Machine Learning Engineer Nanodegree

Udacity Capston Project: Home Credit

Author: Omar Villa

Date: November 2nd 2018

Contents

Definition

Overview

Home Credit strives to broaden financial inclusion for the unbanked population by providing a positive and safe borrowing experience. In order to make sure this underserved population has a positive loan experience, Home Credit makes use of a variety of alternative data--including telco and transactional information--to predict their clients' repayment abilities.

While Home Credit is currently using various statistical and machine learning methods to make these predictions, they're challenging Kagglers to help them unlock the full potential of their data. Doing so will ensure that clients capable of repayment are not rejected and that loans are given with a principal, maturity, and repayment calendar that will empower their clients to be successful.

Problem Statement

The Home Credit Default Risk competition is a supervised classification machine learning task. The objective is to use historical financial and socioeconomic data to predict whether or not an applicant or client will be able to repay the loan, we plan to accomplish this through a scientific and repeatable preprocessing first creating lists of the different features types, use cross-validation to split and test our model, if is necessary create features out of existing features, cleanup data or fill empty values, squash values that are skewed with techniques as log-transform, drop columns if we find is necessary, One-hot encode categorical features with 3 or more values once all this preprocessing is completed we can use the resulting data frame and run it through a Machine Learning algorithm. This is a standard supervised classification task:

• Supervised: The labels are included in the training data and the goal is to train a model to learn and predict the labels from the features

• Classification: The label is a binary variable, 0 (will repay loan on time) and 1 (will have difficulty repaying the loan)

The final application is expected to predict when or who are the best candidates for credit loan.

Metrics

Our evaluation metrics will be based in a ROC Curve first because is what Kaggle competition requests and second because ROC is a probability curve that help us to easily and clearly see a model behavior and performance in a multiclass model meaning we can plot multiple results under the same chart and easily compare them which in this problem domain is something we need to do in order to know which of the Machine Learning models work best to predict if an applicant will be able to pay their loan based on the TARGET feature of our dataset.

[1] Understanding AUC - ROC Curve

The target metric will be "TARGET" under the application_train/test datasets, results are based on loan payed or not payed

Analysis

Data Exploration

application_{train|test}.csv

This is the main table, broken into two files for Train (with TARGET) and Test (without TARGET). Static data for all applications. One row represents one loan in our data sample.

bureau.csv

All client's previous credits provided by other financial institutions that were reported to Credit Bureau (for clients who have a loan in our sample). For every loan in our sample, there are as many rows as number of credits the client had in Credit Bureau before the application date.

bureau_balance.csv

Monthly balances of previous credits in Credit Bureau. This table has one row for each month of history of every previous credit reported to Credit Bureau – i.e the table has (#loans in sample *# of relative previous credits* # of months where we have some history observable for the previous credits) rows.

POS_CASH_balance.csv

Monthly balance snapshots of previous POS (point of sales) and cash loans that the applicant had with Home Credit. This table has one row for each month of history of every previous credit in Home Credit (consumer credit and cash loans) related to loans in our sample – i.e. the table has (#loans in sample *# of relative previous credits* # of months in which we have some history observable for the previous credits) rows.

credit_card_balance.csv

Monthly balance snapshots of previous credit cards that the applicant has with Home Credit. This table has one row for each month of history of every previous credit in Home Credit (consumer credit and cash loans) related to loans in our sample – i.e. the table has (#loans in sample *# of relative previous credit cards* # of months where we have some history observable for the previous credit card) rows.

previous_application.csv

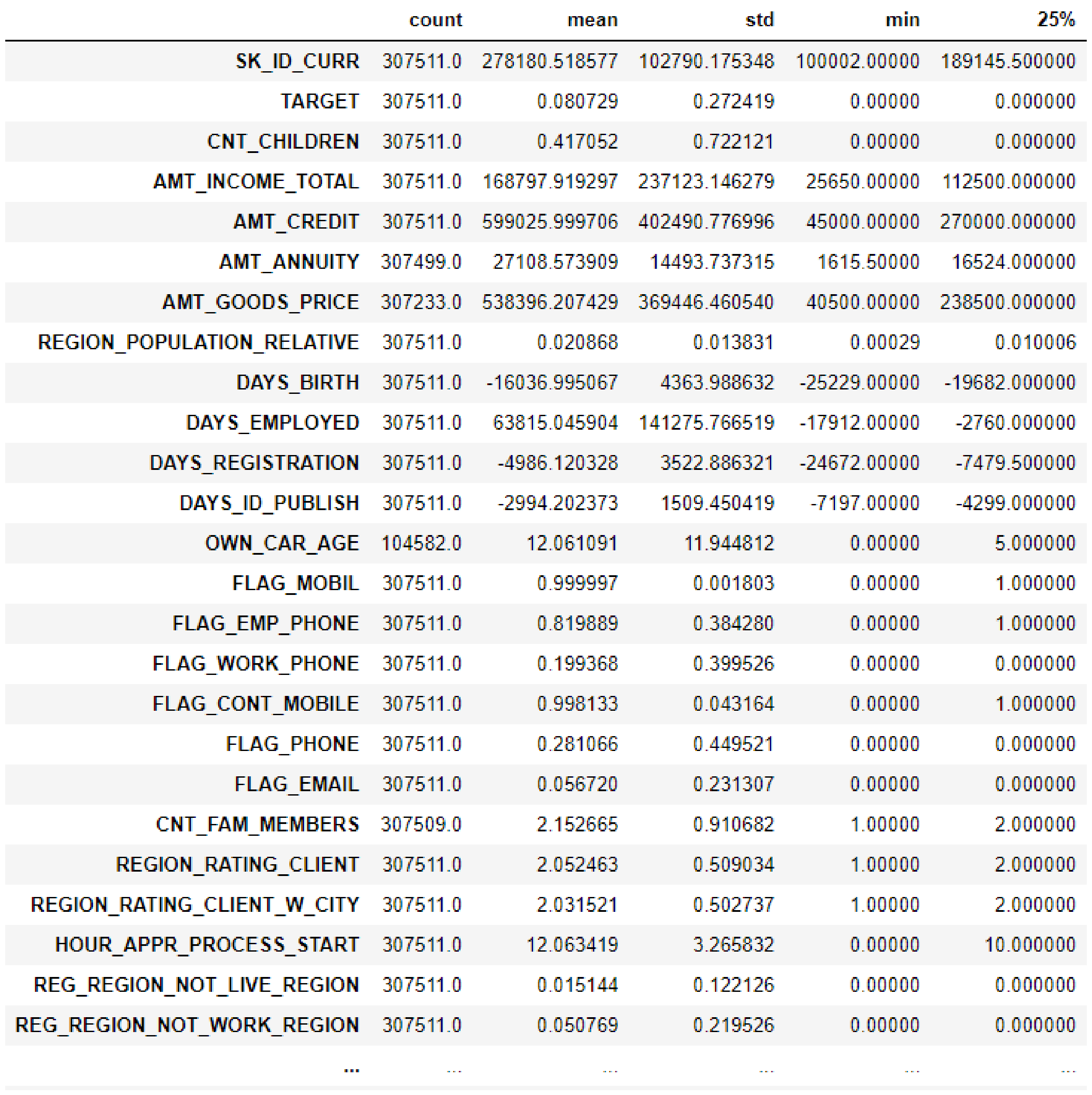
All previous applications for Home Credit loans of clients who have loans in our sample. There is one row for each previous application related to loans in our data sample.

installments_payments.csv

Repayment history for the previously disbursed credits in Home Credit related to the loans in our sample. There is a) one row for every payment that was made plus b) one row each for missed payment. One row is equivalent to one payment of one installment OR one installment corresponding to one payment of one previous Home Credit credit related to loans in our sample.

HomeCredit_columns_description.csv

This file contains descriptions for the columns in the various data files, following is a header and statistics of some of the features, please look at the jupyter notebook if a total output is required:



	count	mean	std	min	25%
SK_ID_CURR	307511.0	278180.518577	102790.175348	100002.00000	189145.500000
TARGET	307511.0	0.080729	0.272419	0.00000	0.000000
CNT_CHILDREN	307511.0	0.417052	0.722121	0.00000	0.000000
AMT_INCOME_TOTAL	307511.0	168797.919297	237123.146279	25650.00000	112500.000000
AMT_CREDIT	307511.0	599025.999706	402490.776996	45000.00000	270000.000000
AMT_ANNUITY	307499.0	27108.573909	14493.737315	1615.50000	16524.000000
AMT_GOODS_PRICE	307233.0	538396.207429	369446.460540	40500.00000	238500.000000
REGION_POPULATION_RELATIVE	307511.0	0.020868	0.013831	0.00029	0.010006
DAYS_BIRTH	307511.0	-16036.995067	4363.988632	-25229.00000	-19682.000000
DAYS_EMPLOYED	307511.0	63815.045904	141275.766519	-17912.00000	-2760.000000
DAYS_REGISTRATION	307511.0	-4986.120328	3522.886321	-24672.00000	-7479.500000
DAYS_ID_PUBLISH	307511.0	-2994.202373	1509.450419	-7197.00000	-4299.000000
OWN_CAR_AGE	104582.0	12.061091	11.944812	0.00000	5.000000
FLAG_MOBIL	307511.0	0.999997	0.001803	0.00000	1.000000
FLAG_EMP_PHONE	307511.0	0.819889	0.384280	0.00000	1.000000
FLAG_WORK_PHONE	307511.0	0.199368	0.399526	0.00000	0.000000
FLAG_CONT_MOBILE	307511.0	0.998133	0.043164	0.00000	1.000000
FLAG_PHONE	307511.0	0.281066	0.449521	0.00000	0.000000
FLAG_EMAIL	307511.0	0.056720	0.231307	0.00000	0.000000
CNT_FAM_MEMBERS	307509.0	2.152665	0.910682	1.00000	2.000000
REGION_RATING_CLIENT	307511.0	2.052463	0.509034	1.00000	2.000000
REGION_RATING_CLIENT_W_CITY	307511.0	2.031521	0.502737	1.00000	2.000000
HOUR_APPR_PROCESS_START	307511.0	12.063419	3.265832	0.00000	10.000000
REG_REGION_NOT_LIVE_REGION	307511.0	0.015144	0.122126	0.00000	0.000000
REG_REGION_NOT_WORK_REGION	307511.0	0.050769	0.219526	0.00000	0.000000
...	...	...	...	...	...

Fig 1. Shows a transposed header and description of the application_train dataset.

Notice that features as DAYS_BIRTH and DAYS_EMPLOYED show big differences between their means and standard values meaning some of this features will need some clean up or rework as one-hot encoding or log-transpose to adapt them to the rest of the data.

Data table statistics

For this project we will use the main table's application_train.csv and application_test.csv as well the bureau.csv table. The application_train table contains 307,511 rows x 122 features each row represents an individual or loan applicant and the features is the data captured by Home Credit regarding the borrower, this table contains the TARGET feature which is the target to predict which is a binary feature where 1 means the borrower made a least one late payment and target value 0 indicates borrowed has paid on time every month.

Of the 122 features we can say that DAYS_BIRTH and DAYS_EMPLOYED have the highest correlation vs the TARGET feature, DAYS_EMPLOYED feature required some feature engineering because it had negative values and counting days employed cannot really say much about the borrower employment situation which is why we created a new feature which only indicates of the borrower haves a job or not.

Bureau table contains summary information of applicant's loans from another lenders. Each row represents a loan and by instance each applicant can have one or more loans. In order to link this table with application_train we will need to create a feature that connects both tables, more of this under the Feature Engineering section.

See figure 1 for more details on some data statistics.

Empty Values or NaN entries

We found around 20 features with more than 60% of missing data or NaN values, because of this we check if the features had good enough data to keep them or if we was force to drop them, fortunately the data was usuable and strategies as Impute work to repopulate the data.

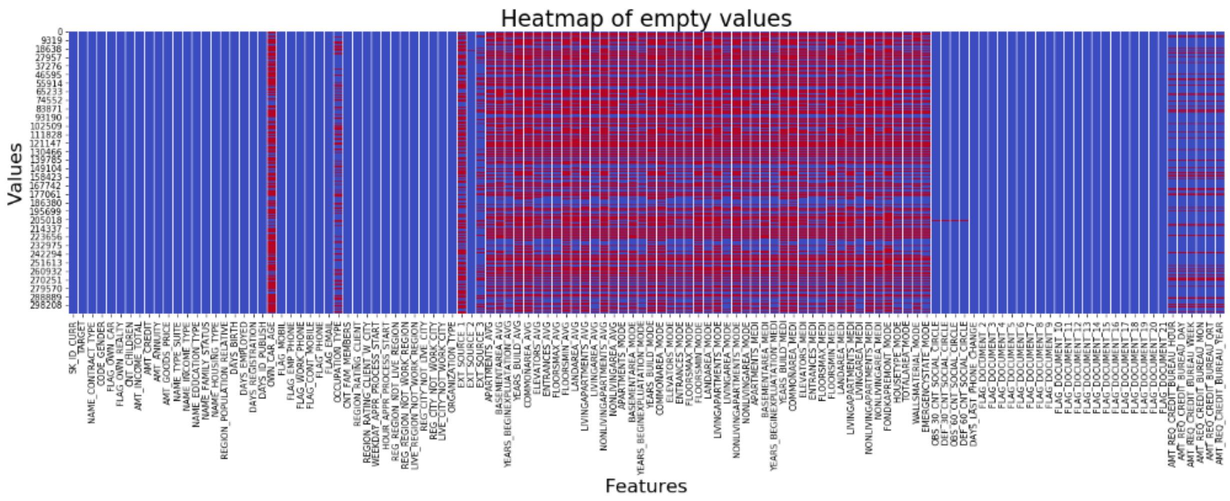


Fig 2. Heatmap shows empty values in red and populated values in blue.

Grouping Features

In order to better split the data and fill the NaN's or execute any Feature Engineering strategy we notice there was different features groups:

1. Categorical Features, divided in binary and Multi-categorical.

2. Numerical Features, divided in features with floating values and whole values at the same time each one of this two had binary entries most of these fall under the Numerical Features with whole values and the rest had several different values up to the thousands.

Once data was categorized it was easier to Engineer other features and move the new feature under another category as the CNT_CHILDREN feature who had several different values, we converted this to another binary category named HAS_CHILDREN indicating 0 for no-children and 1 for children.

This logic was applied to the bureau table but no new features where created.

Anomalies

There was five numerical features with a good level of anomalies: DAYS_BIRTH, DAYS_EMPLOYED, DAYS_REGISTRATION, DAIS_ID_PUBLISH and DAYS_LAST_PHONE_CHANGE which we only end up "fixing" DAYS_BIRTH, DAYS_EMPLOYED and DAYS_REGISTRATION and left the other two alone, critical sample of this is the creation of HAS_JOB feature from DAYS_EMPLOYED

Second anomaly was found with the CNT_FAM_MEMBERS, the size of this feature was represented as a numerical value and we decided to engineer a binary categorical feature named HAS_CHILDREN.

Feature Engineering

Of all the tables outside of application_train/test we found that bereau.csv contains a summary of information of applicant loans from other lenders, being this more useful for our target we have the hypothesis that engineering a feature out of this table that could relate with application_train table was best thing to do to better evaluate the applicants not only based on the data or loans with Home Credit but with other lenders.

The engineered featured was based on CREDIT_DAY_OVERDUE and indicates if a borrower has overdue loans with other lenders, we named this new feature HAS_CREDIT_BUREAU_LOANS_OVERDUE is of the binary type indicating of the applicant have fail to pay any of its loan with the other creditors.

During the exploratory visualization we found an anomaly with the DAYS_EMPLOYED feature, some of the numbers where way to far from the rest of the data but because all what we care for this analysis is if the borrower has a job or not is that we converted this numeric feature in to a binary feature where if the applicant had more than 0 days employed was enough to know if applicant will have the income to pay the loan from this we created the HAS_JOB feature and added to our dataset.

Exploratory Visualization

Distribution of target feature

I plotted the TARGET feature distribution and we found that we have a very small number of applicants who have paid in time their loans, pretty much a 10% of the total pays their loans on time.

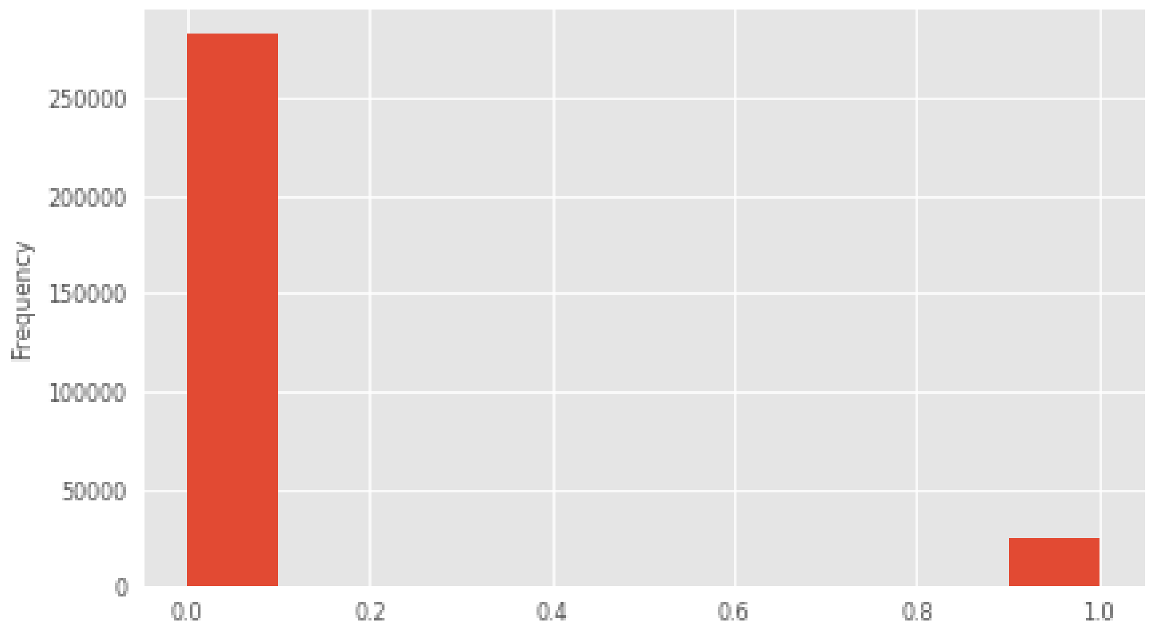


Fig 3. TARGET feature indicating 0.0 for applicants with at least one failed payment and 1 for those who haven't fail a single payment.

Based on above data we wanted to know if the missing data showed in figure 2 had any impact over our target feature, and we found that around 2000 of the total of the applicants had 60% of their data missing, but being the total number of applicants more than 300,000 we consider this number small and is why we decided to keep all the features and later impute the missing values with the mean of each feature.

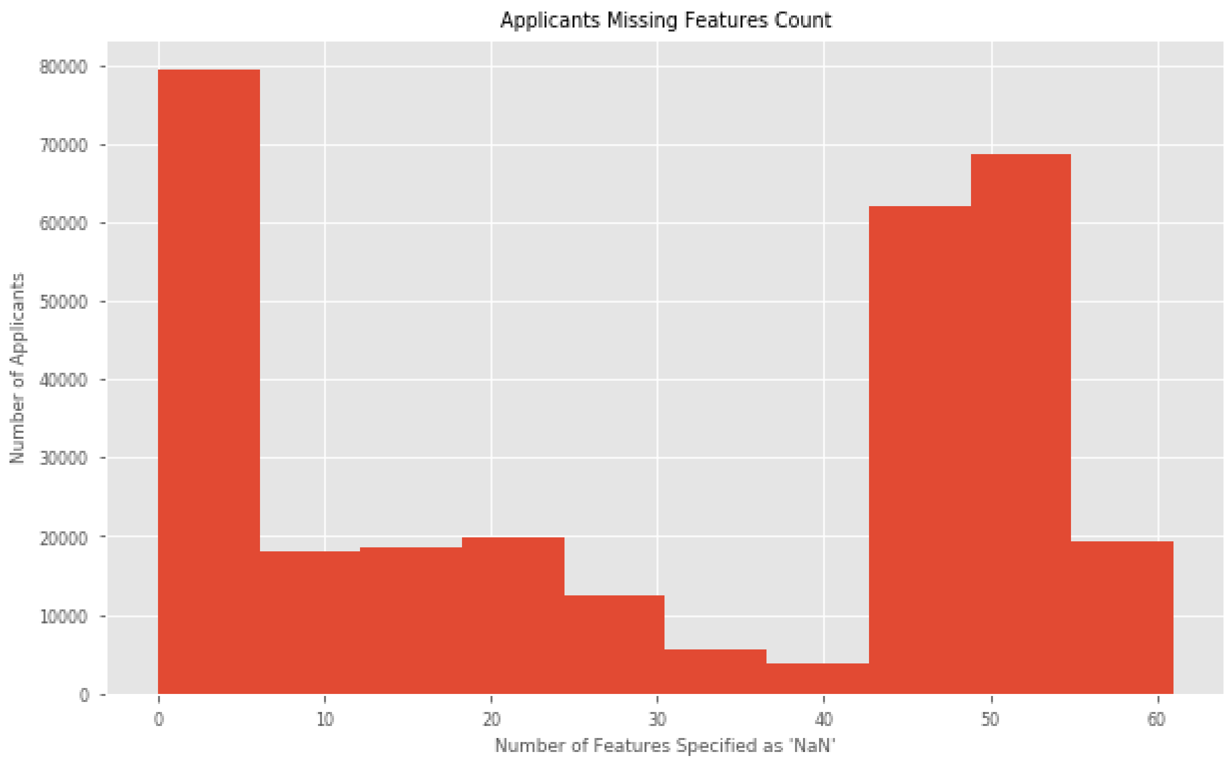


Fig 4. Number of feature with NaN values vs the Number of applicants.

We also found some key features that showed anomalies on their charts, some had negative values as DAYS_BIRTH we could only assume that Home Credit system was counting the number of days that person was alive since date of birth.

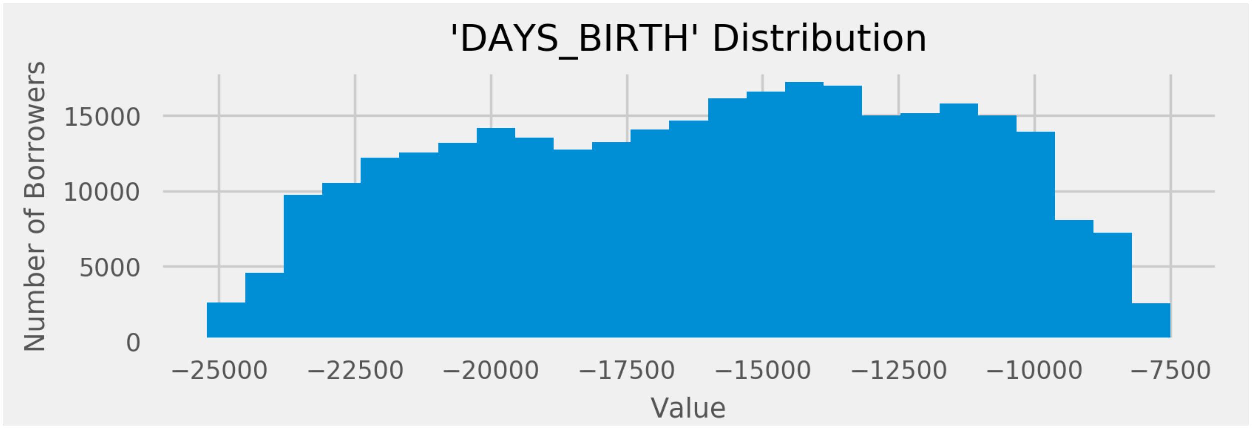


Fig 5. DAYS_BIRTH distribution in negative values anomaly.

Other important feature that we found with anomalies was DAYS_EMPLOYED, here we found some values way too far from the rest of the values this help us to make to create the HAS_JOB feature mentioned in the feature engineering section.

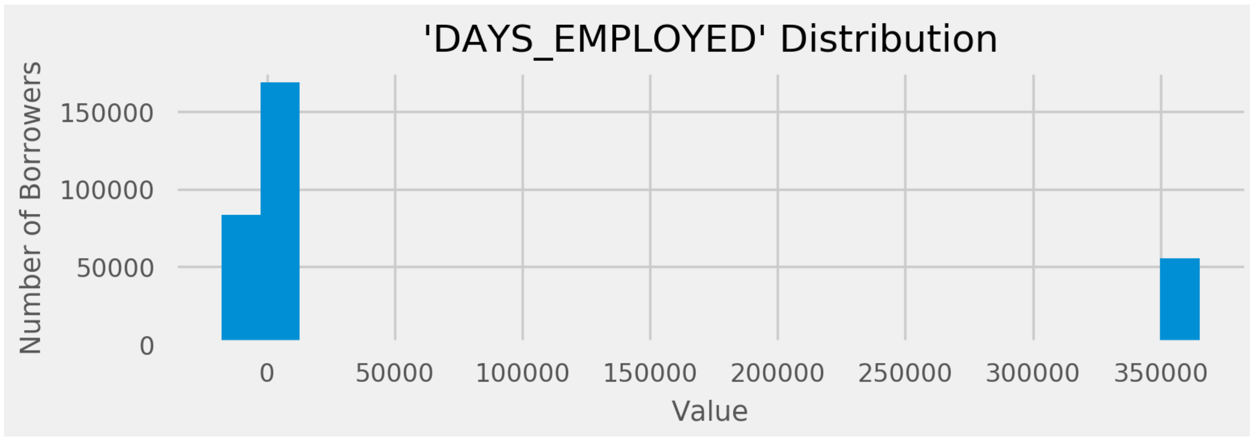


Fig 6. DAYS_EMPLOYED distribution anomaly.

Algorithms and Techniques

The classifier is a Known Neighbors, which is based on comparing neighbor data around a select point then algorithm will evaluate and count this neighbors and determine if the point in evaluation belongs to one group of data or not, the algorithm is a bit slow but work well for analysis and is not part of the regression family which is our benchmark and I think that using another family of algorithms can give us a good view of which model works better, the processing steps

To solve our problem we think that Logistic Regression and Known Neighbors algorithms are very good matches because our target feature is a binary column and this two algorithms can split the data in a different way but they can find patterns that might help our machine learning process to actually learn and predict the target feature.

Our first algorithm is Logistic Regression, this algorithm is based on a Sigmoid function:

$$\sigma(t) = \frac{e^t}{e^t + 1} = \frac{1}{1 + e^{-t}}$$

Fig. 7 Sigmoid Formula

This takes any real value between zero and one and gives a probability if this values is above or below a 0.5 benchmark.

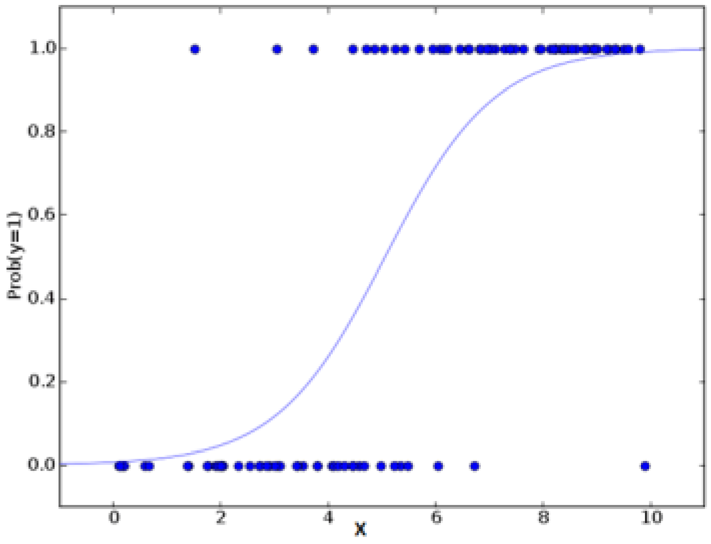


Fig. 8 Logistic Regression S curve [1]

Our second algorithm Known Neighbors is based on clustering pieces of the data and compare them among their neighbors, in example:

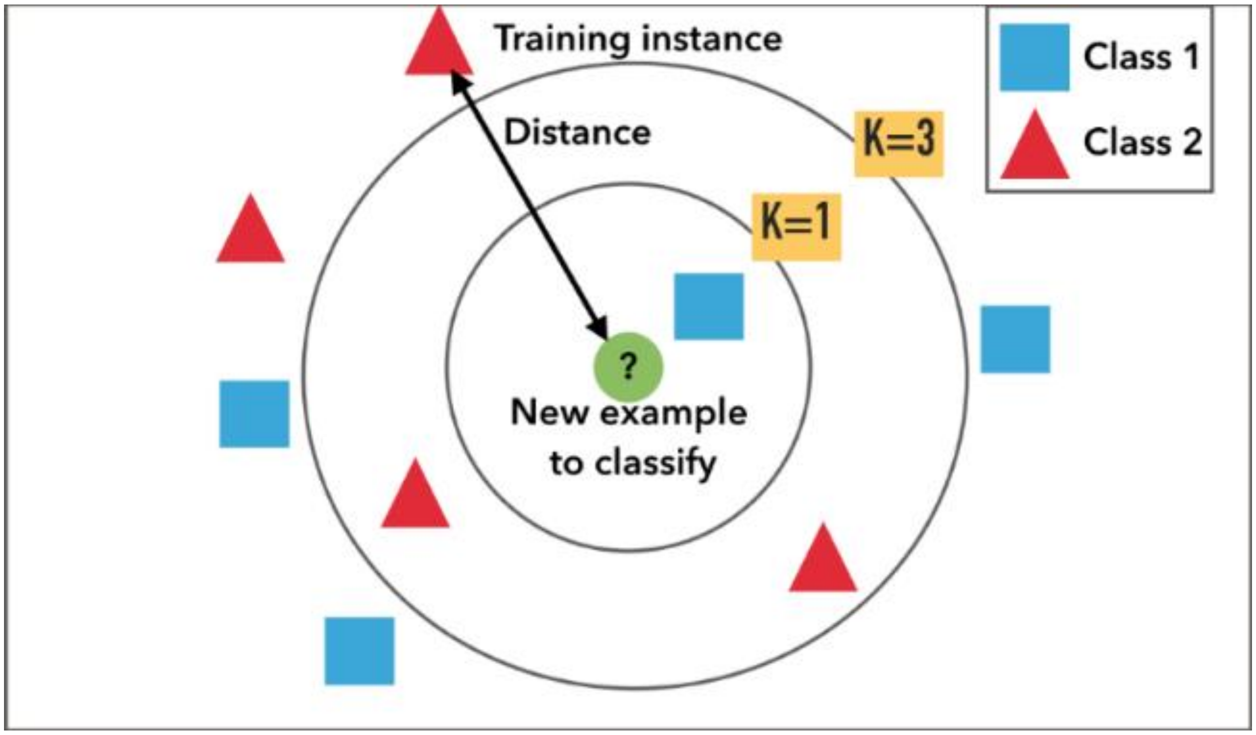


Fig. 9 Example of k-NN classification. The test sample (inside circle) should be classified either to the first class of blue squares or to the second class of red triangles. If k = 3 (outside circle) it is assigned to the second class because there are 2 triangles and only 1 square inside the inner circle. If, for example k = 5 it is assigned to the first class (3 squares vs. 2 triangles outside the outer circle). [2]

Benchmark

As this is a Kaggle competition a benchmark model would be SK_ID_CURR in the test set, who must predict a probability for the TARGET variable. The submission will be evaluated on area under the ROC curve between the predicted probability and the observed target.

We will use a Multi-Layer Perceptron Classifier algorithm as our benchmark model because we can control the output of nodes and it can help us to predict the outcome.

Our goal is not to win the competition but to present a solid model with a scientific method that can be use in other datasets so in the end we will learn the machine learning methodology being this the main objective of this project.

My Solutions will be ranked by the area under the curve (ROC) between the predicted probabilities and the observed targets indicating if the borrower will be able to pay its loan.

I will use the Multi-Layer Perceptron classifier to analyze the dataset and use the ROC as the benchmark to see if there is any major difference between the Logistic Regression and KNN classifiers.

Methodology

Data Preprocessing

1. We split the application_train table by 80/20% based on the cross-validation technique, even when we was provided with the application_test table, is a best practice to use the cross-validation as a pre-step to actual testing and potentially confirm that what have cross-validated it near to the actual test.
2. Fixed anomalies, as we already mentioned in our previous section and in some cases we created new features to squash this anomalies this specifically points to DAYS_EMPLOYED and CREDIT_DAY_OVERDUE.
3. Other Anomalies that required attention where the features with negative values who were out of proportion specifically DAYS_REGISTRATION and DAYS_LAST_PHONE_CHANGE for this features we log-normalized the values to align them with the rest of the features.
4. Missing data imputation, as we shown in figure 2 we found that around 20 features had more than 60% of NaN values and we decided to keep them because they didn’t represented much of problem for the number of applicants who failed to pay their loans because only 2000 of the more than 300,000 applicants who fail on their payments had more than 60% of their , what we did with each one of this features was to calculate the mean of each feature and fill the empty or NaN values with the resulting mean, to accomplish this we used Sklearn Imputer, with the mean strategy and fill each NaN value with the mean of the column or feature.
5. Regarding the categorical features we One-hot encoded all the non-binary features, this was necessary because for machine learning to work we can only present numbers to be analyzed and categorical features will not be computable explaining why it was necessary to use this technique that causes and explosion of features which it can be a big setback but helps with the processing.

6. For the One-hot encoded features who were binary, it was necessary to replace the NaN values with a zero, this was necessary because NaN is not part of a binary value as 0 and 1 so we imputed all this values to 0 using a replacement technique.
7. As final step of the preprocessing we scaled all numeric values in a range of 0.0 to 1.0.

Implementation

1. Logistic Regression classifier. I felt that starting with this classifier was the best option because our target feature is a binary feature and logistic regression is king on this matter and one of the main principles of machine learning is to start with the most simple algorithm before we get in to more complex classifiers, our initial training give us a prediction score of 74%, this using the default values of scikit learn Logistic Regression classifier.
2. GridSearchCV. Using a grid we was able to increase our 74% score to 91% something to mention is that using our ROC AUC Scorer function we scored only 50% accuracy and the resulting hyperparameters of the GridSearch where totally different or opposite while we use the ROC AUC Scorer function we got a C=1000 and penalty=l2 and with the default scorer best score hyperparameters where C=0.001 and penalty=l1
3. KNeighbor classifier. This classifier help us to compare with the benchmark created by the Logistic Regression classifier

Refinement

To refine or analysis we used the Principal Component Analysis to reduce the number of features or dimensionality of the dataset. PCA will help us to find the properties that show as much variations across the dataset fitting the PCA algorithm to the numerical features using 17 components reaching a 90% of variance, we think with this can reduce the characteristics of our problem and increase the performance of the Logistic Regression and KNeighbors algorithm without impacting the results we obtained before the refinement.

As another part of the refinement was the GridSearch technique use to find the best hyperparameters for our Logistic Regression model finding that C=0.0001 with a Penalthy=l1 where the best parameters for the model getting an accuracy of 91.92%.

Results

Model Evaluation and Validation

In the begging I couldn’t tell which algorithm to use or which one will work better this is why I randomly picked the Logistic Regression and KNeighbors models, first one based on our target feature which is a binary feature but once we introduced the Multi-Layered Perceptron Classifier as our benchmark we was able to compare our resulting models and see if any of them was a better solution then our benchmark.

During the development of the model we set a validation and testing parameters using cross-validation to split our data and GridSearchCV to find the best or most convenient hyper-parameters for either the Logistic Regression or Known Neighbors models. The best hyper-parameters for our model found through a Grid Search CV selection for the Logistic Regression classifier was C=0.001 and Penalty=l1 when we was using the ROC AUC Scaler but when we use the default Logistic Regression scaler we also notice a significant improvement on the accuracy of the model and compared with MLP Classifier we can say that the Logistic Regression model gives better results and improvement from a 74% the MLP give us to a 91.92% of Logistic Regression accuracy.

The selection of PCA as our refinement process was based on the number of dimensions or data had, with more than 200 features in our dataset it was almost obvious that a reduction was necessary not only to improve performance but to see if the solution could be achieve with less variables.

Furthermore, after the final hyper-parameter tuning and training the model we found a 91.92% score when we use the default scaler for our Machine Learning algorithms, we think this number must be in the space of over-fitted model because using the ROC AUC scaler we got a 50.62% accuracy for our first runs on the training data and after we run KNeighbors against the PCA data ROC AUC only improve a 3% up to 53.62% but the Logistic Regression give us a 75% score it went up from 20% from 50.62%, telling us that our initial approach was correct regarding the Logistic Regression method but when we applied the GridSearch best score values the results improved by a 5% before PCA values application.

Model robustness & Trust

After all testing we can say our model is robust enough to be trusted because the final results were very similar between the CV testing and the actual testing data, we know we can trust out model parameters because we did a thorough Grid Search of the hyper-parameters and we found which parameters where the most appropriate and best for our training data and after using different data never seen by our model the accuracy score was very similar proving the robustness of the model.

I think that our model performs very well at the state of the art level after comparing the results with the different results in the leader board table of Kaggle.com which most of the models give an 80% of accuracy which is only 5% different compared with ours and we must consider that the Kaggle models are all design by teams of already experienced Data Scientists giving us a very good level of confidence

Justification

Using an Intel i5-86000K CPU @ 3.60GHz with 6 cores and a total of 8GB of RAM our testing took only a couple of minutes to finish with the Logistic Regression algorithm vs KNeighbors that took a couple of hours to run each time, our results were also way better with this method.

With this results we can tell Home Credit that we have a 75% accuracy model on selecting applicants that will pay their loan which it can give more confidence on their selection process.

Conclusion

Free-Form Visualization

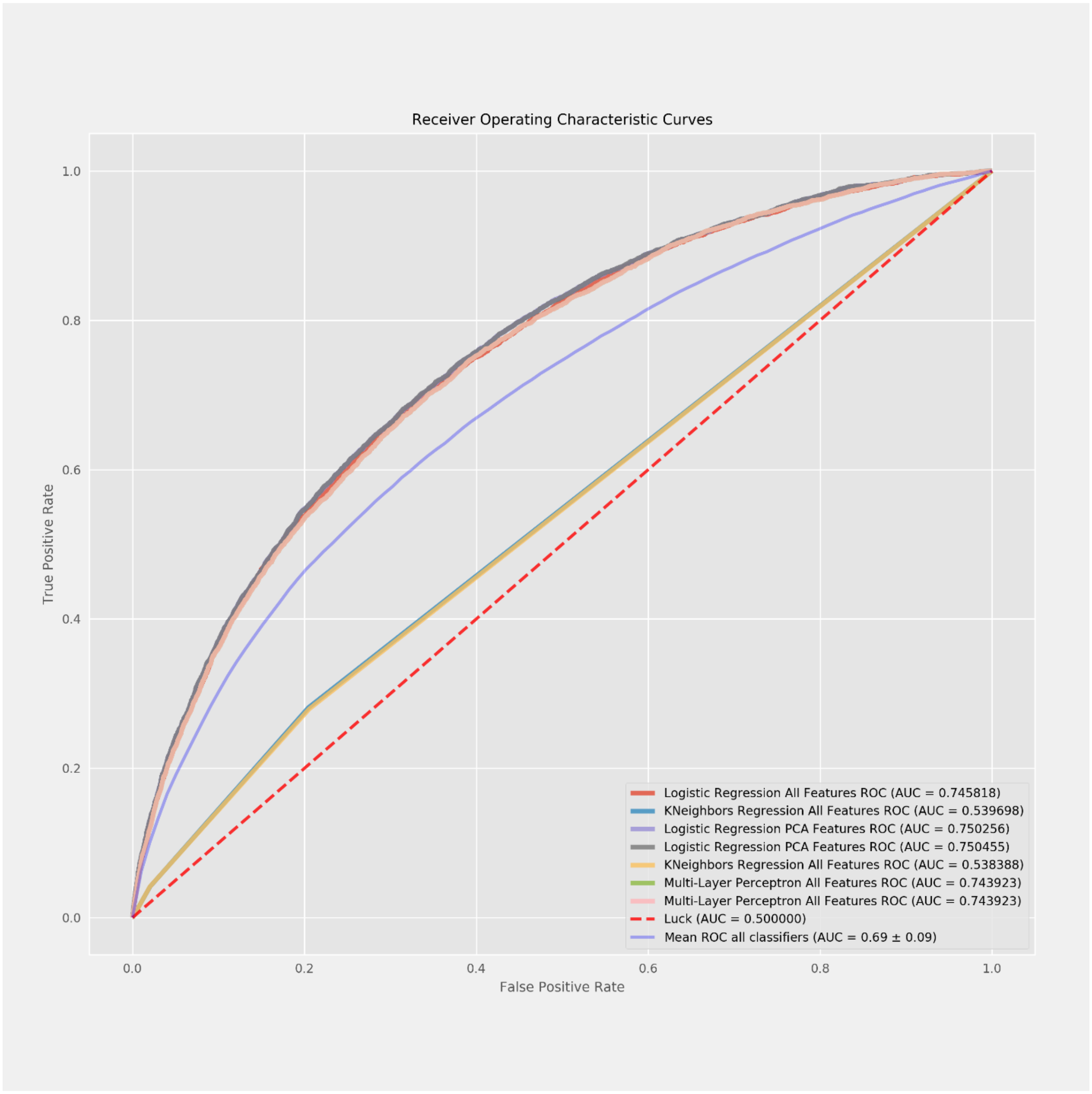


Fig 10. ROC Curves of each classifier and features set.

Looking at the Free-Form visualization we can see each one of our models ROC charts, starting from the red dotted line which is pure 50/50 luck, followed by KNN models which are very near to flipping a coin too, this three models are not robust enough to be use in the real life but prove that our next models are good enough, next ROC is the Mean of all classifiers with a 69% this tell us that just using the simple mean of the data it can give us good results, proving that a human analysis of the data can be 69%

accurate which is what Home Credit associates might be doing at the moment they place credits but with our Logistic Regression model either with All Features or PCA features we see a very good improvement up to 75% which means that our model it will be able to correctly predict 3 of every 4 applicants will be able to pay the loan fully and on time.

Continuing from our Robustness and Trust section our model does very well considering that top worldwide models in kaggle perform up to 80% of accuracy score.

All of our visualization show the progress and process we did to get to this final ROC Free-Form chart the idea is to clearly see all our results in a single picture which is accomplished with this graph.

Reflection

My reflection of the final results and the ROC charts where that simple models or classifiers can do a better job in this type of problems specifically when they are binary outputs, looking at our benchmark of the Multi-Layered Perceptrons Classifier and the results of the Logistic Regression Classifier this proves that simple in better for this type of problems and if we compare with the KNN model the performance was pretty bad in this case which is again a more complex solution then Logistic Regression.

The end to end solution to this problem consisted in identifying the types of data or columns, then separate them in Numeric and Non-Numeric followed by the identification of which of the columns was binary either numeric or not, once this was done we was able to identify pieces of data that where skewed or out of proportion and we implemented different techniques to help with this as Log Transform those columns or features with values that where way out of proportion, second we fill the empty or NaN values using the Sklearn Impute class by calculating the mean of the feature and filling the empty cells with the mean values, after this we One-Hot encoded the Categorical columns expanding the total number of features by each feature category and dropping the old our source features converting all the categorical features in to numeric ones, once our data was ready we added a couple of Engineered features and move to preprocess and validate our resulting dataset.

Our firsts test where very good and encouraging based on our Multi-Layered Perceptron Classified benchmark we was able to tell if the Logistic Regression or KNN models where giving us better accuracy and results, once we confirm this we was able to move in to the refinement section of the project to see if using techniques as PCA to reduce the dimensionality of our project it could help to improve the processing performance of our models without impacting the outcome which it was true because our ROC curve it overlap pretty well with the original Logistic Regression curve and exceeded the performance of the MLP Classifier benchmark as well it was higher then the Mean ROC for all classifiers.

Improvement

The improvement here was far from competitive from kaggle kernels but I was able improve my own initial evaluation, I was able to use the tools I consider where the right ones for this type of project and showed that there was certain level of improvement during the training and testing of the data, what surprised me the most was my intuition to use the Logistic Regression model and how well this worked, I’m sure there are other algorithms out there that I can add and test now that I have the preprocessing ready to be executed, but I wanted to stick with the initial proposal of only use two models.

Finally thanks to the GridSearch technique I was able to tell that I was using the best possible parameters for my classifiers, even if where not the best classifiers I can tell for sure I was using the best parameters for the Logistic Regression and KNeighbors algorithms, this part was the one that took the longest because doing a GridSearch is not a compute cheap process and it took a total of 10-15 hours spread in different runs and after several mistakes, I think next step is to learn to use the cloud resources in a better way and avoid this long waiting and waste of times.

After using the refinement PCA technique and the GridSearch we was able to improve our model from the Multi-Layered Perceptron’s Classifier

References

[1] S. Narkhede, "Understanding Logistic Regression," [Online]. Available: https://towardsdatascience.com/understanding-logistic-regression-9b02c2aec102.

[2] A. Bronshtein, "A Quick Introduction to K-Nearest Neighbors Algorithm," [Online]. Available: https://medium.com/@adi.bronshtein/a-quick-introduction-to-k-nearest-neighbors-algorithm-62214cea29c7.

[3] J. Dellinger, "github.com," [Online]. Available: https://github.com/jamesdellinger/machine_learning_nanodegree_capstone_project/blob/master/report.pdf.

[4] J. Weed, "github," [Online]. Available: https://github.com/udacity/machine-learning/blob/master/projects/capstone/report-example-1.pdf.

[5] Scikit-Learn, "machine_learning_map," 2017. [Online]. Available: http://scikit-learn.org/stable/tutorial/machine_learning_map/index.html.

[6] H. Credit, "home credit default risk," September 2018. [Online]. Available: https://www.kaggle.com/c/home-credit-default-risk.

[7] J. Brownlee, "https://machinelearningmastery.com," [Online]. Available: https://machinelearningmastery.com/how-to-transform-data-to-fit-the-normal-distribution/.

[8] J. Brownleed, "machinelearningmastery.com," [Online]. Available: https://machinelearningmastery.com/handle-missing-data-python/.

[9] C. Moffitt, "pandas_transform," [Online]. Available: http://pbpython.com/pandas_transform.html.

[10] J. Brownleed, "feature-selection-machine-learning-python," [Online]. Available: https://machinelearningmastery.com/feature-selection-machine-learning-python/.

[11] J. Brownlee, "how-to-one-hot-encode-sequence-data-in-python," [Online]. Available: https://machinelearningmastery.com/how-to-one-hot-encode-sequence-data-in-python/.

[12] T. Lee, "a-home-for-pandas-and-sklearn-beginner-how-tos," [Online]. Available: https://www.kaggle.com/timolee/a-home-for-pandas-and-sklearn-beginner-how-tos.

[13] J. VanderPlas, "PythonDataScienceHandbook/05.04-feature-engineering," [Online]. Available: https://jakevdp.github.io/PythonDataScienceHandbook/05.04-feature-engineering.html.

[14] S. Narkhede, "understanding-auc-roc-curve," [Online]. Available: https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5.