COMP9417 Project - Home Credit Default Risk

Authors: Machine Lean Me
Oliver Li - z3372932, Wei Chen - z5275224, Ho Wan Hou - z5315885

1 Introduction

Lending money is a vital component of today's economy and is a source of money creation. Default risk is the risk that a borrower is unable to repay the principal and interest on a loan to their lender in a timely manner. Financial institutions assess default risk of potential borrowers to help inform whether the loan should be made or not, and to help decide on the interest rate charged to adequately compensate the financial institution for the default risk taken on.

Home Credit Group ('Home Credit') have invited the public via Kaggle to use statistical and machine learning methods to help unlock the full potential of their datasets in measuring default risk. The competition can be found in the following link: "https://www.kaggle.com/c/home-credit-default-risk". Machine Lean Me have accepted this invitation and have attempted to apply their newly acquired machine learning skills on the datasets provided by Home Credit Group.

The primary objective is to create a machine learning model on the provided training data to predict the probability of default on the test data.

2 Data

The dataset for this competition can be located here:
https://drive.google.com/drive/folders/1LGXNdR3ur3FSzrm8TQc9bm6n_6WJobFj?usp

2.1 Overview

Home Credit provided 219 columns of data across 7 datasets. These datasets were:

- application_train/test.csv - Primary data table containing static data of all loan applications including the TARGET variable of whether the applicant had payment difficulties or not.
- bureau.csv - Clients previous credits provided by other financial institutions that were reported to the Credit Bureau.

- bureau_balance.csv - Monthly balance snapshots of previous credits in Credit Bureau.
- POS_CASH_balance.csv - Monthly balance snapshots of previous point of sales and cash loans that the applicant had with Home Credit.
- credit_card_balance.csv - Monthly balance snapshots of previous credit cards that the applicant has with Home Credit.
- previous_application.csv - All previous applications for Home Credit loans of clients who have loans in application_train/test.csv.
- installments_payments.csv - Repayment history for previously disbursed credits in Home Credit related to loans in application_train/test.csv.

2.2 Dataset Completeness

All the above datasets are connected by the column SK_ID_CURR. Unfortunately the non-primary datasets do not contain all SK_ID_CURR in the primary datasets (application_train/test.csv). Below is a table showing the coverage of each dataset in relation to application_train/test.

Table 1: SK_ID_CURR coverage in relation to application_train.csv.

Dataset	No. of SK_ID_CURR Matches	Coverage %
application_train.csv	307,511	NA
bureau.csv	263,491	86%
bureau_balance.csv	92,231	30%
POS_CASH_balance.csv	289,444	94%
credit_card_balance.csv	86,905	28%
previous_application.csv	291,057	95%
installments_payments.csv	291,643	95%

Table 2: SK_ID_CURR coverage in relation to application_test.csv.

Dataset	No. of SK_ID_CURR Matches	Coverage %
application_test.csv	48,744	NA
bureau.csv	42,320	87%
bureau_balance.csv	42,311	87%
POS_CASH_balance.csv	47,808	98%
credit_card_balance.csv	16,653	34%
previous_application.csv	47,800	98%
installments_payments.csv	47,944	98%

3 Feature Selection / Feature Engineering

3.1 Criteria

Reducing the amount of columns in the data is required to reduce the computational complexity of a selected machine learning algorithm to a workable level. This makes Feature Selection an important task. Features were selected according to the following list of criteria:

- Whether there is any intuitive link between the feature and default risk.
- Whether the feature is categorical or numeric. Current Scikit-learn models cannot work directly with categorical data. If a feature is categorical:
  - Is the category ordinal or nominal?
  - How many categories are there and whether it is feasible for one-hot encoding to be used.
  - Whether it is related to a numerical feature already present.
- Whether features are related to other features and whether they can be combined to create a new feature.
- Numerical analysis of the features. Features are not expected to have a linear relationship with default risk so Spearman's rank co-efficient is used. Spearman's rank co-efficient is calculated for each feature against:
  - The TARGET variable.
  - And other features.
- The amount of missing data in the feature for each data-point.

The features that were ultimately selected following the discussion below can be found in the Appendix along with their Spearman's rank co-efficient to the TARGET variable.

3.2 Preprocessing

As a reminder, SK_ID_CURR is the primary key of the primary dataset. The non-primary datasets had different primary keys, with SK_ID_PREV or SK_ID_BUREAU forming a primary key or a subset of a primary key. An SK_ID_CURR could have multiple SK_ID_PREV or SK_ID_BUREAU entries in the non-primary datasets. Below is a list of the primary keys for each dataset.

Table 3: SK_ID_CURR coverage in relation to application_train.csv.

Dataset	Primary Key
application_train/test.csv	SK_ID_CURR
bureau.csv	SK_ID_BUREAU
bureau_balance.csv	SK_ID_BUREAU & MONTHS_BALANCE
POS_CASH_balance.csv	SK_ID_PREV & MONTHS_BALANCE
credit_card_balance.csv	SK_ID_PREV & MONTHS_BALANCE
previous_application.csv	SK_ID_PREV
installments_payments.csv	SK_ID_PREV & NUM_INSTALMENT_NUMBER

Before we were able decide which features are desirable, the non-primary datasets need to be processed into a form that is compatible with our primary dataset which can then be used by our machine learning models. This section describes the preprocessing undertaken.

3.2.1 Numerical Features

For the datasets with only SK_ID_BUREAU or SK_ID_PREV as the primary key, the sum and average of these numerical features were calculated, grouping by SK_ID_CURR.

For the features which included MONTHS_BALANCE or NUM_INSTALMENT_NUMBER as part of the primary key,the sum and average of these numerical features were calculated, grouping by SK_ID_PREV or SK_ID_BUREAU and then summed by SK_ID_CURR.

The sum and average of the numerical features have a prefix in their feature name of SUM and AVG respectively.

3.2.2 Categorical Features

Categorical features with an excessive amount of unique values were removed altogether to save on computing time. For those with a workable amount of unique values, a numerical feature with the count of each of value was formed. These counts were summed by SK_ID_PREV or SK_ID_BUREAU and then summed by SK_ID_CURR.

The count of the categorical features have a prefix in their feature name of COUNT with the suffix being the categorical feature value.

3.3 Feature Manipulation

As mentioned above, it was considered whether any features could be combined to create new features to ease computational complexity while potentially adding predictive power.

Further information on the features we manipulated can be found in the Appendix.

4 Modelling

4.1 Model Selection

Two fundamental model types were considered for modelling default risk:

- Logistic Regression
- Decision Trees

Logistic Regression is a natural choice for modelling default risk as it models the probability of an event taking place, the event being default in the context of this competition.

Decision Trees are a non-parametric model useful in classification type problems. They are easy to interpret and intuitive. Scikit-learn has a Decision Tree package that allows the user to model the probability of a class being predicted, which in the context of this competition, is default.

4.2 Performance Criteria

The performance criteria for this competition is the area under the Receiver Operating Characteristics curve (AUC/ROC). This criteria is useful in binary classification tasks, i.e. default or no-default. It measures the ability of the machine learning model to classify defaults as defaults and non-defaults as non-defaults.

An AUC/ROC of 1 indicates that the model is perfectly classifying defaults as defaults and vice-versa, while an AUC/ROC of 0 indicates that the model is perfectly classifying defaults as non-defaults and vice-versa.

The worst case scenario is when the AUC/ROC is 0.5 which means it is sometimes classifying defaults as non-defaults and non-defaults as defaults. The closer the AUC/ROC is to 1, the more ideal.

4.3 Validation

Our machine learning models were validated using stratified 5-fold cross validation. In 5-fold cross validation, application_train.csv is split into 5 groups with the same number of data points, where 4 groups are used to train the model and 1 group is used to test the model. The training process is iterated 5 times, where the training and testing data are alternated. In each iteration, we calculate the AUC/ROC on the test group and take arithmetic mean of the 5 AUC/ROCs generated.

Stratification ensures that each group has roughly equal class distribution, which is important for the application_train.csv dataset where defaults are less than 10% of the data.

The benefit of using this validation technique compared to a standard fixed 80/20 training/test data split is that it has a tendency of reducing the bias of the machine learning model, as the entire dataset is used to train the model.

Scikit-learn's StratifiedKFold function is used to implement this.

4.4 Initial Fitting

A baseline Decision Tree and Logistic Regression model was fit using Scikit-learn's default model parameters except with criterion set to 'Entropy' for Decision Tree.

For Logistic Regression, the features were normalised because the default model parameter in Scikit-learn was to use L2-regularisation, which becomes sensitive to the size of the feature values.

Experimentation was done with the NULL values to decide whether they should be set to 0, set to mean, median or most frequent. We also initially considered a k-nearest neighbours methodology, but this was considerd to be too computationally expensive. The AUC/ROC of the experiment are illustrated below.

Table 4: AUC/ROC with NULL set to different values.

	Logistic Regression	Decision Tree
NULL set to 0	0.73300	0.54205
NULL set to mean	0.74106	0.54350
NULL set to median	0.74081	0.54271
NULL set to most frequent	0.73650	0.54191

In light of the above, it was decided to set NULL values set to the mean. The AUC/ROC plot for both the Logistic Regression and Decision Tree's are shown below.

In addition, the above table highlights that the fitted Decision Tree model does not provide an adequate model in modelling the probability of default. We consider Ensemble Learning techniques for Decision Trees in the next section.

4.5 Ensemble Learning

As seen in the previous section, the default Decision Tree model was not adequate. Ensemble Learning methods often improve the performance of the underlying model, and so as a result, the team decided to experiment with Gradient Boosted Trees and Random Forests

using GradientBoostingClassifier and RandomForestClassifier respectively in Scikit-learn.

The below table reports the AUC/ROC of Gradient Boosted Trees and Random Forests under Scikit-learn's default parameters. Both Gradient Boosted Trees and Random Forests substantially improved performance. The team decided to stick with Gradient Boosted Trees as it improved the performance more.

Table 5: AUC/ROC of Ensemble Learning Methods.

Method	AUC/ROC
Decision Trees	0.54350
Gradient Boosted Trees	0.75468
Random Forest	0.72891

4.6 Hyper-parameter Tuning

Scikit-learn's Logistic Regression and Gradient Boosted Tree models allow variations in hyper-parameters. A list of hyper-parameters were selected for tuning using a Grid Search Algorithm to find the settings that optimised performance. The Grid Search Algorithm iterates through all possible combinations of hyper-parameters and is quite computationally taxing (due to the amount of iterations) and so the list was kept quite limited. The list of hyper-parameters, sample space of possible values, reasons for their selection and results are described are in the subsections below.

4.6.1 Logistic Regression

The options for Logistic Regression are quite limited compared to Gradient Boosted Trees, albeit the following were still selected.

- penalty - {none, l2} - Controls whether a penalty is included in the loss function. The default model includes l2-regularisation (Ridge Regression), which reduces the probability of over-fitting. l1 regularisation and elasticnet not considered as otherwise solver method would have to change too.

- c_values - {0.01, 0.1, 1, 10, 100} - Strength of the l2-regularisation. Lower hyper-parameter values indicates a stronger l2-regularisation.

- class_weight - {none, balanced} - The balanced mode adjusts weights inversely proportional to class size. This could be useful in this challenge as defaults are a minority class, being less than 10% of the data. This means that data which have the TARGET as default are weighed more.

Table 6: Logistic Regression AUC/ROC from hyper-parameter tuning.

penalty	c_value	class_weight	AUC/ROC
l2	0.01	none	0.73542
l2	0.1	none	0.73895
l2	1	none	0.74109
l2	10	none	0.74159
l2	100	none	0.74170
none	NA	none	0.74171
l2	0.01	balanced	0.73746
l2	0.1	balanced	0.74065
l2	1	balanced	0.74189
l2	10	balanced	0.74215
l2	100	balanced	0.74218
none	NA	balanced	0.74219

The results presented in the table above indicate that having no penalty improves the performance of the machine-learning model. Higher c_values, which are inverse to regularisation (penalty) strength are associated with higher AUC/ROC scores, while the highest score is achieved with no penalty.

The results also indicate that placing a higher weight on datapoints with the TARGET variable as default also improves performance. This makes sense given that these datapoints are less than 10% of the training dataset.

4.6.2 Gradient Boosted Trees

- learning rate - {0.1, 0.2, 0.3} - This is a common parameter to experiment with as it changes the rate at which the model adapts to the traing data.
- min_samples_leaf - {0.01, 0.02, 0.03} - Minimum number of samples in an internal node to prevent over-fitting.
- n_estimators - {100, 200, 300} - Number of trees used by Gradient Boosted Trees. Increasing trees can improve performance but there is a chance of over-fitting which reduces performance.

Table 7: Gradient Boosted Trees AUC/ROC from hyper-parameter tuning.

learning_rate	min_samples_leaf	n_estimators	AUC/ROC
0.1	0.01	100	0.75526
0.1	0.01	200	0.75992
0.1	0.01	300	0.76160
0.1	0.02	100	0.75518
0.1	0.02	200	0.75993
0.1	0.02	300	0.76182
0.1	0.03	100	0.75456
0.1	0.03	200	0.75942
0.1	0.03	300	0.76126
0.2	0.01	100	0.75913
0.2	0.01	200	0.76166
0.2	0.01	300	0.76218
0.2	0.02	100	0.75902
0.2	0.02	200	0.76188
0.2	**0.02**	**300**	**0.76266**
0.2	0.03	100	0.75860
0.2	0.03	200	0.76158
0.2	0.03	300	0.76229
0.3	0.01	100	0.76054
0.3	0.01	200	0.76140
0.5	0.01	300	0.76133
0.3	0.02	100	0.75998
0.3	0.02	200	0.76144
0.3	0.02	300	0.76159
0.3	0.03	100	0.75925
0.3	0.03	200	0.76075
0.3	0.03	300	0.76108

The above results indicate that model performance improves when n_estimators increases, however increasing this hyper-parameter increases runtime and the performance improvement is quite marginal between 200 and 300.

The above results also indicate that the sweet spot for learning_rate is 0.2 and min_samples_leaf is 0.02.

4.7 Final Models and Results

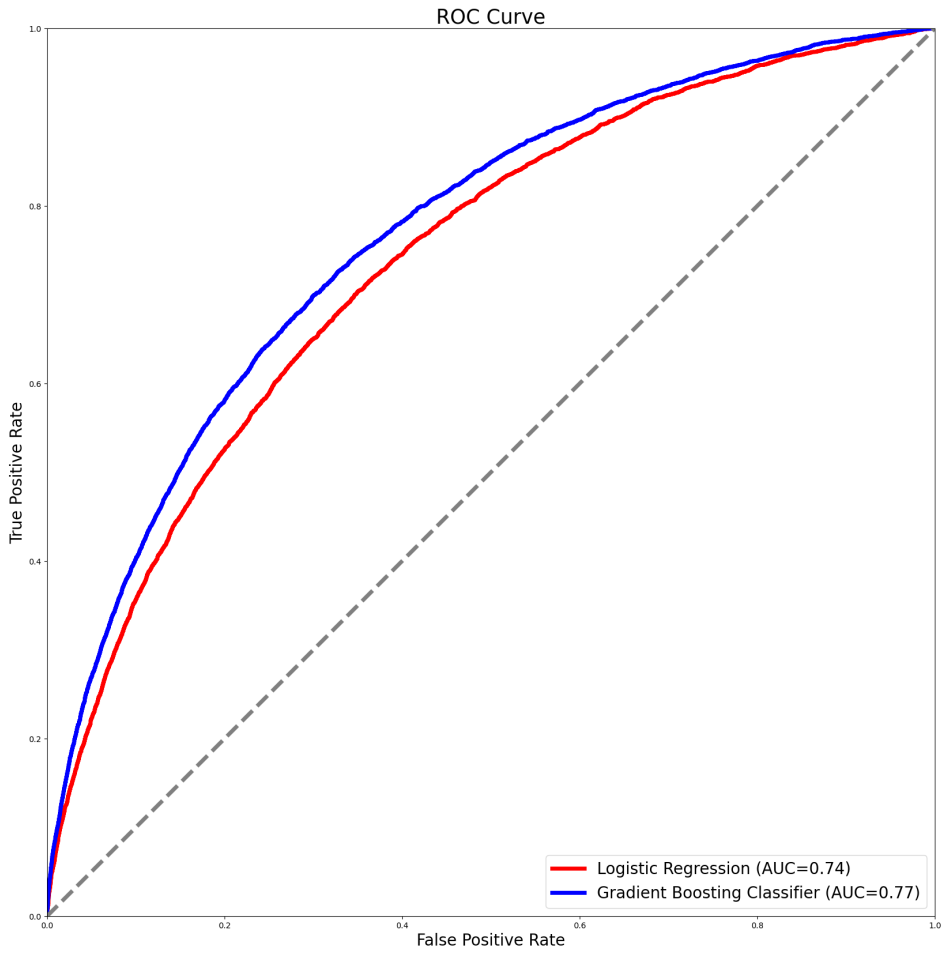
While the Gradient Boosting Trees out-performed the Logistic Regression model on the training data, we decided on trying to submit both models to Kaggle to see which model would perform better. The final parameters of the final models that were submitted to

Kaggle were:

- Logistic Regression - NULL values set to mean, penalty = none, and class_weight = balance.
- Gradient Boosting Trees - NULL values set to mean, learning_rate = 0.2, min_samples_leaf = 0.02, n_estimators = 300.

The AUC/ROC curves for these models are illustrated in the chart below.

Figure 1: AUC/ROC of final models.



4.8 Submission To Kaggle

We submitted both models to Kaggle with the results shown below. The public AUC/ROC is calculated using 20% of application_test.csv while the private AUC/ROC is calculated using 80%.

Table 8: Kaggle Submission Results

Model	Public AUC/ROC	Private AUC/ROC
Logistic Regression	0.51640	0.52035
Gradient Boosting Trees	0.72823	0.71938
Logistic Regression (v2, see below)	0.72022	0.73087

The results show that the Logistic Regression had substantially under-performed compared to what we expected. Further experimentation revealed that this was due to setting the penalty as 'none'. We submitted a version to Kaggle with the penalty set to 'l2' and c-value set to 0.01 and the Public and Private AUC/ROC scores were 0.72022 and 0.73087 respectively - a dramatic improvement. This is a sign of over-fitting in our model training process described earlier.

The Gradient Boosting Trees AUC/ROC also reduced, but not as much as the Logistic Regression. This is also a sign of over-fitting.

A screenshot of our Gradient Boosting Tree submission is provided in the Appendix

4.9 Reflection

We settled on this competition as a couple of the members had an understanding of default risk and found the topic to be interesting. We had spent considerable time researching models and techniques before starting, only to realise that there was considerable time required in thinking of ways to pre-process the data and pulling the data together. Ideally we would have spent less time on this stage of the model fitting process.

A key learning for future machine learning and artificial intelligence projects is to begin implementation as soon as possible, get a feel for the data and what tools we can work with. A higher focus could also have been placed on computational complexity and how long it takes to run an algorithm, especially the Grid Search Algorithm. Despite our difficulties, we learned a lot through experimenting with different techniques such as cross-validation, hyper-parameter tuning, model selection as well as learning any advantages and disadvantages.

5 References

- Scikit-learn.org. (2022). sklearn.linear_model.LogisticRegression — scikit-learn 0.21.2 documentation. [online]. Available at: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html.

- Scikit-learn.org. (2022). sklearn.tree.DecisionTreeClassifier — scikit-learn 0.22.1 documentation. [online] Available at: https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html.

- Scikit-learn.org. (2022). sklearn.ensemble.GradientBoostingClassifier — scikit-learn 0.20.3 documentation. [online] Available at: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html.

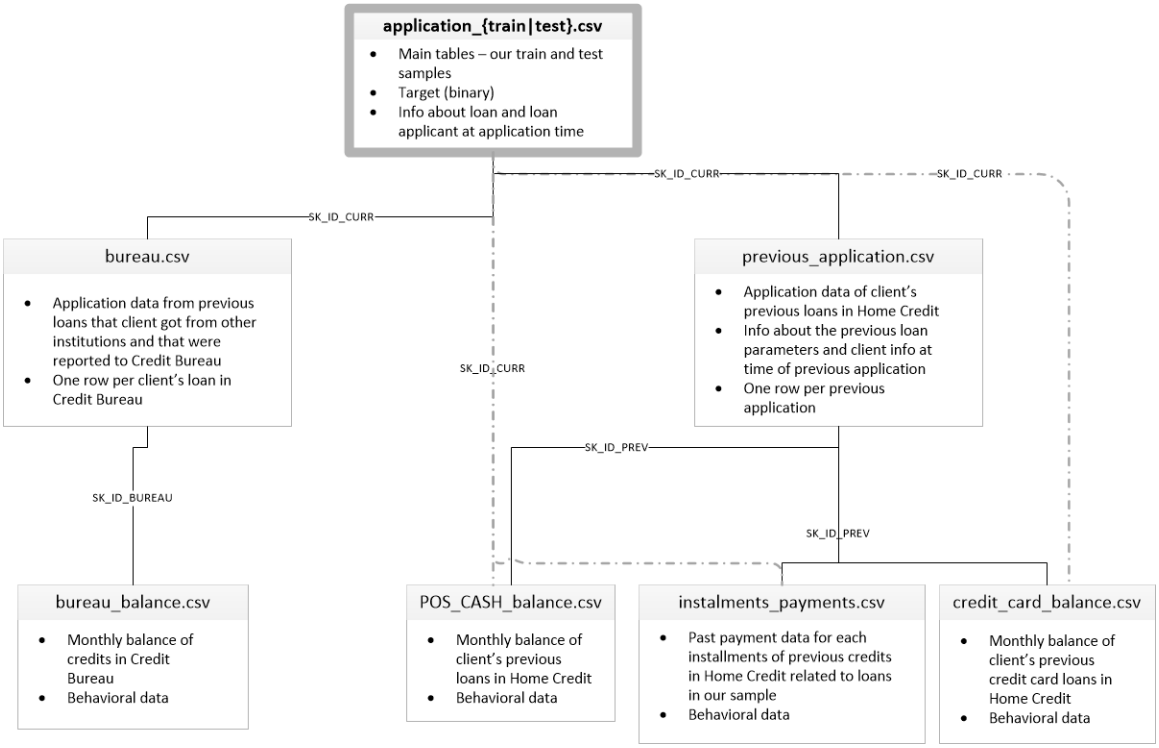
- Scikit-learn.org. (2022). sklearn.ensemble.RandomForestClassifier — scikit-learn 0.20.3 documentation. [online]. Available at: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html.

- Scikit-learn.org. (2022). sklearn.model_selection.StratifiedKFold — scikit-learn 0.21.3 documentation. [online] Available at: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedKFold.html.

- Scikit-learn.org. (2022). sklearn.model_selection.GridSearchCV — scikit-learn 0.22 documentation. [online] Available at: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html.

6 Appendix

6.1 Data Chart



6.2 Feature Manipulation

6.2.1 Application_train/test: Address_score

"Address_score" was formed as the sum of 6 address related indicator variables:

- REG_REGION_NOT_LIVE_REGION
- REG_REGION_NOT_WORK_REGION
- LIVE_REGION_NOT_WORK_REGION
- REG_CITY_NOT_LIVE_CITY
- REG_CITY_NOT_WORK_CITY
- LIVE_CITY_NOT_WORK_CITY

As illustrated in the table below, combining the variables created a variable which almost has the highest Spearman's rank coefficient. It is intuitive to keep these variables together as they measure the inconsistency with the applicant's registration address, work and living arrangements.

Table 9: Spearman rank co-efficient of address variables to TARGET.

Variable	Spearman's ρ
REG_REGION_NOT_LIVE_REGION	0.006
REG_REGION_NOT_WORK_REGION	0.007
LIVE_REGION_NOT_WORK_REGION	0.003
REG_CITY_NOT_LIVE_CITY	0.044
REG_CITY_NOT_WORK_CITY	0.051
LIVE_CITY_NOT_WORK_CITY	0.033
Address_score	0.048

6.2.2 bureau_balance.csv: COUNT_ge1

"COUNT_ge1" was formed as the sum of the below variables which counted the number of times the applicant was past a due date for a payment.

- "COUNT_1" - 0 to 1 months past due date.
- "COUNT_2" - 1 to 2 months past due date.
- "COUNT_3" - 2 to 3 months past due date.
- "COUNT_4" - 3 to 4 months past due date.
- "COUNT_5" - 4 to 5 months past due date.

As illustrated in the table below, combining the variables created a variable which almost has the highest Spearman's rank coefficient. It is intuitive to keep these variables together as they are an indication of the applicant's previous payment issues.

Table 10: Spearman rank co-efficient of COUNT variables to TARGET.

Variable	Spearman's ρ
COUNT_1	0.051
COUNT_2	0.016
COUNT_3	0.016
COUNT_4	0.018
COUNT_5	0.017
COUNT_ge1	0.050

6.3 Selected Features

6.3.1 application_train/test.csv

Table 11: Spearman's ρ of application_train/test.csv variables to TARGET.

	Spearman's ρ
DAYS_BIRTH	0.078
REGION_RATING_CLIENT_W_CITY	-0.060
NAME_INCOME_TYPE_Working	0.057
CODE_GENDER_M	0.055
DAYS_ID_PUBLISH	0.053
NAME_EDUCATION_TYPE_Secondary / secondary special	0.050
Address_score	0.048
OCCUPATION_TYPE_Laborers	0.054
ORGANIZATION_TYPE_XNA	-0.046
NAME_INCOME_TYPE_Pensioner	-0.046
EXT_SOURCE_1	-0.048
TOTALAREA_MODE	-0.048
NAME_EDUCATION_TYPE_Higher education	-0.057
EXT_SOURCE_3	-0.121
EXT_SOURCE_2	-0.147

6.3.2 bureau.csv

Table 12: Spearman's ρ of bureau.csv variables to TARGET.

	Spearman's ρ
COUNT_ACTIVE	0.057
COUNT_CLOSED	-0.044
AVG_DAYS_CREDIT	-0.095
AVG_CREDIT_DAY_OVERDUE	0.034
AVG_DAYS_CREDIT_ENDDATE	0.067
AVG_AMT_CREDIT_SUM_DEBT	0.054

6.3.3 bureau_balance.csv

Table 13: Spearman's ρ of bureau_balance.csv variables to TARGET.

	Spearman's ρ
AVG_COUNT_ge1	0.050

6.3.4 POS_CASH_balance.csv

Table 14: Spearman's ρ of POS_CASH_balance.csv variables to TARGET.

	Spearman's ρ
POS_OCCURENCES	-0.043
AVG_SK_DPD_DEF	0.040

6.3.5 credit_card_balance.csv

Table 15: Spearman's ρ of credit_card_balance.csv variables to TARGET.

	Spearman's ρ
AVG_AMT_BALANCE	0.104
COUNT_ACTIVE	-0.055

6.3.6 previous_application.csv

Table 16: Spearman's ρ of previous_application.csv variables to TARGET.

	Spearman's ρ
COUNT_REVOLVING	0.041
AVG_AMT_ANNUITY	-0.045
AVG_AMT_DOWN_PAYMENT	-0.053
AVG_DAYS_DECISION	0.055
COUNT_REJECT	0.061
COUNT_YIELD_GROUP_HIGH	0.037
COUNT_YIELD_GROUP_LOW	-0.049

6.3.7 installment_payments.csv:

Table 17: Spearman's ρ of installment_payments.csv variables to TARGET.

	Spearman's ρ
AVG_DIFF_DAY_PAYMENT	0.054
AVG_DIFF_AMT_PAYMENT	0.051

6.4 Submission Evidence