ViaBill

Risk factors discovery and predictive modelling

report by Daniel Szponar, Warsaw 4.2021

Table of Contents

1. Business perspective

1.1. Business case

ViaBill allows customers to purchase an item and pay for it in 4 equal instalments over the next 4 weeks. Most customers will pay these required instalments on time. However, some customers will pay them late and some customers will dishonour their agreement (default) and not pay all the instalments at all.

1.2. Value added

As ViaBill company, I would like to know the risks of client's late payment or lack of payment at the time of lending the money. Having this information, I could apply additional business logic that will transfer the predicted risk to the client by increasing the price of the service or applying additional fees.

1.3. Task description

Goals:

- identify factors which may be used to predict which customers will pay late,
- identify factors which may be used to predict which customers will not pay an instalment at all,
- build predictive model that will predict if client pays back the loan or not - will default (for simplicity I will build only one model – predicting default event for customers with at least one historic transaction – existing customer).

The **default** is defined as a loan that has a "never paid" status on, at least one of four instalments:

- PaymentStatus1 = 2 (never paid),
- PaymentStatus2 = 2 (never paid),
- PaymentStatus3 = 2 (never paid),
- PaymentStatus4 = 2 (never paid),

The **late** payment is defined as a loan that has a "paid but late" status, on at least one of four instalments:

- PaymentStatus1 = 1 (paid but late),
- PaymentStatus2 = 1 (paid but late),
- PaymentStatus3 = 1 (paid but late),
- PaymentStatus4 = 1 (paid but late),

2. Data overview

Data available for the task are as follows:

- **Customers.csv**

  Table with 500k rows and six columns.

CustomerID	Integer	1..500000
Sex:	Integer	1 (male), 2 (female) 0 (other)
Age (years):	Integer	1..100
Residential Address:	String	free text
Postal Address	String	free text
Income (dollars per year):	Integer	100..100000

- **Transactions.csv**

  Table with 1 million rows and 8 columns.

Transaction ID	Integer	1..2000000
ShopID	Integer	100.999
CustomerID2	Integer	1..500000
Price (dollars)	Integer	3..200
PaymentStatus1	Integer	0 paid on time 1 paid but late 2 never paid
PaymentStatus2	Integer	
PaymentStatus3	Integer	
PaymentStatus4	Integer	

  Comments:
  - There is no timestamp on the transaction level, which makes it impossible to analyse data in context of time.

2.1. Exploratory data analysis EDA

Notebook: *01_EDA.ipynb*

In this section I will provide general overview of the data before feature engineering.

Customer table describes client features, like sex, age, address (residential and postal) and income. It is connected to **Transaction** table with relation one to many – one customer has at least one relating transaction.

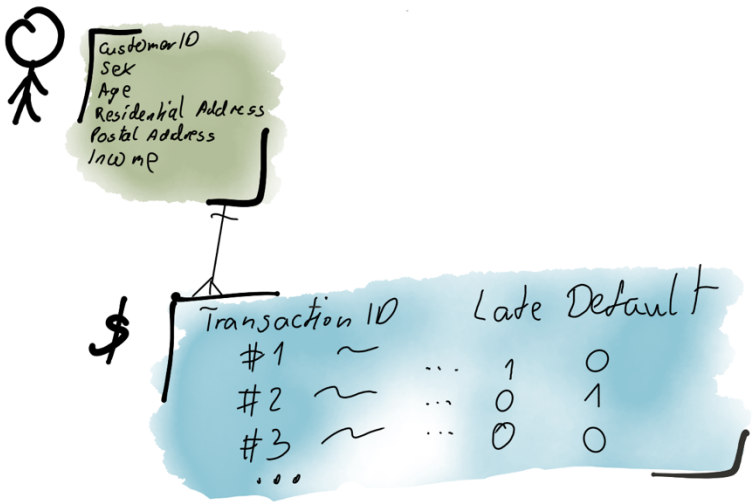

Figure 1 tables schema

2.1.1. Customers

I've created 3 additional features on Customers table:

CustomerID	
Sex:	1 (male), 2 (female) 0 (other)
Age (years):	
Residential Address:	
Postal Address	
Income (dollars per year):	
New features:	
residentialAddress_clean	Residential Address without numbers
postalAddress_clean	Residential Address without numbers
same_address	Binary feature that indicates if residentialAddres and postalAddress is the same (1) or is different (0)

Customers table preview:

	customerID	sex	age	residentialAddress	postalAddress	income	residentialAddress_clean	postalAddress_clean	same_address
0	1	male	25	28 Irvine Place	28 Irvine Place	NaN	Irvine Place	Irvine Place	1
1	2	male	19	72 Bertha Street	72 Bertha Street	43200.0	Bertha Street	Bertha Street	1
2	3	female	22	63 Ladberry Street	63 Ladberry Street	70200.0	Ladberry Street	Ladberry Street	1
3	4	other	24	98 Linneman Close	98 Linneman Close	93900.0	Linneman Close	Linneman Close	1
4	5	male	53	56 Salonica Road	56 Salonica Road	77000.0	Salonica Road	Salonica Road	1

Figure 2 Customer table preview

	count	mean	std	min	25%	50%	75%	max
customerID	500000.000000	250000.50	144337.711635	1.000000	125000.750000	250000.500000	375000.250000	500000.000000
age	500000.000000	44.63	16.178283	5.000000	27.000000	50.000000	58.000000	90.000000
income	490025.000000	50000.04	28899.119696	0.000000	25000.000000	50000.000000	75000.000000	100000.000000
same_address	500000.000000	0.93	0.254364	0.000000	1.000000	1.000000	1.000000	1.000000

Figure 3 Customer table statistics

Distribution of the age and income can be seen below:

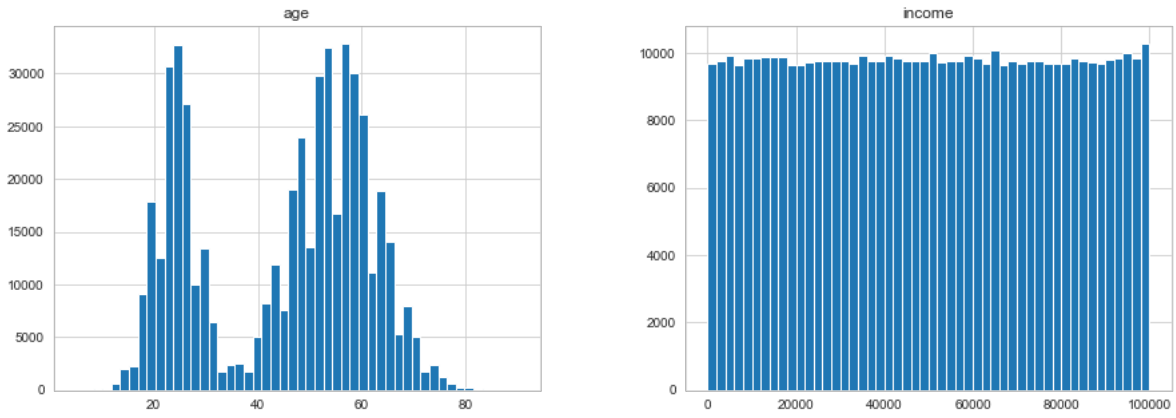

Figure 4 Age and income distribution

We can see that there are 2 age groups that are using the product:

- young, around 25 years old,
- middle-aged, around 55 years old,

There is no dependency between age and sex in the data (chi2 test p-value 0.399).

Income seems to be uniformly distributed, but there might be a dependency between income and sex (chi2 test p-value 0.032.) There are slightly more women that are in the highest pay group in comparison to other genders.

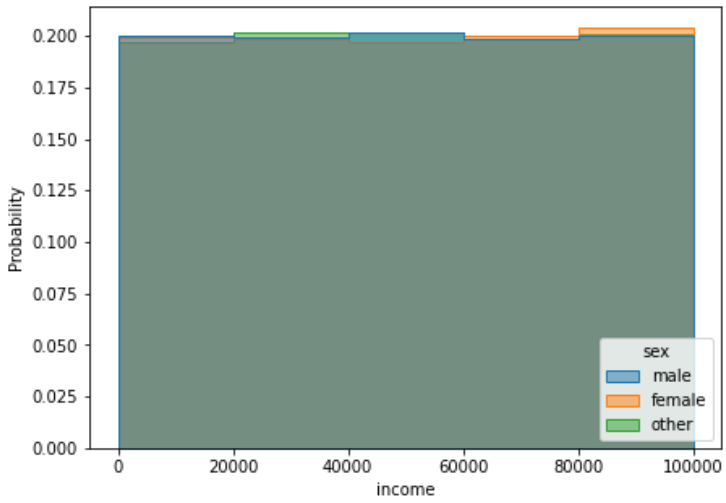

Figure 5 Income vs gender

	customerID	transaction_count	price_avg	money_lost_avg	late_avg	default_avg
count	490533.000000	490533.000000	490533.000000	490533.000000	490533.000000	490533.000000
mean	249973.179978	4.036391	101.538044	4.685877	0.294494	0.091845
std	144332.983398	1.927140	32.957038	12.189746	0.294082	0.185714
min	1.000000	1.000000	3.000000	0.000000	0.000000	0.000000
25%	124989.000000	3.000000	80.600000	0.000000	0.000000	0.000000
50%	249977.000000	4.000000	101.500000	0.000000	0.250000	0.000000
75%	374973.000000	5.000000	122.500000	2.000000	0.500000	0.142857
max	500000.000000	16.000000	200.000000	200.000000	1.000000	1.000000

Figure 6 Customer transactions statistics

We can learn from the statistics that:

- on average single customer performs 4 transactions,
- default rate is 9% - 1 in 10 of transaction will be not paid back (at least 1 instalment),
- average lost on transaction is $4,7 with transaction price equals $101, which converts to 5% loss on transaction,
- 1 in 3 clients will not pay at least one instalment on time.

There is a clear dependency between sex and default as well as probability of being late with payments.

sex	default_rate
female	0.139903
male	0.075917
other	0.140126

Figure 7 Default rate vs sex

sex	late_rate
female	0.228787
male	0.316318
other	0.224040

Figure 8 Late payment rate vs sex

Females and other are twice likely to default then man (chi2 test, p-value 0.000).
Males are 50% more likely to be late with payments (chi2 test, p-value 0.000).

2.1.2. Transactions

I've created 6 additional features in Transaction table:

Transaction ID	1..2000000
ShopID	100.999
CustomerID2	1..500000
Price (dollars)	3..200
PaymentStatus1	0 paid on time 1 paid but late 2 never paid
PaymentStatus2	
PaymentStatus3	
PaymentStatus4	
New features:	
late	An indicator (1) if client hasn't paid at least one instalment on time
default	An indicator (1) if client has default at any instalment
defualted_payment	Which instalment defaulted first
late_payment_first	Which instalment was paid late
money_lost	How much money was lost on transaction, calculated as following: `if defualted_payment > 0 then price * (5-defualted_payment)`

Data distribution overview:

	count	mean	std	min	25%	50%	75%	max
transactionID	2000000.000000	1000000.50	577350.413527	1.000000	500000.750000	1000000.500000	1500000.250000	2000000.000000
shopID	2000000.000000	549.25	259.847443	100.000000	324.000000	549.000000	774.000000	999.000000
customerID	1979983.000000	250095.15	144288.944851	1.000000	125139.000000	250096.000000	375093.000000	500000.000000
price	2000000.000000	101.53	57.175753	3.000000	52.000000	102.000000	151.000000	200.000000
paymentStatus1	2000000.000000	0.12	0.360735	0.000000	0.000000	0.000000	0.000000	2.000000
paymentStatus2	2000000.000000	0.15	0.427886	0.000000	0.000000	0.000000	0.000000	2.000000
paymentStatus3	2000000.000000	0.19	0.505364	0.000000	0.000000	0.000000	0.000000	2.000000
paymentStatus4	2000000.000000	0.27	0.616858	0.000000	0.000000	0.000000	0.000000	2.000000
late	2000000.000000	0.29	0.455622	0.000000	0.000000	0.000000	1.000000	1.000000
default	2000000.000000	0.09	0.289078	0.000000	0.000000	0.000000	0.000000	1.000000
defualted_payment	2000000.000000	0.28	0.926016	0.000000	0.000000	0.000000	0.000000	4.000000
late_payment_first	2000000.000000	0.90	1.360083	0.000000	0.000000	0.000000	2.000000	4.000000
money_lost	2000000.000000	4.70	19.657777	0.000000	0.000000	0.000000	0.000000	200.000000

Figure 9 Transactions table statistics

Highlights:

- max price is $200 while average price is $102,
- average loss on transaction is $5,

There are 1% of transactions that don't relate to any Customer. After performing some investigation and being sure that there isn't any pattern behind it, I deleted these observations.

```
Nulls percentage:

transactionID          0.00%
shopID                 0.00%
customerID             1.00%
price                  0.00%
paymentStatus1         0.00%
paymentStatus2         0.00%
paymentStatus3         0.00%
paymentStatus4         0.00%
late                   0.00%
default                0.00%
defualted_payment      0.00%
late_payment_first     0.00%
money_lost             0.00%
dtype: object
```

Figure 10 Missing values %

One shop "113" has 100% of its transactions in default. Since **it seemed like a fraud,** I decided to remove it for the sake of higher quality of further analysis and prediction model.

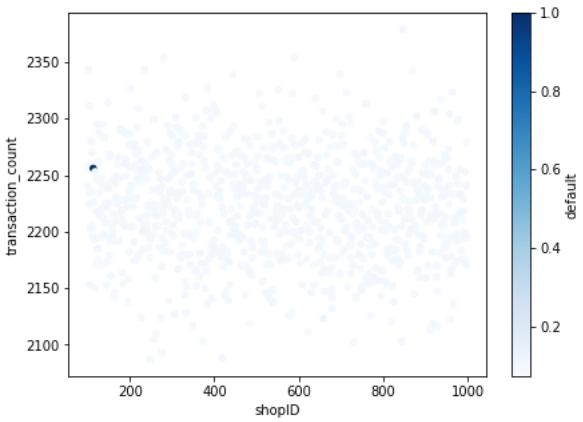

Figure 11 Default rate on each shop (with one fraud)

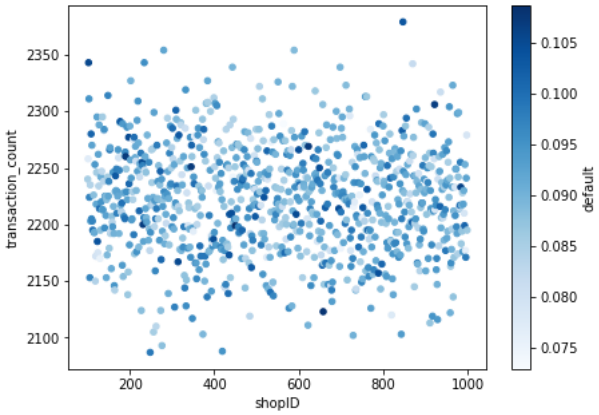

Figure 12 Default rate on shop (without fraud)

2.1.3. Transactions in shops

Graphs below shows distribution of transactions in shops. Shop #113, which was classified as fraud, is excluded.

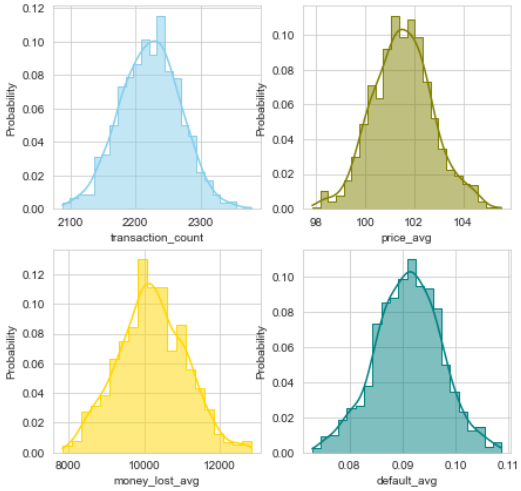

Figure 13 Transaction per shop statistics

Highlights:

- single shop has 2,222 transactions on average (sd 46),
- average item price is 101 (sd 1.3),
- default rate on shop is 9.1% with (sd 0.6 %),
- average loss is $ 10 K.

I've tried to reduce the number of shops using clustering (T-SNE + DBSCAN) but the shops in dataset do not differ between each other.

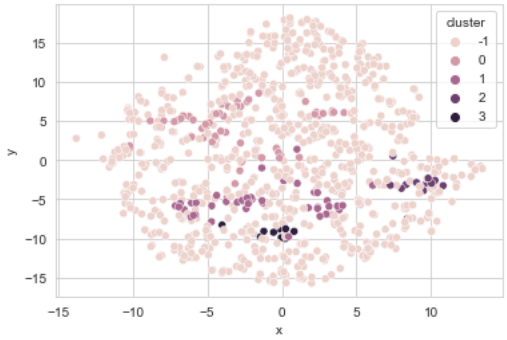

Figure 14 Dimension reduction and clustering on shops

2.1.4. Default and late payment on single transaction

In the next step of my analysis, I wanted to verify it there is any relation between late payment and default on single transaction. In other words - is it true that client who will be late with payment on the first instalment is more likely to not pay back the second instalment?

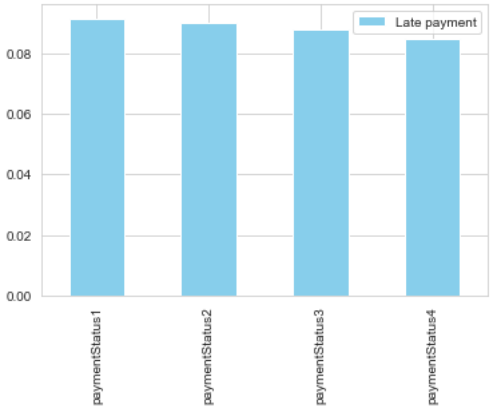

Figure 15 Late payment rate on separate instalments

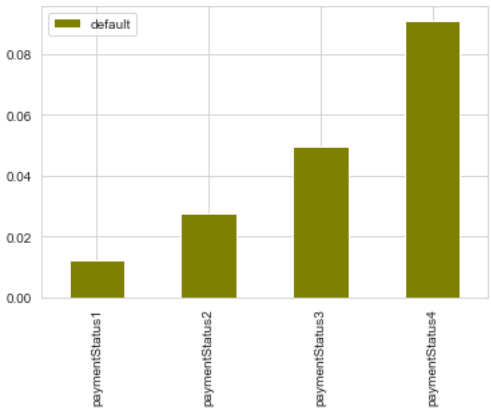

Figure 16 Default rate on each instalments

Tested hypothesis:

- If first late, then next is late - no dependency (chi2 test, p-value 0.38)
- If first late, then next default - no dependency (chi2 test, p-value 0.16)
- If second late, then next late - no dependency (chi2 test, p-value 0.22)
- If second late, then next default - no dependency (chi2 test, p-value 0.86)

In analysed population there seems to be **no** dependency between late payments and default in the context of single transaction.

2.1.5. Geospatial data

One thing that is easy to notice about address data is the following dependency – if postal address isn't the same as residential address, then customer **is 4 times more likely to default.**

Data set contains address fields: residential Address and postal Address and they are of insufficient quality, containing only street name and number. Therefore, I used OpenStreetMap API to geocode the addresses and try to enrich the dataset since geographical data usually gives additional analytical value.

After performing address geocoding on sample data (500), I discovered that around 53% of them are from Australia, Brisbane.

Australia	0.528
United States	0.270
United Kingdom	0.094
Canada	0.040
New Zealand / Aotearoa	0.034
Deutschland	0.012
Nederland	0.008
Polska	0.002
South Africa	0.002
Trinidad and Tobago	0.002
Österreich	0.002
Moldova	0.002
Italia	0.002

Figure 17 The most frequent countries returned by API based on address

Brisbane City	0.496
London	0.036
New York	0.018
Melbourne	0.014
Preston	0.008
	...
Old Toronto	0.002
City of Nottingham	0.002
South Somerset	0.002
Jamestown	0.002
Szczecin	0.002

Figure 18 The most frequent city returned by API based on address

Following this, I've complemented the address, pasting "Brisbane, Australia" to each one, and geocoded again data sample. After I printed out clients coordinates on the plot (I could use a map, but it would be more time consuming), unfortunately there hadn't been any pattern visible. Clustering algorithms didn't reveal anything as well.

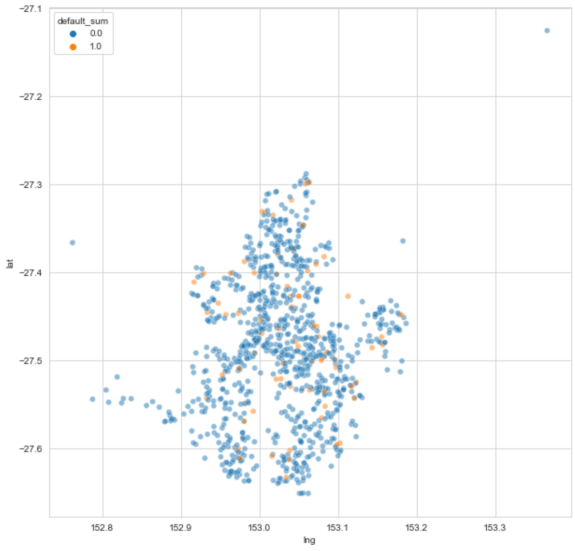

Figure 19 Transaction on "map" - orange dots means defaults

3. Observation and Target

Notebook: *02_Target.ipynb*

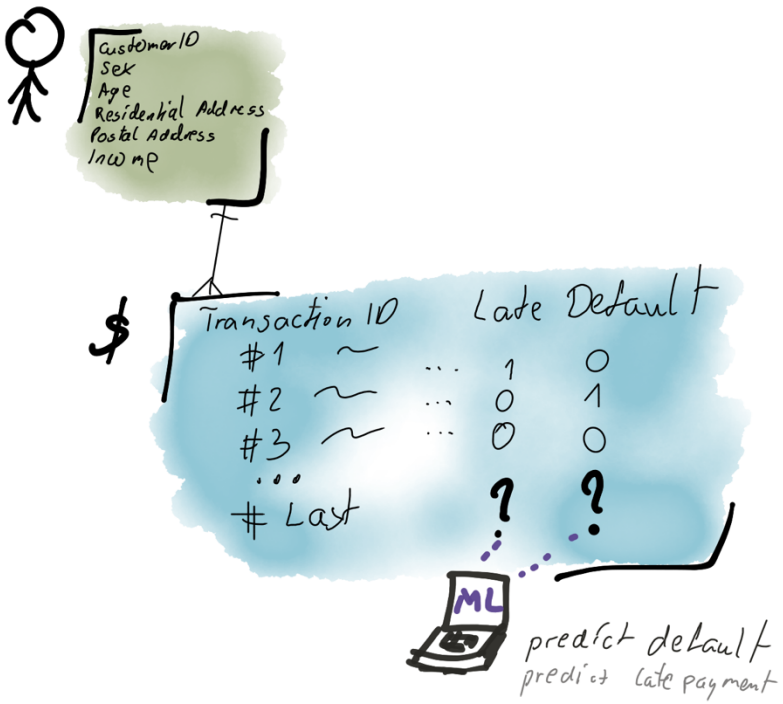

Figure 20 Target (?)

Since I wanted to predict if customer who is applying for a loan will pay back the loan and whether the pay back will happen without delays, I needed to simulate such scenario with available data.

To achieve this, I treated the last available transaction of the client as a credit application, and the previous transactions as my client's history.

It is important to know what the distribution of my target over the time was. Since the data doesn't contain timestamp, I created "pseudo" timestamp based on assumptions that transaction ID is increasing over the time.

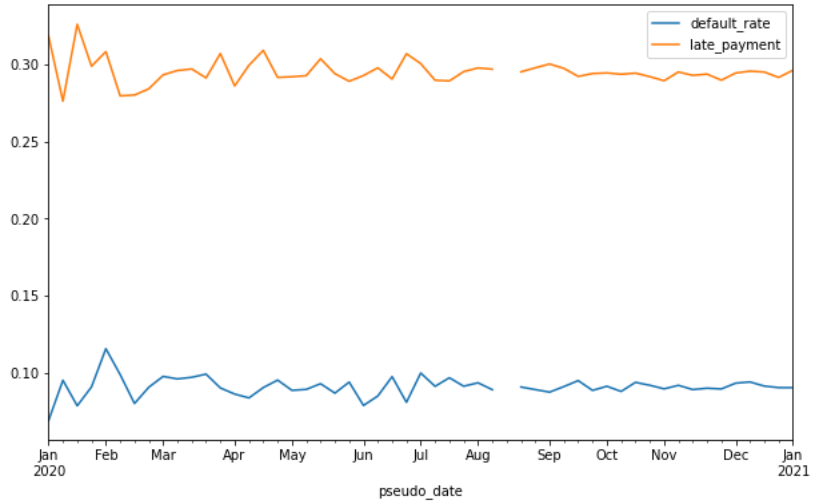

Figure 21 Default rate in context of time

Default rate and late payment rate looked very stable (unnaturally stable) over time.

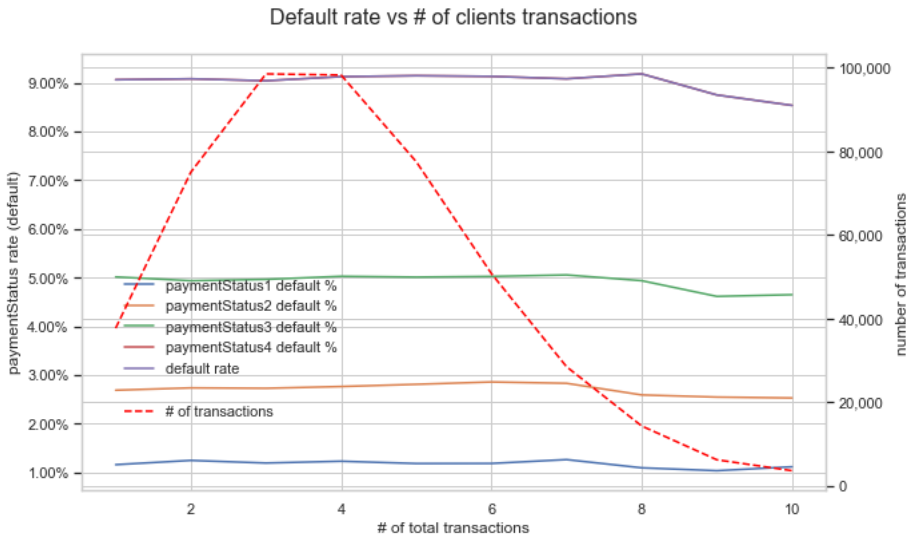

Figure 22 Default rate vs transactions number

Customer lifetime (which here means, how many transactions he or she has performed so far using ViaBill) did not affect the risk of default (which in real life scenario is rarely true).

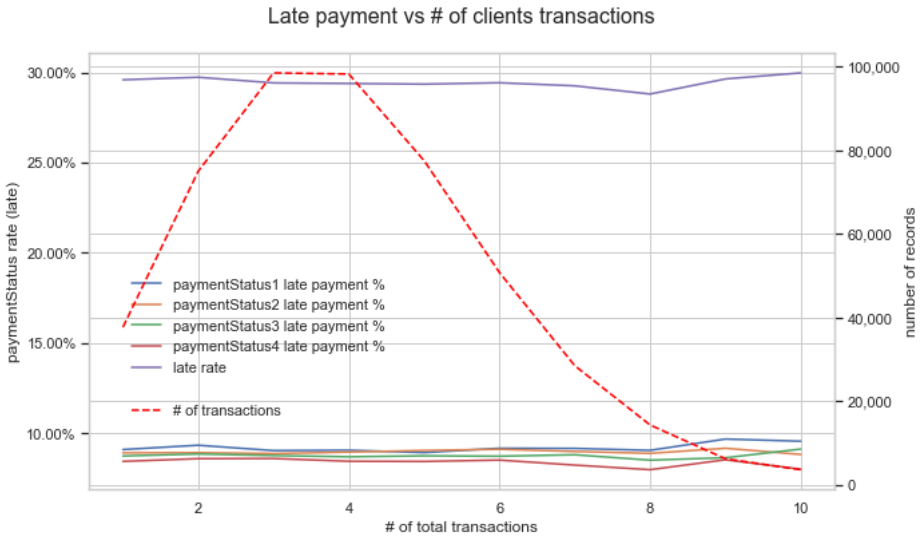

Figure 23 Late payment rate vs number of transactions

Similar pattern appeared with late payments - Clients' history (how many transactions he or she has performed so far using ViaBill) did not affect the risk of late payment.

4. Feature Engineering

Notebook: *03_Feature_engineering.ipynb*

Based on the transaction history I created some new features.

Total history aggregates

Features created based on all available client history – excluding the last transaction which is our 'credit application':

- hist_trans_count – number of transactions, so far,
- hist_default_sum – number of defaults, so far,
- hist_default_avg – average default, so far (default rate on client),
- hist_late_sum – number of late paid transaction, so far,
- hist_late_avg – average late, so far (late rate on client),
- hist_price_sum – sum of money spends, so far,
- hist_price_avg – average item price, so far,

Window aggregates

Features created on defined number of the past transaction (3, 6, 9, 12).
In real life case scenario, it would be some fixed periods of time ex. how many transactions client has perform in the period of past 3, 6, 9 and 12 months. We exclude the last transaction which is our 'credit application'. XX means here 3, 6, 9 or 12 past transactions of client.

- default_lst_XX_sum - number of defaults, in the past X transaction window,
- money_lost_lst_XX_sum – the sum of money that company lost on client, in the past X transaction window,
- late_lst_XX_sum - number of late paid transactions, in the past X transaction window,
- price_lst_X_sum – the sum of item purchased, in the past X transaction window,
- default_lst_XX_avg – default rate on client, in the past X transaction window,
- defualted_payment_lst_XX_avg – which instalment is not being paid on average, in the past X transaction window,
- money_lost_lst_XX_avg – average money lost on single transaction, in the past X transaction window,
- price_lst_XX_avg – average item price, in the past X transaction window,
- late_payment_first_lst_XX_avg –which payment is being late on average, in the past X transaction window,

Other:

- geo_risk_rank – an average default rate on address classified to defined ranges of values, and displayed as rating,

5. Primary feature selection

Notebook: *03_Feature_engineering.ipynb*

In real-life case scenario, at this stage of analysis, I would perform detailed analysis of the following features:

- prediction power,
- stability over time,
- correlations between features,
- missing values,
- outliers detections,

After the analysis, I would get rid of features that are not informative of poor quality.

But since it is time consuming, I decided to skip this part for now and move forward.

6. Segments

Notebook: *04_Segmentation_and_data_split.ipynb*

There are at least two distinct customer groups in the data that deserves to be treated as separated modelling task.

First segment is “existing customer” - he or she has at least one historic transaction (behavioural information’s).

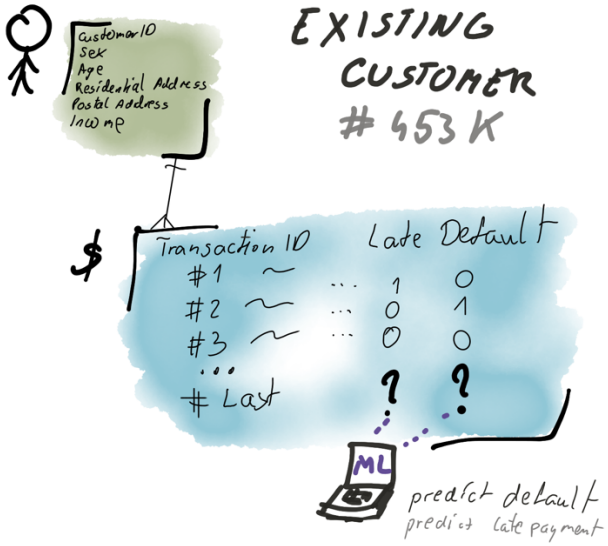

Figure 24 Existing customer schema

The second group is ‘new customer’, with no history at all.

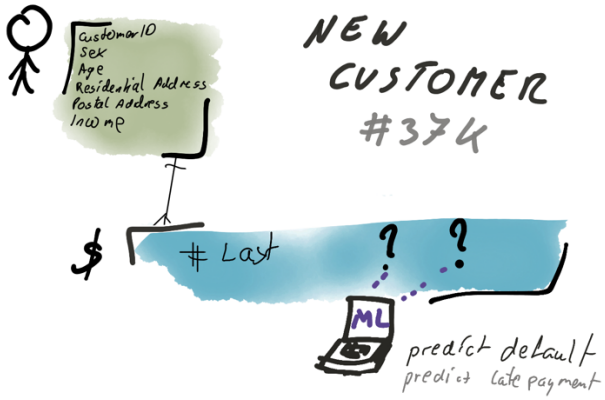

Figure 25 New customer schema

For the simplification of the task, going further I will only focus only on the “Existing Customer” segment.

7. Data split

Notebook: *04_Segmentation_and_data_split.ipynb*

There are around 450K of observations in the segment of “Existing Customer”. I splited the data into:

- **train** - dataset for model training,
- **test** - dataset for model performance assessing (out of sample),
- **valid** - dataset for hp tuning (out of sample),

In cases when there weren't so plenty of data, df_valid could be excluded from training data set at the stage of hyperparameter tuning and put back before final training.

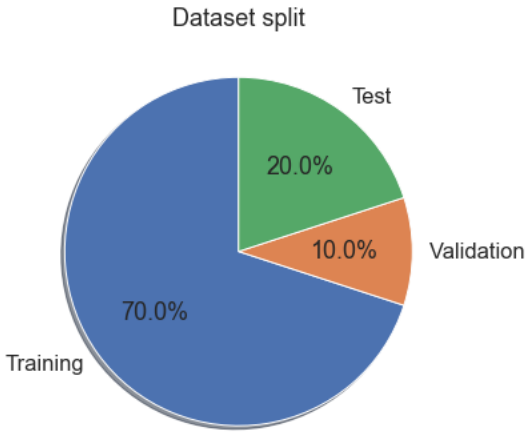

Figure 26 Data split for modelling

8. One-dimension feature analysis

Notebook: *05_One_dim_analysis.ipynb*

I performed analysis of most important features and calculated Information Value (IV).

Information Value:

$$IV = \sum(DistributionGood_i - DistributionBad_i) \times ln(\frac{DistributionGood_i}{DistributionBad_i})$$

IV allows to assess the Predictive Power of the features:

- >0.5 - Suspiciously too good,
- 0.3 to 0.5 - Strong predictor
- 0.1 to 0.3 - Medium predictor
- 0.02 to 0.1 - Weak predictor
- <0.02 – useless for predictions

Weight of Evidence (WOE):

$$Weight of Evidence = ln(\frac{DistributionGood_i}{DistributionBad_i})$$

WOE allows to assess predictive power of each category in Variable, but here I didn't use it.

8.1.1. Category features

Results for categorical features:

- `sex` default IV 0.098 [**Medium** predictor], late payment IV 0.037 [**Weak** predictor],
- `same address`: default IV 0.32 [**Strong**], late payment IV 0.001 [useless],

8.1.2. Numeric features

Since one cannot calculate WOE and IV for continuous values, it had to be discretized first. I didn't manage to find any discretizer that would have meet my needs so I've prepared my own implementation of discretizer "**DecisionTreeDiscretizer_DF**" that:

- is based on DecisionTreeClassifier,
- is fast because uses vectorized computation (numpy arrays),
- is written as a Transformer - ready to work with scikit-learn Pipelines,
- could transform many features in one run,
- allows to provide "manual" cut points for each transformed feature (in dictionary),

The most predictive numerical features for default:

- `default_lst_06_avg` - 0.234 [**Medium** predictor],
- `defualted_payment_lst_06_avg` - 0.225 [**Medium** predictor],

- `money_lost_lst_06_avg` - 0.212 [**Medium** predictor],

The most predictive numerical features for late payments:

- `late_lst_06_avg` - 0.163 [**Medium** predictor],
- `late_lst_06_sum` - 0.15 [**Medium** predictor],
- `late_payment_first_lst_03_avg` - 0.212 [**Medium** predictor],

9. Models

Notebook: *06_Models.ipynb*

9.1. Logistic Regression

Logistic regression regularization is often first choice algorithm for classification problem due to the following reasons:

- is fast to calculate,
- its decision could be interpreted in strait forward way,
- is easy to deploy (production),
- is easy to tune – small number of parameters.

On the other hand:

- it is a linear model thus not able to learn nonlinear dependencies,
- it usually performs worse than state of art algorithms like Gradient Bosting,
- it takes a lot of time to create a good quality model,

Logistic Regression is more prone to have higher bias then variance, so is creates lower risk of overfitting.

Model creation process:

1. Transform features:
    a. Discretize numeric features with custom transformer **DecisionTreeDiscretizer_D,**
    b. Passthrough categorical features – with no change,
2. Perform WoE encoding on categorical features,
3. Feature selection using Forward Feature Selection,
4. Logistic Regression Model without regularization,

Forward Feature Selection algorithm shows that optimal number of features here is between 4 and 8. I choose 6.

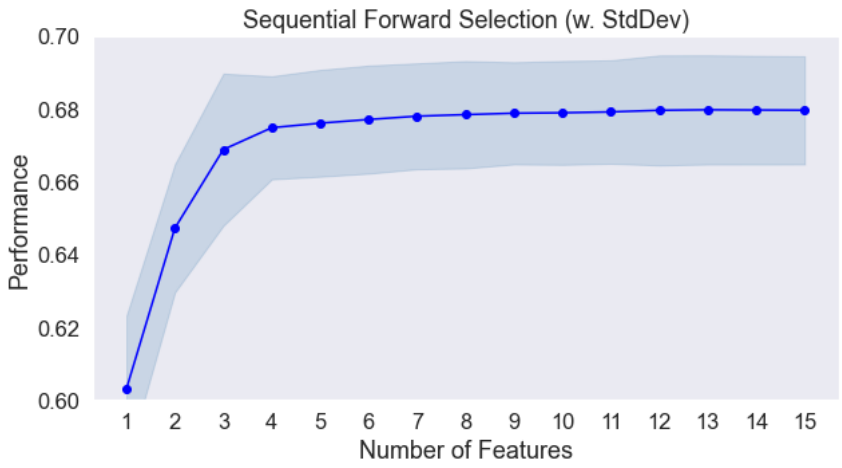

Figure 27 Sequential Forward Selection results (AUC)

The correlation between features shouldn't be too high.

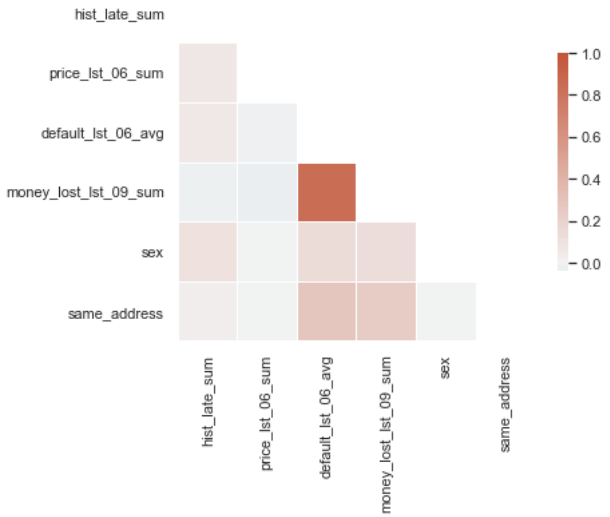

Figure 28 Correlation of features initially selected by SFFS

Here we have 0.84 correlation between *money_lost_lst_09_sum* and *default_lst_06_avg.* Since *default_lst_06_avg* has higher IV I drop *money_lost_lst_09_sum.*

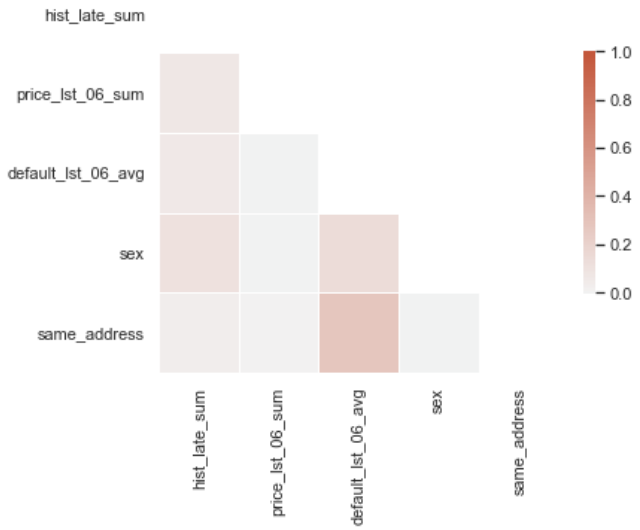

Figure 29 Correlation of features after removing highly correlated feature

Logistic regression model has a predictive power measured in AUC equals 0.66 on test sets, which is equivalent of 32 GINI. This isn't a high predictive power. We should gain more data sources to achieve a score around 50-60 GINI.

To choose good cut-off point I used ROC curve.

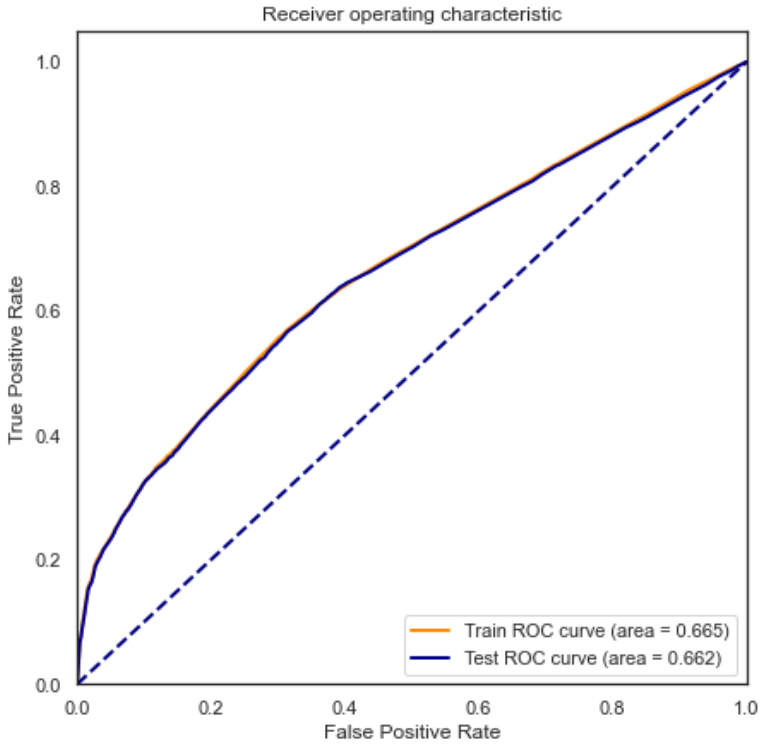

Figure 30 Receiver Operating Characteristic curve - Logistic Regression

"The sensitivity, or true positive rate of the model, is shown on the y-axis. And the false positive rate, or 1 minus the specificity, is given on the x-axis.
The ROC curve always starts at (0, 0) corresponding to a threshold of 1. This means we have 0 sensitivity and we won't catch a good care cases. But since our false positive rate is 0 as well, that means that we correctly label all the poor care cases.
The ROC curve always ends at (1, 1) which corresponds to a threshold of 0. So, the threshold decreases as we move from (0, 0) to (1, 1).
Let's take an approximate point (0.4, 0.65) on the curve. This point signifies that we correctly label 65% of the cases with a false positive rate of 40%" – source

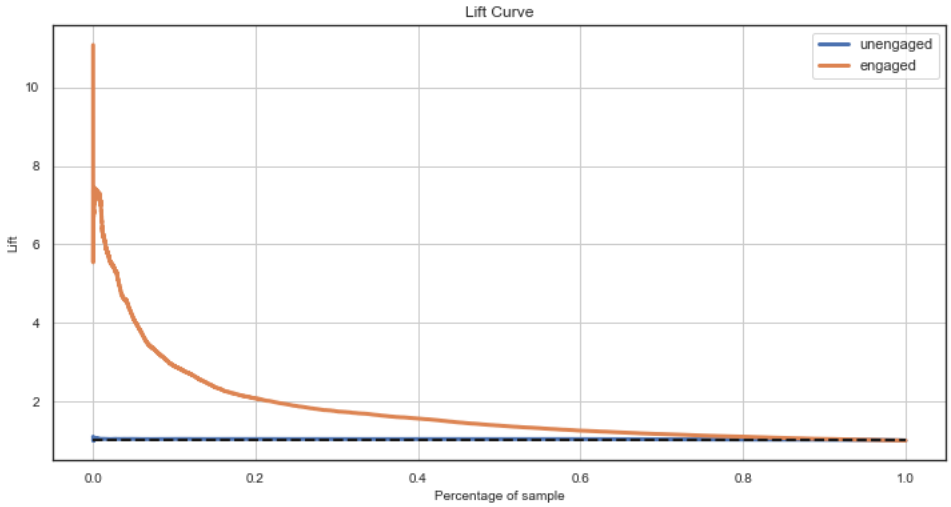

Figure 31 Lift curve - ranking clients from the highest risk to least risk

Lift curve above can be interpreted in the following way: if we take ex. 5% of the customers with highest predicted risk, we will be 4 times more accurate in predicting default than while selecting them at random.

Logit Regression Results

Dep. Variable:	default	No. Observations:	316942
Model:	Logit	Df Residuals:	316936
Method:	MLE	Df Model:	5
Date:	Wed, 21 Apr 2021	Pseudo R-squ.:	0.07560
Time:	07:55:17	Log-Likelihood:	-89259.
converged:	True	LL-Null:	-96559.
Covariance Type:	nonrobust	LLR p-value:	0.000

	coef	std err	z	P>|z|	[0.025	0.975]
const	-2.2984	0.007	-352.184	0.000	-2.311	-2.286
hist_late_sum	0.2152	0.069	3.103	0.002	0.079	0.351
price_lst_06_sum	0.9970	0.220	4.539	0.000	0.566	1.428
default_lst_06_avg	0.6081	0.013	47.763	0.000	0.583	0.633
sex	0.9172	0.020	46.057	0.000	0.878	0.956
same_address	0.8552	0.010	84.289	0.000	0.835	0.875

Figure 32 Logistic Regression parameters

9.2. LightGBM + Hyperopt

Currently there are three most popular applications of Gradient Boosting algorithms:

- XGBoost – the most accurate,
- LightGBM – the fastest and the most memory efficiency,
- CatBoost – the best on multiple categories features, easiest use with GPU,

I chose LightGBM due to performing my calculations on the laptop.

To find out the best hyperparameters for my model, I declared search space and used hyperopt for hyperparameters tuning, along with mlflow for experiment tracking.

My model was trained with early stopping based on validation dataset to prevent overfiting.

Showing 100 matching runs | Compare | Delete | Download CSV | Columns

		Metrics								
Start Time	Run Nam	verbose_eval	best_iter	stopped_	train-auc	train-average_	train-binary_lo	valid-a	valid-average_	valid-binary_lo
2021-04-20 18:	-	False	3	43	0.67	0.252	0.286	0.677	0.263	0.289
2021-04-20 18:	-	False	3	43	0.67	0.249	0.296	0.677	0.26	0.299
2021-04-20 18:	-	False	47	87	0.671	0.255	0.279	0.676	0.265	0.282
2021-04-20 18:	-	False	2	42	0.669	0.255	0.289	0.676	0.265	0.292
2021-04-20 19:	-	False	2	42	0.671	0.253	0.303	0.675	0.26	0.307
2021-04-20 19:	-	False	10	50	0.667	0.238	0.279	0.675	0.249	0.282
2021-04-20 19:	-	False	3	43	0.671	0.255	0.286	0.675	0.263	0.29
2021-04-20 18:	-	False	14	54	0.667	0.245	0.279	0.675	0.255	0.282
2021-04-20 18:	-	False	14	54	0.666	0.243	0.28	0.675	0.254	0.282
2021-04-20 19:	-	False	2	42	0.667	0.255	0.295	0.675	0.263	0.299
2021-04-20 18:	-	False	26	66	0.673	0.264	0.278	0.675	0.272	0.281

Figure 33 Experiments in MlFlow

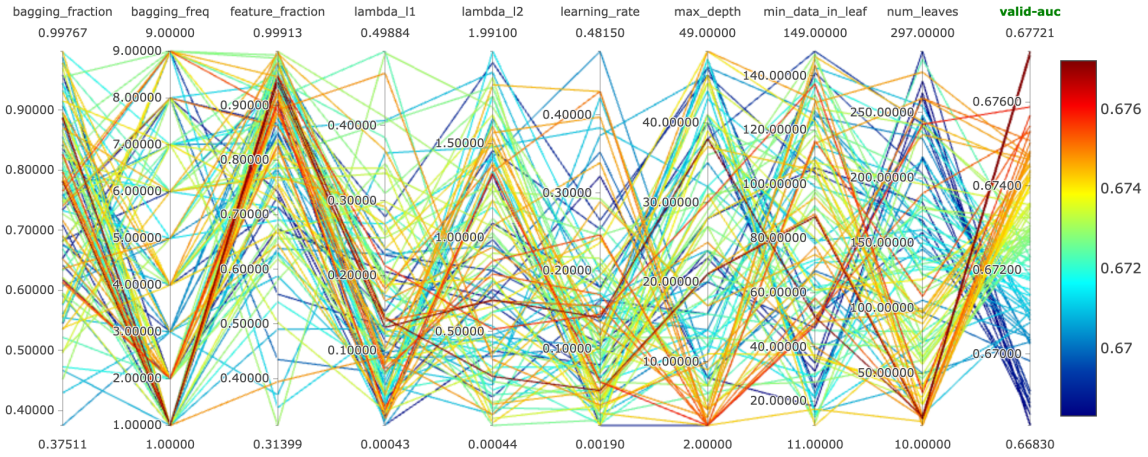

Figure 34 Comparison of hyperparameters in different experiments

The best hyperparameters set is shown below:

```
{
  "bagging_fraction": "0.8880304082464161",
  "bagging_freq": "1",
  "feature_fraction": "0.9431345128661389",
  "lambda_l1": "0.13105830883401792",
  "lambda_l2": "0.668330198543196",
  "learning_rate": "0.14008260036519118",
  "max_depth": "38",
  "min_data_in_leaf": "51",
  "num_leaves": "17"
}
```

- **bagging_fraction** - make training faster, and reduce overfit by reducing the number of data in each iteration,

- **bagging_freq** – how often perform bagging (Bagging: Take N random samples of x% of the samples and y% of the features,
- **feature_fraction** - fraction of features that are randomly selected in each iteration, the lower the more it prevents from strongest features domination,
- **reg_alpha (l1) and reg_lambda (l2)** - regularization factor, when number of features are low it could promote the strongest features and under valuate weaker features,
- **learning_rate** - a learning rate
- **max_depth** - tree max depth, the higher, the more is model prone to overfit,
- **min_data_in_leaf** - minimal number of data in one leaf. Can be used to deal with over-fitting
- **num_leaves** - max number of leaves in one tree

Features list that contributed the most to model predictions.

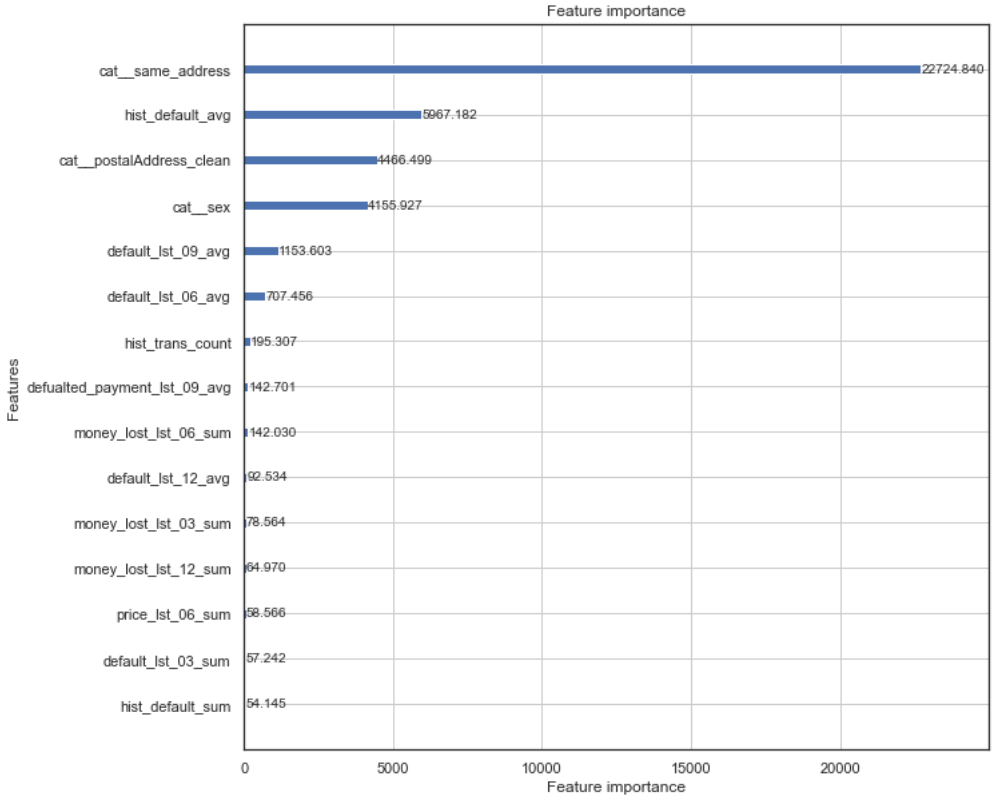

Figure 35 Feature importance

LightGBM model has a predictive power measured in AUC equals 0.67 on test sets, which is equivalent of 34 GINI – not a high predictive power.

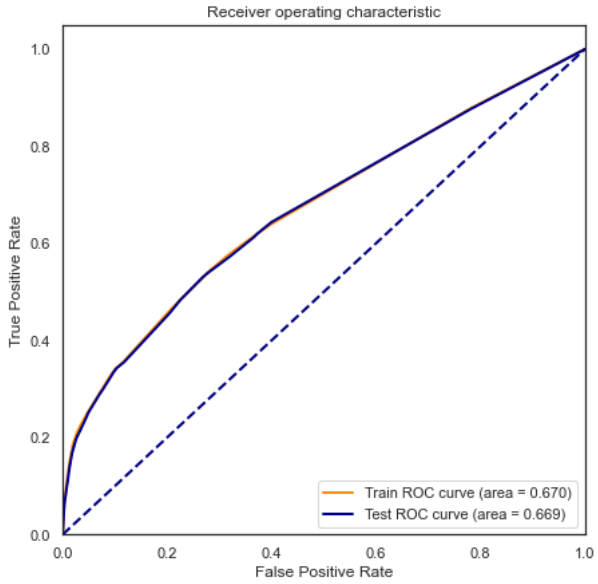

Figure 36 ROC curve for LightGBM

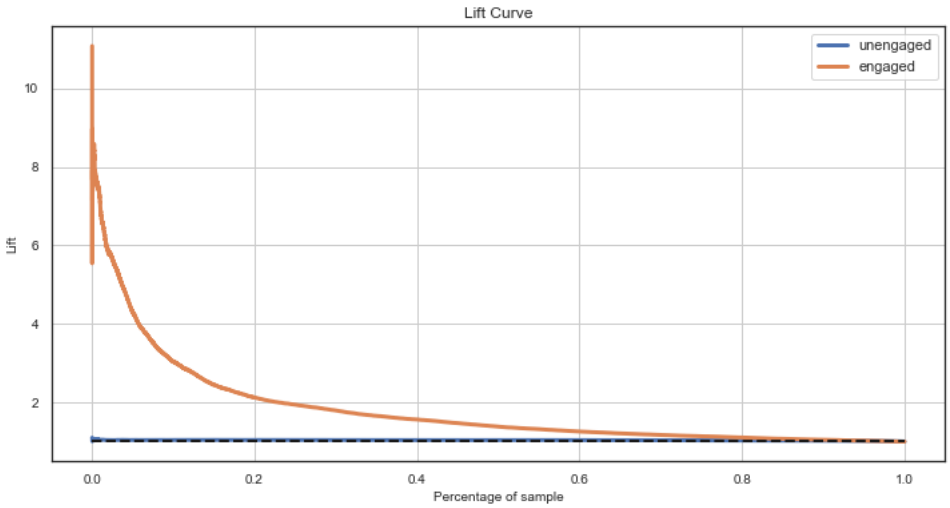

Figure 37 Lift curve for LightGBM

Lift Curve for LightGBM is very similar to Logistic Regression.

9.3. Summary

Comparison of the models

#	Model name	AUC – train	AUC – test	Notes
1	Logistic Regression with SFFS	0.665	0.662	+ results easy to interpret + feature importance + easy to deploy and use - - need some improvements, it takes a lot of time and knowledge to build a good model due to the need to meet statistical assumptions,
3	LightGbm + Hyperopt	0.67	0.669	+ very fast to build, no need to worries about outliers, missing values, values normalization, + non-parametric model with no statistical assumption, - black box – difficulties in interpreting factors that affect results, - more difficult to deploy without Docker,

Recommendation regarding choice:

- If you have a lot of time and need easy to deploy (without containers), easy to interpret model, I would choose **Logistic Regression**,
- If you need quick win and best score, go for **Gradient Boosting model**,

10. Technological aspect

Kedro *is an open-source Python framework for creating reproducible, maintainable and modular data science code. It borrows concepts from software engineering and applies them to machine-learning code; applied concepts include modularity, separation of concerns and versioning.*

Mlflow Tracking f*ocuses on experimental versioning. Its goal is to store all the objects needed to reproduce any code execution. This includes code through version control, but also parameters and artifacts (i.e objects fitted on data like encoders, binarizers). These elements vary wildly during machine learning experimentation phase. Mlflow also enable to track metrics to evaluate runs and provides a User Interface (UI) to browse different runs and compare them.*

I used kedro framework that provided project structure (cookiecutter) and integration between all the tools like jupyter, ipython, mlflow. Additionally, it also provided environment management tool. What is more, it hadsthe Data Catalog and pipelines that allows to build whole training pipeline., as well as good integration with mlflow, which gives experiment tracking and model registry.

Alle the tasks were first performed in Jupiter and then transferred to kedro pipelines (except the model, because of the time constraints). The logical structure of the pipelines can be seen on the graph below.

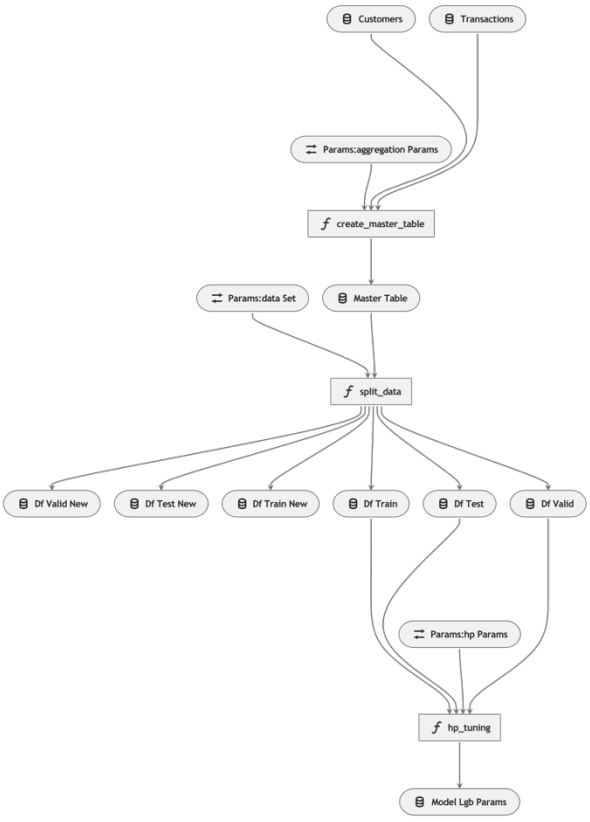

Figure 38 Kedro Pipeline schema