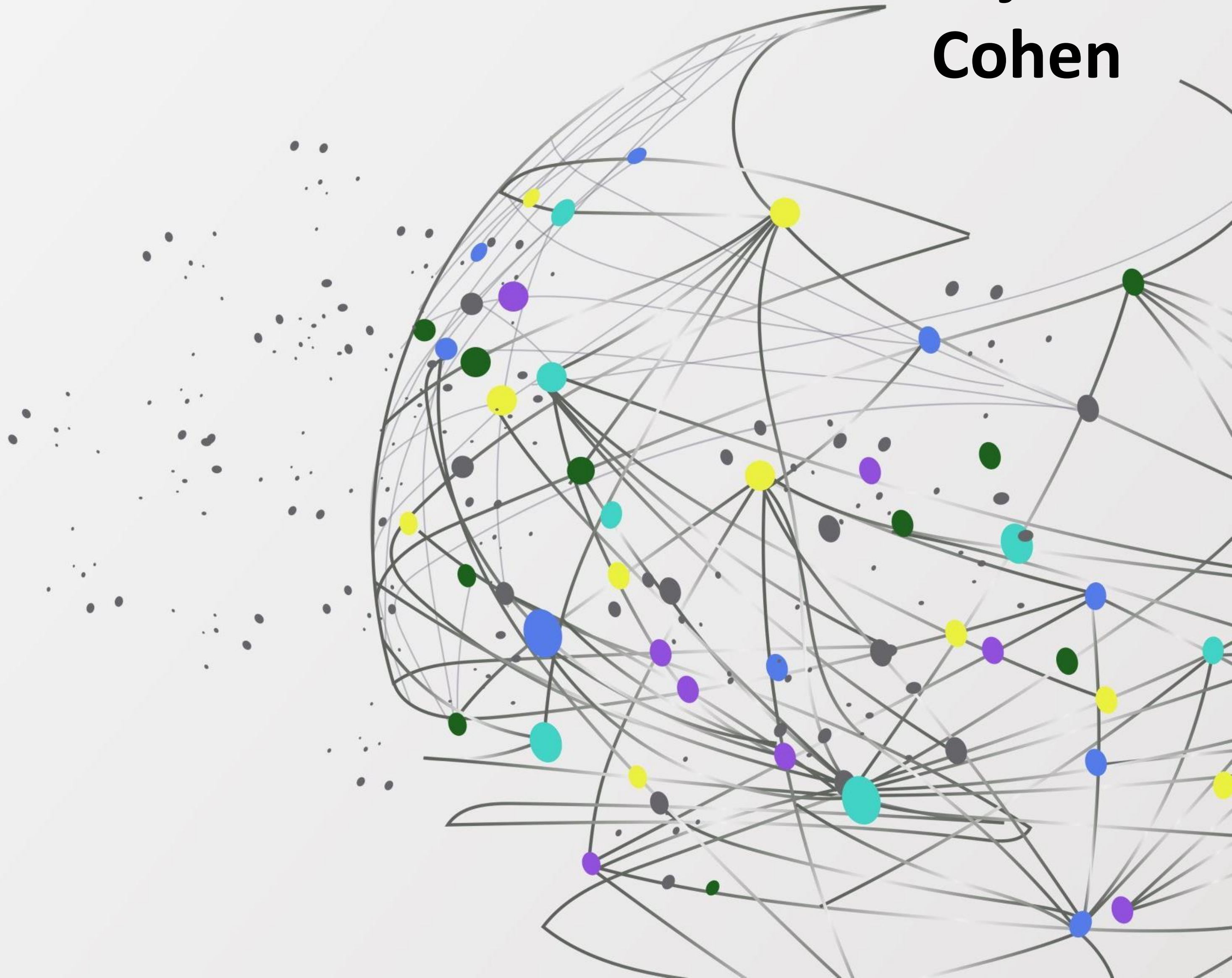
Rakuten Advertising Case Interview for Data Scientist Intern position

Benjamin Cohen



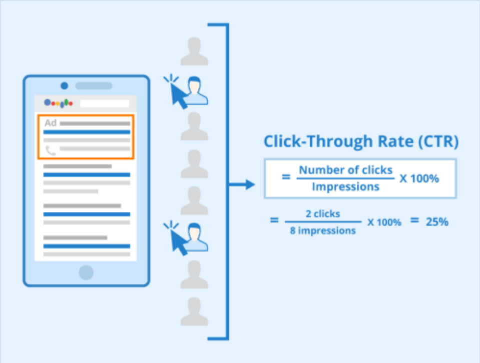
Prediction of the click-through rate for an impression

Rakuten Advertising

Topic of the presentation and problem statement

Problem statement

Rakuten Advertising is informed of a new auction for an ad placement for an user along with information on him/her. Those ads can lead to a bigger exposure and therefore a new chance to contract a sale with users. But those ad placements have a price so we need to **carefully value** them so we make a profit out of them.



- **Purpose**: Optimize ad placements for Rakuten on the internet.
- **Where** and **when**: During auctions in which Rakuten Advertising and its competitors are involved.
- **What**: One auction = one ad placement for one user at one given moment.
- **Time to propose a price**: 100 milliseconds to give a maximum amount to pay for the ad placement called bid response secret to other competitors (sealed auction)
- **Who gets the ad placement**: The participant who proposed the highest priced and paid for it. The others pay 0.

Problem Methodology

So in order to give a coherent price of those ad placements using users'infomation and their contexts, we must **estimate the potential clickthrough rate (CTR)** of the impression (ie: the probability they will click on the ad). This must be done before we inform the auction of the price we are willing to offer to display the banner.

To compute our CTR estimation, we have user-related information (e.g. time since last visit on an advertiser's website, number of banners already displayed to that user), and context-related information (e.g. format of the ad placement, domain of the web page on which it is to be displayed).

Rakuten Advertising

Dataset's features overview

- **ad placement**: A slot on a given webpage inside which a banner can be displayed.
- **cookie id**: An identifier that characterizes a user navigating on the internet. Two different cookie ids will be considered as two different users.
- **impression**: Number of banners displayed.
- **click**: A click that occurred on a banner.
- **advertiser**: A brand Rakuten Advertising are working with and for which it is displaying banners, e.g. Decathlon, Ray-Ban. One banner is related to exactly one advertiser. Our clients.
- **format**: The width and height of an ad placement in pixels. E.g. 728x90, 300x250 etc. If we win the auction, the displayed banner will have the same format as the ad placement.
- **width**: The width in pixels of a banner.
- **height**: The height in pixels of a banner.
- **visibility**: A figure that characterizes the position of the ad placement on the web page (top, bottom). It can happen that it is not provided in the bid request.
- **domain**: An abbreviation of the URL of the webpage on which the ad placement is. “youtube.com” or “lemonde.fr” are example of domains.
- **ad**: The banner template. Each ad is identified by its ad id. In other words, if we display twice the same banner template, it will have the same ad id each time.
- **date**: The date of the impression. In the provided dataset dates are truncated at the day level.
- **media**: A set of ad placements with common characteristics. For instance, a media could be “all ad placements sold by Microsoft in France”.
- **zone**: Each media is split into zones. Thus, a zone identifies a subgroup of ad placements with common characteristics. One zone could be “ad placements of media 28, with format 300x250 and with domain lemonde.fr or lesechos.fr”.
- **campaign**: Campaigns characterizes the qualification of the users. For instance, the advertiser Vans could have the campaigns “users who did a purchase on the Vans.com website”, or “users who visited the Vans.com website but did not do any purchases”.
- **country_ref**: Abbreviation for the country in which the user navigating the internet is located.
- **device**: iPhone, Tablet... or browser or combination or both of the user navigating the internet.

Rakuten Advertising

Dataset's overview

- **5,4 million samples** in the training set and **2 million samples** in the test set with **13 features** collected over a week to predict the next week number of clicks and thus the clickthrough rate of potential new ads to value an amount for the auctions.
- **Distribution of clicks highly skewed** to the left because most people do not click on ads.
- About only **3.73% of visitors click**.
- Number of clicks range from 0 to 694 with 184 different amount of clicks.
- Without any pre-processing, the **skewness** and **kurtosis** of the distribution of clicks are about **162** and **44691** respectively. Far more weight in the left tail of the distribution and more outliers than the normal distribution.

	date	zone_id	media_id	advertiser_id	campaign_id	domain	visibility	ad_id	width	height	country_ref	device	imps	clicks
Number of unique values	8	2910	187	119	503	82016	3	15184	19	144	12	17	1463	184
% of missing values	0	0	0	0	0	**1.77**	**48.52**	0	0	0	0	0	0	0

Rakuten Advertising

Libraries used and methodology

Basic Python Libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import pylab as py
import statsmodels as sm
```

Processing & scoring

```
from sklearn.preprocessing import OrdinalEncoder, StandardScaler
from sklearn.model_selection import train_test_split
from scipy.stats import kurtosis, skew
from sklearn.metrics import mean_squared_error,r2_score
from math import log, exp
```

Machine Learning Regressors

```
from sklearn.linear_model import LinearRegression, Ridge, Lasso, ElasticNet
import xgboost as xgb
import lightgbm as lgb
from catboost import CatBoostRegressor
```

Rakuten Advertising

Pre-processing of the data

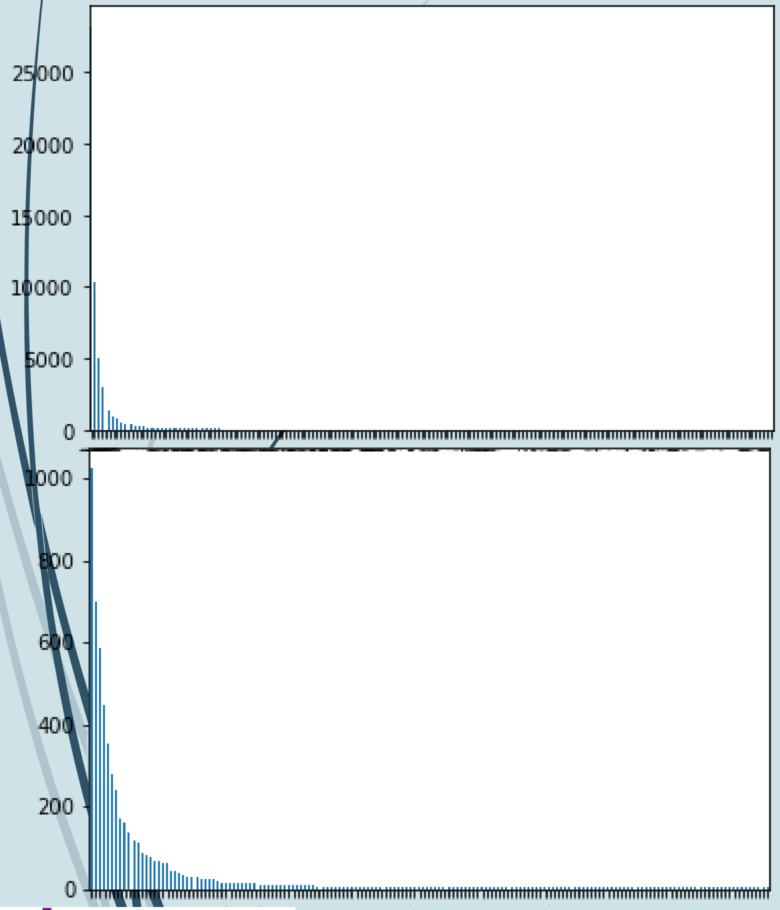
- **Handling missing values**: First try - using **linear interpolation** proposed by df.interpolate. After a few tries and using the round function to make the categorical variables integers again, the results did not improve.
  So I stayed on the second approach: Using **OrdinalEncoder** to convert the string categorical variables into integers ranging from 0 to n_categories - 1 per feature and **replacing the NaN values with –1**.
  .
- **Data transformation**:
  Using **powerTransformer** and **QuantileTransformer** to approach a **normal distribution** for the number of clicks. The results were not satisfying,the preprocessing using StandardScaler.fit_transform on the training set covariates X_train then transformed both the test set and validation set covariates respectively X_test and X_val lead to better results for linear models and decision-trees based models.

  For the **target** (ie: number of clicks that we want to predict) i tried in a first time:
  a Min-Max scaling as methods alike Linear Regression would lead to better results because the target ranges from 0 to 694.
  In the end the best results came from a **logarithmic transformation** using ln(y+1) then inverting after the predictions of our models.

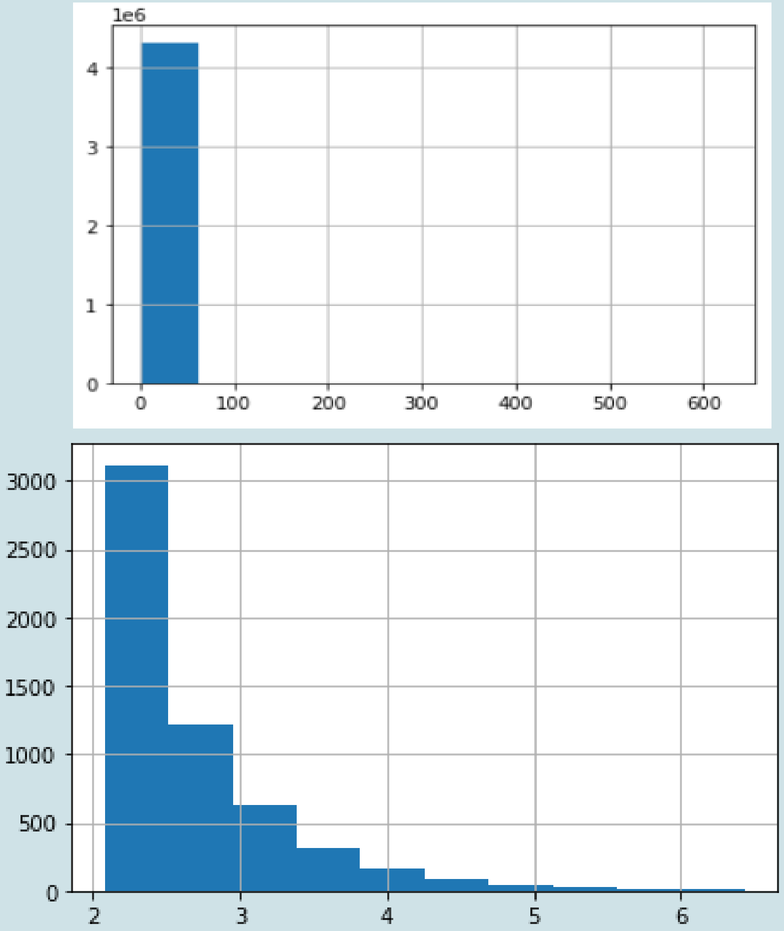
  The **skewness** and **kurtosis** of the click distribution went from **162 to 8** and **44690 to 88** respectively fitting more a normal distribution.
  NB: oversampling, sample_weight and smote did not lead to better results.

Pre-processing of the data

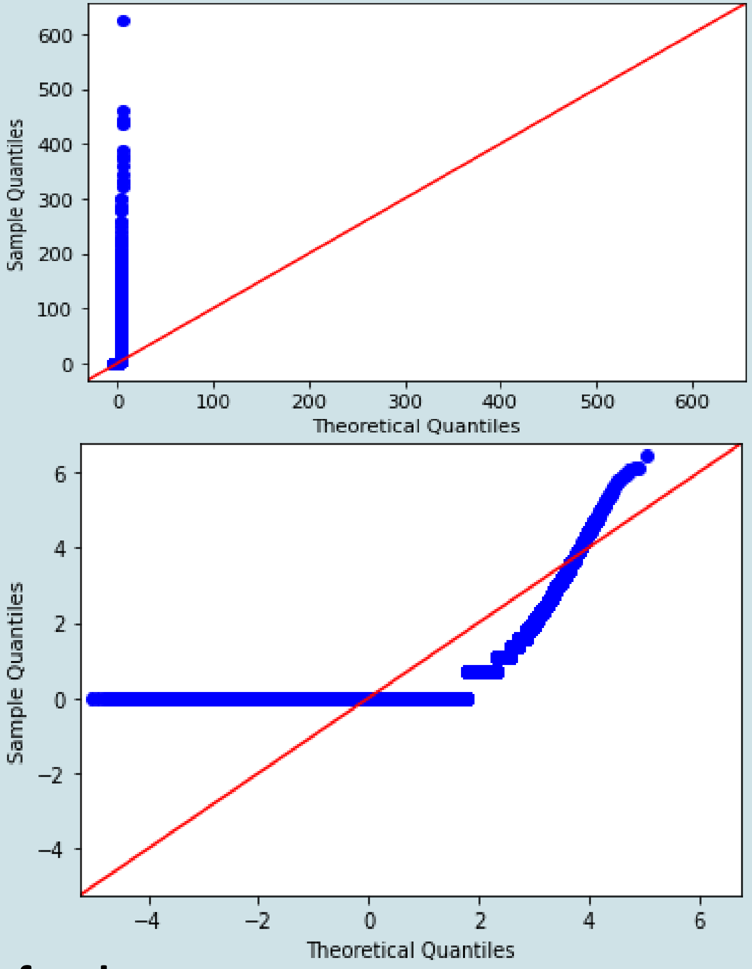
Bar-plot of the number of clicks before and after log-normalization



Histogram of the number of clicks before and after log-normalization



QQplot of the number of clicks before and after log-normalization



Rakuten Advertising

The number of clicks is more alike a normal distribution after log.

First try: Linear, Ridge, Lasso, regressions

Loss used to evaluate the predictions:

$$-\frac{\sum_{line\ i}\{clicks_i \times \ln(\hat{Y}_i) + (imps_i - clicks_i)\ \times \ln(1-\hat{Y}_i)\}}{\sum_{line\ i} imps_i}$$

No scaling on the covariates and the target

```
Error= 0.1267638080056897
Linear regression: difference between actual and predicted values 1.339084630587891
R2: 0.01
/usr/local/lib/python3.8/dist-packages/sklearn/linear_model/_ridge.py:157: LinAlgWarning
  return linalg.solve(A, Xy, sym_pos=True, overwrite_a=True).T
Error= 0.1027674095163307
Ridge: difference between actual and predicted values 1.2754003316091689
R2: 0.10
/usr/local/lib/python3.8/dist-packages/sklearn/linear_model/_coordinate_descent.py:647:
  model = cd_fast.enet_coordinate_descent(
Error= 0.11422434520199694
Lasso: difference between actual and predicted values 1.3235434575312581
R2: 0.03
```

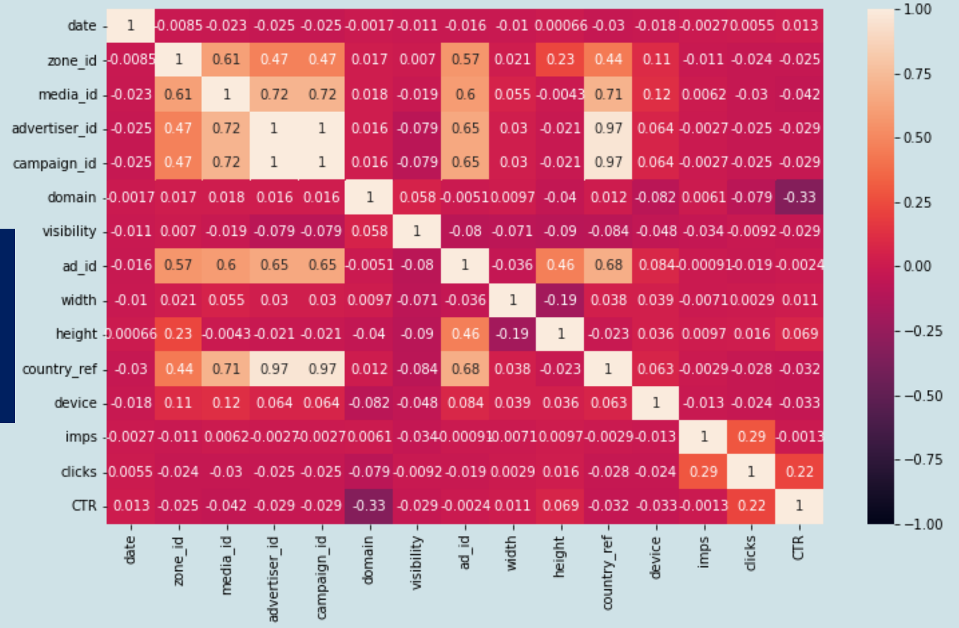
Standard scaling on the covariates and logarithmic scaling on the target

```
0.1138524175414041
Linear regression log-scaled: difference between actual and predicted values 56.881744628271136
R2: -1791.49
0.13994353404747054
Lasso regression log-scaled: difference between actual and predicted values 1.3441699754141496
R2: -0.00
0.11385499168128163
5
Ridge regression log-scaled: difference between actual and predicted values 56.89100034884436
R2: -1792.07
```

Standard scaling on the covariates and min-max scaling on the target

```
Error= 0.10305837064801758
Linear regression scaled: difference between actual and predicted values 1.2745707139803606
R2: 0.10
Error= 0.10304384394322363
Ridge scaled: difference between actual and predicted values 1.2745741117633027
R2: 0.10
Error= 0.13476349209365532
Lasso scaled: difference between actual and predicted values 1.3435223789171065
R2: -0.00
```

Covariance Heatmap Matrix



XGBoost vs LightGBM and CatBoost regressors: parameters

Some models are particularly suited for imbalanced datasets.
For example, in boosting models we give more weights to the cases that get misclassified in each tree iteration.

- **max_depth**: The maximum depth per tree. A deeper tree might increase the performance, but also the complexity and chances to overfit.
  The value must be an integer greater than 0.
- **learning_rate**: The learning rate determines the step size at each iteration while your model optimizes toward its objective. A low learning rate makes computation slower, and requires more rounds to achieve the same reduction in residual error as a model with a high learning rate. But it optimizes the chances to reach the best optimum.
- **n_estimators**: The number of trees in our ensemble. Equivalent to the number of boosting rounds.
  The value must be an integer greater than 0.
  NB: In the standard library, this is referred as **num_boost_round**.
- **colsample_bytree**: Represents the fraction of columns to be randomly sampled for each tree. It might improve overfitting.
  The value must be between 0 and 1.
- **subsample**: Represents the fraction of observations to be sampled for each tree. A lower values prevent overfitting but might lead to under-fitting. The value must be between 0 and 1.

Regularization parameters:

- **alpha** (reg_alpha): L1 regularization on the weights (Lasso Regression). When working with a large number of features, it might improve speed performances. It can be any integer.
- **lambda** (reg_lambda): L2 regularization on the weights (Ridge Regression). It might help to reduce overfitting. It can be any integer.
- **gamma**: Gamma is a pseudo-regularisation parameter (Lagrangian multiplier), and depends on the other parameters. The higher Gamma is, the higher the regularization. It can be any integer.

XGBoost vs LightGBM and CatBoost regressors: Results

XGBoost results

alpha	colsample_bytree	learning_rate	max_depth	n_estimators	loss	time	Method Used
10	0.7	0.4	2	10	0.07868807612924227	2min30	XGBOOST
10	0.6	0.3	10	100	0.05954845967772007	25min	XGBOOST
10	0.7	0.4	10	100	0.05869134292135717	25min	XGBOOST
10	0.7	0.4	10	200	0.05819723396433916	48min	XGBOOST
10	0.7	0.4	20	100	0.06106302747858923	1hour	XGBOOST

LightGBM results

num_leaves	early_stopping_rounds	learning_rate	max_depth	num_iteration	loss	time	early stopped at	Method Used
20	300	0.5	10	1000	0.05885859414947101	6min	1000	LIGHTGBM
20	300	0.5	10	2000	0.05882383889757768	6min	1031	LIGHTGBM
20	300	0.8	10	2000	0.05939603709485562	7min	1101	LIGHTGBM

CatBoost results

l2_leaf_reg	random_strength	learning_rate	depth	iterations	loss	time	Training Set Used	CATBOOST
3	1	0.03	6	1000	0.06053128206543126	10min	X_train, y_train	Grow_policy = 'SymetricTrees'
3	1	0.03	6	1000	0.06428915824615546	10min	X_train_scaled, y_train_scaled	max_leaves=4096
3	1	0.03	6	1000	0.15363053689963857	10min	X_train, y_train_scaled	on 0.67 train, 0.33 validation
3	1	0.03	6	1000	0.060531278094734534	10min	X_train_scaled, y_train_log	on 0.67 train, 0.33 validation
3	1	0.03	6	1000	0.05871649264669241	10min	X_train, y_train_log	on 0.67 train, 0.33 validation
3	1	0.061158	6	3000	0.058450940198761014	30min	X_train, y_train_log	on 0.67 train, 0.33 validation
3	1	0.061158	4	1000	0.06088476859692932	10min	X_train, y_train_log	on 0.67 train, 0.33 validation
3	1	0.149401	8	1000	0.0577331211792265	13min	X_train, y_train_log	on 0.67 train, 0.33 validation
3	1	0.149401	12	1000	0.056863955494348715	30min	X_train, y_train_log	on 0.67 train, 0.33 validation
3	1	0.5	12	100	0.058468244778106894	4min	X_train, y_train_log	on 0.67 train, 0.33 validation
3	1	0.5	12	100	0.05840136014604107	9min	X_train, y_train_log	on 0.67 train, 0.33 validation
1	1		12	100	0.06467261216060241	4min	X_train, y_train_log	on 0.67 train, 0.33 validation
10	1		12	100	0.06562600057706483	4min	X_train, y_train_log	on 0.67 train, 0.33 validation
3	1	0.5	12	200	0.057804681879861756	7min	X_train, y_train_log	on 0.67 train, 0.33 validation
3	1	0.5	10	200	0.05810572636690242	6min	X_train, y_train_log	on 0.67 train, 0.33 validation
3	1	0.5	16	200	0.058197456771305625	17min	X_train, y_train_log	on 0.67 train, 0.33 validation
3	1	0.5	12	100	0.05892583363002603	4min	X_train, y_train_log	on 0.80 train, 0.20 validation
3	1	0.153647	16	1000	0.05767962777351761	1h25	X_train, y_train_log	on 0.80 train, 0.20 validation
3	1	0.153647	10	1000	0.05756711914914552	30min	X_train, y_train_log	on 0.80 train, 0.20 validation
3	1	0.087455	12	2000	0.05718235826294024	1h10	X_train, y_train_log	on 0.80 train, 0.20 validation

Rakuten Advertising

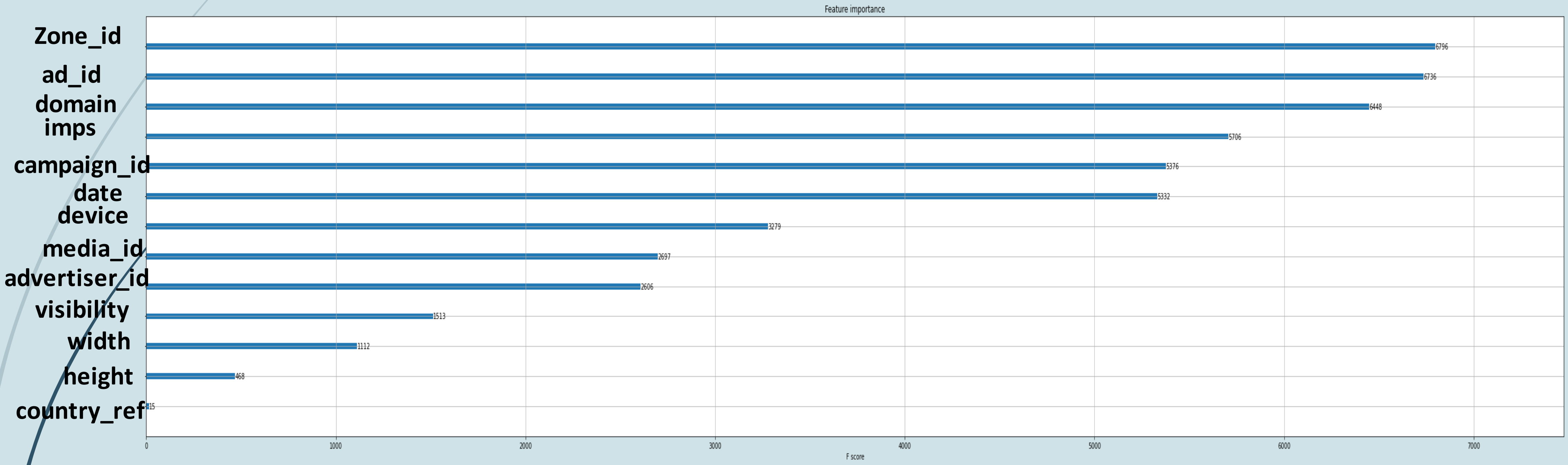
Best results: Catboost after fine-tuning

Using eval_set 80/10/10 - Train/Validation/Test

l2_leaf_reg	random_strength	learning_rate	depth	iterations	loss	time	Training Set Used	Grow_policy	subsample	score_function	max_leaves	model_size_reg	R2	Model used	RMSE
3	1	default	32	100	0.05106602955118768	30min	X_train, y_train_log	Lossguide	1	L2	16384	0	0.90	CatBoost	
3	1	0.189498	32	500 early stopped at 292	**0.047909687032176324**	1h30	X_train, y_train_log	Lossguide	1	L2	16384	0	0.98	CatBoost	0.19

XGBoost vs LightGBM and CatBoost regressors, Feature Importance

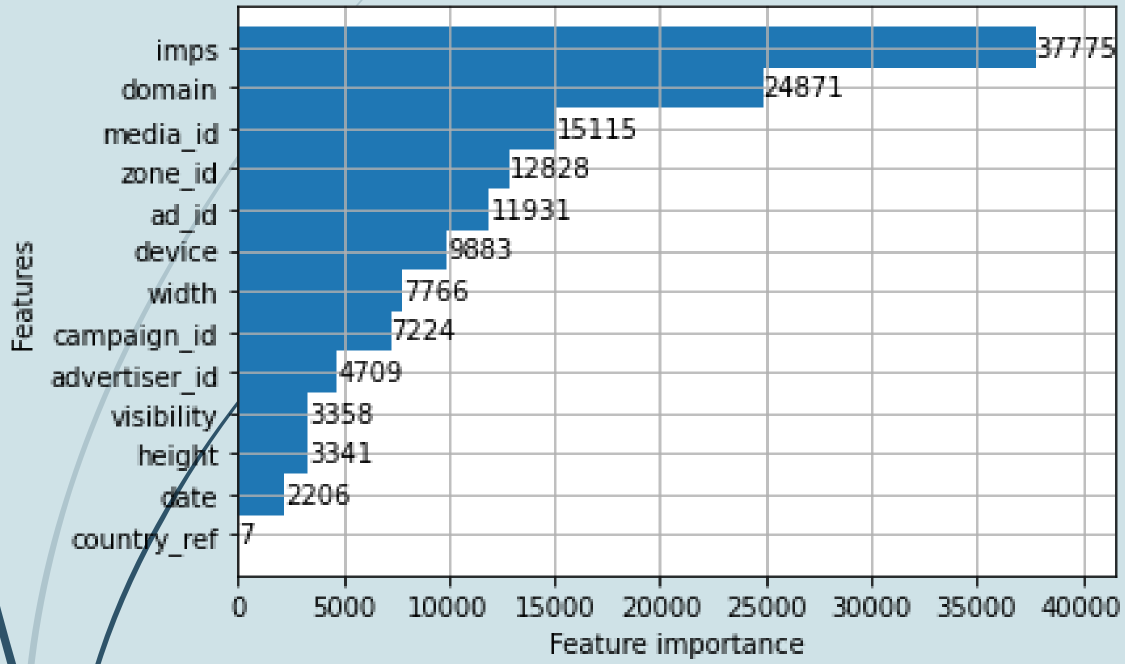
XGBoost Features Importance



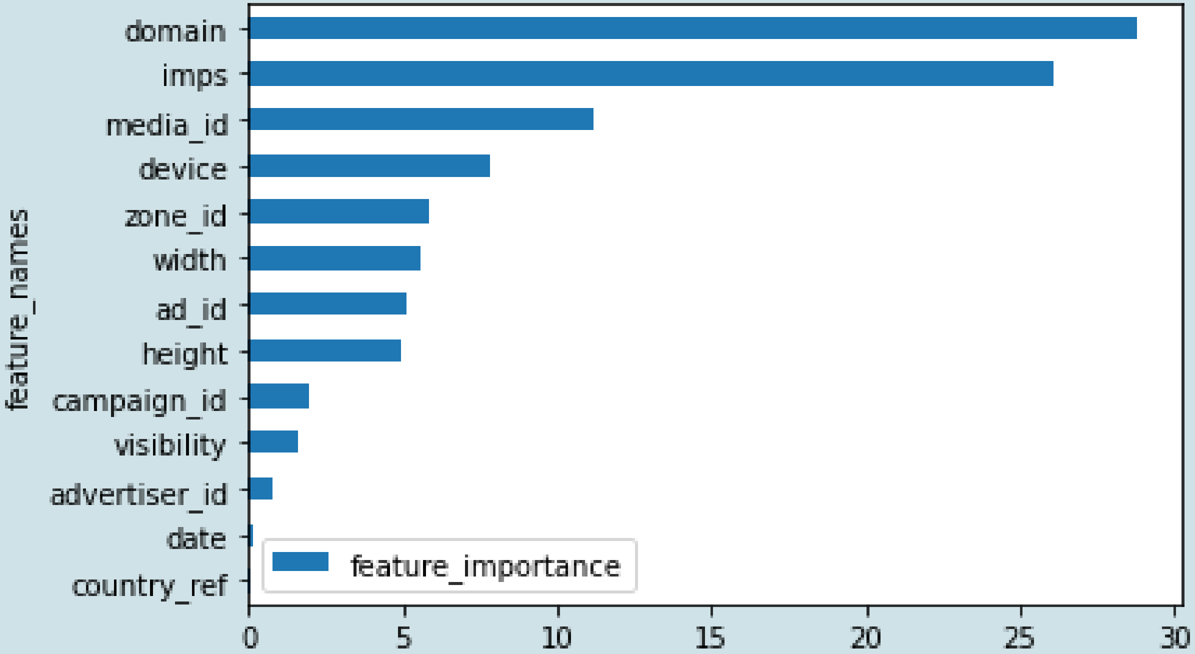
Most important features: zone, domain, ad_id and imps

XGBoost vs LightGBM and CatBoost regressors, Feature Importance

LightGBM Features Importance



CatBoost Features Importance



Most important features: imps, domain and media_id

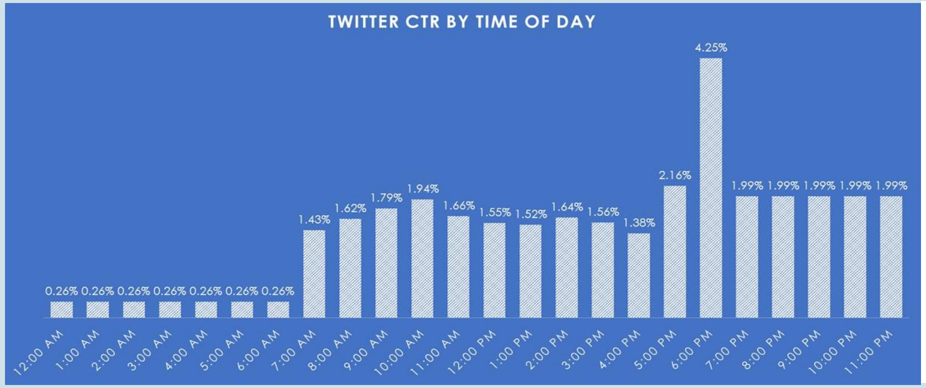
Ideas to increase the clickthrough rate

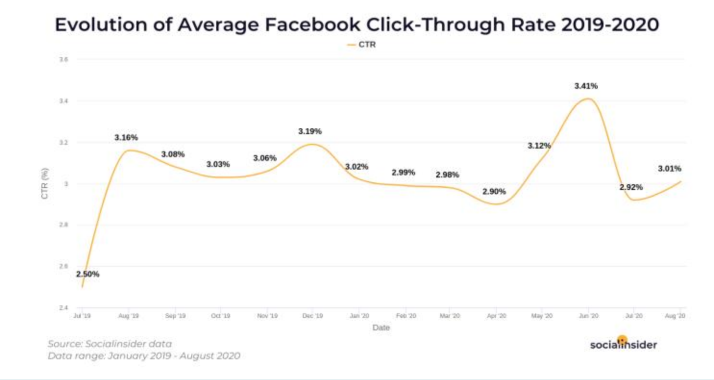
Machine Learning

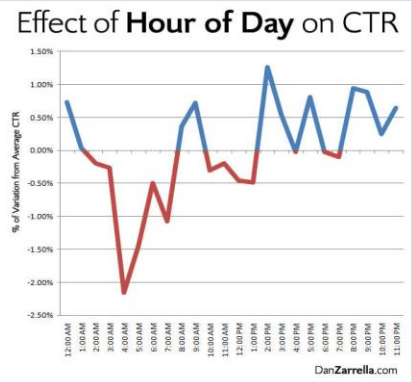
- **Feature engineering**: Add the hour of the day of the auction and extend the data on time.
- Try **Deep Learning** approaches using a **GPU** for computation.
- Use **GridSearch**, **HalfSearch** and **randomized search cross validation** for hyperparameter tuning.
- **Implement the loss function** and input it to the CatBoost Regressor in order to fit it and minimize it.

Qualitative ideas

- Focus certain **zones, medias, domains and number of impressions** as shown by the features importance of our boosting models.
- Focus on the **top of the site** for ad placement.
- The **bigger the ad** the more clicks we will get.
- The **color** helps to push the click.
- Make a **good call to action** (CTA), and compare new ideas with an **A/B test**.







Rakuten Advertising

Thank you for listening!

Any questions?

Benjamin Cohen

TELECOM Paris

INSTITUT POLYTECHNIQUE DE PARIS

COLUMBIA UNIVERSITY

Rakuten Advertising