Photo by Michael Longmire on Unsplash

Credit Risk Analysis with Machine Learning

Predicting the risk of client default using XGBoost, LightGBM and CatBoost

Rafael Bastos Oct 14, 2020 · 7 min read

Note from Towards Data Science's editors: *While we allow independent authors to publish articles in accordance with our rules and guidelines, we do not endorse each author's contribution. You should not rely on an author's works without seeking professional advice. See our Reader Terms for details.*

Minimizing the risk of default is a major concern for financial institutions. For this reason, commercial and investment banks, venture capital funds, asset management companies and insurance firms, to name a few, are increasingly relying on technology to predict which clients are more prone to stop honoring their debts.

Machine Learning models have been helping these companies to improve the accuracy of their credit risk analysis, providing a scientific method to identify potential debtors in advance.

In this article, we will build a model to predict the risk of client default for Nubank, a prominent Brazilian Fintech.

About the Data

Nubank is a Brazilian digital bank and one of the largest Fintechs in Latin America. It is known to be a data-driven company, taking advantage of technology to make decisions and improve services.

The data set can be downloaded here. Some private information was hashed to keep the data anonymous.

Data Analysis

We are working with a data set containing 43 features for 45,000 clients. `target_default` is a True/False feature and is the target variable we are trying to predict. After exploring the data set, we found that some features had outliers and missing values. Other variables, that would add no value to the model, were removed (you can check the code and thorough explanation here).

The following histogram helps us visualize the distribution of the numerical features:

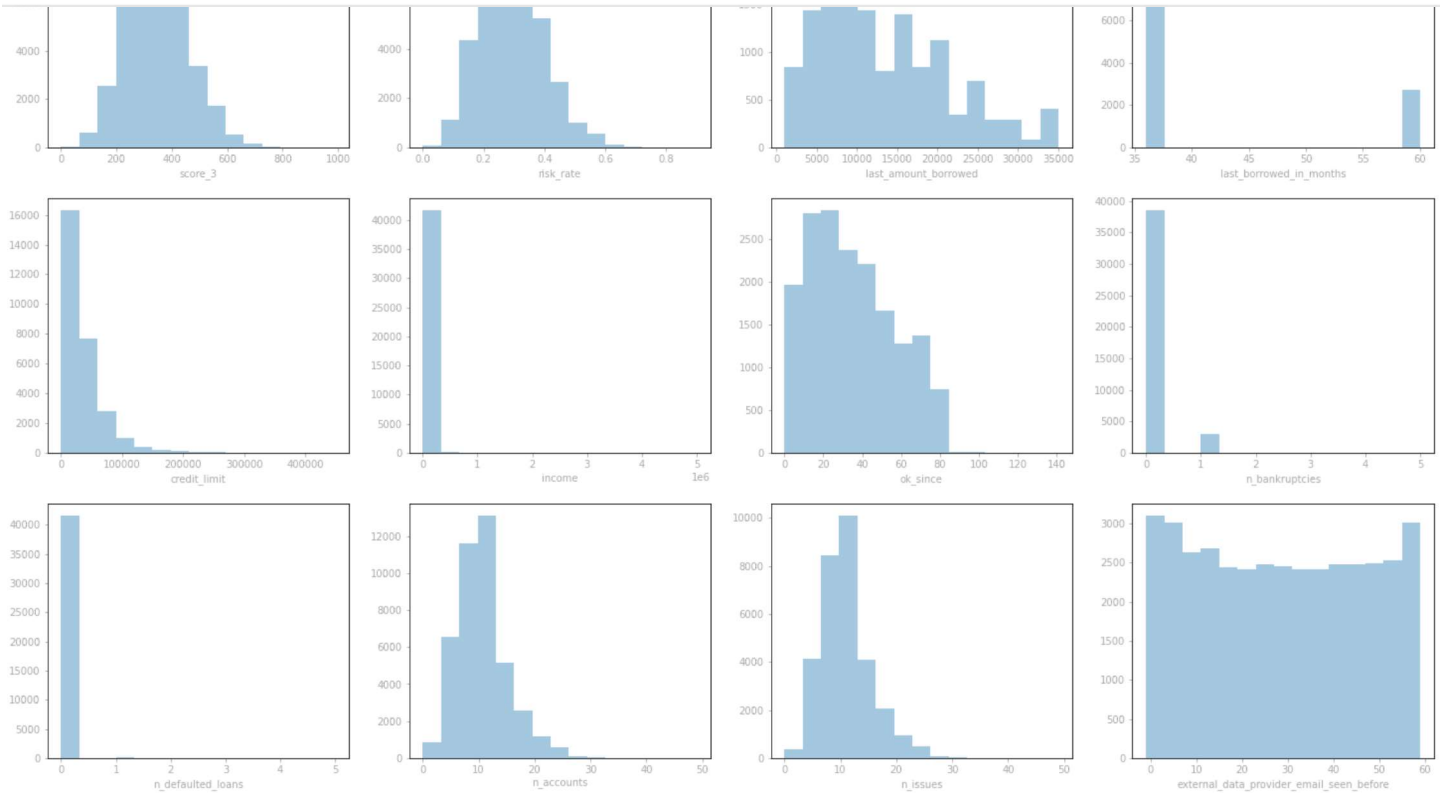

Fig 1. Histogram of the numerical features

The features above have missing values that need to be treated. As we can see, they have skewed distribution, which is an indication that we should fill the missing values with the median value for each feature.

It's time to deal with the missing values from the remaining columns. We are filling these values according to the particularities of each feature, as below:

- Categorical variables will be filled with the most recurrent value.
- Numerical variables will be filled with their median values.
- In some specific cases, we'll fill the missing values with zero, as it is reasonable to believe that not every client would have values assigned to these variables.

After cleaning the data set and handling the missing values, we are now working with the following features:

```
target_default
score_1
score_2
score_3
score_4
```

```
last_borrowed_in_months
credit_limit
income
facebook_profile
state
real_state
ok_since
n_bankruptcies
n_defaulted_loans
n_accounts
n_issues
application_time_in_funnel
external_data_provider_credit_checks_last_month
external_data_provider_credit_checks_last_year
external_data_provider_email_seen_before
external_data_provider_fraud_score
reported_income
shipping_state
```

Fig 2. List of features

Before setting up the machine learning algorithms, we need to perform some preprocessing. Considering that most Machine Learning algorithms work better with numerical inputs, we'll preprocess our data using Scikit Learn's `LabelEncoder` for the binary variables and pandas' `get_dummies` for the other categorical variables.

Machine Learning Models

We are experimenting with the following 3 gradient boosting algorithms to determine which one yields better results:

- XGBoost
- LightGBM
- CatBoost

Before setting up the models, we need to split the data into training and test sets. After that, as we are dealing with an unbalanced data set, we'll standardize and resample the training set, with `StandardScaler` and `RandomUnderSampler`, respectively.

Evaluation Metrics

With regards to the evaluation of the models, it's worth mentioning that we should consider `Precision`, `Recall` and `F1 Score` as evaluation metrics, for the following

- **Precision** will give us the proportion of positive identifications that were indeed correct. It can be defined as:

$$Precision = \frac{TruePositives}{TruePositives + FalsePositives}$$

- **Recall** will determine the proportion of real positives that were correctly identified, and it can be defined as:

$$Recall = \frac{TruePositives}{TruePositives + FalseNegatives}$$

- **F1 Score** is a metric that is useful when we need to seek a balance between precision and recall. The formula is defined as:

$$F_1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

Since our objective is to minimize company loss, predicting the risk of client default, a good recall rate is desirable because we want to identify the maximum amount of clients that are indeed prone to stop paying their debts, thus, we are pursuing a small number of *False Negatives*.

Additionally, we also seek to minimize the number of False Positives because we don't want clients to be mistakenly identified as defaulters. Therefore, a good precision rate is also desirable.

However, there is always a tradeoff between precision and recall. For this article, we chose to give more emphasis to recall, using it as our evaluation metric.

We're also using Cross-Validation to get better results. Instead of simply splitting the data into a train and test set, the `cross_validate` method splits our training data into k number of Folds, making better use of the data. In our case, we'll perform 5-fold cross-validation, as we let the default k value.

	Recall
XGBClassifier	0.662726

Fig 3. Recall values

Notice that all three models yielded similar results. We'll now tune some hyperparameters on the models to see if we can achieve higher score values. The method utilized here is `GridSearchCV`, which will search over specified parameter values for each estimator.

The hyperparameter tuning for each model are the following:

XGBoost

For the XGBoost model, we'll tune the following hyperparameters, according to the official documentation:

- `n_estimators` - The number of trees in the model
- `max_depth` - Maximum depth of a tree
- `min_child_weight` - Minimum sum of instance weight needed in a child
- `gamma` - Minimum loss reduction required to make a further partition on a leaf node of the tree
- `learning_rate` - Step size shrinkage used in the update to prevents overfitting

```
param_grid = {'n_estimators': range(0,1000,50),
              'max_depth': [1, 3, 5],
              'min_child_weight': [1, 3, 6],
              'gamma': [0, 1, 5],
              'learning_rate': [0.0001, 0.001, 0.01, 0.1]}
```

Best recall rate: 0.81 for {'n_estimators': 50, 'max_depth': 3, 'min_child_weight': 6, 'gamma': 1, 'learning_rate': 0.0001*}*

LightGBM

Now, turning to the LightGBM model, another tree-based learning algorithm, we are going to tune the following hyperparameters, referring to the documentation:

- `learning_rate` - Shrinkage rate
- `num_leaves` - Max number of leaves in one tree
- `min_data_in_leaf` - Minimal number of data in one leaf

```
param_grid = {'max_depth': np.arange(5, 75, 10),
              'learning_rate' : [0.001, 0.01, 0.1],
              'num_leaves': np.arange(20, 220, 50),
              'min_data_in_leaf': np.arange(100, 1000, 100)}
```

Best recall rate: 0.69 for {'learning_rate': 0.01, 'max_depth': 5, 'num_leaves': 70, 'min_data_in_leaf': 400*}*

CatBoost

Lastly, we're going to search over hyperparameter values for CatBoost, our third gradient boosting algorithm. The following hyperparameters will be tuned, according to the documentation:

- `depth` - Depth of the tree
- `learning_rate` - As we already know, the learning rate
- `l2_leaf_reg` - Coefficient at the L2 regularization term of the cost function

```
param_grid = {'depth': [6, 8, 10],
              'learning_rate': [0.03, 0.1],
              'l2_leaf_reg': [1, 5, 10]}
```

Best recall rate: 0.65 for {'depth': 6, 'l2_leaf_reg': 5, 'learning_rate': 0.03*}*

Evaluating the models on the test set

After tuning the hyperparameters, all three models displayed betters results. It is worth mentioning that XGBoost presented a great score increase, while LightGBM and CatBoost saw a meager improvement.

XGBoost

	precision	recall	f1-score	support
0	0.92	0.44	0.60	8771
1	0.22	0.81	0.34	1665
accuracy			0.50	10436
macro avg	0.57	0.63	0.47	10436
weighted avg	0.81	0.50	0.56	10436

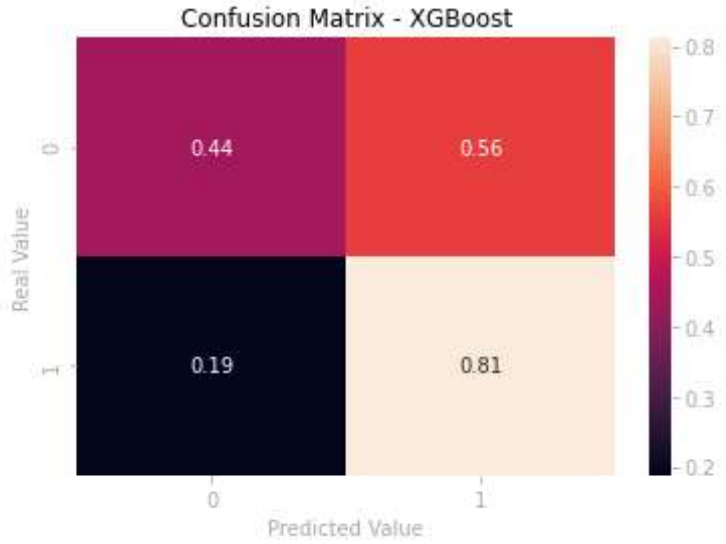

Fig 4. XGBoost Confusion Matrix

LightGBM

	precision	recall	f1-score	support
0	0.91	0.62	0.74	8771
1	0.25	0.68	0.37	1665
accuracy			0.63	10436
macro avg	0.58	0.65	0.55	10436
weighted avg	0.81	0.63	0.68	10436

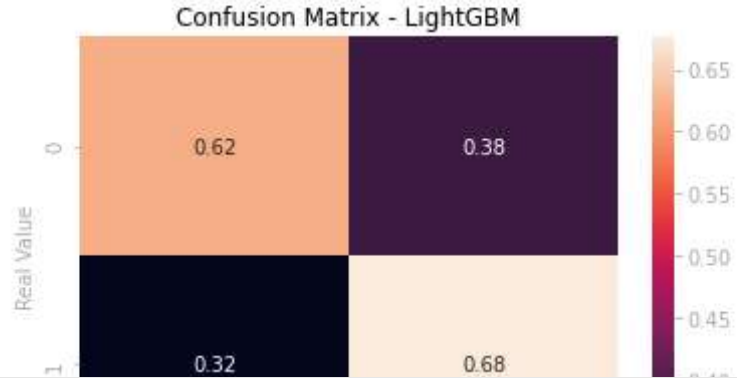

Fig 5. LightGBM Confusion Matrix

CatBoost

	precision	recall	f1-score	support
0	0.91	0.67	0.77	8771
1	0.27	0.64	0.38	1665
accuracy			0.66	10436
macro avg	0.59	0.65	0.57	10436
weighted avg	0.81	0.66	0.71	10436

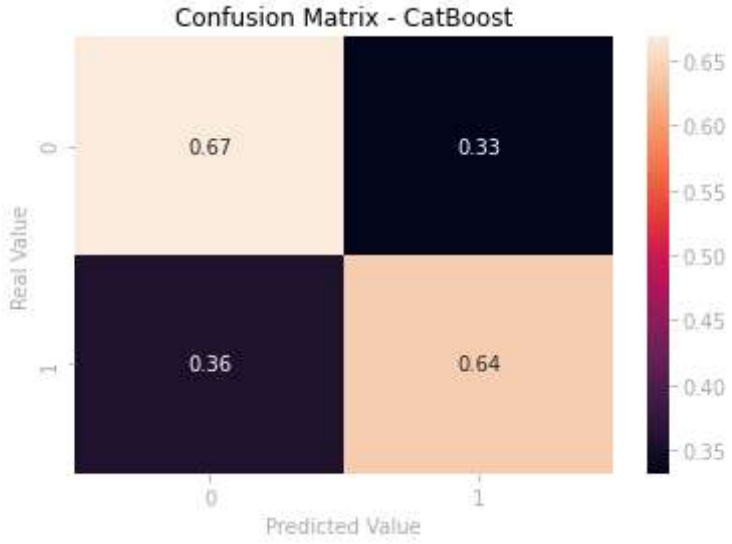

Fig 6. CatBoost Confusion Matrix

Conclusion

The main objective of this article was to build machine learning algorithms that would be able to identify potential defaulters and therefore reduce company loss. The best model possible would be the one that could minimize false negatives, identifying all defaulters among the client base, while also minimizing false positives, preventing clients to be wrongly classified as defaulters.

Meeting these requirements can be quite tricky as there is a tradeoff between precision and recall, meaning that increasing the value of one of these metrics often decreases the value of the other. Considering the importance of minimizing company loss, we decided to give more emphasis on reducing false positives, searching for the best hyperparameters that could increase the recall rate.

positives. On the other hand, **LightGBM** and **CatBoost** delivered a better count of false positives, with 38% and 33% respectively, but their false negatives were substantially higher than that of XGBoost, resulting in a weaker recall rate.

This article presents a classic evaluation metrics dilemma. In this case, it would be up to the company's decision-makers to analyze the big picture, with the aid of the machine learning algorithms, and decide the best plan to follow. Of course, in a future article, we can test a different approach to achieve a more desirable result, such as taking advantage of deep learning models.

For the full code, please refer to the notebook.