categories
predictionMatrix <-
  predictionCategories[, !is.na(match(
    names(predictionCategories),
    str_remove_all(GLMCoefficients$variableName, "\`")
  ))] %>%
  as.matrix()
```

```
predictionMatrix <- cbind(as.vector(rep.int(x = 1, times = dim(predictionMatrix)[1])), predictionMat
dim(predictionMatrix)
```

```
## [1] 261272    168
```

```
TestLogit <- predictionMatrix %*% CoefficientsVector
TestLogit <-
  tibble::enframe(TestLogit[, 1]) %>%
  mutate(
    p = 1 / (1 + exp(-value)),
    oddsGood = if_else(is.infinite(p / (1 - p)), 1e10, p / (1 - p)))

predictionScorecard <- predictionMatrix %*% ( GLMScores$points %>% as.matrix() )
```

```
loansTest %>%
  cbind(TestLogit) %>%
  cbind(predictionScorecard) %>%
  filter(predictionScorecard > 0) %>%
  ggplot(aes(predictionScorecard)) +
  geom_density(col = "blue", fill = "lightblue", adjust = 3)
```

```
median(predictionScorecard)
```

```
## [1] 1636
```

Same downward dynamics as training set

```
loansTest %>%
  cbind(TestLogit) %>%
  filter(between(value, -2, 5)) %>%
  ggplot(aes(value, col = grade)) +
  geom_density(adjust = 2)
```

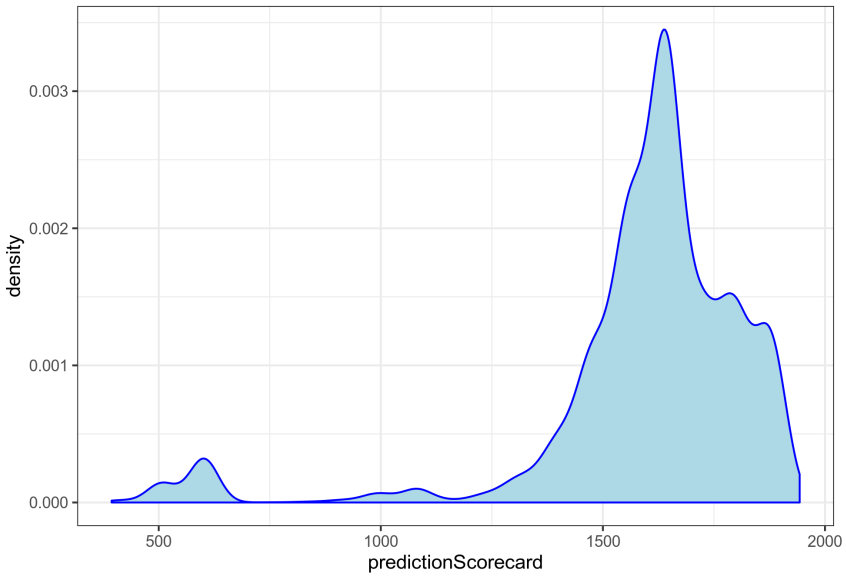


Figure 3.2: Density of loans by scorecard

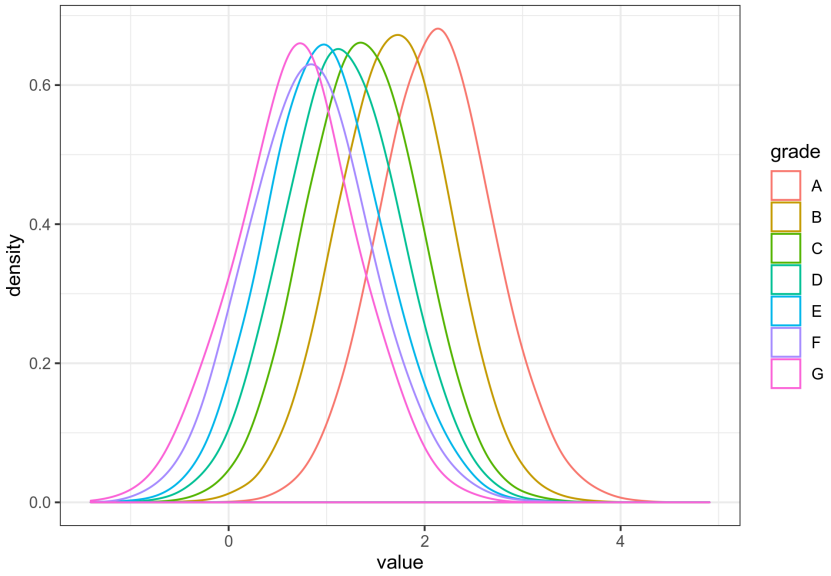


Figure 3.3: Logit value predicted by the model

```
loansTest %>%
  cbind(TestLogit) %>%
  ggplot(aes(p, col = grade)) +
  geom_density(adjust = 2) +
  scale_x_log10()
```

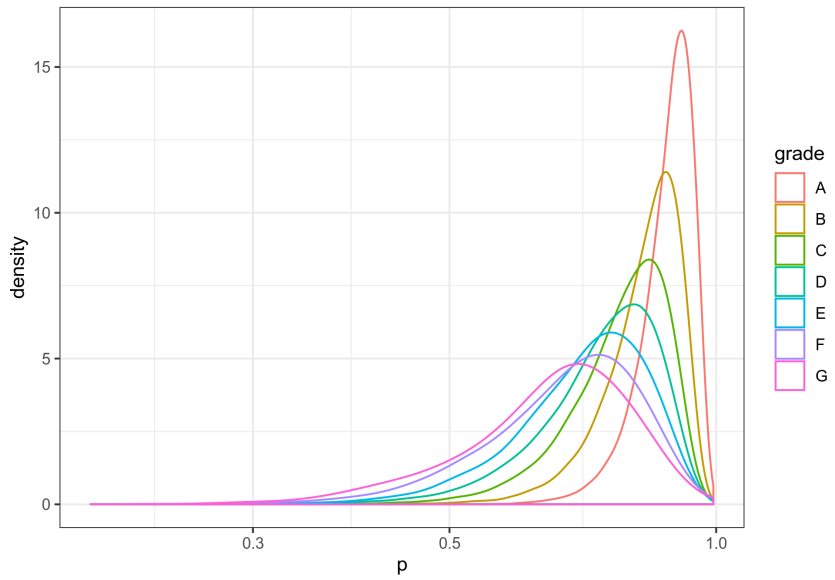


Figure 3.4: Probability of a loan being GOOD predicted by the model

```
loansTest %>%
  cbind(TestLogit) %>%
  filter(between(oddsGood, 0.1, 100)) %>%
  ggplot(aes(oddsGood, col = grade)) +
  geom_density(adjust = 2) +
  scale_x_log10()
```

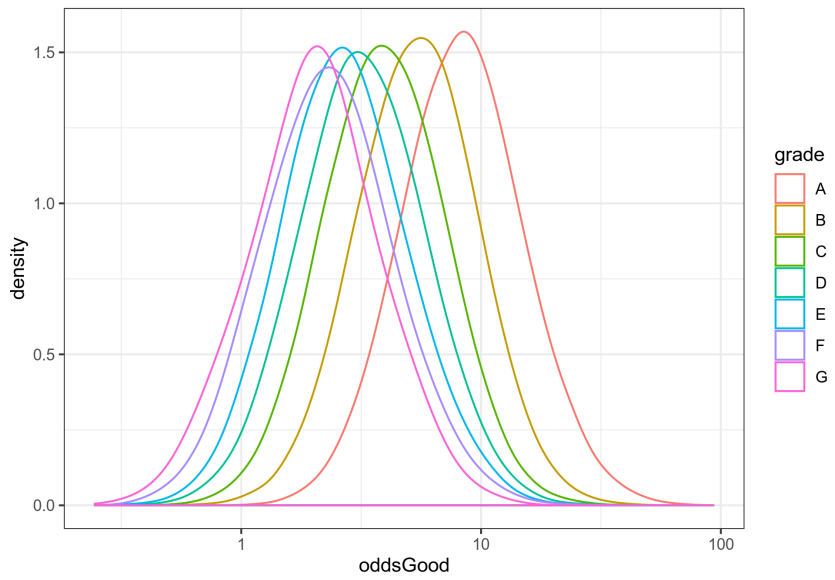


Figure 3.5: Odds of a loan being GOOD predicted by the model

3.8 Confusion matrix

Given a probability p from the model, we use a $p = 0.50$ cut-off point to decide whether a loan is Good or Bad. The Confusion Matrix results are:

```
tCM <- loansTest %>%
  cbind(TestLogit) %>%
  select(p, isGoodLoan) %>%

  mutate(p = if_else(p >= 0.50, "GOOD", "BAD"),
         isGoodLoan = if_else(isGoodLoan, "GOOD", "BAD")) %>%
  rename(Predicted = p,
         Actual = isGoodLoan) %>%

  table() %>%
  caret::confusionMatrix(positive = "GOOD")

tCM
```

```
## Confusion Matrix and Statistics
##
##           Actual
## Predicted    BAD   GOOD
##      BAD    1864   1512
##      GOOD  50657 207239
##
##                Accuracy : 0.8003
##                  95% CI : (0.7988, 0.8019)
##     No Information Rate : 0.799
##     P-Value [Acc > NIR] : 0.043
##
##                   Kappa : 0.0435
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.99276
##             Specificity : 0.03549
##          Pos Pred Value : 0.80358
##          Neg Pred Value : 0.55213
##              Prevalence : 0.79898
##          Detection Rate : 0.79319
##    Detection Prevalence : 0.98708
##       Balanced Accuracy : 0.51412
##
##        'Positive' Class : GOOD
##
```

The results suggest that the model is effrective at predicting good loans (measured by the sensitivity $= \frac{TP}{TP+FN} = 99.28\%$). However, this is deceptive since the dataset is unbalanced. The model is performing poorly at detecting bad loans (measured by the specificity $= \frac{TN}{TN+FP}$ $= 3.55\%$) The dataset is unbalanced and measuring the model's performance with a confusion matrix is imprecise. A much better approach would be to train many models on balanced datasets (by sampling a reduced 'good loans' dataset) and study the distribution of that resulting models and their parameters.

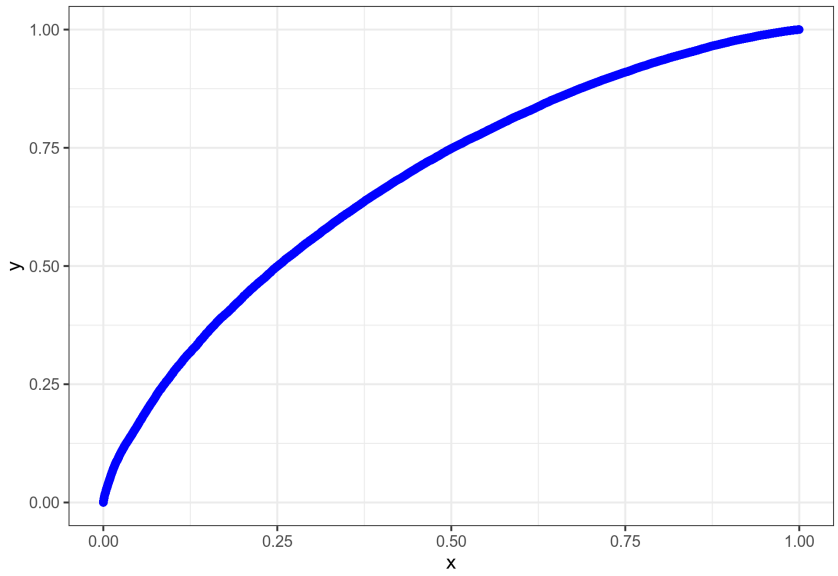
More critically, tThe confusion matrix does not (and cannot) reflect the consequence of getting predictions wrong. At the end of the day, the only relevant consequence is estimating the number of dollars lost on a loan. A misqualified loan might lead to a loss of a single dollar, or a million.

Predicting a probability of default is not enough. We need to subsequently predict the expected loss when a particular loan defaults. In the conclusion, we suggest one possible avenue.

3.9 ROC Curve

A popular measure for the performance of such a logistic regression model is to consider its *Receiver Operating Characteristic* curve (ROC) and calculate its area under curve (AUC). Ideally, a model able to perfectly classify

```
ROCRPrediction <- ROCR::prediction(TestLogit$value, loansTest$isGoodLoan)
ROCRPerformance <- ROCR::performance(ROCRPrediction, "tpr", "fpr")

ROCData <- tibble(x = ROCRPerformance@x.values[[1]],
       y = ROCRPerformance@y.values[[1]])

ROCData %>%
  ggplot(aes(x, y)) +
  geom_point(alpha = 0.2, col = "blue")
```




```
stats::ks.test(x = ROCData$x, y = ROCData$y)
```

```
##
##  Two-sample Kolmogorov-Smirnov test
##
## data:  ROCData$x and ROCData$y
## D = 0.25527, p-value < 2.2e-16
## alternative hypothesis: two-sided
```

Chapter 4

Conclusion

Is the interest rate proposed by LC high enough?

This report was one of those "it-builds-character" endeavour. Facing such a large dataset, it took a very long time before settling to address a particular question. Exploring the data lead to many blind alleys with little interest.

As a practical tool, this approach would not work in real life: we have no data on rejected loans. Using this model to accept/reject loans would need more work (using Reject Inference).

Currently only look at PD. Extend to LGD with Good Dollars and Bad Dollars instead of Good Loans / Bad Loans

Further possible explorations:

- Address the unbalanced dataset by sampling a number of training samples whose size is identical to the test dataset.
- Perform PCA beforehand
- Poisson GLM
- Improve the regularisation of the model parameters.
- Explore economic cycles/situations as additional entry.
- We use the entire dataset to estimate the impact of time as a polynomial curve. To assess future loans, this would not be acceptable. Only an online algorithm should be used. For example ARMA/ARIMA, Kalman filtering of the time-trend trajectory.
- Loss amount where good loan/bad loan proportion is replaced by a good dollar (repaid) / bad dollars (lost) fraction. See SAS Global Forum 2012.
- The size of the dataset is an issue to apply other techinques. But with stochastics methods, possible other models could be:
  - Tree models which have numerous variations (CART generally, and simple or aggregated boosted decision trees specifically)
  - Neural network

We initiated the exploration of potential models with Principal Component Analysis, Linear Regression, Extreme Boosting and Random Forest. No model could be trained on the full set.

We therefore came to limit the training set on a random sample of 0.1% (1 thousandth) of the initial full set.

Any model will require training in batch (e.g. stochastic gradient descent) or online. Our untested intuition is that online methods are not adapted to an unbalanced dataset: the number of defaults/write-offs is fairly low for high quality ratings. However, the dataset is evidently a time series which points to online training. We will not explore that line of investigation, although exploring that tension would be an interesting subject.

Chapter 5

Errands and post-mortem

This project took much longer that the MovieLens capstone. Here is a list of ideas explored, abandoned and lessons learned.

5.1 Geographical data

We sourced US zip and FIPS (*Federal Information Processing Standards*) codes, and macroeconomical data for possible geographical statistics. The source code for the data import and reformatting is available in the GitHub repository.

Macro-economical datasets were sourced from the same website as Microsoft Excel files. They were converted as-is to tab-separated csv files with LibreOffice.

- Median income per household: https://geofred.stlouisfed.org/map/?th=pubugn&cc=5&rc=false&im=fractile&sb&lng=-112.41&lat=44.31&zm=4&sl&sv&am=Average&at=Not%20Seasonally%20Adjusted,%20Annual,%20Dollars&sti=2022&fq=Annual&rt=county&un=lin&dt=2017-01-01
- Per capita personal income: https://geofred.stlouisfed.org/map/?th=pubugn&cc=5&rc=false&im=fractile&sb&lng=-112.41&lat=44.31&zm=4&sl&sv&am=Average&at=Not%20Seasonally%20Adjusted,%20Annual,%20Dollars&sti=882&fq=Annual&rt=county&un=lin&dt=2017-01-01
- Unemployment: https://geofred.stlouisfed.org/map/?th=rdpu&cc=5&rc=false&im=fractile&sb&lng=-90&lat=40&zm=4&sl&sv&am=Average&at=Not%20Seasonally%20Adjusted,%20Monthly,%20Percent&sti=1224&fq=Monthly&rt=county&un=lin&dt=2019-08-01

Of key interest was indicators of economic stress with the following intuition: if a borrower has a good credit standing, traditional banks would provide cheaper access to credit. In times of financial distress (indicated by higher unemployment, lower GDP growth, income per household, ...), we should see higher volumes of loans, and/or changes in the loan application patterns.

Given the time available, that data turned out to be too difficult to use:

- it was incomplete since the reported figures were not available over the entire timespan of the LendingClub dataset;
- ZIP codes and FIPS location reference change over time, meaning that determining economic indicators for a given loan was no easy to automatise or would have require substantial time-consuming hand-made adjustment.

Chapter 6

Stochastic Gradient Descent

The diagram 6.1[1] shows a useful decision tree.

```
knitr::include_graphics("images/scikit-learn-mlmap.png", auto_pdf = TRUE)
```

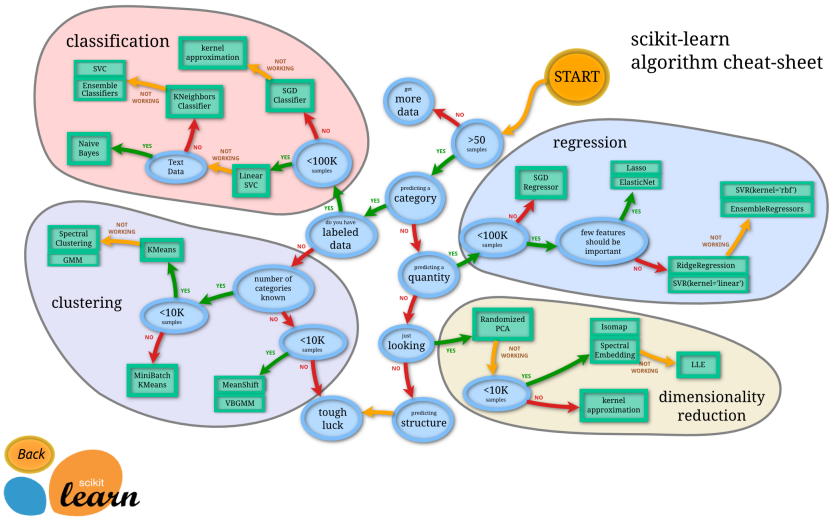


Figure 6.1: Scikit Learn algorithm cheat-sheet

Following the disappointing results of the previous section, we will explore a simpler model, but iteratively trained on a wider set of random samples.

6.1 Description of the model

6.2 Gradient descent

6.2.1 Generalities

Gradient descent is a generic numerical optimisation algorithm to iteratively converge towards a (sometimes local) minumum of a given function. It is extensively used in statistical learning to minimise error functions.

In the case of a simple linear regression model, the model training error J (the *cost function*) as a function of θ is:

[1] Source: https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html

$$J_{(\theta)} = \frac{1}{2} \sum_{i=1}^{N} \epsilon(y_i, \theta X_i)$$

where the model parameters are denoted θ_i, $X_i \in \mathbb{R}^{\times}$ are the predictors, $Y_i \in \mathbb{R}^{\times}$ are the responses and ϵ is a distance function. Typically, ϵ will be the Manhattan error or the Euclidian norm ($A = (a_1, \cdots, a_n), B = (b_1, \cdots, b_n)$).

Manhattan: $\epsilon(A, B) = \sum_{i=1}^{n} |a_i - b_i|)\$$

Euclidian norm: $\epsilon(A, B) = \sqrt{\sum_{i=1}^{n} (a_i - b_i)^2}$

The gradient descent algorithm uses the gradient of the error function, $\nabla J_{(\theta)}$, defined as:

$$\nabla J_{(\theta)} = \left(\frac{\partial J}{\partial \theta_0}, \frac{\partial J}{\partial \theta_1}, \cdots, \frac{\partial J}{\partial \theta_p} \right)$$

And in the case of linear regression is in a matrix form that can be computed efficiently:

$$\nabla J_{(\theta)} = \left(y^T - \theta X^T \right) X$$

The gradient decent algorithm finds parameters in the following manner iterating over the training samples:

While $||\alpha \nabla J_{(\theta)}|| > \eta$, $\theta := \theta - \alpha \nabla J_{(\theta)}$

In practice, the cost function will add a penalty term to regularise the model parameters (see below).

6.3 Stochastic Gradient Descent

With realistic datasets, gradient descent can experience slow convergence because (1) each iteration requires calculation of the gradient for every single training example, and (2) since each individual sample is potentially very different from another, the calculated gradient may not be optimal. In such case, the gradient descent can be done using a batch of several training samples and use the average of the cost function (**batch gradient descent**). This addresses those two sources of inefficiency.

This method however still requires iterating over the entire dataset. We can instead iterate over batched of random training samples drawn from the entire dataset, instead of being drawn sequentially. This is the *stochastic gradient descent*.

Aside from the choice of the initial choice of samples, and the averaging of the cost function, the update of θ remains identical.

6.3.0.1 Cost function regularisation and derivative

Regularisation is a way to decrease the complexity of models by narrowing the range that parameters can take. It has long been established that it assists the stability and robustness of learning algorithms. See Chapter 13 of (Shalev-Shwartz and Ben-David, 2014).

Our regularised cost function is written:

TODO

$$J_{P,Q} = \sum_{(i,j)\in\Omega} \left(r_{i,j} - \sum_{k=1}^{N_{features}} p_{i,k} q_{j,k} \right)^2 + \frac{\lambda}{2} \left(\sum_{i,k} p_{i,k}^2 + \sum_{j,k} q_{j,k}^2 \right)$$

The gradient descent algorithm seeks to minimise the $J_{(\theta)}$ cost function by step-wise update of each model parameter x as follows:

$$\theta_{t+1}^i \leftarrow \theta_t^i - \alpha \frac{\partial J_{(\theta)}}{\partial \theta_i}$$

The parameters are the matrix coefficients $p_{i,k}$ $q_{j,k}$. α is the learning parameter that needs to be adjusted.

6.3.0.2 Cost function partial derivatives

The partial derivatives of the cost function is:

$$\frac{\partial J_{P,Q}}{\partial x} = \frac{\partial}{\partial x} \left(\sum_{(i,j)\in\Omega} \left(r_{i,j} - \sum_{k=1}^{N_{features}} p_{i,k} q_{j,k} \right)^2 + \frac{\lambda}{2} \left(\sum_{i,k} p_{i,k}^2 + \sum_{j,k} q_{j,k}^2 \right) \right)$$

$$\frac{\partial J_{P,Q}}{\partial x} = \sum_{(i,j)\in\Omega} 2 \frac{\partial r_{i,j} - \sum_{k=1}^{N_{features}} p_{i,k} q_{j,k}}{\partial x} \left(r_{i,j} - \sum_{k=1}^{N_{features}} p_{i,k} q_{j,k} \right) + \frac{\lambda}{2} \left(\sum_{i,k} 2 \frac{\partial p_{i,k}}{\partial x} p_{i,k} + \sum_{j,k} 2 \frac{\partial p_{i,k}}{\partial x} q_{j,k} \right)$$

We note that $r_{i,j}$ are constants

$$\frac{\partial J_{P,Q}}{\partial x} = 2 \sum_{(i,j)\in\Omega} \sum_{k=1}^{N_{features}} \left(\frac{\partial - p_{i,k} q_{j,k}}{\partial x} \left(r_{i,j} - \sum_{k=1}^{N_{features}} p_{i,k} q_{j,k} \right) \right) + \lambda \left(\sum_{i,k} \frac{\partial p_{i,k}}{\partial x} p_{i,k} + \sum_{j,k} \frac{\partial p_{i,k}}{\partial x} q_{j,k} \right)$$

If x is a coefficient of P (resp. Q), say $p_{a,b}$ (resp. $q_{a,b}$), all partial derivatives will be nil unless for $(i,j) = (a,b)$.

Therefore:

$$\frac{\partial J_{P,Q}}{\partial p_{a,b}} = -2 \sum_{(i,j)\in\Omega} q_{j,b} \left(r_{i,j} - \sum_{k=1}^{N_{features}} p_{i,k} q_{j,k} \right) + \lambda p_{a,b}$$

and,

$$\frac{\partial J_{P,Q}}{\partial q_{a,b}} = -2 \sum_{(i,j)\in\Omega} p_{i,b} \left(r_{i,j} - \sum_{k=1}^{N_{features}} p_{i,k} q_{j,k} \right) + \lambda q_{a,b}$$

Since $\epsilon i,j = r_{i,j} - \sum_{k=1}^{N_{features}} p_{i,k} q_{j,k}$ is the rating prediction error, this becomes:

$$\frac{\partial J_{P,Q}}{\partial p_{a,b}} = -2 \sum_{(i,j)\in\Omega} q_{j,b} \epsilon_{i,j} + \lambda p_{a,b}$$

and,

$$\frac{\partial J_{P,Q}}{\partial q_{a,b}} = -2 \sum_{(i,j)\in\Omega} p_{i,b}\epsilon_{i,j} + \lambda q_{a,b}$$

We note that the recent implementations of autodifferentiation algorithm (key to deep learning implementations) would avoid making those calculations by hand. We did not attempt to use the R `madness` package [2].

6.3.1 Stochastic Gradient Descent (SGD)

The size of the datasets is prohibitive to do those calculations across the entire training set.

Instead, we will repeatedly update the model parameters on small random samples of the training set.

Chapter 14 of (Shalev-Shwartz and Ben-David, 2014) gives an extensive introduction to various SGD algorithms.

We implemented a simple version of the algorithm and present the code in more detail.

```
# The algorithm is implemented from scratch and relies on nothing but the `Tidyverse` libraries.
library(tidyverse)

# The quality of the training and predictions is measured by the _root mean squared error_
# (RMSE), for which we define a few helper functions (the global variables are defined
# later):
rmse_training <- function() {
  prediction_Z <- rowSums(LF_Model$P[tri_train$userN,] *
                            LF_Model$Q[tri_train$movieN,])
  prediction <- prediction_Z * r_sd + r_m
  sqrt(sum((tri_train$rating - prediction) ^ 2 / nSamples))
}

rmse_validation <- function() {
  prediction_Z <- rowSums(LF_Model$P[tri_test$userN,] *
                            LF_Model$Q[tri_test$movieN,])
  prediction <- prediction_Z * r_sd + r_m
  sqrt(sum((tri_test$rating - prediction) ^ 2) / nTest)
}

sum_square <- function(v) {
  return (sqrt(sum(v ^ 2) / nrow(v)))
}

# The key function updates the model coefficients. Its inputs are:
#
# + a list that contains the $P$ an $Q$ matrices, the training RMSE of those matrices, and
# a logical value indicating whether this RMSE is worse than what it was before the update
# (i.e. did the update diverge).
#
# + a `batch_size` that defines the number of samples to be drawn from the training set. A
# normal gradient descent would use the full training set; by default we only use 10,000
# samples out of 10 million (one tenth of a percent).
#
# + The cost regularisation `lambda` and gradient descent learning parameter `alpha`.
```

[2] https://github.com/shabbychef/madness

```r
#
# + A number of `times` to run the descent before recalculating the RMSE and exiting the
# function (calculating the RMSE is computationally expensive).
#
#
# The training set used is less rich than the original set. As discussed, it only uses the
# rating (more exactly on the z_score of the rating). Genres, timestamps,... are
# discarded.


# Iterate gradient descent
stochastic_grad_descent <- function(model,
                                    times = 1,
                                    batch_size = 10000,
                                    lambda = 0.1,
                                    alpha = 0.01,
                                    verbose = TRUE) {
  # Run the descent `times` times.
  for (i in 1:times) {
    # Extract a sample of size `batch_size` from the training set.
    spl <- sample(1:nSamples, size = batch_size, replace = FALSE)
    spl_training_values <- tri_train[spl,]


    # Take a subset of `P` and `Q` matching the users and
    # movies in the training sample.
    spl_P <- model$P[spl_training_values$userN,]
    spl_Q <- model$Q[spl_training_values$movieN,]

    # rowSums returns the cross-product for a given user and movie.
    # err is the term inside brackets in the partial derivatives
    # calculation above.
    err <- spl_training_values$rating_z - rowSums(spl_P * spl_Q)

    # Partial derivatives wrt p and q
    delta_P <- -err * spl_Q + lambda * spl_P
    delta_Q <- -err * spl_P + lambda * spl_Q

    model$P[spl_training_values$userN,]  <- spl_P - alpha * delta_P
    model$Q[spl_training_values$movieN,] <- spl_Q - alpha * delta_Q

  }

  # RMSE against the training set
  error <- sqrt(sum((
    tri_train$rating_z - rowSums(model$P[tri_train$userN,] *
                                   model$Q[tri_train$movieN,])
  ) ^ 2)
  / nSamples)

  # Compares to RMSE before update
  model$WORSE_RMSE <- (model$RMSE < error)
  model$RMSE <- error

  # Print some information to keep track of success
  if (verbose) {
    cat(
      "  # features=",
```

```
      ncol(model$P),
      "  J=",
      nSamples * error ^ 2 +
        lambda / 2 * (sum(model$P ^ 2) + sum(model$Q ^ 2)),
      "  Z-scores RMSE=",
      model$RMSE,
      "\n"
    )
    flush.console()
  }

  return(model)
}


rm(list_results)
list_results <- tibble(
  "alpha" = numeric(),
  "lambda" = numeric(),
  "nFeatures" = numeric(),
  "rmse_training_z_score" = numeric(),
  "rmse_training" = numeric(),
  "rmse_validation" = numeric()
)

# The main training loop runs as follows:
#
# + We start with 3 features.
#
# + The model is updated in batches of 100 updates. This is done up to 250 times. At each
# time, if the model starts diverging, the learning parameter ($\alpha$) is reduced.
#
# + Once the 250 times have passed, or if $\alpha$ has become incredibly small, or if the
# RMSE doesn't really improve anymoe (by less than 1 millionth), we add another features
# and start again.

initial_alpha <- 0.1
for (n in 1:100) {
  # Current number of features
  number_features <- ncol(LF_Model$P)

  # lambda = 0.01 for 25 features, i.e. for about 2,000,000 parameters.
  # We keep lambda proportional to the number of features
  lambda <- 0.1 * (nUsers + nMovies) * number_features / 2000000

  alpha <- initial_alpha

  cat(
    "CURRENT FEATURES: ",
    number_features,
    "---- Pre-training validation RMSE = ",
    rmse_validation(),
    "\n"
  )

  list_results <- list_results %>% add_row(
    alpha = alpha,
    lambda = lambda,
```

```
    nFeatures = number_features,
    rmse_training_z_score = LF_Model$RMSE,
    rmse_training = rmse_training(),
    rmse_validation = rmse_validation()
  )

  for (i in 1:250) {
    pre_RMSE <- LF_Model$RMSE
    LF_Model <- stochastic_grad_descent(
      model = LF_Model,
      times = 100,
      batch_size = 1000 * number_features,
      alpha = alpha,
      lambda = lambda
    )

    list_results <- list_results %>% add_row(
      alpha = alpha,
      lambda = lambda,
      nFeatures = number_features,
      rmse_training_z_score = LF_Model$RMSE,
      rmse_training = rmse_training(),
      rmse_validation = rmse_validation()
    )

    if (LF_Model$WORSE_RMSE) {
      alpha <- alpha / 2
      cat("Decreasing gradient parameter to: ", alpha, "\n")
    }

    if (initial_alpha / alpha > 1000 |
        abs((LF_Model$RMSE - pre_RMSE) / pre_RMSE) < 1e-6) {
      break()
    }
  }


  # RMSE against validation set:
  rmse_validation_post <- rmse_validation()
  cat(
    "CURRENT FEATURES: ",
    number_features,
    "---- POST-training validation RMSE = ",
    rmse_validation_post,
    "\n"
  )

  # if (number_features == 12){
  #   break()
  # }


  # Add k features
  k_features <- 1
  LF_Model$P <- cbind(LF_Model$P,
                      matrix(
                        rnorm(
                          nrow(LF_Model$P) * k_features,
```

```
                          mean = 0,
                          sd = sd(LF_Model$P) / 100
                        ),
                        nrow = nrow(LF_Model$P),
                        ncol = k_features
                      ))

  LF_Model$Q <- cbind(LF_Model$Q,
                      matrix(
                        rnorm(
                          nrow(LF_Model$Q) * k_features,
                          mean = 0,
                          sd = sd(LF_Model$Q) / 100
                        ),
                        nrow = nrow(LF_Model$Q),
                        ncol = k_features
                      ))

}

saveRDS(list_results, "datasets/LRMF_results.rds")
```

6.4 Various errands

6.4.1 Optimising the scorecard (or rating) depending on the NPV curve.

Input - Layer - Single Q - Distribution

Cost function as a function of the Q parameter (equivalent to scorecard).

Looking back at the distribution of the NPV between the -1 and about 1.5, it is multimodal and looks like the sum of 4 log-normal distributions with modes centered on about

$$PDF(x) = \frac{1}{x\sigma\sqrt{2\pi}} exp\left(-\frac{1}{2}\left(\frac{\log(x)-\mu}{\sigma}\right)^2\right)$$

$$\text{mode} = m = e^{\mu-\sigma^2}, \text{ therefore: } \mu = log(\text{mode}) + \sigma^2$$

$$PDF(x) = \sqrt{\frac{e}{2\pi}}\frac{1}{x\sigma} exp\left(-\frac{1}{2}\left(\frac{\log(\frac{x}{m})-\sigma^2}{\sigma}\right)^2\right)$$

We will center the distribution on the mode, therefore:

$$PDF(x) = \sqrt{\frac{e}{2\pi}}\frac{1}{(x-m)\,\sigma} exp\left(-\frac{1}{2}\left(\frac{\log(\frac{x-m}{m})-\sigma^2}{\sigma}\right)^2\right)$$

The distribution's tail is towards positive infinity. For the symmetric result, we would replace $(x-m)$ by $-(x-m)$.

If we use 4 log-normal distributions, the cost function is:

$$\mathrm{J}\left(x,Q\right)=-[x-\left(\alpha_1\,\mathrm{PDF}_1\left(x,Q\right)+\alpha_2\,\mathrm{PDF}_2\left(x,Q\right)+\alpha_2\,\mathrm{PDF}_3\left(x,Q\right)+\alpha_4\,\mathrm{PDF}_4\left(x,Q\right)\right)]^2$$

To optimise the shape of the total multi-modal distribution, we will assume that each α, m and σ is a linear function of Q. The derivative $\frac{\partial \mathrm{J}}{\partial Q}$ is[3]:

$$\begin{aligned}\frac{\partial \mathrm{J}}{\partial Q}\left(x,Q\right)=-\sqrt{\frac{2}{\pi}}x \quad &- \quad \sqrt{\frac{2}{\pi}}\frac{\alpha_1}{\sigma_1(-x+m_1)}e^{-\frac{1}{2}\left(\frac{\log\left(\frac{-x+m_1}{m_1}\right)+\sigma_1^2}{\sigma_1}\right)^2}\\ &- \quad \frac{1}{\sqrt{2\pi}}\frac{\alpha_2}{\sigma_2(-x+m_2)}e^{-\frac{1}{2}\left(\frac{\log\left(\frac{-x+m_2}{m_2}\right)+\sigma_2^2}{\sigma_2}\right)^2}\\ &- \quad \sqrt{\frac{2}{\pi}}\frac{\alpha_3}{\sigma_3(-x+m_3)}e^{-\frac{1}{2}\left(\frac{\log\left(\frac{-x+m_3}{m_3}\right)+\sigma_3^2}{\sigma_3}\right)^2}\\ &- \quad \sqrt{\frac{2}{\pi}}\frac{\alpha_4}{\sigma_4(x-m_4)}e^{-\frac{1}{2}\left(\frac{\log\left(\frac{x-m_4}{m_4}\right)+\sigma_4^2}{\sigma_4}\right)^2}\end{aligned}$$

[3]This was actually generated using Maxima (code in appendix) which allows for quicker iterations.

Appendix

6.5 List of assumptions / limitations regarding the dataset

As mentioned during this report, we had to make numerous assumptions given the lack of clarity of the variable descriptions.

- *Dataset quality*: Aside from cents rounding issues, the dataset does not contain any flagrant errors that we could see (e.g. minor error of amount or rate, zipcode). Quality of the variable description is a different matter altogether.
- *Ratings*: The day-1 rating is between A1 and (and no lower than) G5. No note is rated lower than E5 after 6 November 2017, and lower than D5 after 30 June 2019.
- *Credit history*: Credit history information for the principal borrower relates to pre-approval and not post-funding. This is clear for the joint applicants, but simply an assumption for the principal borrower.
- *Recoveries*: Recoveries (if any) are assumed to be paid 3 months after the last scheduled payment date (variable `last_pymnt_d`)
- *Survival effect*: The dataset does not include applications that were rejected by the lender (for whatever reason) or by the borrower (for example because the interest rate quote is too high). It may also be the case that some actual loans were excluded as and when the dataset changed over the years.
- LIBOR funding rate: we use the 3-year and 5-year swap rates. In reality, we should have used average tenor-weighted swap rates (i.e. ca. 1.5 Y and 2.5 Y). This requires a full swap curve and more calculation than necessary for our purpose. The principles of this report should not be significantly affted by this approximation.

We do hope that LendingClub investors receive information of much better quality!

6.6 Data preparation and formatting

We used different sources of information:

- The LendingClub dataset made available on Kaggle;
- US geographical data about zip and FIPS codes;
- Market interest rates from the Saint Louis Federal Reserve Bank; and,
- Macro data from the same source.

We here show the code used to prepare the data. It was automatically formatted by *RStudio*.

6.6.1 LendinClub dataset

```
local({
  #
  # STEP 1: Download the dataset
  #
  #   Got to https://www.kaggle.com/wendykan/lending-club-loan-data
  #
  #   Download into the 'datasets' subdirectory
  #   Unzip the file.
  #   WARNING: The unzipping will be about 2.4GB
  #
  #   Name the sql database "datasets/lending_club.sqlite"
  #

  #
  # STEP 2: Prepare the dabase as a tibble
  #

  ##
  ## WARNING: THIS ASSUMES A 'datasets' DIRECTORY WAS CREATED
  ##
  library(RSQLite)
  db_conn <-
    dbConnect(RSQLite::SQLite(), "datasets/lending_club.sqlite")
  dbListTables(db_conn)

  # Returns a 2.96GB data frame
  lending_club <- dbGetQuery(db_conn, "SELECT * FROM loan")
  lending_club <- as_tibble(lending_club)

  # Close the database
  dbDisconnect(db_conn)

  # Compressed to ca.285MB on disk
  saveRDS(lending_club, "datasets/lending_club.rds")


  library(tidyverse)
  library(lubridate)
  library(hablar)

  # Before reformat in case the previous step was already done
  # lending_club <- readRDS("datasets/lending_club.rds")
  #
  # str(lending_club)
  #

  # Leave the original dataset untouched and work with a copy.
  lc <- lending_club

  lc <- lc %>%

    # Add loan identification number to track the loans across calculations
    mutate(loanID = row_number()) %>%

    # Remove useless strings
    mutate(
```

```
    term       = str_remove(term, " months"),
    emp_length = str_replace(emp_length, "<1", "0"),
    emp_length = str_replace(emp_length, "10+", "10"),
    emp_length = str_remove(emp_length, "years")
  ) %>%

  # Creates dates out of strings - Parse errors will be raised when no dates.
  mutate(
    debt_settlement_flag_date = as_date(dmy(
      str_c("1-", debt_settlement_flag_date)
    )),
    earliest_cr_line          = as_date(dmy(str_c(
      "1-", earliest_cr_line
    ))),
    hardship_start_date       = as_date(dmy(str_c(
      "1-", hardship_start_date
    ))),
    hardship_end_date         = as_date(dmy(str_c(
      "1-", hardship_end_date
    ))),
    issue_d                   = as_date(dmy(str_c("1-", issue_d))),
    last_credit_pull_d        = as_date(dmy(str_c(
      "1-", last_credit_pull_d
    ))),
    last_pymnt_d              = as_date(dmy(str_c(
      "1-", last_pymnt_d
    ))),
    next_pymnt_d              = as_date(dmy(str_c(
      "1-", next_pymnt_d
    ))),
    payment_plan_start_date   = as_date(dmy(str_c(
      "1-", payment_plan_start_date
    ))),
    sec_app_earliest_cr_line  = as_date(dmy(str_c(
      "1-", sec_app_earliest_cr_line
    ))),
    settlement_date           = as_date(dmy(str_c(
      "1-", settlement_date
    )))
  ) %>%

  # Bulk type conversion with convert from the `hablar` package
  convert(
    # Strings
    chr(emp_title, title, url, zip_code),

    # Factors
    fct(
      addr_state,
      application_type,
      debt_settlement_flag,
      desc,
      disbursement_method,
      grade,
      hardship_flag,
      hardship_loan_status,
      hardship_reason,
      hardship_status,
```

```
        hardship_type,
        home_ownership,
        id,
        initial_list_status,
        loan_status,
        member_id,
        policy_code,
        purpose,
        pymnt_plan,
        settlement_status,
        sub_grade,
        verification_status,
        verification_status_joint
      ),

      # Integers
      int(
        acc_now_delinq,
        acc_open_past_24mths,
        chargeoff_within_12_mths,
        collections_12_mths_ex_med,
        deferral_term,
        delinq_2yrs,
        emp_length,
        hardship_dpd,
        hardship_length,
        inq_fi,
        inq_last_12m,
        inq_last_6mths,
        mo_sin_old_il_acct,
        mo_sin_old_rev_tl_op,
        mo_sin_rcnt_rev_tl_op,
        mo_sin_rcnt_tl,
        mort_acc,
        mths_since_last_delinq,
        mths_since_last_major_derog,
        mths_since_last_record,
        mths_since_rcnt_il,
        mths_since_recent_bc,
        mths_since_recent_bc_dlq,
        mths_since_recent_inq,
        mths_since_recent_revol_delinq,
        num_accts_ever_120_pd,
        num_actv_bc_tl,
        num_actv_rev_tl,
        num_bc_sats,
        num_bc_tl,
        num_il_tl,
        num_op_rev_tl,
        num_rev_accts,
        num_rev_tl_bal_gt_0,
        num_sats,
        num_tl_120dpd_2m,
        num_tl_30dpd,
        num_tl_90g_dpd_24m,
        num_tl_op_past_12m,
        open_acc,
        open_acc_6m,
```

```
        open_act_il,
        open_il_12m,
        open_il_24m,
        open_rv_12m,
        open_rv_24m,
        sec_app_chargeoff_within_12_mths,
        sec_app_collections_12_mths_ex_med,
        sec_app_inq_last_6mths,
        sec_app_mort_acc,
        sec_app_mths_since_last_major_derog,
        sec_app_num_rev_accts,
        sec_app_open_acc,
        sec_app_open_act_il,
        term
      ),

      # Floating point
      dbl(
        all_util,
        annual_inc,
        annual_inc_joint,
        avg_cur_bal,
        bc_open_to_buy,
        bc_util,
        collection_recovery_fee,
        delinq_amnt,
        dti,
        dti_joint,
        funded_amnt,
        funded_amnt_inv,
        hardship_amount,
        hardship_last_payment_amount,
        hardship_payoff_balance_amount,
        il_util,
        installment,
        int_rate,
        last_pymnt_amnt,
        loan_amnt,
        max_bal_bc,
        orig_projected_additional_accrued_interest,
        out_prncp,
        out_prncp_inv,
        pct_tl_nvr_dlq,
        percent_bc_gt_75,
        pub_rec,
        pub_rec_bankruptcies,
        recoveries,
        revol_bal,
        revol_bal_joint,
        revol_util,
        sec_app_revol_util,
        settlement_amount,
        settlement_percentage,
        tax_liens,
        tot_coll_amt,
        tot_cur_bal,
        tot_hi_cred_lim,
        total_acc,
```

```
        total_bal_ex_mort,
        total_bal_il,
        total_bc_limit,
        total_cu_tl,
        total_il_high_credit_limit,
        total_pymnt,
        total_pymnt_inv,
        total_rec_int,
        total_rec_late_fee,
        total_rec_prncp,
        total_rev_hi_lim
      )
    ) %>%

    # Converts some values to 1/-1 (instead of Boolean)
    mutate(
      pymnt_plan =           if_else(pymnt_plan == "y",           1, -1),
      hardship_flag =        if_else(hardship_flag == "Y",        1, -1),
      debt_settlement_flag = if_else(debt_settlement_flag == "Y", 1, -1)
    ) %>%

    # Some values are percentages
    mutate(
      int_rate = int_rate / 100,
      dti = dti / 100,
      dti_joint = dti_joint / 100,
      revol_util = revol_util / 100,
      il_util = il_util / 100,
      all_util = all_util / 100,
      bc_open_to_buy = bc_util / 100,
      pct_tl_nvr_dlq = pct_tl_nvr_dlq / 100,
      percent_bc_gt_75 = percent_bc_gt_75 / 100,
      sec_app_revol_util = sec_app_revol_util / 100
    ) %>%

    # Create quasi-centered numerical grades out of grade factors with "A" = 3 down to "G" = -3
    mutate(grade_num = 4 - as.integer(grade)) %>%

    # Ditto with sub_grades. "A1" = +3.4, "A3" = +3.0, down to "G3" = -3.0, "G5" = -3.4
    mutate(sub_grade_num = 3.6 - as.integer(sub_grade) / 5) %>%

    # Keep the first 3 digits of the zipcode as numbers
    mutate(zip_code = as.integer(str_sub(zip_code, 1, 3))) %>%

    # order by date
    arrange(issue_d) %>%

    # Remove empty columns
    select(-id, -member_id, -url)

  saveRDS(lc, "datasets/lending_club_reformatted.rds")


  # Select loans which have matured or been terminated
  past_loans <- lc %>%
    filter(
      loan_status %in% c(
        "Charged Off",
```

```
        "Does not meet the credit policy. Status:Charged Off",
        "Does not meet the credit policy. Status:Fully Paid",
        "Fully Paid"
      )
    )

  saveRDS(past_loans, "datasets/lending_club_reformatted_paid.rds")
})
```

6.6.2 Zip codes and FIPS codes

The R package `zipcode` was installed.

```
#
# ZIPCodes dataset.
#

library(zipcode)
data(zipcode)
zips <- zipcode %>%
  as_tibble() %>%
  mutate(zip = as.integer(str_sub(zip, 1, 3)))

saveRDS(zips, "datasets/zips.rds")
```

A csv file containing zip codes, FIPS codes and population information was downloaded from the *Simple Maps* [4] website.

```
local({
  kaggleCodes <- read.csv("datasets/csv/ZIP-COUNTY-FIPS_2017-06.csv")

  kaggleCodes <-
    kaggleCodes %>%
    as_tibble() %>%
    mutate(zip = floor(ZIP/100),
           FIPS = STCOUNTYFP,
           COUNTYNAME = str_replace(COUNTYNAME, pattern = "County", replacement = ""),
           COUNTYNAME = str_replace(COUNTYNAME, pattern = "Borough", replacement = ""),
           COUNTYNAME = str_replace(COUNTYNAME, pattern = "Municipio", replacement = ""),
           COUNTYNAME = str_replace(COUNTYNAME, pattern = "Parish", replacement = ""),
           COUNTYNAME = str_replace(COUNTYNAME, pattern = "Census Area", replacement = "")) %>%
    rename(county = COUNTYNAME) %>%
    select(zip, county, FIPS) %>%
    arrange(zip)

  saveRDS(zipfips, "datasets/kaggleCodes.rds")
})
```

6.6.3 Market interest rates

Market interest rates (3-year and 5-year swap rates) were download from the Saint Louis Federal Reserve Bank. Datasets are split between before and after the LIBOR fixing scandal. The datasets are merged with disctinct dates.

Download sources are:

- Pre-LIBOR 3-y swap https://fred.stlouisfed.org/series/DSWP3

[4] https://simplemaps.com/data/us-zips

- Post-LIBOR 3-y swap https://fred.stlouisfed.org/series/ICERATES1100USD3Y
- Pre-LIBOR 5-y swap https://fred.stlouisfed.org/series/MSWP5
- Post-LIBOR 5-y swap https://fred.stlouisfed.org/series/ICERATES1100USD5Y

```
local({
  LIBOR3Y <- read.csv("datasets/csv/DSWP3.csv") %>%
    as_tibble() %>%
    filter(DSWP3 != ".") %>%
    mutate(DATE = as_date(DATE),
           RATE3Y = as.numeric(as.character(DSWP3)) / 100) %>%
    select(DATE, RATE3Y)

  ICE3Y <- read.csv("datasets/csv/ICERATES1100USD3Y.csv") %>%
    as_tibble() %>%
    filter(ICERATES1100USD3Y != ".") %>%
    mutate(DATE = as_date(DATE),
           RATE3Y = as.numeric(as.character(ICERATES1100USD3Y)) / 100) %>%
    select(DATE, RATE3Y)


  LIBOR5Y <- read.csv("datasets/csv/DSWP5.csv") %>%
    as_tibble() %>%
    filter(DSWP5 != ".") %>%
    mutate(DATE = as_date(DATE),
           RATE5Y = as.numeric(as.character(DSWP5)) / 100) %>%
    select(DATE, RATE5Y)

  ICE5Y <- read.csv("datasets/csv/ICERATES1100USD5Y.csv") %>%
    as_tibble() %>%
    filter(ICERATES1100USD5Y != ".") %>%
    mutate(DATE = as_date(DATE),
           RATE5Y = as.numeric(as.character(ICERATES1100USD5Y)) / 100) %>%
    select(DATE, RATE5Y)

  RATES3Y <- LIBOR3Y %>% rbind(ICE3Y) %>%
    arrange(DATE) %>% distinct(DATE, .keep_all = TRUE)

  RATES5Y <- LIBOR5Y %>% rbind(ICE5Y) %>%
    arrange(DATE) %>% distinct(DATE, .keep_all = TRUE)

  saveRDS(RATES3Y, "datasets/rates3Y.rds")
  saveRDS(RATES5Y, "datasets/rates5Y.rds")


  # Note there are 7212 days from 1 Jan 2000 to 30 Sep 2019
  #
  # (ymd("2000-01-01") %--% ymd("2019-09-30")) %/% days(1)
  RATES <- tibble(n = seq(0, 7212)) %>%

    # Create a column with all dates
    mutate(DATE = ymd("2000-01-01") + days(n)) %>%
    select(-n) %>%

    # Add all daily 3- then 5-year rates and fill missing down
    left_join(RATES3Y) %>%
    fill(RATE3Y, .direction = "down") %>%

```

```
    left_join(RATES5Y) %>%
    fill(RATE5Y, .direction = "down")

  saveRDS(RATES, "datasets/rates.rds")
})
```

6.7 List of variables

This table presents the list of variables provided in the original dataset. The descriptions come from a spreadsheet attached with the dataset and, unfortunately, are not extremely precise and subject to interpretation. We added comments and/or particular interpretations in *CAPITAL LETTERS*.

```
LC_variable %>%
  select(variable_name, inModel, description) %>%
  mutate(inModel = if_else(inModel, "YES", "NO")) %>%

  # Format the table.
  kable(
    "latex",
    caption = "Description of the dataset variables as provided in the dataset downloaded from Kaggle
    booktabs = T,
    longtable = T,
    col.names = c("Variable Name", "Used in model?", "Description")
  ) %>%
  kable_styling(full_width = F,
                latex_options = c("repeat_header")) %>%
  column_spec(1, width = "4cm") %>%
  column_spec(2, width = "3cm") %>%
  column_spec(3, width = "7cm")
```

Table 6.1: Description of the dataset variables as provided in the dataset downloaded from Kaggle

Variable Name	Used in model?	Description
loanID	YES	NOTE THIS IS NOT AN ORIGINAL VARIABLE. IT WAS ADDED FOR THE PURPOSE OF TRACKING LOANS INDIVIDUALLY AS AND WHEN NEEDED.
loan_amnt	YES	The listed amount of the loan applied for by the borrower. If at some point in time, the credit department reduces the loan amount, then it will be reflected in this value.
funded_amnt	NO	The total amount committed to that loan at that point in time.
funded_amnt_inv	NO	The total amount committed by investors for that loan at that point in time.
term	YES	The number of payments on the loan. Values are in months and can be either 36 or 60.
int_rate	YES	Interest Rate on the loan
installment	NO	The monthly payment owed by the borrower if the loan originates.
grade	YES	LC assigned loan grade
sub_grade	YES	LC assigned loan subgrade

Table 6.1: Description of the dataset variables as provided in the dataset downloaded from Kaggle *(continued)*

Variable Name	Used in model?	Description
emp_title	NO	The job title supplied by the Borrower when applying for the loan.
emp_length	YES	Employment length in years. Possible values are between 0 and 10 where 0 means less than one year and 10 means ten or more years.
home_ownership	YES	The home ownership status provided by the borrower during registration or obtained from the credit report. Our values are: RENT, OWN, MORTGAGE, OTHER, NONE
annual_inc	NO	The self-reported annual income provided by the borrower during registration. NOT USED AS A VARIABLE SINCE JOINT INCOME ALREADY INCLUDES IT.
verification_status	YES	Indicates if income was verified by LC, not verified, or if the income source was verified
issue_d	YES	The month which the loan was funded
loan_status	NO	Current status of the loan
pymnt_plan	NO	Indicates if a payment plan has been put in place for the loan
url	NO	URL for the LC page with listing data.
desc	NO	Loan description provided by the borrower
purpose	YES	A category provided by the borrower for the loan request.
title	NO	The loan title provided by the borrower
zip_code	NO	The first 3 numbers of the zip code provided by the borrower in the loan application.
addr_state	YES	The state provided by the borrower in the loan application
dti	YES	A ratio calculated using the borrower s total monthly debt payments on the total debt obligations, excluding mortgage and the requested LC loan, divided by the borrower s self-reported monthly income. NOT USED AS A VARIABLE. ONLY USE JOINT DTI.
delinq_2yrs	YES	The number of 30+ days past-due incidences of delinquency in the borrower s credit file for the past 2 years
earliest_cr_line	YES	The month the borrower s earliest reported credit line was opened
inq_last_6mths	YES	The number of inquiries in past 6 months (excluding auto and mortgage inquiries)
mths_since_last_delinq	YES	The number of months since the borrower s last delinquency.
mths_since_last_record	YES	The number of months since the last public record.
open_acc	YES	The number of open credit lines in the borrower s credit file.
pub_rec	YES	Number of derogatory public records
revol_bal	YES	Total credit revolving balance

Table 6.1: Description of the dataset variables as provided in the dataset downloaded from Kaggle *(continued)*

Variable Name	Used in model?	Description
revol_util	YES	Revolving line utilization rate, or the amount of credit the borrower is using relative to all available revolving credit.
total_acc	YES	The total number of credit lines currently in the borrower s credit file
initial_list_status	NO	The initial listing status of the loan. Possible values are – W, F
out_prncp	NO	Remaining outstanding principal for total amount funded. NOTE ONCE A LOAN IS REPAID OR CHARGED OFF, THIS AMOUNT BECOMES 0.
out_prncp_inv	NO	Remaining outstanding principal for portion of total amount funded by investors. NOTE ONCE A LOAN IS REPAID OR CHARGED OFF, THIS AMOUNT BECOMES 0.
total_pymnt	NO	Payments received to date for total amount funded
total_pymnt_inv	NO	Payments received to date for portion of total amount funded by investors
total_rec_prncp	NO	Principal received to date. NOTE THIS AMOUNT WILL SHOW WHETHER A BORROWER DID NOT REPAY IN FULL
total_rec_int	NO	Interest received to date
total_rec_late_fee	NO	Late fees received to date
recoveries	NO	Post charge off gross recovery. NOTE IF A LOAN IS REPAID, THIS AMOUNT IS 0.
collection_recovery_fee	NO	Post charge off collection fee
last_pymnt_d	NO	Last month payment was received
last_pymnt_amnt	NO	Last total payment amount received
next_pymnt_d	NO	Next scheduled payment date
last_credit_pull_d	NO	The most recent month LC pulled credit for this loan
collections_12_mths_ex_med	NO	Number of collections in 12 months excluding medical collections
mths_since_last_major_derog	NO	Months since most recent 90-day or worse rating
policy_code	NO	Publicly available policy_code=1 / New products not publicly available policy_code=2
application_type	YES	Indicates whether the loan is an individual application or a joint application with two coborrowers
annual_inc_joint	YES	The combined self-reported annual income provided by the coborrowers during registration
dti_joint	YES	A ratio calculated using the coborrowers total monthly payments on the total debt obligations, excluding mortgages and the requested LC loan, divided by the coborrowers combined self-reported monthly income
verification_status_joint	YES	Indicates if income was verified by LC, not verified, or if the income source was verified

Table 6.1: Description of the dataset variables as provided in the dataset downloaded from Kaggle *(continued)*

Variable Name	Used in model?	Description
acc_now_delinq	YES	The number of accounts on which the borrower is now delinquent.
tot_coll_amt	NO	Total collection amounts ever owed
tot_cur_bal	NO	Total current balance of all accounts
open_acc_6m	NO	Number of open trades in last 6 months
open_act_il	NO	Number of currently active installment trades
open_il_12m	NO	Number of installment accounts opened in past 12 months
open_il_24m	NO	Number of installment accounts opened in past 24 months
mths_since_rcnt_il	NO	Months since most recent instalment accounts opened
total_bal_il	NO	Total current balance of all installment accounts
il_util	NO	Ratio of total current balance to high credit/credit limit on all install acct
open_rv_12m	YES	Number of revolving trades opened in past 12 months
open_rv_24m	YES	Number of revolving trades opened in past 24 months
max_bal_bc	YES	Maximum current balance owed on all revolving accounts
all_util	NO	Balance to credit limit on all trades
total_rev_hi_lim	NO	Total revolving high credit/credit limit
inq_fi	YES	Number of personal finance inquiries
total_cu_tl	NO	Number of finance trades
inq_last_12m	NO	Number of credit inquiries in past 12 months
acc_open_past_24mths	NO	Number of trades opened in past 24 months.
avg_cur_bal	YES	Average current balance of all accounts
bc_open_to_buy	YES	Total open to buy on revolving bankcards.
bc_util	YES	Ratio of total current balance to high credit/credit limit for all bankcard accounts.
chargeoff_within_12_mths	NO	Number of charge-offs within 12 months
delinq_amnt	NO	The past-due amount owed for the accounts on which the borrower is now delinquent.
mo_sin_old_il_acct	YES	Months since oldest bank instalment account opened
mo_sin_old_rev_tl_op	YES	Months since oldest revolving account opened
mo_sin_rcnt_rev_tl_op	YES	Months since most recent revolving account opened
mo_sin_rcnt_tl	YES	Months since most recent account opened
mort_acc	YES	Number of mortgage accounts.
mths_since_recent_bc	YES	Months since most recent bankcard account opened.
mths_since_recent_bc_dlq	YES	Months since most recent bankcard delinquency
mths_since_recent_inq	YES	Months since most recent inquiry.
mths_since_recent_revol_delinq	YES	Months since most recent revolving delinquency.

Table 6.1: Description of the dataset variables as provided in the dataset downloaded from Kaggle *(continued)*

Variable Name	Used in model?	Description
num_accts_ever_120_pd	YES	Number of accounts ever 120 or more days past due
num_actv_bc_tl	YES	Number of currently active bankcard accounts
num_actv_rev_tl	YES	Number of currently active revolving trades
num_bc_sats	YES	Number of satisfactory bankcard accounts
num_bc_tl	YES	Number of bankcard accounts
num_il_tl	YES	Number of installment accounts
num_op_rev_tl	YES	Number of open revolving accounts
num_rev_accts	YES	Number of revolving accounts
num_rev_tl_bal_gt_0	YES	Number of revolving trades with balance >0
num_sats	YES	Number of satisfactory accounts
num_tl_120dpd_2m	YES	Number of accounts currently 120 days past due (updated in past 2 months)
num_tl_30dpd	YES	Number of accounts currently 30 days past due (updated in past 2 months)
num_tl_90g_dpd_24m	YES	Number of accounts 90 or more days past due in last 24 months
num_tl_op_past_12m	YES	Number of accounts opened in past 12 months
pct_tl_nvr_dlq	YES	Percent of trades never delinquent
percent_bc_gt_75	YES	Percentage of all bankcard accounts > 75% of limit.
pub_rec_bankruptcies	YES	Number of public record bankruptcies
tax_liens	YES	Number of tax liens
tot_hi_cred_lim	YES	Total high credit/credit limit
total_bal_ex_mort	YES	Total credit balance excluding mortgage
total_bc_limit	YES	Total bankcard high credit/credit limit
total_il_high_credit_limit	YES	Total installment high credit/credit limit
revol_bal_joint	YES	Total credit revolving balance
sec_app_earliest_cr_line	NO	Earliest credit line at time of application for the secondary applicant. VARIABLE NOT USED. WE RELY ON THE MAIN BORROWER IN THE FIRST INSTANCE.
sec_app_inq_last_6mths	NO	Credit inquiries in the last 6 months at time of application for the secondary applicant. VARIABLE NOT USED. WE RELY ON THE MAIN BORROWER IN THE FIRST INSTANCE.
sec_app_mort_acc	NO	Number of mortgage accounts at time of application for the secondary applicant. VARIABLE NOT USED. WE RELY ON THE MAIN BORROWER IN THE FIRST INSTANCE.
sec_app_open_acc	NO	Number of open trades at time of application for the secondary applicant. VARIABLE NOT USED. WE RELY ON THE MAIN BORROWER IN THE FIRST INSTANCE.

Table 6.1: Description of the dataset variables as provided in the dataset downloaded from Kaggle *(continued)*

Variable Name	Used in model?	Description
sec_app_revol_util	NO	Ratio of total current balance to high credit/credit limit for all revolving accounts. VARIABLE NOT USED. WE RELY ON THE MAIN BORROWER IN THE FIRST INSTANCE.
sec_app_open_act_il	NO	Number of currently active installment trades at time of application for the secondary applicant. VARIABLE NOT USED. WE RELY ON THE MAIN BORROWER IN THE FIRST INSTANCE.
sec_app_num_rev_accts	NO	Number of revolving accounts at time of application for the secondary applicant. VARIABLE NOT USED. WE RELY ON THE MAIN BORROWER IN THE FIRST INSTANCE.
sec_app_chargeoff_within_12_mths	NO	Number of charge-offs within last 12 months at time of application for the secondary applicant. VARIABLE NOT USED. WE RELY ON THE MAIN BORROWER IN THE FIRST INSTANCE.
sec_app_collections_12_mths_ex_med	NO	Number of collections within last 12 months excluding medical collections at time of application for the secondary applicant. VARIABLE NOT USED. WE RELY ON THE MAIN BORROWER IN THE FIRST INSTANCE.
sec_app_mths_since_last_major_derog	NO	Months since most recent 90-day or worse rating at time of application for the secondary applicant. VARIABLE NOT USED. WE RELY ON THE MAIN BORROWER IN THE FIRST INSTANCE.
hardship_flag	NO	Flags whether or not the borrower is on a hardship plan
hardship_type	NO	Describes the hardship plan offering
hardship_reason	NO	Describes the reason the hardship plan was offered
hardship_status	NO	Describes if the hardship plan is active, pending, cancelled, completed, or broken
deferral_term	NO	Amount of months that the borrower is expected to pay less than the contractual monthly payment amount due to a hardship plan
hardship_amount	NO	The interest payment that the borrower has committed to make each month while they are on a hardship plan
hardship_start_date	NO	The start date of the hardship plan period
hardship_end_date	NO	The end date of the hardship plan period

Table 6.1: Description of the dataset variables as provided in the dataset downloaded from Kaggle *(continued)*

Variable Name	Used in model?	Description
payment_plan_start_date	NO	The day the first hardship plan payment is due. For example, if a borrower has a hardship plan period of 3 months, the start date is the start of the three-month period in which the borrower is allowed to make interest-only payments.
hardship_length	NO	The number of months the borrower will make smaller payments than normally obligated due to a hardship plan
hardship_dpd	NO	Account days past due as of the hardship plan start date
hardship_loan_status	NO	Loan Status as of the hardship plan start date
orig_projected_additional_accrued_interest	NO	The original projected additional interest amount that will accrue for the given hardship payment plan as of the Hardship Start Date. This field will be null if the borrower has broken their hardship payment plan.
hardship_payoff_balance_amount	NO	The payoff balance amount as of the hardship plan start date
hardship_last_payment_amount	NO	The last payment amount as of the hardship plan start date
disbursement_method	YES	The method by which the borrower receives their loan. Possible values are: CASH, DIRECT_PAY
debt_settlement_flag	NO	Flags whether or not the borrower, who has charged-off, is working with a debt-settlement company.
debt_settlement_flag_date	NO	The most recent date that the Debt_Settlement_Flag has been set
settlement_status	NO	The status of the borrower's settlement plan. Possible values are: COMPLETE, ACTIVE, BROKEN, CANCELLED, DENIED, DRAFT
settlement_date	NO	The date that the borrower agrees to the settlement plan
settlement_amount	NO	The loan amount that the borrower has agreed to settle for
settlement_percentage	NO	The settlement amount as a percentage of the payoff balance amount on the loan
settlement_term	NO	The number of months that the borrower will be on the settlement plan

6.8 Calculations of the internal rate of returns and month-to-default

The following shows two versions of the same code. The R version is provided because of the description of the assignment. However, the R version takes just under a full day to run. A Julia version, which is a direct translation of the R code, runs in about 150s (ca. 500x faster).

6.8.1 R code

```r
#############################################################################################################
#
# Given some numerical parameters describing a loan in the dataset, returns its Internal Rate
# of Return.
#
# In the first instance, the function creates a schedule of payments.
# In many cases, the schedule will be extremely simple: a series of 36 or 60 equal instalements.
#
# But in some cases, a loan repayment are accelerated. Therefore the total amount of interest will
# be lower than expected (but this is good for the investor because highe interest rate over
# shorter tenor.).
#
# In other cases, the borrower defaults. Overall payments are less than expected.
#
# Based on the limited information of the dataset, the function makes educated guesses on the exact
#
# WARNING: THIS IS NOT OPTIMISED. RUNNING THIS FOR ALL LOANS (1.3 MLN OF THEM) TAKES CA.20 HOURS !!!
#
calculateIRR <- function(loanNumber = 1,
                         loan = 1000,
                         intRate = 0.02,
                         term = 36,
                         totalPaid = 1000,
                         totalPrincipalPaid,
                         totalInterestPaid,
                         recoveries = 0,
                         lateFees = 0,
                         showSchedule = FALSE) {
  require(tidyverse)

  # number of monthly payments.
  # It exceeds 60 months in case recoveries on a 60-month loan takes the schedule after 60 months.
  nMonths <- 90

  # Months after which a loan defaults (normal tenor if no default or early prepayment)
  monthDefault = term

  # Note: *100 /100 to calculate in cent because ceiling cannot specify significant digits.
  installment <-
    ceiling(100 * loan * intRate / 12 / (1 - 1 / (1 + intRate / 12) ^ term)) / 100

  # We create a schedule
  schedule <- tibble(
    month = 0:nMonths,
    monthlyPayment = 0.0,
    totalPandI = 0.0,
    totalI = 0.0,
    totalP = 0.0
  )

  for (i in 2:(nMonths + 2)) {
    # Get situation at the end of previous month
    previousTotalPandI <- as.numeric(schedule[i - 1, "totalPandI"])
    previousTotalP     <- as.numeric(schedule[i - 1, "totalP"])
    previousTotalI     <- as.numeric(schedule[i - 1, "totalI"])

```

```
    # This is the beginning of a new month. First and foremost, the borrower is expected to pay the
    # accrued interest on amount of principal outstanding.
    # ceiling doesn't seem accept to accept significative digits.
    accruedInterest <-
      ceiling(100 * (loan - previousTotalP) * intRate / 12) / 100

    # If that amount takes the schedule above the total amount of interest shown in the data set,
    # we should stop the schedule at this point
    if (previousTotalI + accruedInterest > totalInterestPaid) {
      # We stop the normal schedule at this date.
      # Interest is paid (although less than scheduled)
      schedule[i, "monthlyPayment"] <-
        totalInterestPaid - previousTotalI

      # As well as whatever principal is left as per the dataset
      schedule[i, "monthlyPayment"] <-
        schedule[i, "monthlyPayment"] + totalPrincipalPaid - previousTotalP

      # Then 3-month after the last payment date, recoveries and and late fees are paid
      schedule[i + 3, "monthlyPayment"] <-
        schedule[i + 3, "monthlyPayment"] + recoveries + lateFees

      # Not really useful, but for completeness
      schedule[i, "totalPandI"] <- totalPaid
      schedule[i, "totalI"]     <- totalInterestPaid
      schedule[i, "totalP"]     <- totalPrincipalPaid

      # If total principal paid is less than borrower, then it is a default, and the monthDefault
      # is adjusted.
      if (totalPrincipalPaid < loan) {
        monthDefault = i
      }

      # No more payments to add to the schedule
      break()

    } else {
      # Deal with normal schedule
      schedule[i, "monthlyPayment"] <- installment
      schedule[i, "totalPandI"] <-
        schedule[i - 1, "totalPandI"] + installment
      schedule[i, "totalI"]     <-
        schedule[i - 1, "totalI"]   + accruedInterest
      schedule[i, "totalP"]     <-
        schedule[i - 1, "totalP"]   + installment - accruedInterest
    }
  }

  # At this point schedule[, "monthlyPayment"] contains the schedule of all payments, but needs to
  # include the initial loan.
  schedule[1, "monthlyPayment"] <- -loan

  if (showSchedule) {
    schedule %>% view()
  }

  NPV <- function(interest, cashFlow) {
    t = 0:(length(cashFlow) - 1)
```

```
    sum(cashFlow / (1 + interest) ^ t)
  }

  IRR <- function(CF) {
    res <- NA
    try({
      res <- uniroot(NPV, c(-0.9, 1), cashFlow = CF)$root
    },
    silent = TRUE)
    return(res)
  }

  return(tibble(
    loanID = loanNumber,
    IRR = round(as.numeric(IRR(
      schedule$monthlyPayment
    ) * 12), digits = 4),
    monthDefault = monthDefault
  ))
}

loanNumberIRR <- function(loanNumber) {
  require(tidyverse)

  l <- loans %>% filter(loanID == loanNumber)
  calculateIRR(
    loanNumber = l$loanID,
    loan = l$funded_amnt, intRate = l$int_rate, term = l$term,
    totalPaid = l$total_pymnt, totalPrincipalPaid = l$total_rec_prncp, totalInterestPaid = l$total_r
    recoveries = l$recoveries, lateFees = l$total_rec_late_fee,
    showSchedule = TRUE
  )
}
```

```
#
# Calculate all the IRRs and month of default for all the loans.
# WARNING: This takes around a full day to run!!!!
#
# The actual data was generated by the Julia version, with cross-checks.
# Julia version takes about 150 sec on the same unoptimised code.
#
local({
  loansIRR <-
    loans %>%
    rowwise() %>%
    do(
      calculateIRR(
        loanNumber = .$loanID,
        loan = .$funded_amnt,
        intRate = .$int_rate,
        term = .$term,
        totalPaid = .$total_pymnt,
        totalPrincipalPaid = .$total_rec_prncp,
        totalInterestPaid = .$total_rec_int,
        recoveries = .$recoveries,
        lateFees = .$total_rec_late_fee
      )
    )
```

```
  saveRDS(loansIRR, "datasets/lending_club_IRRs.rds")

})
```

6.8.2 Julia code

6.8.2.1 Internal Rate of Return

6.8.3 Maxima derivation of the cost function

```
PDF1(x, Q) := alpha1( Q) * sqrt( 1 / ( 2 * pi)) *
              exp( - 1 / 2*(( log( -( x - m1( Q)) / m1( Q)) + sigma1( Q) ^ 2) /
              sigma1( Q)) ^ 2) / ( -( x - m1(Q)) * sigma1( Q)) ;

PDF2(x, Q) := alpha2( Q) * sqrt( 1 / ( 2 * pi)) *
              exp( - 1 / 2*(( log( -( x - m2( Q)) / m2( Q)) + sigma2( Q) ^ 2) /
              sigma2( Q)) ^ 2) / ( -( x - m2( Q)) * sigma2( Q)) ;

PDF3(x, Q) := alpha3( Q) * sqrt( 1 / ( 2 (* pi)) *
              exp( - 1 / 2*(( log( -( x - m3( Q)) / m3( Q)) + sigma3( Q) ^ 2) /
              sigma3( Q)) ^ 2) / ( -( x - m3( Q)) * sigma3( Q)) ;

PDF4(x, Q) := alpha4( Q) * sqrt( 1 / ( 2 * pi)) *
              exp( - 1 / 2*(( log( ( x - m4( Q)) / m4( Q)) + sigma4( Q) ^ 2) /
              sigma4( Q)) ^ 2) / ( ( x - m4( Q)) * sigma4( Q)) ;


alpha1(Q) := am1* Q + an1 ;
alpha2(Q) := am2* Q + an2 ;
alpha3(Q) := am3* Q + an3 ;
alpha4(Q) := am4* Q + an4 ;

m1(Q) := mm1* Q + mn1 ;
m2(Q) := mm2* Q + mn2 ;
m3(Q) := mm3* Q + mn3 ;
m4(Q) := mm4* Q + mn4 ;

sigma1(Q) := sm1* Q + sn1 ;
sigma2(Q) := sm2* Q + sn2 ;
sigma3(Q) := sm3* Q + sn3 ;
sigma4(Q) := sm4* Q + sn4 ;

J(x, Q): = -( x - ( PDF1( x, Q) + PDF2( x, Q) + PDF3( x, Q) + PDF4( x, Q))) ^ 2 ;

diff( PDF1(x, Q), Q) ;
```

6.9 System version

```
Sys.info()
```

```
##                                    sysname                                    release
##                                    "Linux"                        "5.3.0-24-generic"
```

```
##                                       version                                      nodename
## "#26-Ubuntu SMP Thu Nov 14 01:33:18 UTC 2019"                                        "x260"
##                                       machine                                        login
##                                     "x86_64"                                    "unknown"
##                                          user                                effective_user
##                                    "emmanuel"                                    "emmanuel"
```

```
# automatically create a bib database for R packages
knitr::write_bib(c(.packages(), "bookdown", "knitr", "rmarkdown"), "packages.bib")
```

Bibliography

Bokhari, M. M. (2019). Credit risk analysis in peer to peer lending data set: Lending club.

California, L. C. S. F. (2019). Prospectus Regulatory Filing S3-ASR for Member Payment Dependent Notes. https://www.sec.gov/Archives/edgar/data/1409970/000140997019000988/0001409970-19-000988-index.htm. [Note: Accessed 31 October 2019].

Kan, L. C. P. W. (2019). Kaggle - LendingClub dataset. https://www.kaggle.com/wendykan/lending-club-loan-data. [Note: Accessed 31 August 2019].

Kim, A. and Cho, S.-B. (2019). An ensemble semi-supervised learning method for predicting defaults in social lending. *Engineering Applications of Artificial Intelligence*, 81:193–199.

Peng, R. (2012). *Exploratory data analysis with R*. Lulu. com.

Shalev-Shwartz, S. and Ben-David, S. (2014). *Understanding machine learning: From theory to algorithms*. Cambridge university press.