

Prediction of “Give Me Some Credit” Dataset

目次

| | |
|-------------------------------------|----|
| 一、緒論..... | 2 |
| 1.1 研究背景及動機..... | 2 |
| 1.2 研究目的 | 2 |
| 1.3 資料簡介 | 2 |
| 二、文獻回顧..... | 2 |
| 三、研究方法..... | 3 |
| 3.1 缺失值填補 | 3 |
| 3.2 探索性資料分析..... | 4 |
| 3.3 特徵工程 | 4 |
| 3.4 建模..... | 5 |
| 3.5 交叉驗證 | 5 |
| 3.6 不平衡分類 | 5 |
| 四、研究結果..... | 6 |
| 4.1 不平衡分類 | 6 |
| 4.2 缺失值填補 | 6 |
| 4.3 探索性資料分析..... | 6 |
| 4.4 特徵工程 | 10 |
| 4.5 建模與變數選取..... | 11 |
| (1) Logistic Regression..... | 11 |
| (2) Treelike methods – tuning | 11 |
| (3) Treelike methods – result | 12 |
| (4) Stacking..... | 12 |
| (5) 提交至 Kaggle 的結果統整 | 12 |
| 4.6 R code 的解釋 | 15 |
| 五、結論與建議 | 16 |
| 六、參考文獻..... | 16 |

Prediction of “Give Me Some Credit” Dataset

摘要

本資料使用 2011 年 Kaggle 上的已結束的比賽，是份預測信用違約的資料¹。因為是一份不平衡分類資料，因此 Kaggle 上要求提交的結果是可用來計算 AUC 的 entry。在做了詳盡的 feature engineering 後，我們使用 LR、RF、XGB 三種模型，接著對三種模型的結果做雙層 stacking，最後在 Kaggle 上 public score 的最高成績 137 名($137/924 = 14.83\%$)，該結果得到了等於比賽結束時排名的第 63 名($63/924 = 6.82\%$)(以 private score 衡量最終排名)，相當於拿到了比賽的銅牌 (51~100 名為銅牌)。

一、緒論

1.1 研究背景及動機

在 Kaggle 上尋覓適合我們程度的 competition，扣掉時間序列(time series)的資料、扣掉檔案過大的資料，終於找到這份大小 14MB 的非時間序列資料。

1.2 研究目的

得到更好的預測準確率，評斷標準為提交到 Kaggle 上的成績。

1.3 資料簡介

2011 年在 Kaggle 上有獎金的比賽，為預測信用違約的比賽。參賽人數 924 人，最後根據 private leaderboard score，第 1~3 名頒發獎金，第 4~11 名頒發金牌，第 12~50 名頒發銀牌，第 51 名~第 100 名頒發銅牌。training dataset 為 150,000 x 11；test dataset 為 101,503 x 11。反應變數(response) 為 SeriousDlqin2yrs，其為一個二元類別變數，(0, 1) = (93.32%, 6.68%)。在附檔 Data/Data Dictionary.csv 中有全部 11 個變數的詳細介紹。附檔 Data/sampleEntry 為提交到 Kaggle 的範例。

二、文獻回顧

我們只有參考在 Give Me Some Credit 討論區² 的評論，找到了第一名、第二名、第五名留下的心得。

1. Alec Stephenson (1st in this Competition)：

¹ 本資料來源取自 <https://www.kaggle.com/c/GiveMeSomeCredit/overview>

² <https://www.kaggle.com/c/GiveMeSomeCredit/discussion>

這個比賽的重點在於 ensemble model。我們從原本資料的 9 個特徵中，提取出了 25 ~ 35 個特徵，然後應用了 5 種不同的模型，包含 a regression random forest、a classification random forest、a feed-forward neural network with a single hidden layer、a gradient regression tree boosting algorithm、a gradient classification tree boosting algorithm。NN 方法在調參、變數選取上讓我們吃進苦頭，但帶來了顯著的進步，不是 bagging 或 boosting 可以企及的進步。

2. Xavier Conort (2nd in this Competition) :

我在這比賽中得到的心得，是「不能」全然地信任 public leaderboard scores 上的結果去選擇模型。在最後 16 天，我在 public score 上完全沒進步，但在我手上的訓練資料做 cross validation 時有持續進步，最後在 private score 結果出來時，證實了我這 16 天的努力沒有白費。我使用了 15 種模型，包括 GBMs、weighted GBMs、Random Forest、balanced Random Forest、GAM、weighted GAM (all with bernoulli/binomial error)、SVM、bagged ensemble of SVMs。我其實還沒對每個模型做 fine tune，只是盡量尋求模型的多樣性。

3. Shea Parkes (5th in this Competition) :

很多人在討論最佳單一模型，但我不認為這些單一模型是「單一的」，因為實際上 random forest, gbms 都是 ensemble models。我們最佳的 random forest 使用了約 8000 棵樹，我們並沒有「平衡」random forest，因此我們只好多跑幾次去彌補這點。我們在 RF 的最佳值為 0.8578，NN 最佳值為 0.8677，gbm 最佳值為 0.8674，elastic net'd glm 最佳值為 0.8644；從這裡可以看出，我們真的應該花點時間平衡我們的 random forest。我們並沒有很依據 public score，去調參、去選擇 ensemble 方法，這是我們的一大敗筆。

三、研究方法

3.1 缺失值填補

當缺失值(missing value)比例過少可忽略，且為 missing randomly 時，使用 na.omit() 是妥當的做法。但當缺失值比例過高不可忽略，或是 missing systematically 時，填補缺失值就是較佳的作法³。

最簡單的方式，是以 mean 或 median 來補遺漏值，但會與真實值差距過大。因此，多重差補(Multiple Imputation Missing Data)誕生了。在 R 中，多重差補函數為 mice()，為 Multivariate Imputation by Chained Equations 之縮寫⁴。

有兩位豐富實務經驗的中國人，在書中提及⁵，他們傾向使用較簡單的手法填補缺失值，因為複雜的缺失值填補方法很能導致過度擬合(overfitting)，並羅列他們常使用的兩種方法：一為平均數、中

³ Zumel, N., Mount, J. (2014) *Practical Data Science with R*.

⁴ 果醬珍珍(2018)。Missing Value Treatment | 遺失值處理 | 統計 R 語言。取自 <https://bit.ly/2Ngdp5X>

⁵ 孫亮、黃倩(2016)。實用機器學習。

位數；二為使用 kNN。

因為以平均數、中位數填補過於簡單，會減少分布變異、扭曲資料的分布、削弱相關性，因此我們不考慮此方法。kNN 則是為懶惰學習(lazy learning)，在本資料及龐大的資料量下，執行時間過長，因此我們也放棄使用。至於迴歸差補，也有受到 outliers 的影響較大、替代數值可能超出合理範圍、變數間必須有充分相關才能產生有效數值等缺點⁶。因此，我們最理想的填補缺失值模型為 random forest，無奈本資料及過大執行時間過久，最後我們退而求其次使用 decision tree 的 cart 來填補缺失值。

最後，在補缺失值時一個常見的錯誤，就是沒有把 training data 和 test data 分開來各自補缺失值。當將 training / test data 放一起補缺失值時，會產生 data leakage，導致 overfitting。⁷

3.2 探索性資料分析

探索性資料分析(explanatory data exploration, EDA) 藉由圖、表，有助於發掘如何執行特徵工程、建模。從各變數的直方圖(histogram)、箱型圖(boxplot) 來觀察數值變數的分布；用表格(table) 來觀察類別變數的分布。生成相關係數的圖，觀察各變數間的關係。.....

另外，我們沒有使用老師上課教的 K-means、PCA 方法的原因很簡單，因為本 training dataset 為 150,000 x 11，使用上述 3 種方法的運算量過於龐大。

3.3 特徵工程⁸⁹

從 EDA 到特徵工程(feature engineering)，接著從特徵工程到建模，並根據建模的結果回來進行特徵工程。EDA、特徵工程、建模，這是資料科學、機器學習中，持續循環的步驟。

特徵工程，包含以現有微調現有特徵、特徵生成新的特徵、標準化(standardization)、常態化(normalization)、將特徵在數值和類別間轉換：

1. 微調現有特徵：在進行 LR 時，可將數值變數的 outliers 更改到較接近中心位置的數值。
2. 相加或相乘除生成全新特徵：觀察變數間理論上的關性，生成比方「平均收入 = 總收入 / 人數」等各種新特徵。
3. 標準化：大多數模型都受到尺度(scale)的影響，包括 Logistic / Linear regression、kNN、K-means、PCA、LDA、SVM、NN。只有基於樹的模型(treelike model) 不受尺度影響。常見方法有 z-score standardization、max-min standardization。z-score standardization 可使用 scale() 函數來達成。
4. 常態化：Logistic / Linear regression 的假設包含了回應變數(response) 和預測變數(predictor) 的常態性假設。可以使用 log() (僅對正數預測變數使用) 或進階的 box-cox 方法轉換。
5. 將特徵在數值和類別間轉換：在 treelike method 中不必將數值轉換為類別，因為 decision tree 的原因對數值變數的處理即為切分，且根據經驗 decision tree 做的切分，總是比人工切來得好。而在

⁶ 劉正山、莊文忠(2012)。項目無反應資料的多重差補分析。

⁷ 有賀康顯、中山心太、西林孝(2018)。機器學習—工作現場的評估導入與實作。

⁸ Ozdemir, S., Susarla, D. (2018). *Feature Engineering Made Easy*.

⁹ Zheng, A., Casari A. (2018). *Feature Engineering for Machine learning*.

NN 和 XGBT 時，必須將類別變數根據 one-hot encoding 轉換。要特別注意，數值資料轉換為類別資料時的資訊損失。

3.4 建模

使用 3 個模型：

1. LR (linear regression)
2. RF (random forest)
3. XGBT (X gradient boosting decision tree)

迴歸作為傳統統計分析方法，也是資料科學、機器學習，在探索資料、了解變數性質時的第一步。因為 kNN、LDA、SVM 一般在實務上表現較差，因此我們直接使用比賽常勝軍---- treelike 方法們。

Random forest 可視為 decision tree 的改良版，使用 bagging 手法，有效地減少了 noise 影響，以及減少 overfitting。

GBT(gradient boosting decision tree) 基於 RF，對錯誤分類、預測誤差較大之項目，加大權重。Xgboost 為 GBT 的改良版，加入了 [i] 更好的優化手法(加入二階 Taylor expansion 項) [ii] 加入 regularization [iii] 加入 bootstrap [vi] 併行處理降低運算時間¹⁰。Xgboost 為 GBT 系列套件中一般公認最優者。

最後，在得到上述 3 個模型結果後，再使用集成學習(ensemble learning) 中的 stacking，加起來平均；混合以上 3 種模型結果，得到更佳的預測結果。有具實務經驗研究者指出，簡單的 stacking 方法，比方簡單平均、regression，會比加權平均、RF 等複雜方法來的好¹¹。最後，參考到一本書¹²提到雙層 stacking，所以我們不只使用單層的 stacking，採用雙層 stacking。

3.5 交叉驗證

交叉驗證(cross validation)和正規化(regularization)，是兩個最常使用來避免過度擬合的方法¹³。我們使用 2-way k-fold validation。

3.6 不平衡分類¹⁴

在分類問題中，面對不平衡分類(Imbalanced classification)的狀況，常見的解法有：

1. 使用不同的評價標準。比方從準確率(accuracy) 改為使用 AUC(area under the curve)。
2. 取樣方法。使用下取樣(down-sampling) 或上取樣(up-sampling)。

在本數據集中的反應變數(response) 為二元類別變數，其分佈比例為 (0,1) = (93.22% ,6.78%)。這可能是阻礙得到更好的預測結果的原因。本資料集的 Kaggle 競賽評分標準即為 AUC。而上取樣、

¹⁰ yyHaker(2019)。GBT、GBDT、GBRT 与 Xgboost。取自 <https://zhuanlan.zhihu.com/p/57814935>

¹¹ 孫亮、黃倩(2016)。實用機器學習。

¹² Battiti, R., Brunato, M.(2018)。機器學習與優化。

¹³ 有賀康顯、中山心太、西林孝(2018)。機器學習—工作現場的評估導入與實作。

¹⁴ 孫亮、黃倩(2016)。實用機器學習。

下取樣，經過我們初步嘗試使用在 LR 上，但沒有得到更好的結果。

四、研究結果

4.1 不平衡分類

下圖是二元反應變數(response)視覺化，這是一個不平衡分類問題，因此 Kaggle 要求提交的是可計算 AUC 的 entry，而非計算 accuracy 的 entry。同樣，我們在 training data 使用的衡量標準也是 AUC。

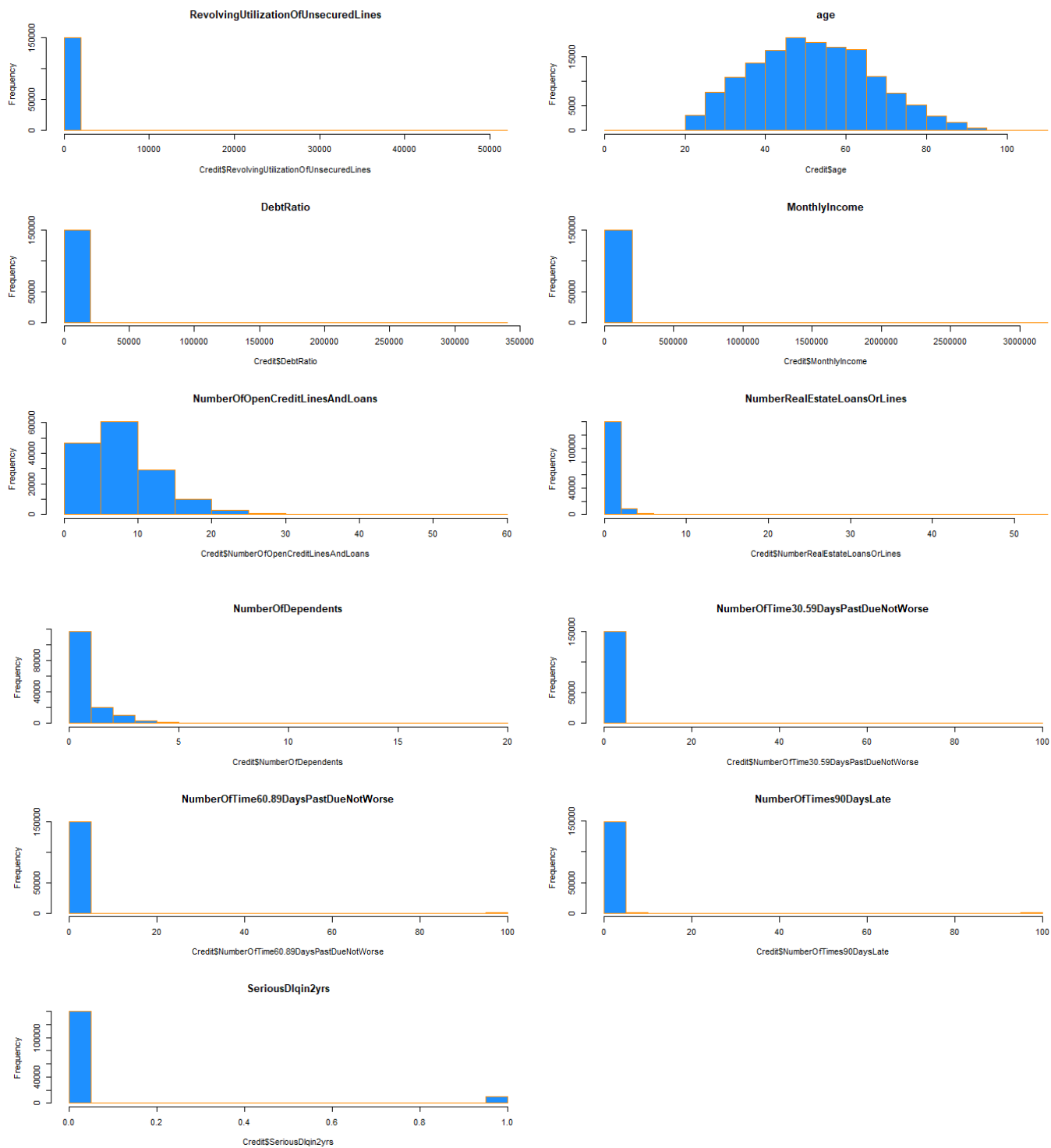


4.2 缺失值填補

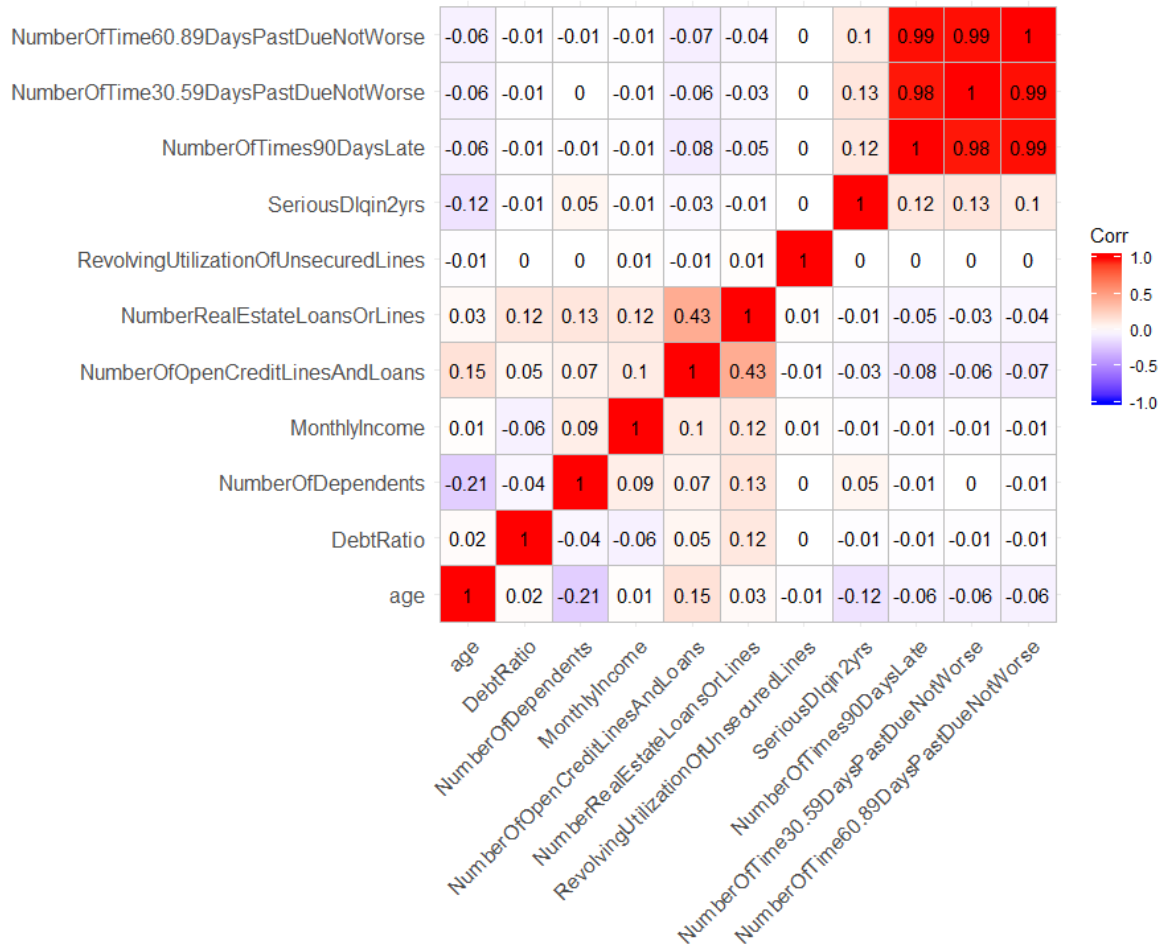
將 training data / test data 拆開來(避免 data leakage)，使用 mice() 函數，設定填補方法為 decision tree 的 CART，因為其兼具準確(不受 outliers 影響)與速度(RF 填補過於耗時)。

4.3 探索性資料分析

1. Histogram & correlation Matrix BEFORE feature engineering



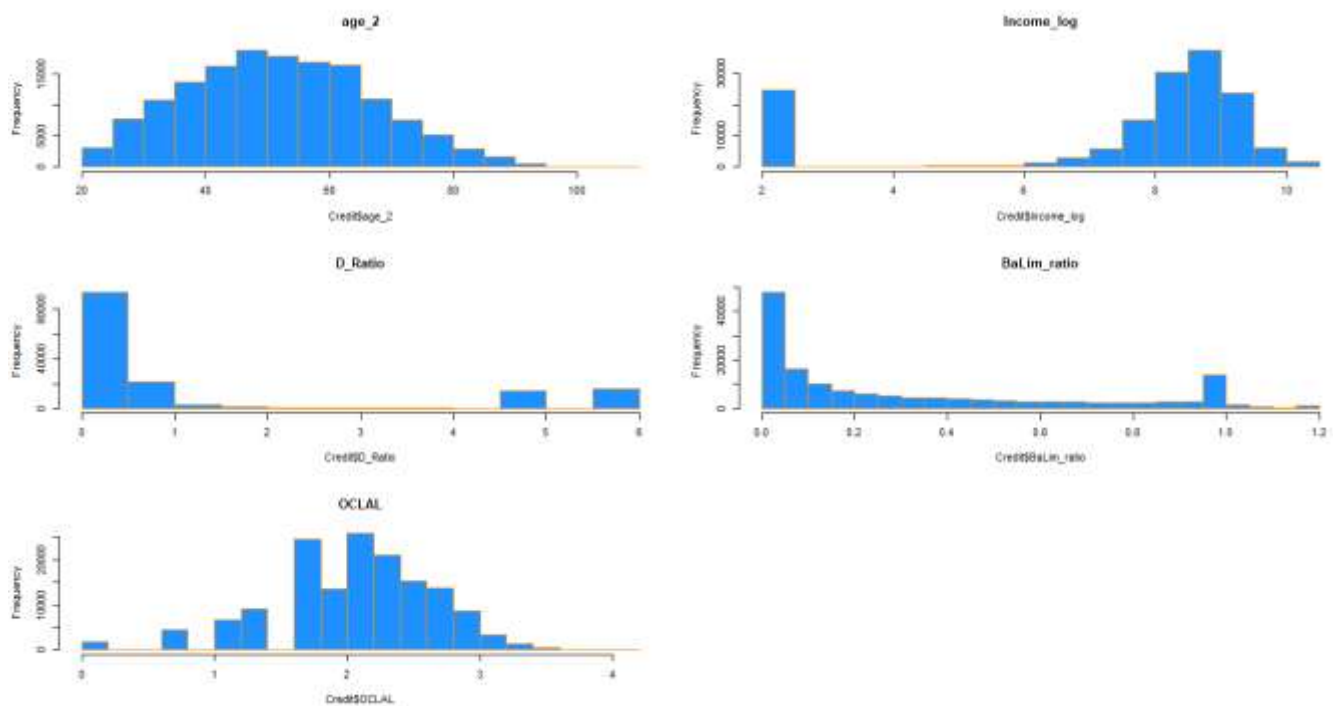
發現大部分資料都是右偏(positively skewed) 的，因此我們試著取 $\log()$ ，發現對預測效果變更好了。



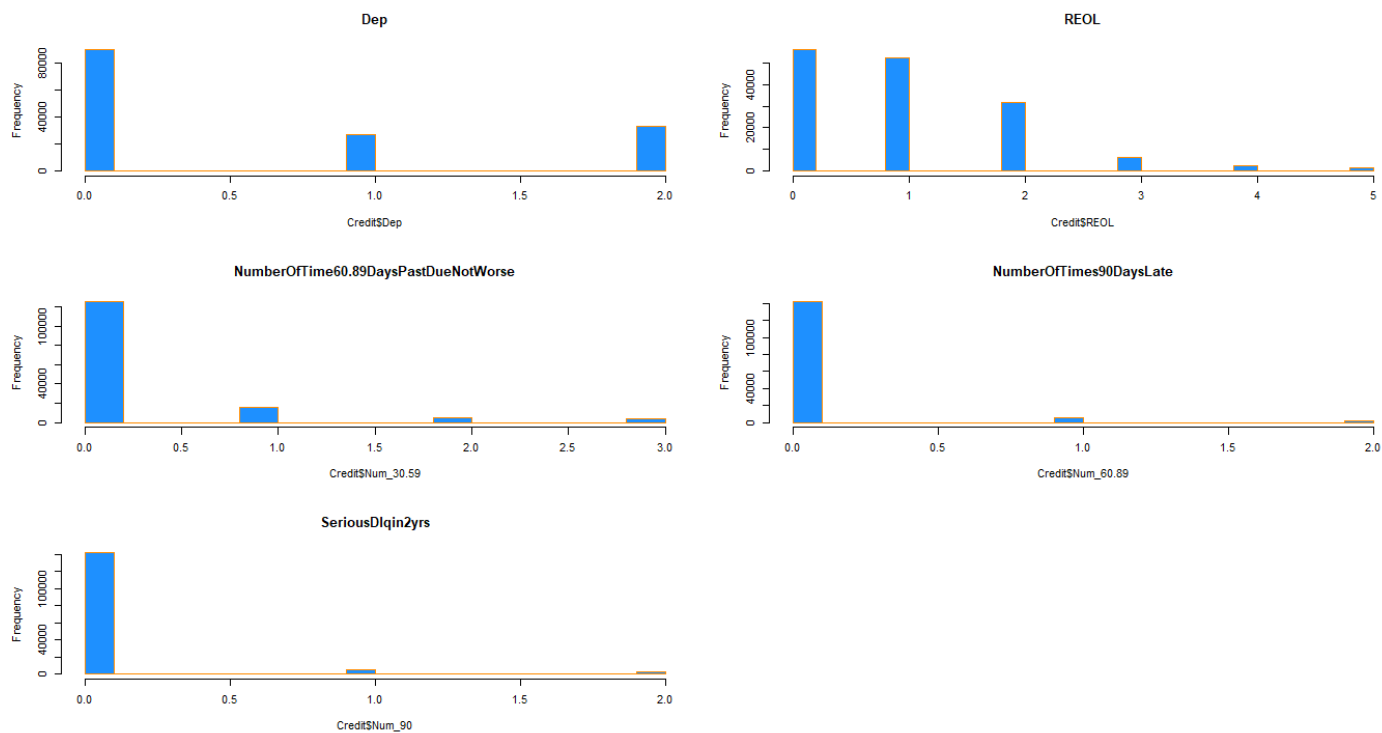
發現最後三項變數的相關係數特別的高！

2. Histogram & correlation Matrix AFTER feature engineering

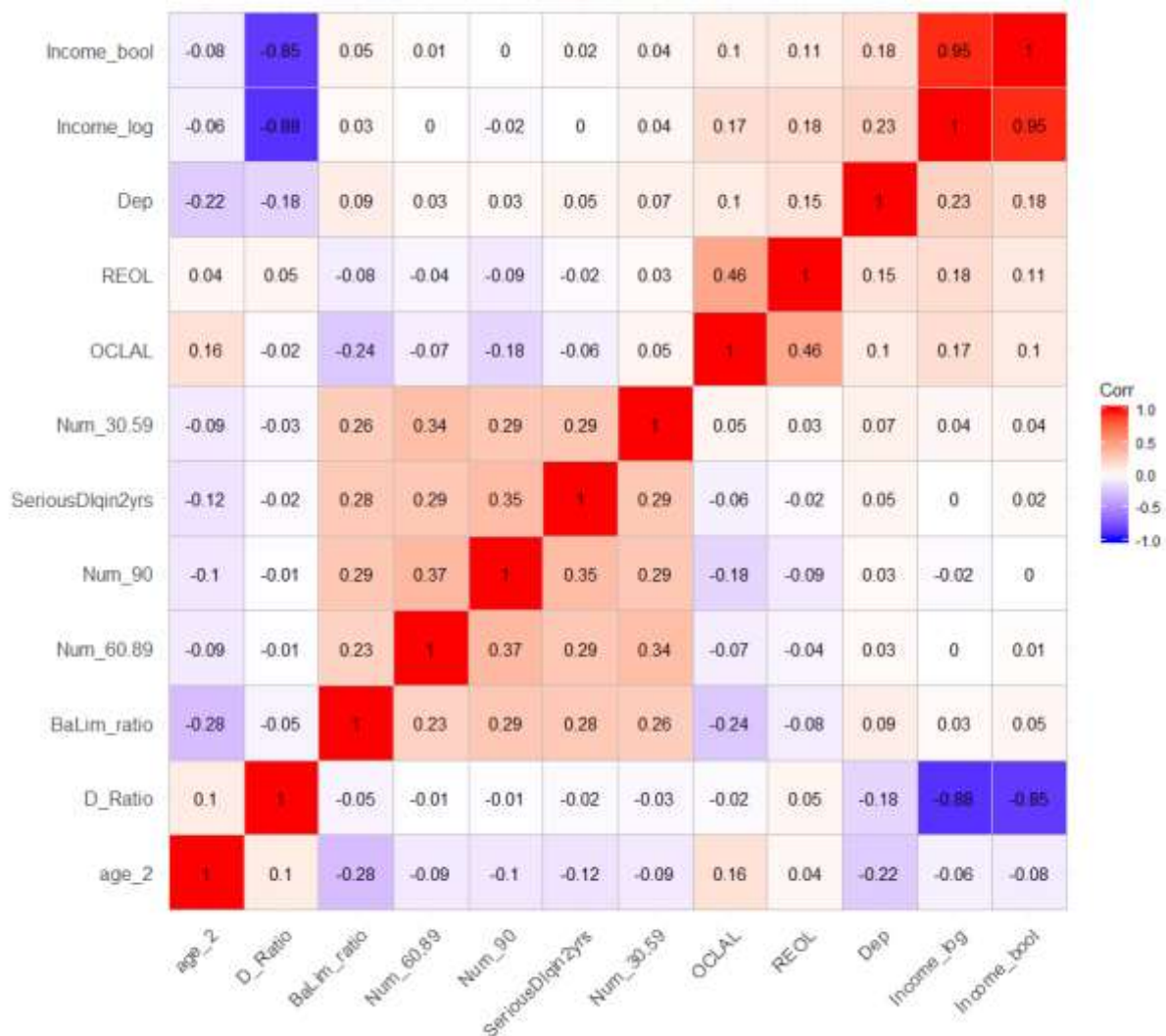
下一小節我們才會詳細解釋 feature engineering 的手法，在此先呈現經過 feature engineering 後的變化。



Histogram & Correlation Matrix after feature engineering



變數的分布更接近常態了！



相關係數高的格子變少了。Income_log 和 Income_bool 的高相關性並不是問題。因為 Income_bool 是將 MonthlyIncome 分成 0 和非 0 兩項，分別改 0 與 1，這會使其與 Income_log 高度相關，但 Income_bool 的加入確實讓 LR 的預測能力提升。

4.4 特徵工程

本文使用了微調現有特徵、相加或相乘除生成全新特徵、標準化、常態化、將特徵在數值和類別間轉換，共 5 種轉換方式。而我們對 10 個 predictor 的處理如下，以使變數常態化，助於 LR 的預測：

1. RevolvingUtilizationOfUnsecuredLines：將 > 1.2 的極端值改成 1.2，產生新變數 BaLim_ratio。
2. age：把一個 age = 0 的改成次小的 20，產生新變數 age_2。
3. MonthlyIncome：把大於 > 23300 的改成 23300，然後全部 +10 後，再取 $\log()$ ，產生新變數 Income_log。接著，篩出 MonthlyIncome = 0 與 MonthlyIncome $\neq 0$ ，產生一個二元類別變數 Income_bool；這裡是猜想 沒收入卻能刷信用卡 的人，可能會是 學生、全職媽媽，這些人會有不一樣的特點。
4. NumberOfDependents：將 NumberOfDependents > 2 的極端值改成 2，產生新變數 Dep。
5. DebtRatio：將 DebtRatio ≥ 1200 的改成 6，將 $1200 > \text{DebtRatio} >$ 的改成 5，產生新變數 D_Ratio。

6. NumberOfOpenCreditLinesAndLoans : +1 後取 $\log()$, 產生新變數 OCLAL 。
7. NumberRealEstateLoansOrLines : 將 NumberRealEstateLoansOrLines >5 的改成 5 , 產生新變數 REOL 。
8. NumberOfTime30.59DaysPastDueNotWorse : 將 59DaysPastDueNotWorse >3 的改成 3 , 產生新變數 Num_30.59 。
9. NumberOfTime60.89DaysPastDueNotWorse : 將 NumberOfTime60.89DaysPastDueNotWorse >2 的改成 2 , 產生新變數 Num_60.89 。
10. NumberOfTimes90DaysLate : 將 NumberOfTimes90DaysLate >2 的改成 2 , 產生新變數 Num_90 。

4.5 建模與變數選取

使用 3 個模型：

1. LR (linear regression)
2. RF (random forest)
3. XGBT (X gradient boosting decision tree)

(1) Logistic Regression

根據 10 個 predictor , 加上 polynomial 至 4 次項作為 full model ; 以對常數的回歸作為 null model , 用 step() 做 stepwise regression , parameter 選用 direction="both", criterion = "BIC" 。接著將得到的變數當作 null model , 以 null model 加上所有一次項的 interaction 作為 full model , parameter 選取如前述 , 做出最後的變數選取 , 並以 AIC 、 BIC 、 變數的 p-value 微調變數選取 。

```
model <- glm(formula = SeriousDlqin2yrs ~ age_2 + Income_log + I(Income_log^2) +
  I(Income_log^3) + D_Ratio + I(D_Ratio^2) + I(D_Ratio^3) + Income_bool +
  BaLim_ratio + OCLAL + I(OCLAL^2) + Dep + REOL + Num_30.59 +
  Num_60.89 + Num_90 + Num_30.59:Num_90 + Num_30.59:Num_60.89 +
  age_2:Num_90 + Num_60.89:Num_90 + age_2:Num_60.89 + D_Ratio:BaLim_ratio +
  REOL:Num_90 + age_2:Num_30.59 + age_2:Dep + age_2:D_Ratio +
  REOL:Num_60.89 + Income_log:Num_30.59 + age_2:REOL + Income_log:REOL,
  family = binomial(link = "logit"), data = Credit_train)

summary(model)
AIC(model); BIC(model)
```

(2) Treelike methods – tuning

取出全部 15 萬筆資料中的 1 萬筆做 RF 、 XGBT , 以快速地調參。特徵工程部分 , 因為 treelike methods 不受極端值影響 , 並且會自行切分數值變數 , 因此我們完全不對變數做特徵工程。在 RF , 我們得到 ntree = 1000, mtry = 3 。而 XGBT 的參數過多 , 在此不羅列 。

(3) Treelike methods – result

將上點得到的調參結果用於全部資料(共 15 萬筆)。

(4) Stacking

採用簡單平均做雙層 stacking。先分別用 LR、RF、XGBT，3 者各自用多個相同模型與簡單平均，做出第一層 stacking；接著，將多個 LR_stacking、RF_stacking、XGBT_stacking 混合產生第二層 stacking，得到最終的 AUC 結果。

(5) 提交至 Kaggle 的結果統整

Kaggle 要求提交計算可 AUC 的 entry (機率而非分類結果)，原因如研究方法提及，不平衡分類會造成使用 accuracy 跑出的結果幾乎沒有差距，使用 AUC 當標準更能看到預測確實更準確了，且 AUC 也有企業實務上之意義。以下列出重要的 11 個 submission。

(左方為 Private Score，右方為 Public Score)

1. LR

| | | |
|-----------------------------|---------|---------|
| predict.csv | 0.86387 | 0.85865 |
| 19 days ago by Yu Cheng Kuo | | |
| Logistic Regression | | |

2. RF

| | | |
|--|---------|---------|
| Kaggle_RF_15w.csv | 0.85117 | 0.84558 |
| an hour ago by Yu Cheng Kuo | | |
| add submission details | | |

3. XGBT

| | | |
|-------------------------------------|---------|---------|
| Kaggle_xgbt_005.csv | 0.86005 | 0.84826 |
| 4 hours ago by Yu Cheng Kuo | | |
| eta = 0.05 | | |

從 XGBT 的表現好過 RF 這點事實，可以判定本資料集的 noise 小到可以忽略。確認 noise 是否存在，對之後的 tuning 是非常重要的指引。

4. Stacking

| | | |
|---|---------|---------|
| stacking.csv 13 hours ago by Yu Cheng Kuo stacking = (s01_dart + s02_xgbt + s03_predict + s04_predict_06 + s05_predict_05) / 5 | 0.86709 | 0.85964 |
|---|---------|---------|

5. Stacking02

| | | |
|--|---------|---------|
| stacking02.csv 13 hours ago by Yu Cheng Kuo stacking02 = (s01_dart + s02_xgbt + s03_predict + s04_predict_06 + s05_predict_05 + s05_predict_04 + s05_predict_03) / 7 | 0.86693 | 0.86002 |
|--|---------|---------|

6. Stacking03

| | | |
|--|---------|---------|
| stacking03.csv 4 hours ago by Yu Cheng Kuo stacking03 = (dart + xgbt + predict + predict_06 + predict_05 + predict_04 + predict_03 + predict_02 + Kaggle_xgbt_0 + Kaggle_xgbt_005 + Kaggle_xgbt_001) / 11 | 0.86717 | 0.85963 |
|--|---------|---------|

7. Stacking04

| | | |
|--|---------|---------|
| stacking04.csv 4 hours ago by Yu Cheng Kuo stacking04 = (dart + xgbt + predict + predict_06 + predict_05 + predict_04 + predict_03 + predict_02 + Kaggle_xgbt_0 + Kaggle_xgbt_005 + Kaggle_xgbt_001 + stacking + stacking02) / 13 | 0.86716 | 0.85968 |
|--|---------|---------|

8. Stacking05

| | | |
|--|---------|---------|
| stacking05.csv 4 hours ago by Yu Cheng Kuo stacking05 = (stacking + stacking02 + stacking03) / 3 | 0.86712 | 0.85981 |
|--|---------|---------|

9. Stacking06

[stacking06.csv](#)

0.85531

0.84929

an hour ago by [Yu Cheng Kuo](#)

```
stacking06 = (Kaggle_RF_10w_ntree_1000 +  
Kaggle_RF_10w_ntree_1000_02 +  
Kaggle_RF_10w_ntree_1000_03) / 3 summary(stack)
```

10. Stacking07

[stacking07.csv](#)

0.85744

0.85054

an hour ago by [Yu Cheng Kuo](#)

```
stacking07 = (Kaggle_RF_5w_ntree_1000 +  
Kaggle_RF_7.5w_ntree_1000 +  
Kaggle_RF_7.5w_ntree_1000_02 +  
Kaggle_RF_10w_ntree_1000 +  
Kaggle_RF_10w_ntree_1000_02 +  
Kaggle_RF_10w_ntree_1000_03 + Kaggle_RF_15w) / 7
```

11. Stacking08

[stacking08.csv](#)

0.86786

0.86089

an hour ago by [Yu Cheng Kuo](#)

```
stacking08 = (stacking + stacking02 + stacking03 +  
stacking04 + stacking07) / 5
```

接著，從上列 11 個重要 submission 中，特別點出 3 項來看。第一項，是在嘗試過 LR、RF、XGBT 後，單一模型之最高的準確率由「LR」拔得頭籌。第二項，是單層 stacking 中，最高的表現為「stacking02」。第三項，是雙層 stacking 中，「stacking08」取得了最好的成績。

而最好的成績「stacking08」，在 Public Score、Private Score 皆進到 15%，更在 Private Score 進到前 100 名，位居第 63 名。而比賽最後 final standings 是看 Private Score，頒發的獎項最多至第 100 名的銅牌。

** 此 3 項各情況下「最優異成績」的選取，皆為「最高的 Public Score」，因為要模擬比賽時的情況。而 Private Score 才是最後比賽結束時的排名。

| Submission | Content | Public/ Private | AUC | Ranking |
|----------------|----------------------------------|-----------------|---------|--------------------|
| (1) LR | Logistic Regression | Public | 0.85865 | 396/ 924 = 42.86 % |
| | | Private | 0.86387 | 435/ 924 = 47.08 % |
| (2) stacking02 | stacking02= (s01_dart + s02_xgbt | Public | 0.86002 | 208/ 924 = 22.51 % |

| | | | | |
|----------------|--|---------|---------|--------------------|
| | + s03_predict + s04_predict_06 + s05_predict_05 + s05_predict_04 + s05_predict_03) / 7 | Private | 0.86693 | 120/ 924 = 12.99 % |
| (3) stacking08 | Stacking08=(stacking+stacking02+stacking03+stacking04 + stacking07) / 5 | Public | 0.86089 | 137/ 924 = 14.83 % |
| | | Private | 0.86786 | 63/ 924 = 6.82 % |

1. Stacking08

[stacking08.csv](#)





an hour ago by [Yu Cheng Kuo](#)

0.86786

0.86089

stacking08 = (stacking + stacking02 + stacking03 + stacking04 + stacking07) / 5

2. Public Score leaderboard (因比賽已結束，我們無法被列入排行榜)

| | | | | | |
|-----|--------------|---|---------|----|----|
| 134 | BarrenWuffet |  | 0.86090 | 38 | 9y |
| 135 | esslor1 |  | 0.86089 | 2 | 9y |
| 136 | humel |  | 0.86089 | 2 | 9y |
| 137 | elad bensal |  | 0.86088 | 33 | 9y |

2. Private Score leaderboard (final standings) (因比賽已結束，我們無法被列入排行榜)

| | | | | | | |
|----|--|--------------------|---|---------|----|----|
| 61 |  32 | Boltzmann's Hammer |  | 0.86791 | 31 | 9y |
| 62 |  42 | Blind Ape |  | 0.86789 | 6 | 9y |
| 63 |  16 | bushnet |  | 0.86782 | 4 | 9y |
| 64 |  5 | jcolander |  | 0.86781 | 6 | 9y |

4.6 R code 的解釋

1. BDA_final_01_mice: 輸入原始資料「Data/train」、「Data/test」, 缺失值填補, 得到了「cs_all_cart」、「train_ans」兩個 csv, 由於缺失值填補得多上一些時間, 因此特別拆分出來。

2. BDA_final_02_LR: 輸入「cs_all_cart」、「train_ans」。做 EDA。使用 Logistic Regression, 並輸出預測結果, 然後將結果上傳 Kaggle。

3. BDA_final_03_Treelike_tuning: 取全部 15 萬筆資料中的 1 萬筆做 RF、XGBT, 以快速地調參。

4. BDA_final_04_Treelike_stacking: 調參完後, 使用全部的 15 萬筆資料, 將結果上傳至 Kaggle。

五、結論與建議

我們在選取提交到 Kaggle 上 public score 的最高成績 137 名($137/924 = 14.83\%$)，得到了等於比賽結束時排名的第 63 名($63/924 = 6.82\%$)(以 private score 衡量最終排名)，相當於拿到了比賽的銅牌(51~100 名為銅牌)，這給了我們相當大的鼓勵；不過，我們依然發現一些可以改進的地方。

在變數選取，我們並沒有將在 LR 得到的 interaction 變數加入 RF、XGBT，而是用了原始變數丟進去 RF、XGBT，所以得到的 RF、XGBT 在 public score 的 AUC 都顯著低於其他參賽者。也許，在這方面改進，我們可以得到更好的 public leaderboard 名次，進而得到更高的 private leaderboard 名次。

經過本次研究，我們體會到「feature engineering 是一個持續的過程」，從 EDA 到 feature engineering 到 modeling，這三者階段是不斷循環的。也深切體會到 stacking 的強大，使用多個不弱的模型結果，可以混合出最佳的預測結果。

六、參考文獻

資料來源取自 <https://www.kaggle.com/c/GiveMeSomeCredit/overview>

(一) 中文文獻

1. 孫亮、黃倩(2016)。實用機器學習。
2. 有賀康顯、中山心太、西林孝(2018)。機器學習—工作現場的評估導入與實作。
3. 劉正山、莊文忠(2012)。項目無反應資料的多重差補分析。
4. Battiti, R., Brunato, M.(2018)。機器學習與優化。
5. yyHaker(2019)。GBT、GBDT、GBRT 与 Xgboost。取自 <https://zhuanlan.zhihu.com/p/57814935>
6. 果醬珍珍 (2018)。Missing Value Treatment | 遺失值處理 | 統計 R 語言。取自 <https://www.jamleecute.com/missing-value-treatment-%E9%81%BA%E5%A4%B1%E5%80%BC%E8%99%95%E7%90%86/>

(二) 英文文獻

1. Zumel, N., Mount, J. (2014) Practical Data Science with R.
2. Zheng, A., Casari A. (2018). Feature Engineering for Machine Learning.
3. Ozdemir, S., Susarla, D. (2018). Feature Engineering Made Easy.
4. Online forum of the dataset "GiveMeSomeCredit"(2011). Retrieved from <https://www.kaggle.com/c/GiveMeSomeCredit/discussion>