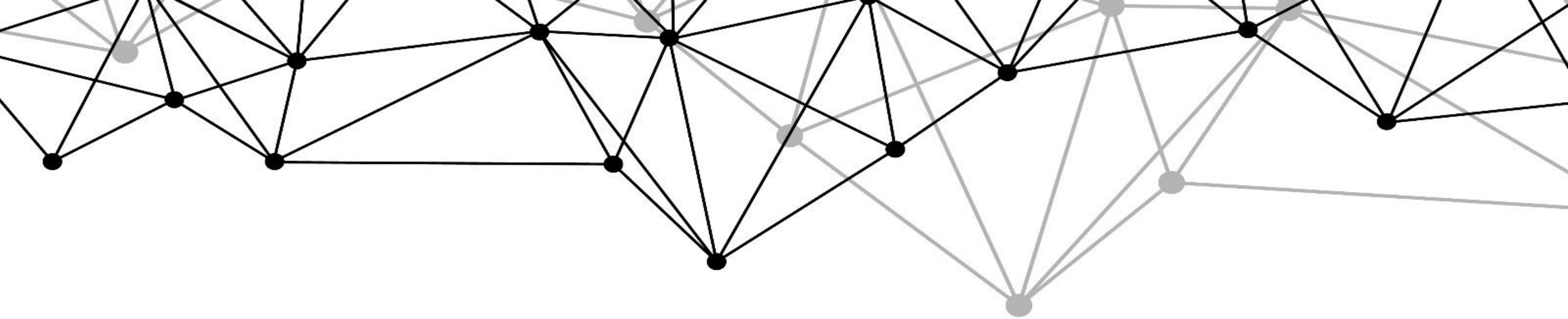


Predicting which subscribers will buy new phone soon

with binary classification model

using Vodafone Ukraine dataset

Author: Omelchuk Igor

Contents

1. Project timeline
2. Executive summary
3. Business understanding
4. Data exploration
5. Data preprocessing & feature engineering
6. Algorithms, models and features selection
7. Model evaluation
8. Analyzing key factors
9. Getting model ready for deployment

1. Project timeline

1. Feb 15 – Project start

   setting up Google cloud VM (e2 instance with 16vCPU and 64 Gb RAM),
   installing Anaconda and some libraries.

2. Feb 16 to Feb 21 – Exploratory analysis & Data preprocessing

3. Feb 22 to Feb 24 – Creating baseline models

   selecting several algorithms, creating simple models w/o tuning, using feature importance
   to drop unimportant features for models of each algorithm type.

4. Feb 25 to Feb 27 – Hyperparameters tuning and choosing best model

5. Feb 27 to Mar 1 – Evaluating best model on test set

6. Mar 13 to Mar 17 – Analyzing key factors and preparing model for deployment

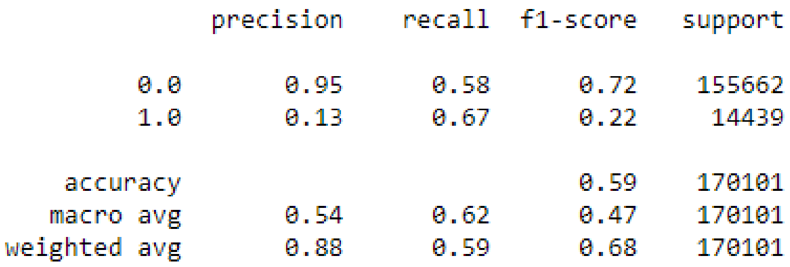
2. Executive summary

- Result of this project – trained model(s) that predict whether subscriber will buy a new phone next month. (LightGBM
- Best model identifies **67%** of users that will buy new phone, while also having specificity of **58%**,

Main model skill evaluation metrics:

ROC-AUC: **0.68**
F1-score: **0.22**
Recall: **0.67**
Precision: **0.13**

Classification report

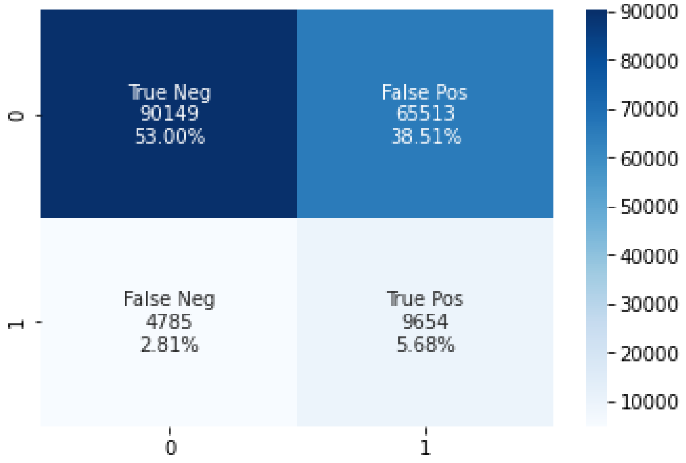


```
              precision    recall  f1-score   support

         0.0       0.95      0.58      0.72    155662
         1.0       0.13      0.67      0.22     14439

    accuracy                           0.59    170101
   macro avg       0.54      0.62      0.47    170101
weighted avg       0.88      0.59      0.68    170101
```

Confusion matrix



3. Business understanding

Business target: In the scope of Vodafone Retail promotion (sale of phones in stores), we need to identify subscribers inclined to replace their phone.

Target audience: Subscribers who are very likely to buy new phone next month.

Expected outcome: notified subscribers will buy phones in Vodafone Retail shops.

Data used: Vodafone dataset with features from CDR, DPI and Network statistics

Objective: train model that detects as many subscribers inclined to change their phone as possible, while keeping specificity at reasonable level

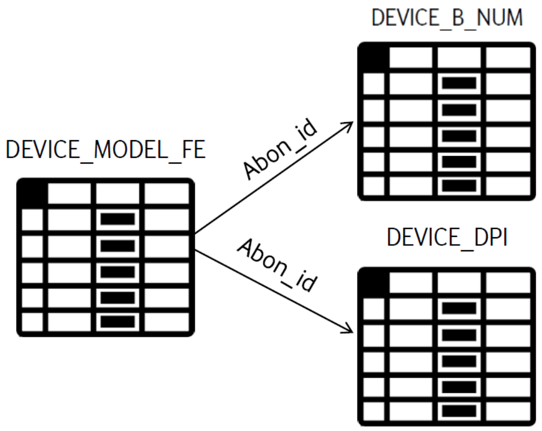
Key metrics: ROC-AUC, F1-score, recall, precision (with lean towards recall)

Data understanding (1/4)

1.Datasets description

Type	Dataset Name	Description	Shape	Dtypes	Unique id	Size
Train (60%)	DEVICE_MODEL_FE	Vitrin with data on subscribers usage patterns from different sources	205 625 rows X 878 columns	float/int: 878 cols	abon_id	1,4 Gb
	DEVICE_B_NUM	Data on subscribers interaction with short numbers	836 780 rows X 7 columns	float/int: 6 cols bytes: 1 col	abon_id + bnum	0,07 Gb
	DEVICE_DPI	Data on Applications usage from DPI system	6 647 895 rows X 6 columns	float/int: 6 cols	abon_id + Application	0,31 Gb
Test (40%)	DEVICE_MODEL_T_FE	*Same as on train dataset*	170 101 rows X 878 columns	*Same as on train dataset*	*Same as on train dataset*	1,2 Gb
	DEVICE_B_NUM_TEST	*Same as on train dataset*	772 356 rows X 7 columns	*Same as on train dataset*	*Same as on train dataset*	0,07 Gb
	DEVICE_DPI_TEST	*Same as on train dataset*	5 463 705 rows X 6 columns	*Same as on train dataset*	*Same as on train dataset*	0,26 Gb

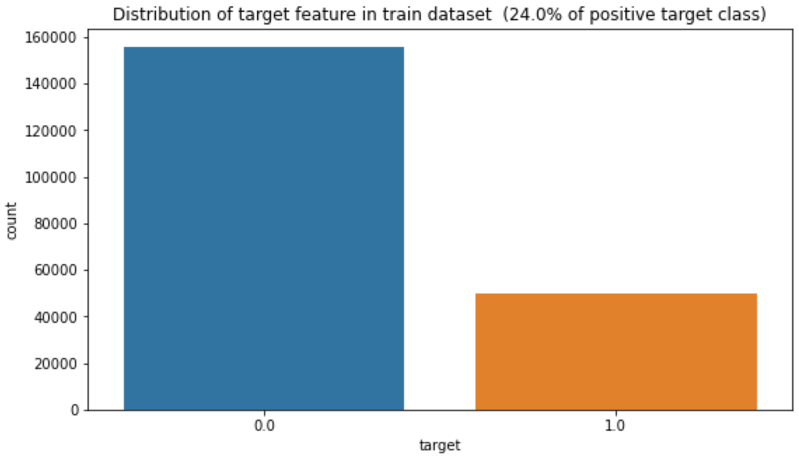
Data scheme:



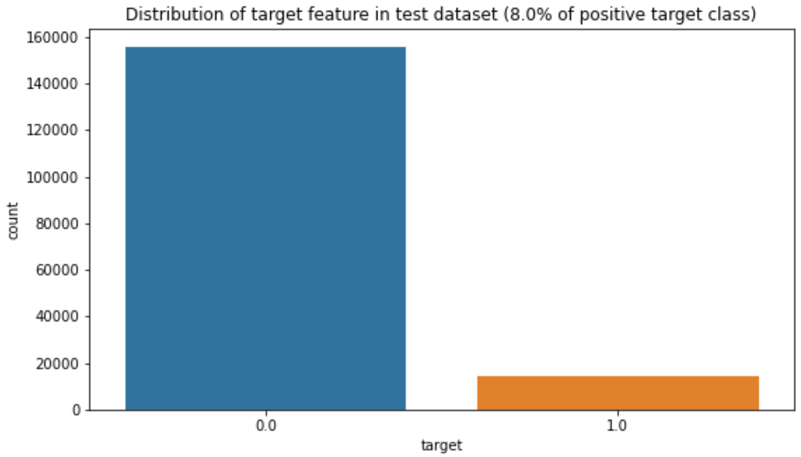
Data understanding (2/4)

2.Target variable distribution

Train set (3:1 ratio)



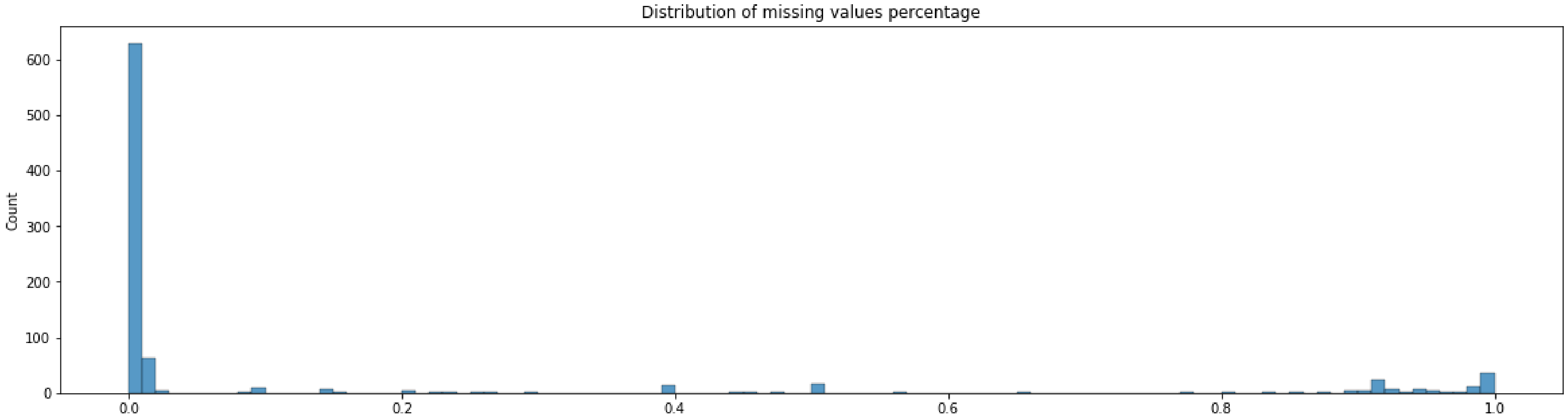
Test set (10:1 ratio)



❗Different target variable distribution in train and test sets and unbalanced classes – needs to be treated

(no missing target values found)

3. Fraction of missing values distribution

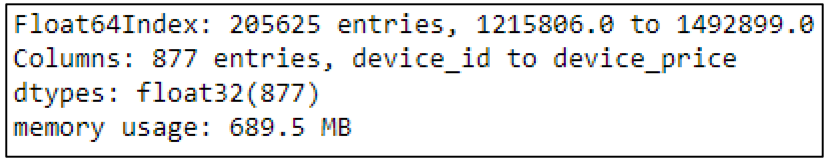


Most features have less than 1% of missing values

Data understanding (3/4)

4. Categorical features analysis (encoding possibilities)

3.1 All features in main vitrine are of the float type:



```
Float64Index: 205625 entries, 1215806.0 to 1492899.0
Columns: 877 entries, device_id to device_price
dtypes: float32(877)
memory usage: 689.5 MB
```

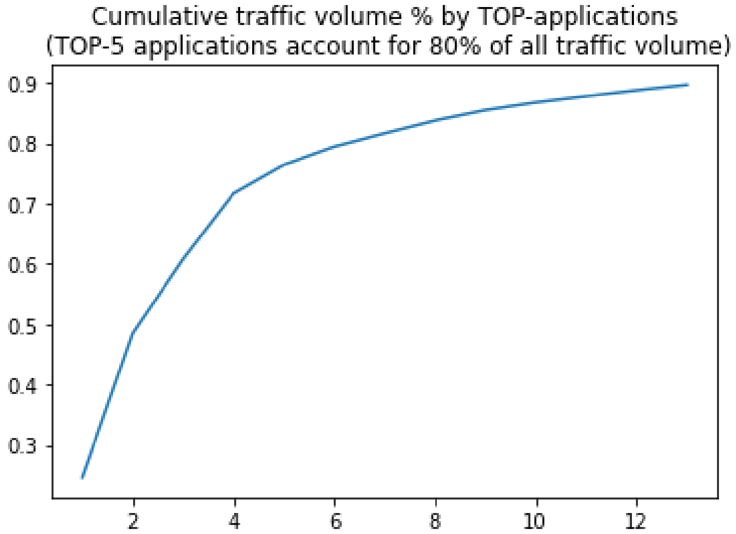
3.2 Most of the categorical features have only 2 unique values and description suppose binary type, no need for encoding

3.3 Only device OS version and Tariff Plan features may be encoded.

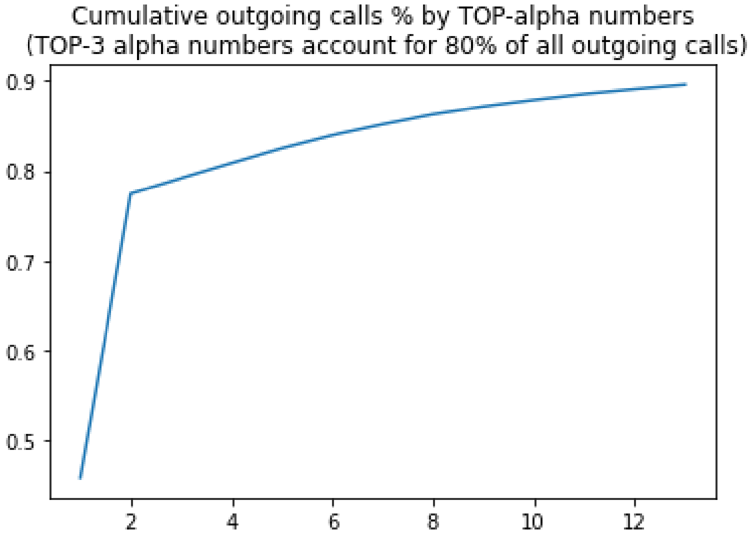
5. Analysis of DPI and bnum datasets

Only small fraction of Applications/bnums account for most part of events/traffic, most bnums used by small amount of subscribers

719 unique Applications



17k unique bnums



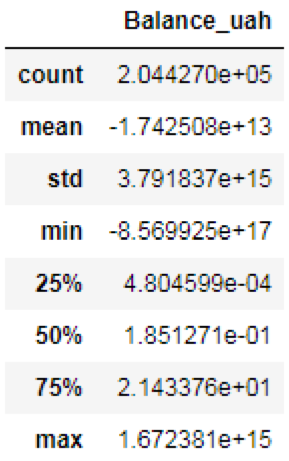
TOP-3 bnums by outgoing calls:

1) 111 – Vodafone CC
2) 3700 – PrivatBank
3) 102 – Police

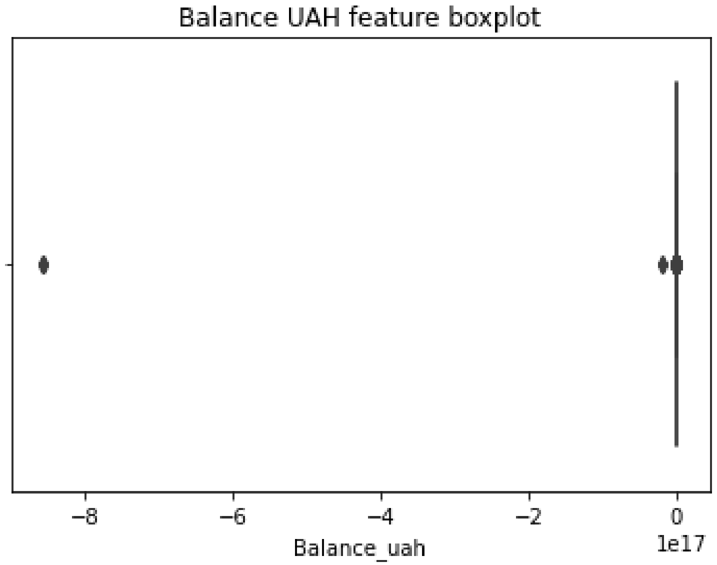
Data understanding (4/4)

6. Outliers

Some features had enormous outliers which probably appeared after data encryption.
Example:



	Balance_uah
count	2.044270e+05
mean	-1.742508e+13
std	3.791837e+15
min	-8.569925e+17
25%	4.804599e-04
50%	1.851271e-01
75%	2.143376e+01
max	1.672381e+15



Balance UAH feature boxplot

- Most values centered around zero, but with huge outliers on both sided
- Needs to be handled keeping in mind that non-linear transformations were applied during encryption and we may not know anything about true sample distribution

Data preprocessing & enrichment

1) Selecting TOP-200 bnums by events sum (account for 99% of total events) → Dropped a lot of very sparse features

2) Making pivots from DPI and bnum datasets to join with main vitrine → Enriched original dataset with lots of features by several network usage metrics (traffic volume, calls and SMS count, days of usage)

3) Grouping DPI and bnum datasets by users to create “Total usage” features → Created new features that represent generalized network usage patterns for each subscriber (sum, mean, etc.)

4) Imputing missing values with median value for each feature → Handled missing values, replacing with median is more robust than with mean especially when we have somewhat distorted data (and showed better res.)

5) Removing features having only 1 unique value → Dropped bad features that would not be useful in any way

6) Removing severe outliers with Isolation Forest method (~0.1% of values) → Treated values that skewed feature distributions a lot

✓ Formed merged train dataset with 4.9k features (to be reduced on feature importance step)

My approach for choosing best model and tuning hyperparameters:

1) Choosing several model types, performance-oriented and known for producing good results
2) Training simple model of each type with “educated guess”-like hyperparameters value. Additionally train simple baseline model (logistic regression) to later compare results.
3) Using feature importance plots to remove non-important features
4) Tuning hyperparameters with K-fold Cross-validation for each model, applying weights to positive examples to account for imbalanced data, and using early stopping to prevent overfitting
5) Comparing best models of each type by key metrics and average fit/predict time using CV

Algorithms selection

- Models that were included in my short-list are: **LightGBM**, **XGBoost** and **RandomForest** binary classifiers

- All of them are tree-based methods that are known for good performance.

(I also tried Support Vector Classifier model, but it was more than 10x slower comparing to other chosen models)

- Pros of tree-based algorithms:

  -Relatively fast to train
  -Interpretable (to certain degree at least)
  -Can classify not linearly separable data
  -Robust to unscaled data
  -Robust to outliers

Feature importance step

- I have trained baseline models of each type, with AUC evaluation metric, using validation set (70-30)
- Algorithms based on decision-trees can all show feature importance by several metrics.

For each model:

Final features list = set(features that account for **99%** of importance by split metric + features that account for **99%** of importance by gain metric)

As a result, around **900** features of **4900** left after feature importance step for each model (x5.5 reduction).

(X% importance threshold can be tuned if we need to drop more features, final model was fast enough with selected value)

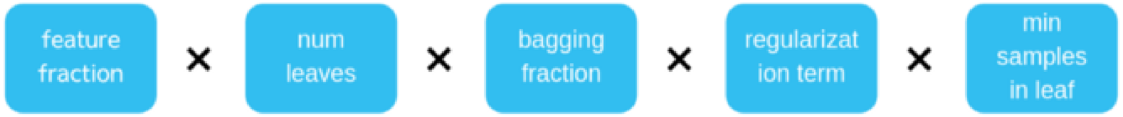
Hyperparameter tuning

- During Hyperparameter tuning, I used **Optuna** library to make this process easier.
- For each model, I used randomized Grid Search with 3-Fold CV, accounting for class imbalance via weighting positive class instances. Evaluation metric – ROC AUC.
- Example of params range list for XGB model:

```
{
    "objective": "binary:logistic“, "eval_metric": "auc", ”scale_pos_weight”: 3,
    "lambda": trial.suggest_float("lambda", 1e-8, 1.0, log=True),
    "alpha": trial.suggest_float("alpha", 1e-8, 1.0, log=True),
    "max_depth": trial.suggest_int("max_depth", 1, 9),
    "gamma": trial.suggest_float("gamma", 1e-8, 1.0, log=True)
}
```

- In case of LightGBM model, **Optuna** library provides step-wise tuning approach option.

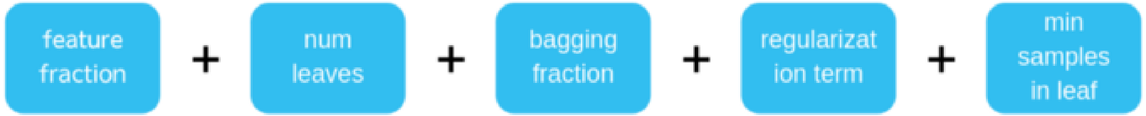
So instead of this:



feature fraction × num leaves × bagging fraction × regularization term × min samples in leaf

number of options

We get this:



feature fraction + num leaves + bagging fraction + regularization term + min samples in leaf

number of options

Final model selection

Evaluation process of best models:

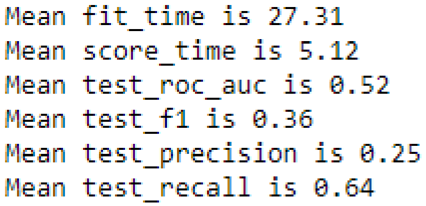
1) Get best hyperparameters for each model type from tuning step

2) Use 10-Fold Stratified CV to train and evaluate models.
Code snippet:

```
def evaluate_model(model, folds, X,y, scoring_metrics = ['roc_auc','f1','precision','recall']):
    kfold = StratifiedKFold(n_splits=folds, random_state=17, shuffle=True)
    results = cross_validate(model,X,y, cv=kfold, scoring=scoring_metrics)
    return results
```

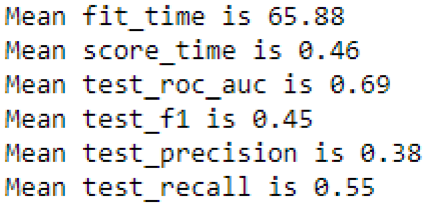
Validation Results:

Baseline model
(logistic regression classifier)



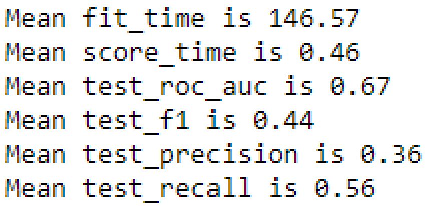
```
Mean fit_time is 27.31
Mean score_time is 5.12
Mean test_roc_auc is 0.52
Mean test_f1 is 0.36
Mean test_precision is 0.25
Mean test_recall is 0.64
```

LightGBM



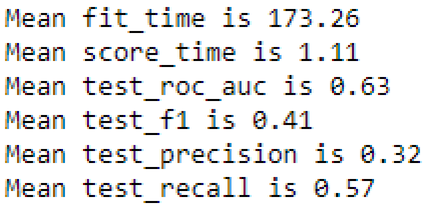
```
Mean fit_time is 65.88
Mean score_time is 0.46
Mean test_roc_auc is 0.69
Mean test_f1 is 0.45
Mean test_precision is 0.38
Mean test_recall is 0.55
```

XGBoost



```
Mean fit_time is 146.57
Mean score_time is 0.46
Mean test_roc_auc is 0.67
Mean test_f1 is 0.44
Mean test_precision is 0.36
Mean test_recall is 0.56
```

RandomForest



```
Mean fit_time is 173.26
Mean score_time is 1.11
Mean test_roc_auc is 0.63
Mean test_f1 is 0.41
Mean test_precision is 0.32
Mean test_recall is 0.57
```

Among tree-based models, **LightGBM** model has slightly better ROC-AUC and F1-score, and is 2-3x times faster.
Baseline model showed substantially worse results than tree-based ensemble algorithms.

Best model evaluation on test set (1/2)

1) Loaded test dataset, preprocessed it same way as train dataset.

2) Dropped features that are in only one of train/test datasets

3) Chose best binarization threshold on validation set using 5-Fold CV (2 options):

1. F-beta score metric, beta=2

(lean towards recall as we want to find as many subscribers who will buy new phone next month as possible)

best threshold

0.45

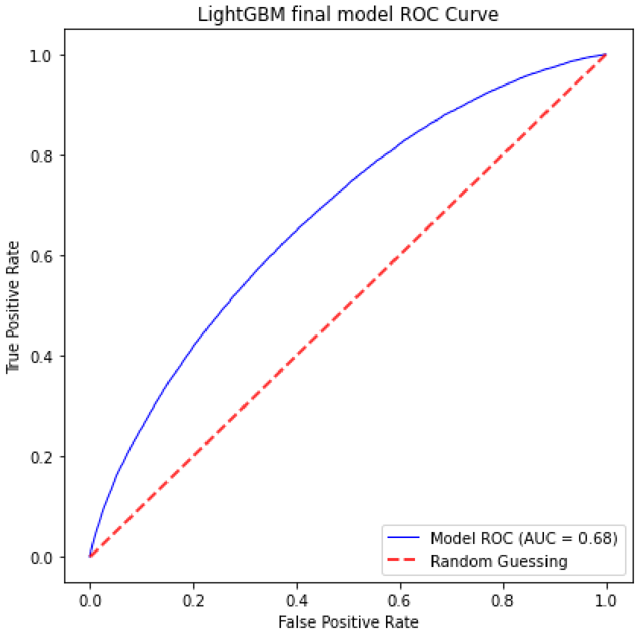
2. G-mean metric

(balance between performance for majority and minority classes)

best threshold

0.51

4) Train model on whole train dataset, getting predicted probabilities on test dataset, plotting ROC-curve:

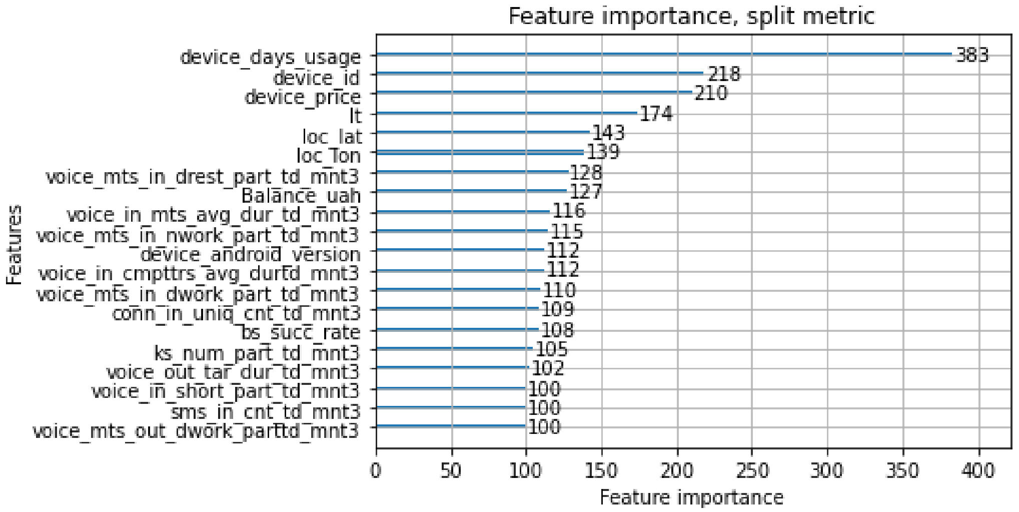


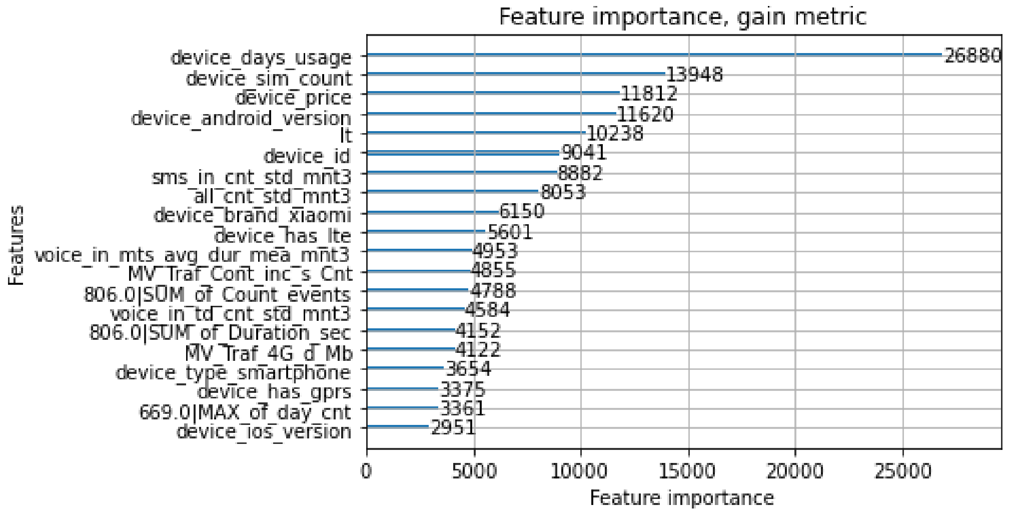
AUC on test set = 0.68

(Mean AUC on validation sets = 0.69)

Best model evaluation on test set (2/2)

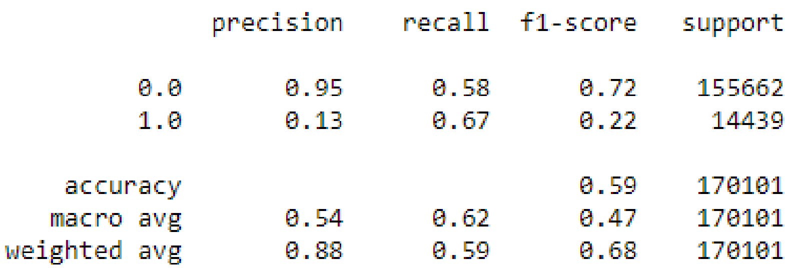
5) Plotting TOP-20 most important features (overall **913** features used in final model):


Feature importance, split metric


Feature importance, gain metric

6) Calculating evaluation metrics (with best threshold by f-beta score metric) :

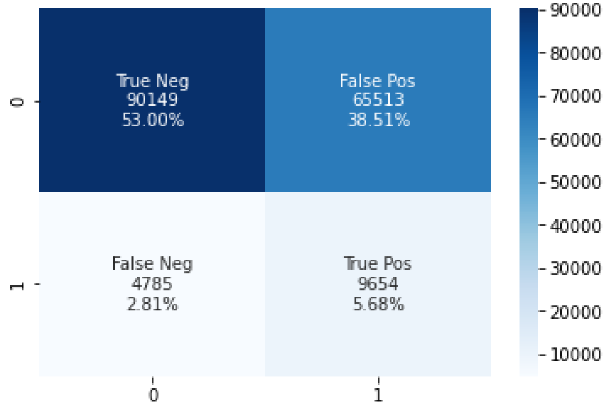
Classification report


```
              precision    recall  f1-score   support

         0.0       0.95      0.58      0.72    155662
         1.0       0.13      0.67      0.22     14439

    accuracy                           0.59    170101
   macro avg       0.54      0.62      0.47    170101
weighted avg       0.88      0.59      0.68    170101
```

Confusion matrix



Best model Hyperparameters:

{'metric': 'auc',
'verbosity': -1,
'boosting_type': 'gbdt',
'is_unbalance': True,
'feature_pre_filter': False,
'lambda_l1': 4.3,
'lambda_l2': 6.7,
'num_leaves': 31,
'feature_fraction': 0.58,
'min_child_samples': 5}

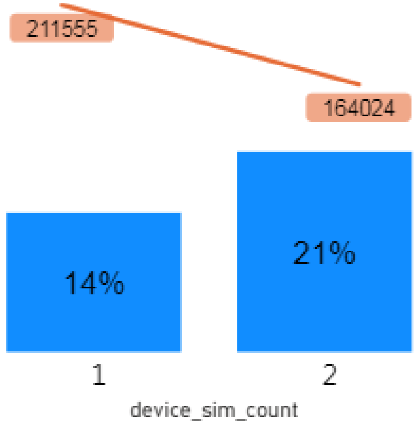
Key results:

ROC-AUC: **0.68**
F1-score: **0.22**
Recall: **0.67**
Precision: **0.13**

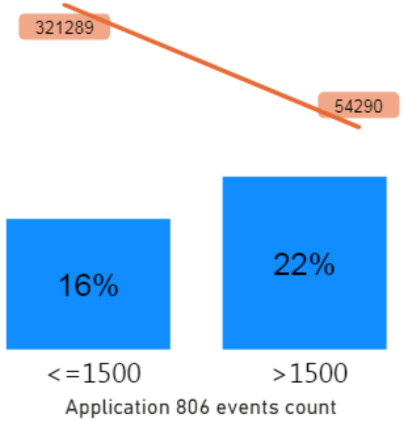
Key factors analysis (1/2)

- Below is analysis of key factors that are inherent to subscribers who bought new device:

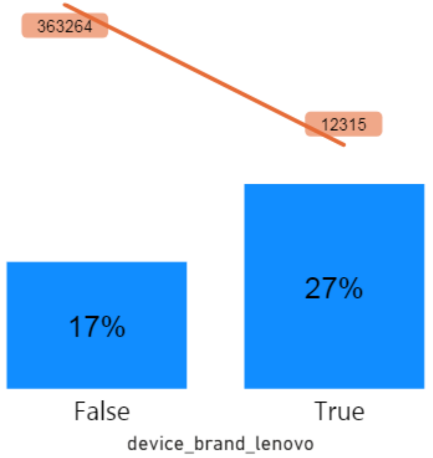
Subscribers who had phones with 2 SIM cards changed phones more frequently



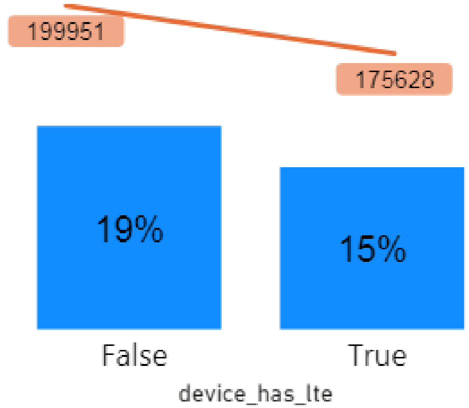
Active users of Application 806 changed devices more often than those with less or no events in this app



Lenovo device owners were more prone to changing device than owners of devices of other brands



Subscribers are more likely to replace old device with no LTE support

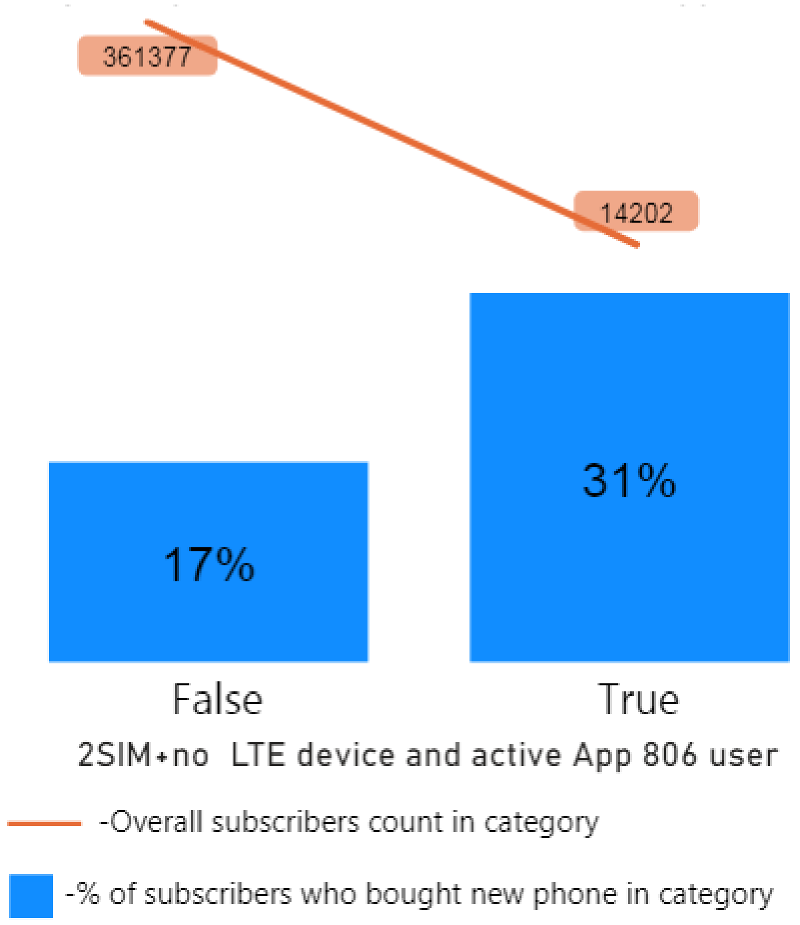


■ -% of subscribers who bought new phone in category — -Overall subscribers count in category

Key factors analysis (2/2)

- By combining factors shown on previous slide, we extract subgroup of subscribers, in which almost every third changed their device, as opposed to 17% on average:

Filters: Device with 2 SIM and no LTE support, active App#806 user



Wrapping model in a Class instance and serializing it

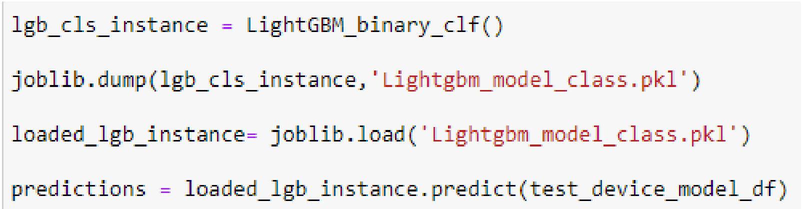
- I have wrapped model in a Class with preprocess and predict methods

-preprocess method checks and accounts for data consistency (if prediction set columns match model features)

-predict method outputs target prediction (either 0 or 1) for each subscriber row that is passed to method, using previously selected binarization threshold

- Then I have created instance of this Class and serialized it in **.pkl** object

Serializing and loading model



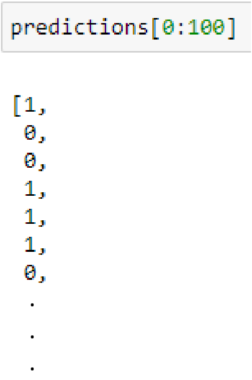
```
lgb_cls_instance = LightGBM_binary_clf()

joblib.dump(lgb_cls_instance,'Lightgbm_model_class.pkl')

loaded_lgb_instance= joblib.load('Lightgbm_model_class.pkl')

predictions = loaded_lgb_instance.predict(test_device_model_df)
```

Loaded model output example



```
predictions[0:100]
```

```
[1,
 0,
 0,
 1,
 1,
 1,
 0,
 .
 .
 .
```

Further deployment steps vary greatly depending on ML&data infrastructure

10. Reflexion, a-lessons-learned report, future steps

1) Things I tried during project implementation but which did not improve results:

-Scaling data (MinMax, Standard scalers). Tree-based methods are robust to scaling so I skipped it

-Using PCA on less important features to reduce features amount

-Using SMOTE & majority class undersampling to treat class imbalance on preprocessing step

-Dividing subscribers by groups (by Tariff Plan) and making predictions inside those groups

2) Recommendations:

-Select some of most important features from final model related to DPI data,
try to collect additional metrics by most important Applications if possible and retune model

3) Lessons learned:

-Try everything you think is well-reasoned for improving model/reaching business goal.

-Enrich data as much as you can before feeding into models.

CONTACTS:

Omelchuk Ihor

Data Analyst, M.Sc. in System Analysis

+380 (97) 333-77-71

ihoromelchuk505@gmail.com

Thank you!