AIX - MARSEILLE SCHOOL OF ECONOMICS

AUTOMATIC MODEL SELECTION, PREDICTIVE METHODS AND MACHINE LEARNING

Advertisement CTR (Click-Through-Rate) Prediction

Annabelle Filip
Bora Numani

January, 2021

Table of Contents

Advertisement CTR Prediction: Reduction Methods and ML Techniques

Annabelle FILIP and Bora NUMANI

Keywords: *Advertisement, CTR Prediction, Accuracy Score, Area Under the Curve, Accuracy Paradox, Automatic Feature Selection, Ridge, Lasso, Elastic Nets, FAMD, Machine Learning, Supervised learning: Random Forests, Support Vector Machines, Gradient Boosting Classifier, XGBoost Classifier, Neural Networks, unsupervised learning, K-Means Clustering.*

Abstract

In this study we have explored the subject of Advertisement Click-Through-Rate Prediction in Huawei User Data collected for seven consecutive days, through automatic feature selection techniques and Machine Learning Models. Up to date, the latest studies have focused on CTR Prediction by implementing Deep Neural Networks to achieve the highest area under the curve. These models have proved to be highly accurate in their classification, but as the networks' complexity gets higher, interpretability becomes more challenging. Our analysis starts with a Logistic Regression on the target variable, whether the user clicked or not, predicted its probability based on the information that was provided and improved results through Ridge, Lasso and Elastic Net Regressions. These techniques have lower predictive power compared to advanced ML classifiers but their estimations can provide more useful information from a marketing point of view.

In addition, our learning was based on supervised ML and unsupervised K-Means Clustering, which we fine tuned. Besides Neural Networks, that achieved an accuracy score of 77% on the training sets using k-fold cross validation and binary cross entropy, the following best classifiers for the task proved to be Random Forests and XGBoost, with AUC scores of 0.66 and 0.58 respectively on the test sets. The main constraint in the prediction is the high imbalance between the target classes, with only a little over 3% CTR, making the accuracy score an unreliable evaluation metric, observing the AUC (Area Under the Curve) score obtained by the ROC (Receiver Operating Characteristic) curve is rather desired.

For this reason, millions of rows of available data and powerful computational machines are also necessary. Because of computational expenses, we trained the models on a sample of 1.2million observations where we applied SMOTE (Synthetic Minority Oversampling Technique) to obtain a 50%-50% distribution between classes to avoid the Accuracy Paradox on the testing sample.

The purpose of the study was to explore techniques that achieve high precision on CTR Prediction, besides using Deep Neural Networks. As the hyperparameters have been fine tuned to best fit the data, the key to obtaining higher AUC scores is to run the models on larger sets of data.

1. Introduction

Prediction of click-through rate (CTR) in online advertising is critical in recommender systems, where the major interest is to estimate the probability a user will click on a recommended advertisement. As many search engines operate on a 'pay-per-click' rather than a 'pay-per-view' basis, the goal is to maximize the number of clicks, and so the advertisements presented to a user can be ranked by estimated CTR; In such scenarios, by maximizing the number of clicks, search engines also maximize their revenue, and so the ranking strategy can be revised as *CTR x bid* across all clients, where bid is the benefit the system receives if the advertisement is clicked by the targeted user.

Through effective targeting, advertisement service companies can offer users relevant advertisements to their intentions, while their predicted CTR also determines how much would need to be bid to beat a competitor in the auction. With a growing clientele, so do website visits, online sales, bookings or mailing list signups with online ads that direct people to the company's website.
On the other end, a high CTR predictive accuracy is bound to increase customer calls with ads that feature the phone number and a click-to-call button, thus increasing awareness and public knowledge about the product or service offered.

A proper targeting can also increase in-store visits, with business advertisements that help people find the business on the map, resulting in potential in the door customers. This again increases the probability that the product or service offered will be bought by the targeted customer. Clearly, the most important task to achieve these goals is estimating CTR correctly.
For CTR prediction, it is imperative to discover implicit feature interactions behind user click behavior. Our study is based on Kaggle datasets that have been collected and anonymized by Huawei Digix and contain the advertising behavior data collected from seven consecutive days.
As the data has been anonymized to protect users' personal data privacy, it imposed a constraint in the interpretation of the descriptive statistics in this study. The exploratory descriptive statistics are interpreted in function of their respective encoded classes, backed up by literature review on the correlations between the described feature and advertisement CTR.

The key challenge is in modelling feature interactions effectively. Following an economic intuition, some feature interactions can be easily understood and estimated, while other interactions are hidden among the data, but can be captured with automatic feature selection techniques, while high precision and predictive power is obtained through Neural Networks.

MAIN RESULTS: Given the complex non-linear relationships between the features, the models with the best predictive power were Deep Neural Networks, with 2 fully connected hidden layers, achieving an accuracy of 77%, followed by Random Forests with a 0.66 Area Under the Curve Score.

Through XGBoost Classifier from the xgboost package, we were able to obtain a 0.58 AUC and also extract feature importances. The most decisive features in the prediction of CTR were: Application Score, the Device Size and City Rank.

2. Literature Review

There are a large number of studies and open competitions to date, whose aim is to have the highest possible accuracy of prediction to a user's behavior facing an advertisement while surfing through apps or online. The studies range from using Support Vector Machines, to Gradient Boosting Classifiers, to complex Deep Neural Networks. Amongst them, Kumar et al. in 2015 used a Logistic Regression algorithm on a one-week advertisement data by considering position and impression as a predictor variable, and they achieved an accuracy of 90% for CTR estimation.

King et al., in 2015 compared four base classification models with four ensemble methods (Voting, Boosting, Stacking and MetaCost) for pay-per-click campaign management and found that ensemble learning methods were superior classifiers based on profit per campaign evaluation compared with the four main simple classifiers. (Naïve Bayes, LR, DT, SVM).
More recently, G. Zhou et al., 2017, proposed a new method to deal with user features compressed into a fixed-length vector – Deep Interest Network (DIN). Results demonstrate the higher effectiveness and precision of Deep Learning approaches, which achieve superior performance compared with state-of-art methods.

Also related to our methodology framework, Google published an article describing how they present advertisements to users based on search queries. They use probabilistic methods to reduce the number of features in order to save memory, as we have in the following sections. In addition, they report best results for the logistic regression model with regularization along with the use of modified gradient descent algorithms that allow them to save memory while maximizing prediction accuracy.

3. Materials and Methods

Once we have defined the study question, our analysis can be summarized into five stages of data manipulation in each of the sections:

- Data preprocessing :
  - Resample the data
  - Rescale the data
- Automatic Feature Selection.
- Evaluate algorithms performance.
- Improve results.
- Present Results

In the following sections we have detailed each level of the process.

3.1. Data Presentation

The dataset we have chosen to respond to the study question comes from Kaggle and has been collected by Huawei DIGIX from 7 consecutive days.
The original dataset contained a whole of 42 million observations with no missing values. To save computational time, we took a random sample of 1.2 million observations, and a random sample of 356.000 observations as our test sample. The data are as presented below:

Input Feature	***Description***
Label	Equals 1 if the user clicked and 0 if not.
Adv_id	Unique ID of an ad material
Inter_type_cd	Display form of an ad material
Slot_id	Ad Slot ID
Tags	Application tag of an ad task
App_first_class	App level-1 Category of an ad taks
App_second_class	App level-2 Category of an ad task
Device_name	Phone Model used by a User
Device_size	Size of the phone used by a User
Career	User Occupation
Gender	User Gender
Net_type	Network status when a behavior occurs.
City_rank	Level of the resident city of a user
His_app_size	Application storage size
His_on_shelf_time	Release time
App_score	Application rating score
Emui_dev	EMUI (operating system) version
List_time	Model release time
Device_price	Device Price
Up_life_duration	HUAWEI ID lifecycle
Up_membership_grade	Service Membership Level
Membership_life_duration	Membership lifecycle
Consume_purchase	Paid user tag.
Communication_avgonline_30d	Daily active time by mobile phone
Indu_name	Advertisement industry information
Pt_d	Date when a behavior occurs.

3.2. Data preprocessing

3.2.1. Unbalanced data, rescaling

- Rescale the data

To reveal the structure of our dataset, taking into account algorithms to be applied later on, we choose to preprocess the numerical features using:

MinMaxScaler() from the Sklearn package, we rescale all the chosen input features. It translates each feature individually such that it is in the given range on the training set.

The process of rescaling the data is given below:

$$\widehat{X}_i = \frac{X_i - Xmin_i}{XMax_i - Xmin_i}$$

$$XScaled_i = \widehat{X}_i(\widehat{XMax}_i - \widehat{Xmin}_i) - \widehat{Xmin}_i$$

Logistic Regression and Gradient Boosting algorithms are typically not sensitive to the magnitude of features, so standardization is not needed before. But it is a crucial step on Principal Component Analysis, Lasso and Ridge Regressions, thus we have applied this step to the numerical features in the preprocessing phase.

- Unbalanced Data

The area under the curve for our models would have been misleading due to our data being highly unbalanced, with only 3.45% of observations having clicked on the recommended advertisement. To reduce this type of error, we used the Synthetic Minority Random Oversampling and Random Under Sampling techniques.

SMOTE works as follows. It selects examples that are close in the feature space, drawing a line between the examples in the feature space and drawing a new sample at a point along that line.
We first oversampled the 'clicked' (feature *label* = 1) class to have 10 percent the number of observations of the 'not clicked' (feature label equal to 0) class via SMOTE, then used random under sampling to reduce the number of observations in the 'not clicked' class to have the same number of observations in both classes..

```
over = SMOTE(sampling_strategy=0.1)
under = RandomUnderSampler(sampling_strategy=1)
steps = [('o', over), ('u', under)]

pipeline = Pipeline(steps=steps)
X_train, y_train = pipeline.fit_resample(X_train, y_train)
```

We gathered these two functions in a Pipeline and applied it to our training datasets, X containing the features and y being the variable we want to predict, whether the user clicked on the advertisement or not. After this process, we end up with a balanced out dataset of 364113 observations between clicks and non-clicks in the binary 'label' variable.

We have randomly extracted a chunk of 20% of the original dataset (~10700 observations), where the CTR is a little over 3% to use to test our models.

- Distinction between Categorical and Numerical features

All the input features in the dataset have been anonymized and their values have been codified into integers. This was a challenge on the specification of the categorical and numerical features, thus we have made assumptions on their nature based on the description that was provided.

We have decided to consider the following features as numerical, assuming that their encoding in integers is positively correlated to the real values in the dataset, in short, 1 more unit represents an increase in the respective level interval on the feature:

- *Age* : Represented in codified intervals ranging from 1 to 9 , showing the user's age.
- *City_Rank* : Represented in codified values ranging from 1 to 9 , showing the level of the user's resident city.
- *His_App_Size* :Represented in encoded values ranging from 1 to 22 , reflecting the application storage size.
- *Device_Size*: Represented in encoded values ranging from 1 to , showing the size of the user's phone, alias the size of the screen.
- *Device_Price* : Represented in encoded values ranging from 1 to 9 , reflecting the price of the user's phone.
- *Up_membership_grade* : Represented in encoded values ranging from 1 to 5 , reflecting the user's service membership level.
- *Membership_life_duration* : Represented in encoded values ranging from 1 to 22, reflecting the user's membership lifecycle.
- *Communication_avgonline_30d* : Represented in encoded values ranging from 1 to 15 , showing the user's average daily active time by phone.

The rest of the features we have assumed categorical.

3.2.2. Principal Component Analysis and Factorial Analysis of Mixed Data

To reduce the complexity of our dataset, we wanted to use a Principal Component Analysis (PCA). It's an unsupervised technique that helps to reduce the dimensionality by using an orthogonal transformation of our independent features in the model. It's for information reduction in data and it only helps the econometrician to discover structure in the variables used to predict the dependent variable. It attempts to learn about the relationship between the X and y, by finding a list of the principal axes (principal components) in the data. It computates the covariance matrix and finds the main eigenvectors that represent a principal component that collects most of the information. It is a widely used technique in dimension reduction and works very well for large datasets that contain redundant information, where several features could have significant correlations.

However, our dataset is a mix of numerical and categorical variables. Factor analysis of mixed data (FAMD) is a principal component method that (roughly) combines PCA for continuous variables and multiple correspondence analysis (MCA) for categorical variables. Standardisation is critical here, as it is in PCA, in order to balance the influence of each variable. Indeed, non-scaled data can affect the results since these methods are

sensitive to the features variance. Variables with a larger variance tend to dominate those with lower variance. At the end results are biased since most of the information is retrieved from the variables with larger dispersion.

We will use the *prince* package in python. This package automatically does the standardization so we skip this step for now.

Applying this method on our dataset provides very poor results. Even when tuning parameters we end up with a very weak explanation of our inertia by our factors.

Factors	Explained Inertia
FAMD1	0.003
FAMD2	0.002
FAMD3	0.002
FAMD4	0.002
FAMD5	0.0015

Table 1 : Explained inertia

We also tried to apply PCA by converting our categorical variables into dummies. As a result we obtain the following explained variance :

Principal Components	Explained variance
PC1	0.076
PC2	0.047
PC3	0.038
PC4	0.034
PC5	0.028
PC6	0.024
PC7	0.022
PC8	0.03

Table 2 : Principal components and explained variance

We need at least 8 components to explain 30% of the variance.

3.3. The choice of the metric

We needed a way to evaluate different types of models, to tune parameters, to choose attributes, etc. A model evaluation procedure is used to estimate how well the model is able to generalize to new data. This requires performance indicators in order to quantify how well our model predicts the class and to be able to compare models with each other.

- The accuracy is the percentage of correct predictions (so the number of correctly predicted classes over the total number of prediction).The accuracy is easy to understand but has some drawbacks. It does not take into account the distribution of predicted values and says nothing about the "types" of errors the classifier makes.

- The null accuracy is the prediction rate that can be achieved by always predicting the dominant class. When the accuracy is similar to the null accuracy, the model is bad since it always predicts a certain class. It's usually the case when you deal with unbalanced data.

- The confusion matrix is a table that describes in detail the performance of a classification model. It's composed of :

- **True Positives** (TP) : the positive observations (positive class=1) for which we correctly predicted that they are positive.
- **True Negatives** (TN) : the negative observations (negative class=0) for which we have correctly predicted that they are negativ.
- **False Positives** (FP): the negative observations for which we have falsely predicted that they are positive ("Type I error").
- **False negatives** (FN): the positive observations for which we have falsely predicted that they are negative ("Type II error").

It gives "more details" on the performance of the classifier and allows the computation of other metrics, which can be used to select models.

- Other metrics are used to evaluate a model and the choice of it can depend on our subject. There are the classification error which is the percentage of incorrectly classified individuals, the sensitivity (or True Positive Rate) which is the percentage positive class correctly predicted . On the contrary we have the False Positive Rate: which represents how many times do I predict 'positive' when it's actually 'negative'.

  We have to mention the Area Under Curve (AUC). AUC represents the probability that the classifier will assign a greater predicted probability to the positive class than to the negative one. AUC is very useful when the classes are not balanced (unlike accuracy).

So, in our work we will first look at the accuracy which is a common thing to do, but then we will focus more on optimizing the sensitivity. Therefore, we will try to minimize the false negatives since in our case, predicting that someone won't click on the ad when they will is more problematic than predicting that the person won't click on the ad but will when it appears. Also we will try to maximize the area under the curve.

3.4. Logistic Regression

A logistic regression is a regression analysis where the target variable is binary like in our case y takes the value 1 or 0. It's a type of Generalized Linear Models that predicts the probability of occurrence of a binary event using a logit function. It applies the sigmoïd function on the linear regression. Instead of fitting a line on the data, logistic regression fits a s-shaped logistic function. Instead of using R^2, it uses the maximum likelihood. The algorithm designs a candidate line to fit our data (y-axis being the $log(odds)$) and then project the data points onto this line. Then transform the $log(odds)$ into probabilities :

$$log(odds) = log(\frac{p}{1+p}) <=> p = \frac{e^{log(odds)}}{1+e^{log(odds)}}$$

Making this transformation allows us to compute the log-likelihood (the sum of individual log-likelihood). Then, rotating the $log(odds)$ line and projecting the data onto it, give us a new log-likelihood. It chooses the line that maximizes the log-likelihood. Coefficients are expressed in $log(odds)$.

We apply this on our data.

3.5. Ridge Regression

With Ridge Regression we introduce a little bit of bias in the logistic regression to have a bigger drop in variance. Previously, in the logistic regression we select coefficients for each independent variable that minimizes the loss function. When training the model we can easily end up with an overfitting problem of our training set if the coefficients are too large. It can become a big problem when applying our regression on new data. To overcome this issue, we do regularization which penalizes large coefficients by their magnitudes. The purpose is to avoid overfitting and thus perform better out of the training sample.

The degree of regularization is controlled by the scalar λ, it's the weight given to the regularization term. The penalized regression is :

$$\widehat{\beta}_{\lambda}^{RIDGE} = argmin\,(n^{-1}\sum_{i=1}^{N}\rho_{(\beta)}(X_i, Y_i) + \lambda\sum_{j=1}^{N}\beta_j{}^2)$$

Where ρ is the loss function : $\rho_{(\beta)}(x,y) = -y(\sum_{j=0}^{p}\beta_j x^{(j)}) + log(1 + exp(\sum_{j=0}^{p}\beta_j x^{(j)}))$

Ridge regression uses a "L2 regularization technique" adding a "squared magnitude" of coefficient as penalty term to the loss function. When $\lambda = 0$ we come back to Logistic Regression. The bigger the λ parameter is, the more weight you put on your penalization term. If λ is too big, then you may face underfitting.

So now the hyperparameter λ has to be chosen. We will select the one that maximizes the AUC. A common way to do so is to use k-cross validation technique. It first splits the dataset in k sets called folds, for example 5 folds, then it designates 1 of them as the test set and the remaining ones as the train set. Then it moves to another group to be the test and the rest becomes the train, so on and so forth. Resulting results are more reliable than with train/test split due to the algorithm is trained and evaluated k times. However, as we were encountering many computational problems (especially the use of RAM) we decided to test the λ on our original train and test sets. We use a range from 0.001 to 0.75. Results will be presented in the section 'Result'.

3.6. LASSO Regression

LASSO (Least Absolute Shrinkage and Selection Operator) regression, similarly to Ridge Regression, introduces a penalty to the objective function with the help of a parameter λ. Similarly to Ridge, we have a trade-off between the bias and the variance obtained from the model. In both regressions, the bias-variance trade-off is decided by a continuous shrinkage (penalization) but the advantage of using Lasso, the "L1 Regularization technique", is that it can be used select the pertinent variables (and reduce overfitting), as it encourages the estimated coefficients to be sparse. What the model does is it shrinks the estimated coefficients of unimportant features to 0 as we increase the parameter λ, while only keeping the coefficients related to the most important features in predicting the target value positive in absolute value. This method performs two tasks which are regularization and feature selection. To do so, values that still have a non-zero coefficient after the shrinking process are selected.

$$\widehat{\beta}_{\lambda}^{\,LASSO} = argmin\,(n^{-1}\sum_{i=1}^{N}\rho_{(\beta)}(X_i, Y_i) + \lambda\sum_{j=1}^{N}|\beta_j|)$$

Where ρ is the loss function : $\rho_{(\beta)}(x,y) = -y(\sum_{j=0}^{p}\beta_j x^{(j)}) + log(1 + exp(\sum_{j=0}^{p}\beta_j x^{(j)}))$

Normally, as λ (the degree of regularization) grows, the regularization term has greater effect and variables in the model are shrunk to zero. Lasso is used for variable selection because depending on the value of λ it sets variables exactly equal to zero.

The next step is to choose the best λ that maximizes the AUC. Like we did for Ridge we take a range between 0 and 0.75 and compute the AUC value for each λ. We will present the results in the 'Result section'.

3.7. Elastic Net

Elastic Net Regression combines both L1 and L2 regularization penalties when performing a Logistic Regression. It can remove weak variables altogether as with Lasso or shrink them towards zero as with Ridge. The objective function in this method is then:

In addition to setting and choosing a λ_1 (for LASSO) and λ_2 (for Ridge) values (according to a common α), the method incorporates a hyperparameter $l1\ ratio$. When $l1\ ratio$ is equal to 0, the Elastic Net corresponds to Ridge and when $l1\ ratio$ equal to 1, it corresponds to Lasso regression. Anything in between is a combination of both. $l1\ ratio$ is the Elastic Net mixing parameter and $\lambda_1 = \alpha \times l1 ratio$; $\lambda_2 = 0.5\alpha * (1 - l1 ratio)$
Thus we have :

$$\widehat{\beta}_{\lambda}^{\,LASSO} = argmin\,(n^{-1}\sum_{i=1}^{N}\rho_{(\beta)}(X_i, Y_i) + \lambda_1\sum_{j=1}^{N}|\beta_j| + \lambda_2\sum_{j=1}^{N}\beta_j{}^2)$$

Where ρ is the loss function : $\rho_{(\beta)}(x,y) = -y(\sum_{j=0}^{p}\beta_j x^{(j)}) + log(1 + exp(\sum_{j=0}^{p}\beta_j x^{(j)}))$

We used the *LogisticRegression()* library in Sklearn where we can specify the type of penalizations in the parameters (the degree of penalization and l1 ratio).

We looked at the best degree of penalization and *l1 ratio* like we did for Ridge and LASSO. We set a range from 0.001 to 0.75 for the regulation strength and from 0 to 1 for the mixing parameter. We tested 198 combinations and picked the one that maximizes the AUC. Results will be presented in the section 'Results'.

3.8. Supervised Learning

3.8.1 Decision Tree and Random Forest

3.8.1.1. Decision tree classifier

Before doing a random forest we would like to look at what results can be obtained with a simple decision tree. A decision tree is a non-parametric machine learning modeling technique used both in regression and classification problems. It's composed in 3 parts : internal nodes, leaf nodes (at the end of the tree), and the root node (at the beginning). At each node the algorithm has to do the optimal choice to classify individuals which minimize what is called the impurity. The most used metrics in classification problems are entropy and Gini impurity (which measures how often a randomly chosen element is incorrectly labeled). The feature with highest decrease in impurity is selected for the internal node.

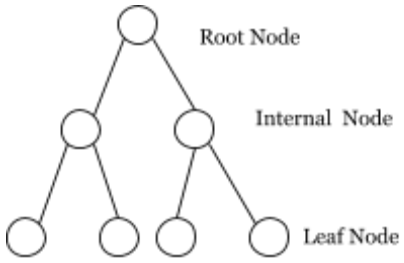


Figure 1 : Decision tree structure

We apply it on our database. Results will be in the section 'Results'.

3.8.1.2. Random Forest Classification

A way to improve the performance of your model is the use of ensemble algorithms. It improves individual algorithms by achieving a more accurate prediction. So, to improve the classic bagged decision trees we use random forests.
First of all, a random forest classifier creates a set of decision trees from a randomly selected subset of training set. It's a supervised method using bagging methods (bootstrap aggregating). The principle of bagging methods is to average the forecasts of several independent models to reduce the variance and therefore the forecast error. To build these different models, we select several bootstrap samples, i.e. drawn with replacement. Random forests are therefore an improvement of bagging for CART decision trees in order to make the trees used more independent (less correlated). It also decreases the risk of overfitting by using this average/mode method.
There are several advantages of using a Random Forest approach. It's a White box model meaning that the result is easy to conceptualize and interpret. Also the data preprocessing required is low. The classification of new data is quick (meaning that the cost of using trees is logarithmic). Random forest can deal with both categorical and continuous features and also with multi-class classification problems (which is not the case here). Finally they can handle missing data well.

We will use the *RandomForestClassifier()* library from Sklearn in python.

So the algorithm first creates a bootstrapped dataset and then creates a classification tree on this dataset, but using a random subset of variables at each step. Once it's done, it repeats these operations again and again until reaching the desired number of trees.

We can tune hyperparameters to improve our random forest. In this algorithm there are several hyperparameters (and there default settings in Sklearn) :

- The number of trees in our random forest (*n_estimator*). When this value is increasing, the prediction will be better but the time to compute will be higher. The default setting is 10.
- The maximal number of features (*max_features*) that our Random Forest considers when splitting the nodes. For no restriction use 'auto' (which is the default setting), 'sqrt' uses the square root of the number of features and 'log2' is logarithms of base 2 of the number of features.
- The maximal depth of each tree (*max_depth*) in the forest. A reduction of the depths can reduce the risk of overfitting but capture less information. The default setting is 'None' meaning there is no restriction.
- The minimum samples in each leaf (*min_sample_leaf*) in a tree. The default setting is 1.
- The minimum number of observations needed to split an internal node (*min_samples_split*). The default setting is 2.
- The number of processes (*n_jobs*) the algorithm is allowed to use; -1 represents an infinite of processes. The default setting is 1.

We will tune parameters "by hand" to better understand the algorithm.

As we said previously, the classification problem is not always useful to compute the accuracy score (percentage of correct predictions). Mostly, we will look at the area under the curve (AUC) to analyse our model performances but we will use accuracy for some, when it's more complicated to define a parameter with the AUC. First we will see how these metrics evolve with the number of trees per random forest.

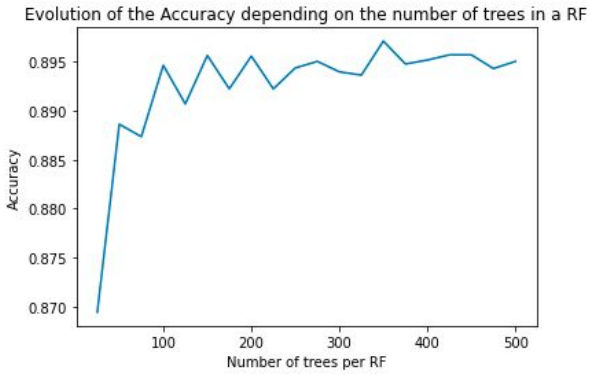


Figure 2 : number of trees per Random Forest and the accuracy

For a better accuracy we kept the value 500 for this parameter.

Then we focused on the parameter *min_samples_leaf* to see how it impacts the AUC. We choose a range from 1 to 110. So increasing the number of samples required to be at a leaf node decreases the AUC.

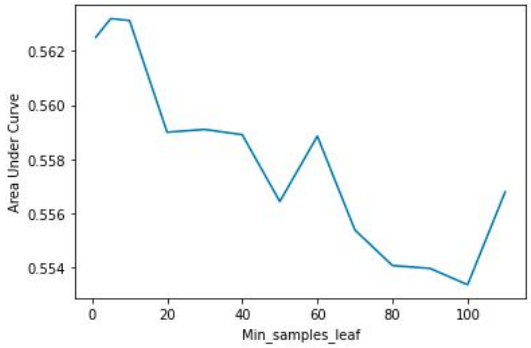


Figure 3 : Minimum number of samples in a leaf and AUC

The relationship is negative and we will keep the default value for this one.

For the maximum depth of the tree we keep the same range to see how accuracy score and AUC evolves with this parameter.

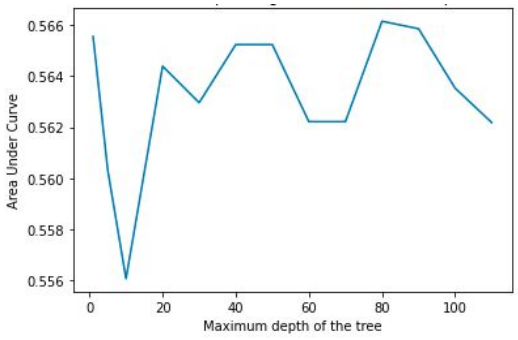


Figure 4 : Maximum depth of the tree and AUC

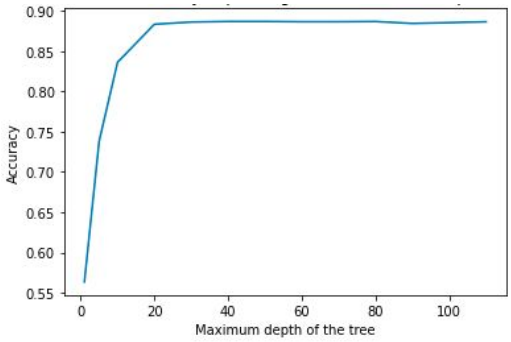


Figure 5 : Maximum depth of the tree and accuracy score

This increasing relationship with accuracy is quite logical since a deeper tree can fit more accurately the data. We will keep the value 40 for this parameter since the accuracy doesn't increase after this value and also because the AUC is at its higher point .

The next parameter is the minimum number of samples required to split an internal node. We take a range between 2 and 110. We got the following relationship between accuracy and the parameter :

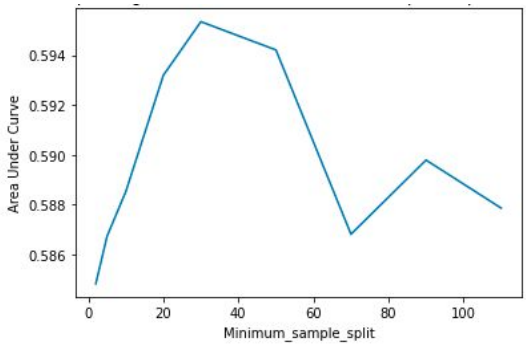


Figure 6 : Minimum sample split and Area Under Curve

We may take the value of 30 for this parameter.

For the maximum number of features to consider in our random forest we have this result regarding the Area Under the Curve (that we want to maximize). For this parameter we will choose the default setting.

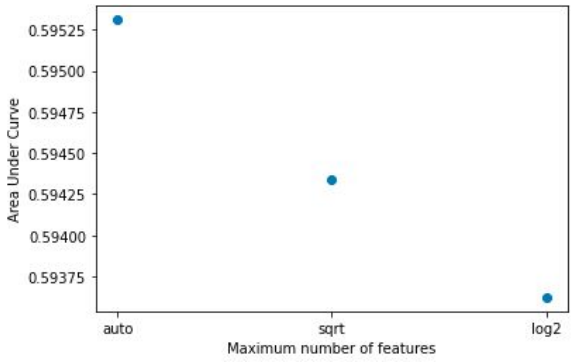


Figure 7 : maximum number of feature and value of the AUC

To summarize, the hyperparameters chosen in the final random forest will be : *n_estimator* = 500 ; *max_features* = 'AUTO' *; max_depth* = 40 *; min_simples_split* = 30 *; min_samples_leaf* = 1*;*

An interesting part of random forest is the concept of feature importance which is calculated as the decrease in the impurity of the node. For each feature it collects the average of the impurity decrease. The average over all trees in the forest is the measure of the feature importance. Our final random forest with all previous hyperparameters will be presented in the section 'Result'.

3.8.2. Support Vector Machines

Support Vector Machines used for classification tasks are machine learning models whose aim is to find the best hyperplane, also known as the decision boundary, separating the classes in the target variable.

The Support vectors will be data-points that are located nearest to the separating hyperplane. Removing them would inevitably alter its position. In this context, these are the most important samples that define the orientation and location of the best decision boundary. The goal is to maximize the SVM Margin, the distance between the hyperplane and its nearest data points.

We used the *Support Vector Classifier* built in the Sklearn.SVM package in Python.

Before parameter tuning, we ran the algorithm using all kernels available to construct the hyperplane, the linear, polynomial and radial basis function. These kernels differ on the kind of separation they achieve on the data. The linear one constructs a linear hyperplane, while the poly and rbf ones use non-linear hyper-planes.

The best results were obtained with the *poly* kernel hyperplane, so we proceeded with the fine tuning of its parameters using a chunk of 50.000 observations from the train set and 15.000 observations from the test set. We have presented the tuned parameters and results in Section 4.

3.8.3. K-Nearest Neighbors

K Nearest Neighbors (KNN) is a supervised machine learning algorithm that classifies individuals based on their neighbors. For example, if a new data is surrounded by data that has 'Click' as target variable, it concludes that the new data belongs also to the class 'Click'. So we have to choose the K nearest neighbors to look at. Then it computes the distance between the new data point and the K others. Several metrics exist. Among all that exist we have tested the standard Euclidean distance, the Manhattan distance, the Hamming distance and finally the Canberra distance. After computing a chosen distance, the algorithm keeps a list of all the distances and sorts them (in ascending order) in order to take the K first elements, the K neighbors that are the closest to the new data.

We wanted to test various parameters : the type of distance and the number of neighbors.

Now, how do we choose the value of K ? The common way is to use the Elbow Curve. We take a range for K, going from 1 to 20. For each K neighbors we compute the error rate (the number of all incorrect predictions divided by the total number of the dataset). Unfortunately our elbow curve was absolutely not angled so we tested different values for distance and number of neighbors parameters.

We will present the KNN results in the 'Results' section.

3.9. Improve Results

A common way to improve the performance of the above predictive techniques, is to apply a set of powerful Machine Learning Algorithms that combine several techniques, based on the principle of ensemble. Our chosen methods are Gradient Boosting Classifier, whose results we then improved by using XGBoost Classifier. We have also applied Support Vector Machines as well as Artificial Neural Networks to iterate through the results and conclude on the best adapted technique to our dataset and study question.

3.9.1. Gradient Boosting Classifier

Gradient boosting is a sequential machine learning technique used for regression and classification problems and is based on the principle of ensemble. It combines a set of weak learners and delivers improved prediction accuracy. It is typically used to discover non-linear relationships between variables in supervised learning. Like other boosting methods, it builds the modes in an iterative fashion and achieves optimization including an arbitrary loss function. It generally involves three elements:

1. Optimization of a loss function
2. A weak learner to make predictions
3. An Additive model to add weak learners to minimize the loss function.

To properly use the Gradient Boosting Classifier, we fine tuned the parameters to best fit our data. To save computational time, we run the iterations on chunks of 50.000 observations from the training set and 15.000 observations from the test set. At every iteration, we varied one parameter at different rates, while keeping the others unchanged. We used the Area Under the Curve Score as a valuation metric then iterated one final time with the optimal hyper parameter values to obtain better results.

The Learning Rate shrinks the contribution of each tree and we varied it between 0.05 and 1.
Number of Estimators, the number of trees in the forest, the higher the number of trees the better to learn the data, but the slower the computation process. We iterated the number of estimators ranging from 1 to 100.
Defining Tree-Specific Parameters:

Maximum Depth indicates how deep the tree built can be, The deeper the tree is, the more splits it has and it captures more information about the data. We observed that at high numbers, the AUC on the test data was not necessarily increasing, so we iterated it between only 1 and 5.

Minimum samples split indicates the minimum number of samples required to split an internal node, we varied it between 0.1 and 1, with 10 separate splits.

Min samples leaf indicates the minimum number of samples required to be at a leaf node, we varied it between 0.1 and 0.5, with 5 separate splits.

Max_features indicates the number of features to consider when looking for the best split, we vary it in the range of 1 and 27.

The loss function is a miscellaneous parameter that affects overall functionality as it refers to the loss function to be minimized in each split. Given the complexity of the study, we will keep the default value to start, which generally works well for classification problems, as other values should be chosen only if we fully understand their impact on the model.

3.10. XGBoost Classifier

Gradient Boosting Decision Trees and Logistic Regression is a widely used method in CTR prediction. XGBoost stands for "Extreme Gradient Boosting Classifier" and uses a Gradient Boosting framework. The objective and loss function at iteration $\boldsymbol{t}$ that is minimized is as follows:

$$L^{(t)} = \sum_{i=1}^{n} l(y_i, \widehat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t)$$

Where y_i is the real value (label) known from the training data-set and $y_i, \widehat{y}_i^{(t-1)} + f_t(x_i)$ can be seen as $f(x + \Delta x)$ where $x = \widehat{y}_i^{(t-1)}$.

We will use the *XGBoostClassifier()* from the xgboost package in python with the hyperparameters we tuned in the Gradient Boosting Classifier Algorithm to see how results are improved.

3.11. Unsupervised Learning

3.11.1. K-Means Clustering

Apart from supervised learning predictive algorithms, we decided to also apply unsupervised learning methods to the dataset, observe, improve and evaluate results. The algorithm we decided to go with is K-Means, a partitional clustering method which divides objects into non overlapping groups. The framework the algorithm is based on is the following:

1. Choose the number K of clusters.
2. Randomly select K distinct points, names the initial centroids, representing the center of the cluster.
3. Compute the Euclidean distance between the first data point and the first K centroids. The process is repeated for all data points.
4. Measure all distances between points and initial centroids and assign each point to the closest centroid, creating the initial K clusters.
5. Calculate the mean of each cluster and repeat step 3 and 4 in order to minimize the in-class variance and maximize the intra-class variance. The iteration is repeated until there is no change in the centroids and clusters we formed.

An important constraint within our dataset is that most of the features are categorical, resulting in over 5000 dummy variables once we leave one out to remove feature collinearity. We decided to extract only the features that we assumed categorical, whose coefficients were also stable when we ran Lasso Regression. As this is a binary classification problem, our objective is to try and divide the data in two clusters

To minimize computational time, we took a chunk of 50.000 observations from the test set to apply the clustering..As always, we did feature scaling beforehand and applied Kmeans to our test set, as the train set we have been using to train the models in the supervised learning methods has synthesized observations.
We used the MinMaxScaler() from the sklearn package and decided to standardize all numerical and categorical features.

The SSE is defined as the sum of the squared Euclidean distances of each point to its closest centroid. The objective function is therefore to minimize this value, the parameters used are as follows:

- *init* : control the centroid initialization technique
- *n_clusters* : sets the number of clusters, we will set it to
- *n_init* : sets the number of initializations to perform. The default behaviour is to run 10 k-mean and return the results obtained from the one with the lowest SSE.
- *max_iter* : sets the number of maximum iterations for each initialization of the algorithm.

In the graph of the elbow, if it is not very clear at the number of clusters that the curve starts to bend, using the *KneeLocator()* from the kneed package computes the optimal number of clusters that minimizes the errors.
An alternative to the elbow method is the silhouette coefficient, which is a measure of cluster cohesion and separation. It finds how well a datapoint fits into its assigned cluster by using two evaluation factors: How close the data point is to other points within the cluster. It ranges between -1 and 1 and the higher this value, the closer the samples are to their clusters compared to other clusters.

The K-means purpose is to minimize inertia, or the within clusters sum of squared criterion:

$$\sum_{i=0}^{n} \min_{\mu_j \in C}(||x_i - \mu_j||^2)$$

Inertia recognizes how internally coherent the clusters are, the lower it is, the more compact and distinguished the clusters are. Through visualization we can also observe how compact the created clusters are.

3.12. Artificial Neural Networks

Neural Networks form the base of deep learning which is a subfield of Machine Learning. Algorithms are inspired by the human brain. Neural networks take data and train themselves to recognize some patterns in the data. Finally it predicts an output.

A neural network is composed of several layers of neurons. First have the input layer receiving the inputs. And at the end the output layer. In between we have hidden layers which performs most of the computations required by the network. Channels connect neurons from one layer to another. Each channel has a weight and each input is multiplied by its corresponding weight. The resulting sum is the new input of the next hidden layer.

We created a neural network model using Keras and stratified k-fold cross-validation from scikit-learn to evaluate it, which is a resampling technique that estimates the performance of the model.

The process is the following:

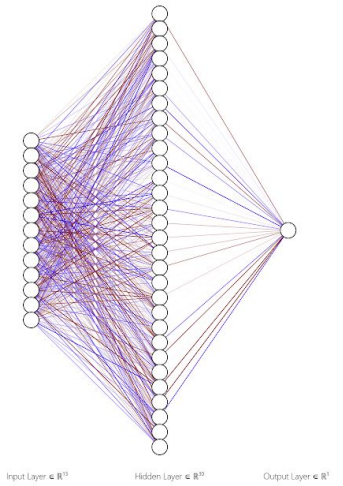
1. It splits the data into K parts.
2. It trains the model on all parts while leaving one out.
3. It tests the model on the fold left out.
4. Repeats the process K-times until all folds have been used once for testing.
5. Average score across all models is used as a performance metric.

The model looks at the output values and attempts to balance the number of instances belonging to each class in the k-splits of the data.

The KerasClassifier wrapper takes a function and returns our neural network model, it takes arguments that it passes along to call the fitting for the 10 epochs that we defined and the batch_size equal to 5.

The initial model that we created has a single fully connected hidden layer with 60 neurons from our 27 input feature.The initial weights are defined using a small Gaussian random number and then we used the Rectifier activation function. Finally, the output layer returns one single neuron in order to make predictions.

The probability output is estimated using the sigmoid activation function, which is then converted to class values.During training, we used the logarithmic loss function, the binary_crossentropy, which is the best suited function for binary classification. The efficient Adam optimization algorithm for gradient descent was used to collect the accuracy metrics when the model was trained.



The model achieved 75.26% Accuracy on the synthetized training set, with a standard deviation of 1.41%, where we took a sample of 50.000 observations, while when trained on a sample of 50.000 observations of the test set, it achieved 96.63% accuracy with 0.01% standard deviation. Here again, we meet the accuracy paradox.

Figure 8 : Fully connected 1 Hidden Layer Neural Network

To achieve better results, we added one fully connected layer to the model, resulting in a MLP (Multi-Layer Perceptron), containing 30 neurons. With 10 epochs, the MLP model scored an accuracy of 76.7% on the same 50.000 observations sample from the test set.

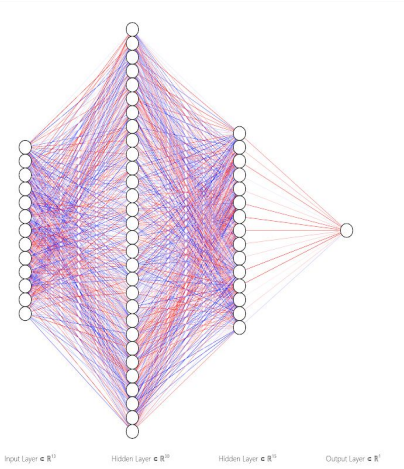


Figure 9 : Fully connected 2 Hidden Layers (60 & 30 neurons) Neural Network

As previous papers have concluded, Deep Neural Networks are the most precise mechanisms on prediction tasks as they can model complex relationship patterns between features. But to further improve precision, there is a high trade-off with interpretability. As we increase the number of layers and depending on the number of neurons they have, the model becomes of a complexity of which the results are not understandable by the human mind. Therefore, the model is useful when we have a large set of data available and works.

4. Results

4.1. Descriptive Statistics

In order to do a complete analysis of the dataset and the study question, descriptive statistics are indispensable. One of the main constraints in this study was that the feature values were all encoded, thus descriptive statistics on the numerical features are to be interpreted in function of their encoding. The table summarizes the basic statistics: Mean, Standard Deviation, Minimum value etc.

Feature	Mean	SD	Min	25%	50%	75%	Max
Age	5.05	1.41	1	6	7	8	9
City Rank	2.86	0.9	1	2	3	4	4
App Storage Size	8.55	5.89	1	6	7	13	23
Device Size	158.13	44.93	102	141	141	16	349
Device Price	5.47	1.15	1	5	6	6	9
Membership Level	1.24	0.65	1	1	1	1	5
Membership Lifecycle	1.02	0.72	1	1	1	1	22
Average Online Communication Rate	13.14	1.65	1	12	13	14	15

When looking for correlations between the numerical features, using the Standard Pearson Coefficient, the coefficients remain low and insignificant. We have visualized them in Figure 10.

Figure 10 : Correlation matrix between numerical features

4.2. Click Through Rate prediction Results

A priori we can't say in advance which algorithm will perform better. First we tried four logistic regression using different penalization terms (A classic Logistic Regression, then using Ridge regularization term, then another

using LASSO and finally using Elastic Net). Secondly, we tried with two non linear algorithms (KNN and SVM). Finally, we applied Random Forest, XGboost and neural network algorithms.

In this section and in the following ones we will look at the results we obtained, compare the Area Under Curve values and look at the False positive rate (that we want the smallest as possible).

4.2.1. Logistic regression Results

For the classic logistic regression (without regularization), we found that 62% of clicked ads are well predicted. The AUC score is 0.569. This is a good result to begin with.

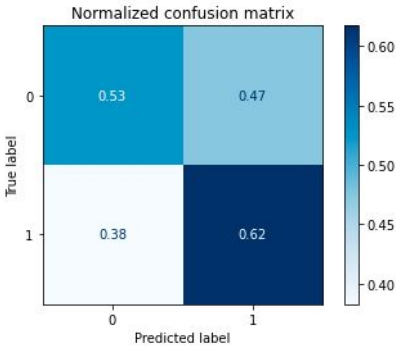


Figure 11: Confusion matrix in a simple logistic regression

The true negative rate can be considered as high (being almost as important as the true negative rate) but with regard to our topic, the most important thing is to minimize the predicted ads 'not clicked' but which are in fact 'clicked'.

4.2.2. Ridge Regression Results

The Ridge penalty will shrink the coefficients related to the input features towards 0, but they will never become 0. The graph below shows that.

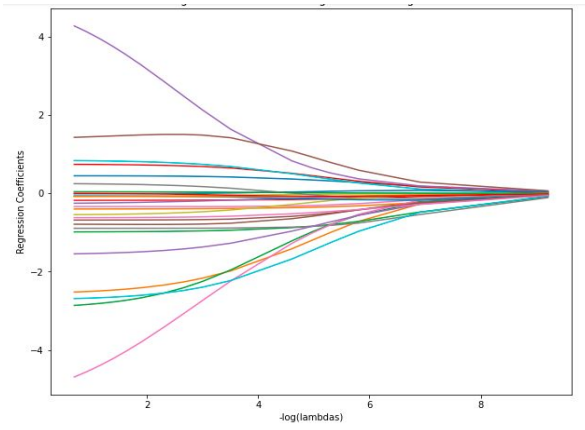


Figure 12 : Ridge coefficients as a function of lambda

We seek to maximize the AUC by tuning the value of the parameter λ from 0 to 1. For $\lambda = 0.001$ the AUC is maximized and equals to 0.573.

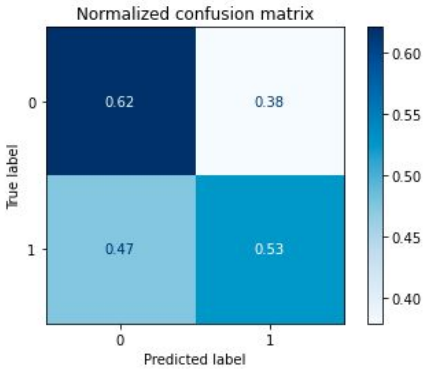


Figure 13 : Confusion matrix with L2 penalization term

The AUC is a bit better, but at the same time the number of ads predicted 'not clicked' but actually 'clicked' is higher.

4.2.3. Lasso Regression Results

As we have gone through different values of λ, we have presented the different estimated values for the corresponding betas related to the input features in the graph below:

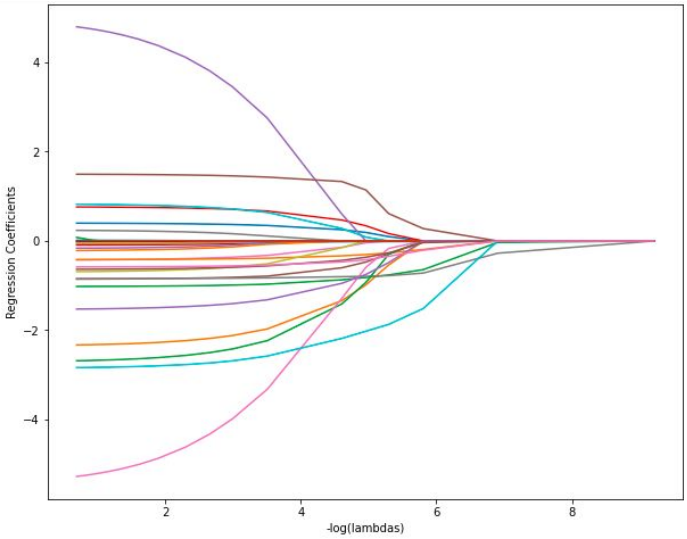


Figure 14 : Lasso coefficients as a function of the regularization

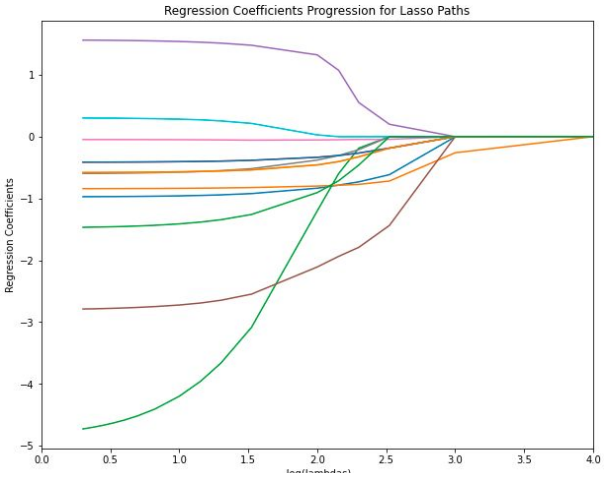


Figure 15 : Feature selection with LASSO

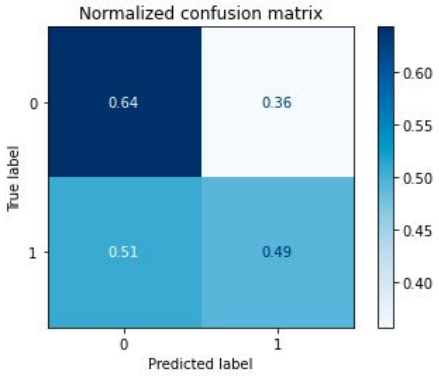


Figure 16 : Confusion matrix with L1 penalization term

As shown in the previous graph, when λ increases, the beta coefficients are skrinked toward 0 and equal 0 at a certain threshold. Variables whose coefficients later converge to 0 are those that the feature selection method will choose (*Figure 15*). We should keep the *'slot_id', 'city_rank', 'app_score', 'gender', 'his_on_shelf_time', 'device_price', 'up_life_duration', 'indu_name', 'pt_d', 'communication_avgonline_30d'* and *'emui_dev'*.

Tuning the λ parameter in this case leads us to a $\lambda = 0.005$. It maximizes the AUC at a value of 0.567. We obtain the following confusion matrix :

Moreover , in this second graph we can see the evolution of the AUC value depending on λ.

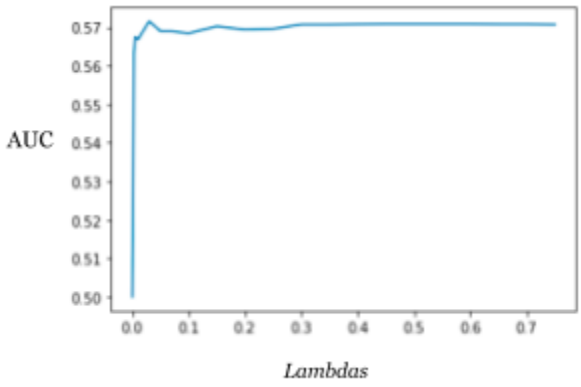


Figure 17 : AUC and lambda values LASSO

The one that maximizes the AUC is $\lambda = 0.005$. For the features selected, for example, we can think that the user's gender can clearly influence whether he clicks or not. The propensity to click on an ad is influenced by the user's gender. Also the place where the advertisement is positioned. For example if the advertisement is placed at the top of a page, it will be more visible (so probably more 'clickable'). Also if an application is well rated, it will be much more downloaded. The ads that will appear in this one will be potentially more clicked.

So for both LASSO and Ridge the improvement from a classic Logistic Regression using penalty is not significant as the value of lambda has to be very low to maximize the AUC.

4.2.4. Elastic Net Regression Results

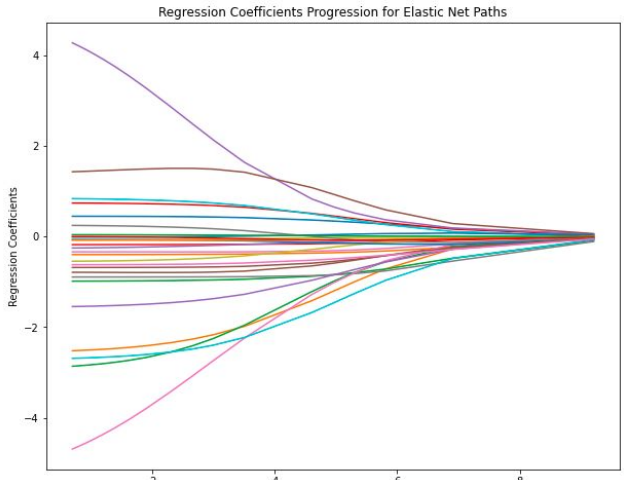


Figure 18 : Coefficients as a function of lambda

We set a range from 0.001 to 0.75 for the regulation strength and from 0 to 1 for the mixing parameter. We tested 198 combinations and picked the one that maximizes the AUC.

As a result the optimal α is equal to 0.03 and the l1 ratio to 0.8.

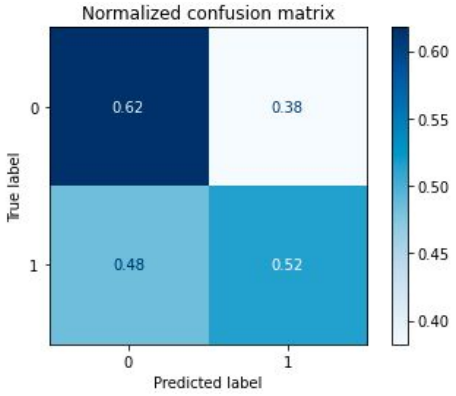


Figure 19 : Confusion matrix with L1 and L2 penalization terms

This means that we are closer to LASSO than to Ridge. For this value of parameters, the AUC is equal to 0.566 which is good but not as good as the simple logistic regression one.

4.2.5. Support Vector Classifier Results

Going in depth, the optimal value for the degree of the polynomial used to find the hyperplane to split the data was of 6, while the penalty parameter C of the error term, which controls the trade off between smooth decision boundary and classification of all training points correctly, gave the best results when set to 10. The algorithm

was so set with these values for the degree and C, in combination with the rest of the parameters set to their default values.

We then trained and tested the algorithm in a set of 150.000 observations from the training set and 50.000 observations from the test set, as it was computationally costly to run it in the original synthetized training set of nearly 1.213.000 observations and the original test sample of nearly 356.000 observations.

The model resulted in 53% Area Under the Curve Score in the test set. The Receiver operating curve is presented below:

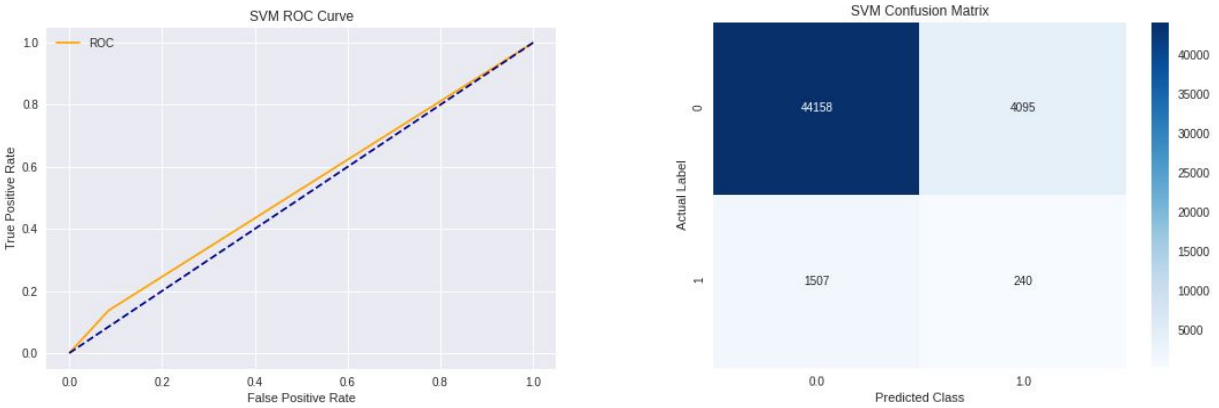


Figure 20 : ROC curve and confusion matrix of SVM classifier

4.2.6. K-Nearest Neighbors Classification Results

For the closest K neighbor algorithm we tested several parameters. The calculation of the distance, the number of neighbors and leaf size.

After a long computational time we find the following optimal values for those parameters :
Number of neighbors = 13 ; leaf size = 30 and metric = manhattan

We wanted to test other distances like the Hamming distance or the Canberra distance. So we test for different values for these parameters. The resulting best combination is : number of neighbors = 15 ; leaf size = 20 and metric = manhattan. This combination gives us an AUC equal to 0.587.

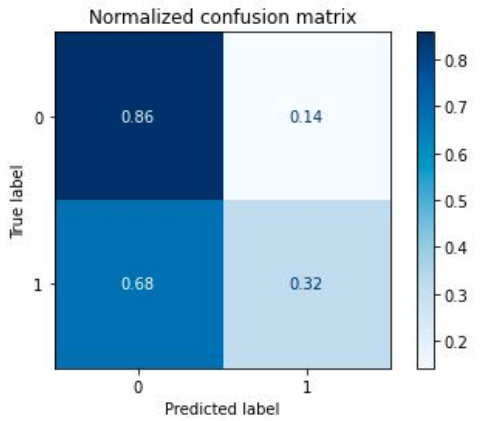


Figure 21. KNN Classification Confusion Matrix

In this case, even if we have a high AUC (compared to other methods), we have to look at the false positive. Here the percentage of false positive (an ad predicted clicked but not actually clicked) is pretty high (0.68%). The value of the AUC reflects in a way the fact that the algorithm predicted the unclicked ads rather well (0.86%).

4.2.7. Decision Tree Classifier Results

For this simple decision tree we use all default settings to see what result we have. The confusion matrix is the following one :

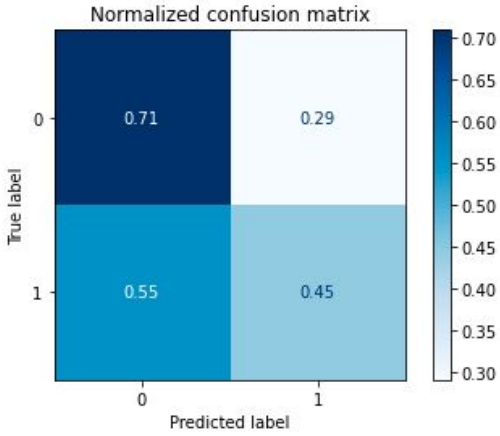


Figure 22. Confusion matrix for the Decision Tree Classifier

The AUC equals to 0.577. Regarding the confusion matrix, the tree is pretty good at predicting unclicked ads. For clicked ads, it has more trouble but it's close to 50-50.

4.2.8. Random Forest Classification Results

In the previous section we saw that it is possible to see the features importance in a random forest. Here our results :

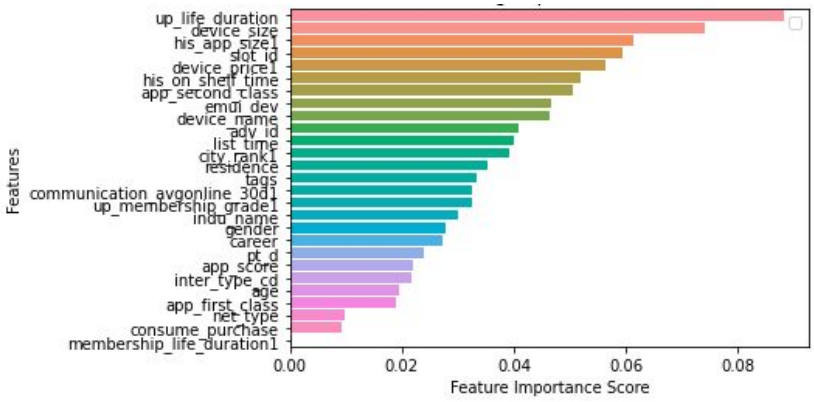


Figure 23 : feature importance with random forest

The larger importance is for *up_life_duration*, and *device_size*. As a reminder, *up_life_duration* is how long the user has been using huawei. The more you are used to using a device brand (here Huawei) the easier it will be for the user to click on an ad. But as you can see there is no big difference in importance between most of the features.

Also, using all the tuned hyperparameter (*n_estimator* = 500 ; *max_features* = 'AUTO' *; max_depth* = 40 *; min_simples_split* = 30 *; min_samples_leaf* = 1) we obtain the following results :

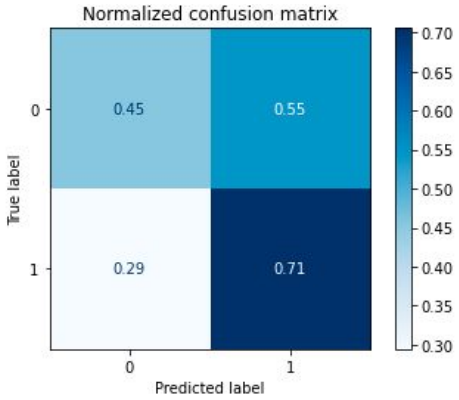


Figure 24. Random Forest Classifier Confusion matrix

The random forest is able to predict that an ad will be clicked more than 70% of the time. This is a good improvement from the simple decision tree. Moreover the AUC is equal to 0.66 which is the best result so far.

4.2.9. Gradient boosting Classifier Results

When fine tuning the hyperparameters, the optimal results that maximized the AUC in the test set, we have presented in Figure , and the parameters are as follows:

- *Learning Rate* shrinks the contribution of each tree and we varied it between 0.05 and 1. The best results were obtained at 0.75.

- *Number of Estimators*, the number of trees in the forest. We iterated in a range from 1 to 100 and obtained the highest accuracy at 8 estimators on the test dataset.

Defining Tree Specific Parameters:

- *Maximum Depth* indicates how deep the tree built can be. We observed that at high numbers, the AUC on the test data was not necessarily increasing, so we iterated it between only 1 and 5 and the highest AUC was obtained at 5.

- *Minimum samples split* indicates the minimum number of samples required to split an internal node, we varied it between 0.1 and 1, with 10 separate splits. The minimum number of samples which gives the highest AUC in the test set is thus equal to 0.1.

- *Min samples leaf* indicates the minimum number of samples required to be at a leaf node, we varied it between 0.1 and 0.5, with 5 separate splits. The highest AUC score was obtained at a minimum number of samples leaf at 0.4.

- *Max_features* indicates the number of features to consider when looking for the best split, we vary it in the range of 1 and 27. The highest AUC score on the test set was obtained at 21 features, thus that is the number as a reference value.

The loss function is a miscellaneous parameter that affects overall functionality as it refers to the loss function to be minimized in each split. Given the complexity of the study, we will keep the default value to start, which generally works well for classification problems, as other values should be chosen only if we fully understand their impact on the model.

The Gradient Boosting Classifier set with the optimal values for the parameters achieved an accuracy score of 64% on the test set. However, the classifier still had difficulty correctly classifying the users that had actually clicked on the advertisement, resulting in a 0.56 Area under the Curve. The corresponding confusion matrix is as presented below:

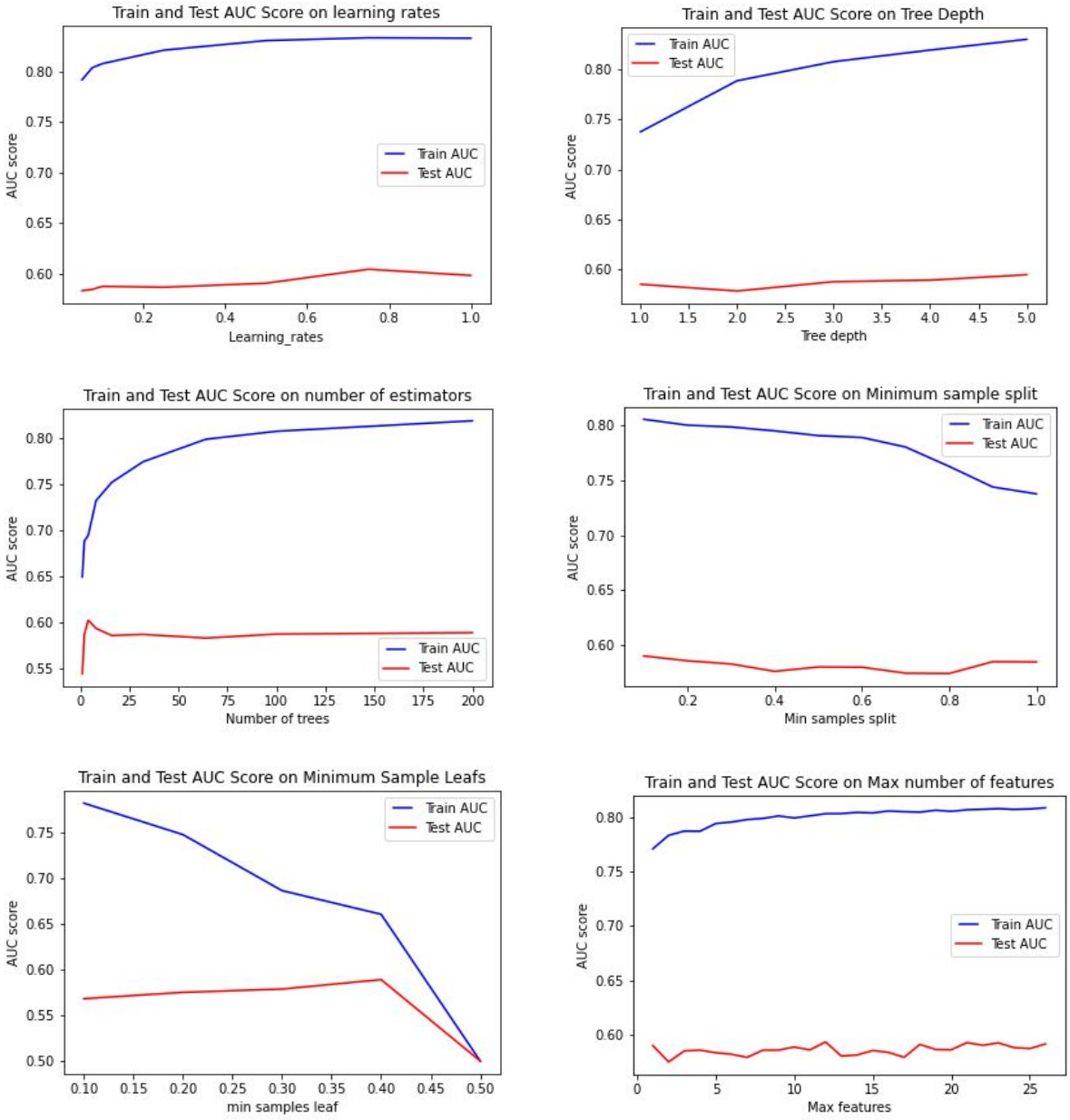


Figure 25. AUC plots on train and test sets for different parameter values of the Gradient Boosting Classifier.

The Gradient Boosting Classifier set with the optimal values for the parameters achieved an accuracy score of 64% on the test set. However, the classifier still had difficulty correctly classifying the users that had actually clicked on the advertisement, resulting in a 0.56 Area under the Curve. The corresponding confusion matrix is as presented in Figure .

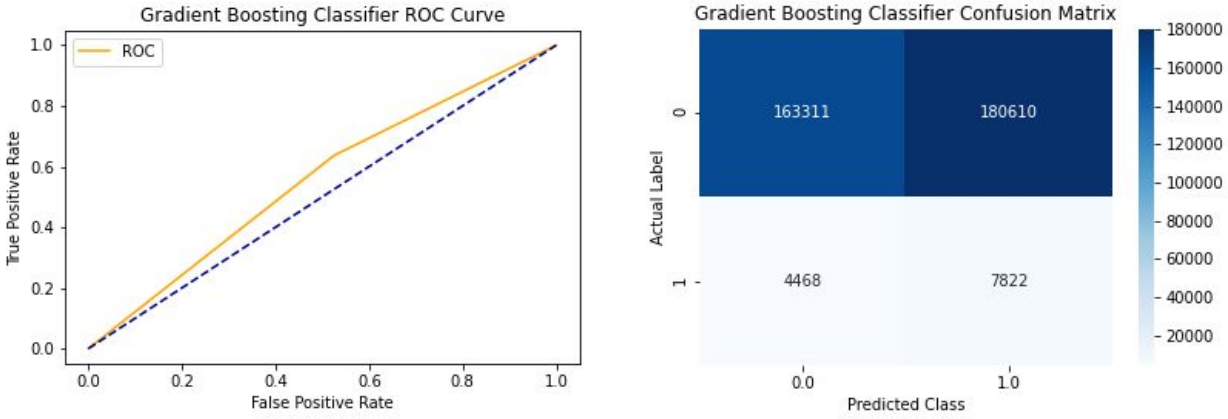


Figure 26. ROC-AUC Curve and corresponding Confusion Matrix for Gradient Boosting Classifier

4.2.10. XGBoost Classifier Results

Using the XGBoostClassifier() from the xgboost package in python and the same optimal hyperparameters that we used to run the Gradient Boosting Classifier on, we obtained a 87% Accuracy Score on the test set, with an Area under the curve of 0.58.

The hyper parameters used are as follows:

- *The Learning Rate* : 0.75
- *Number of Estimators* : 8
- *Maximum Depth* : 5
- *Minimum samples split* : 0.1
- *Min samples leaf* : 0.4

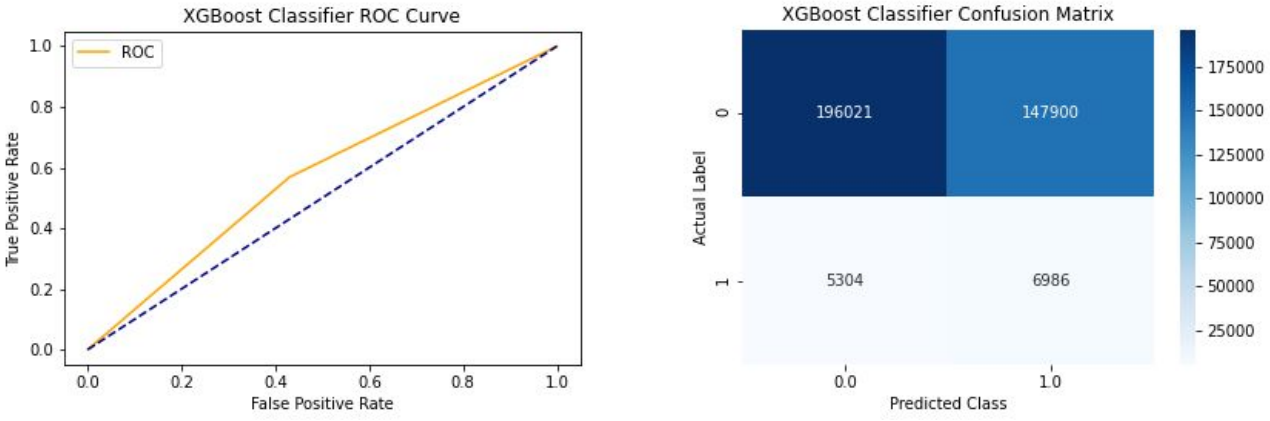


Figure 27. ROC Curve and Corresponding Confusion Matrix for XGBoost Classifier

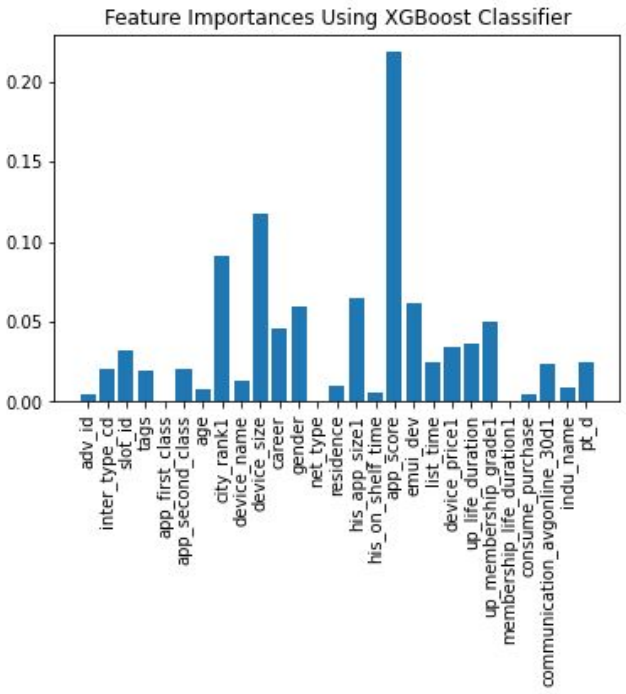


Figure 28: Feature importances using XGBoost Classifier

The *feature importances* function from the algorithm gave us histograms of the feature coefficients estimated when the classification model was trained. It resulted that the most important feature in predicting the Click-Through-Rate of Huawei users in advertisements was the *application score*, followed by the *device size*, *city rank* as the three most important predictors. Scrolling down on feature importance, we find *application storage size, emui version, gender, career* and the *service membership level* as predecessors.

The application score is an indirect reflection of how much users trust it. Applications with high scores have also more propensity to succeed, thus also having more probability to be able to choose through the advertisements they serve their users, and at the same time having the means to target their users with customized content, in the purpose of improving their own CTR.

Device size is unsurprisingly the second most important predictor in CTR, because it reflects clearly the mobile screen size. (Have to look onto the coefficient sign to interpret)

City Rank reflects the level of the resident city of a user, typically, users living in developed cities are more likely to be interested in services or products online because they are accessible to them, while residents in areas with lower urban development are less likely to click on an advertisement, as it might be harder or longer of a process to reach the final stage of the supply chain.

What is interesting in our findings is that *age* is amongst the most useless features in the model, suggesting that there is a homogeneous spread across all ages on their activity regarding online advertisements.

4.3. Unsupervised Learning Results

4.3.1. KMeans Clustering Results

In the graph of the elbow, if it is not very clear at the number of clusters that the curve starts to bend, using the KneeLocator() from the kneed package computes the optimal number of clusters that minimizes the errors.
An alternative to the elbow method is the silhouette coefficient, which is a measure of cluster cohesion and separation. It finds how well a datapoint fits into its assigned cluster by using two evaluation factors: How close the data point is to other points within the cluster. It ranges between -1 and 1 and the higher this value, the closer the samples are to their clusters compared to other clusters.

With the silhouette coefficient, we can see that it is maximized when the number of clusters equals 5.

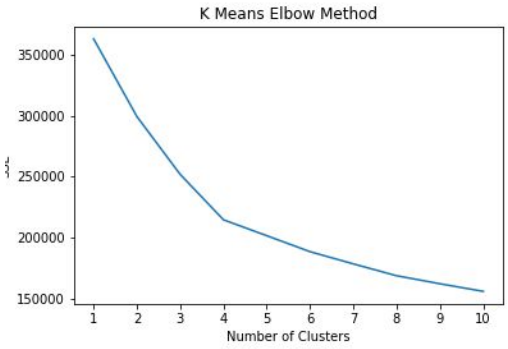


Figure 29: K-Means Elbow method

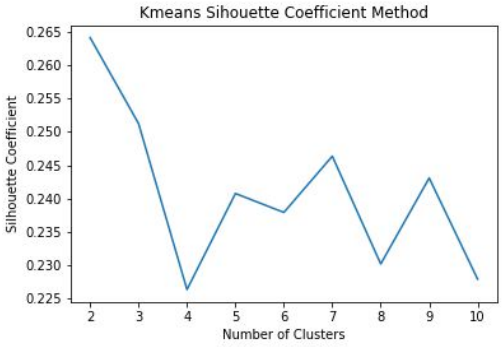


Figure 30. K-Means Silhouette Coefficient

Inertia recognizes how internally coherent the clusters are, and in our case, with 2 clusters, it is of 10655.49 and was achieved at 5 total iterations.
Nevertheless, it is a binary classification problem, so we decided to go with the number of clusters equal to 2. After running the DBSCAN, we obtained the following plots:

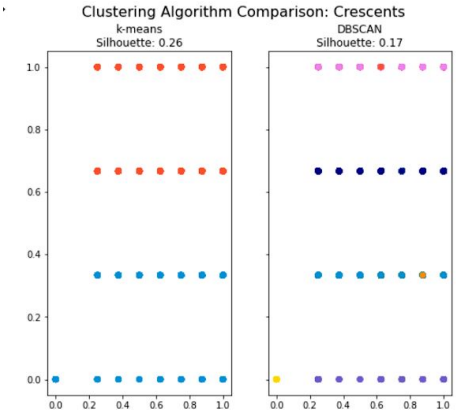


Figure 31. K Means and DBSCAN using 2 Clusters

We can clearly see that there are 4 distinguishable clusters, and when looking for 2 separate clusters, Kmeans is not able to perform well when it is run on all our numerical features.

We decided to create clusters using K Means on the features *Application Storage Size* and *Device Size* and apply the same framework.

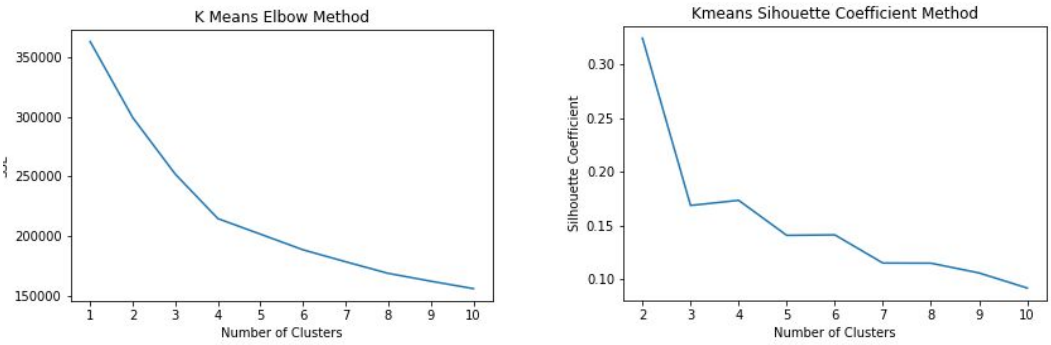


Figure 32. K-Means Elbow method and Silhouette Coefficient using 2 features

The KneeLocator() function outputs an optimal k of 4, as observable in the Elbow graph as well. With 4 clusters inertia of 62963.84 achieved by the 9th iteration. This higher inertia is a result of using only these two features, and that we have few classes within our features which are not continuous. This makes it difficult to define crisp clusters.

When we visualize, we can see that the clusters are highly heterogeneous and points in different clusters are laying on top of each other. This method was thus not effective in distinction of clusters in datasets where numerical features have few classes and underlying relationships are complex to uncover.

Figure 33. K Means and DBSCAN using 4 Clusters on 2 Features

5. Comments and Conclusions

In this report we have explored techniques and methods to better predict the click on an advertisement. The best model to predict if an advertisement will be clicked or not, was a Deep Neural Network with two hidden layers, but the results of which are uninterpretable and the output we obtained is the accuracy score and the standard deviation. Following, the second best model was the Random Forest giving both a high Area Under the Curve and a rather low false positive rate. Some of the algorithms were time consuming and we had to take smaller chunks in order to minimize computational time. Nevertheless, the results we obtained using K-Means Clustering and supervised KNN were useless in terms of CTR prediction.

We used Automatic Feature Selection techniques to observe which of the features were stable and which converged to 0 as we introduced a penalization term to the objective function. The purpose of the study was exploring these techniques and the ML models separately in regard to responding to the research question.

A hybrid model using both, trained on a larger set of data would have given a better accuracy when tested.

The main constraints in the research were the high imbalance between the target classes, with only 3% CTR, as well as the data being encoded. We had to make assumptions on which were the numerical features and which were categorical, along with the line of ordering of values in the numerical features. Otherwise, coefficient and feature interpretation would not have been possible.

Another limit to the research is that not all information is provided and there might be omitted features that have a significant impact on a user's propensity to click on an ad, thus limited resources regarding the study question preclude high accuracy on our predictions.

References

[1]Kaggle Dataset
https://www.kaggle.com/louischen7/2020-digix-advertisement-ctr-prediction

[2]Random Forest : Feature importance default
https://explained.ai/rf-importance/

[3]Tanagra Data Mining : Regression Lasso sous Python
http://eric.univ-lyon2.fr/ ricco/tanagra/fichiers/frTanagraRegressionLassoP ython:pdf

[4]Factor Analysis
https://www.datacamp.com/community/tutorials/introduction-factor-analysis/

[5]StatQuest : Elastic Net
https://www.youtube.com/watch?v=1dKRdX9bfI0

[6] KNN in Sklearn
https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

[7] SVM in Sklearn
https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

[8] KMeans in Sklearn
https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html

[9] A. Moneera, Maram and al., 'Click through Rate Effectiveness Prediction on Mobile Ads Using Extreme Gradient Boosting' in Computer, Material and Continua, december 2020.

[10]
https://www.alpha-quantum.com/blog/ctr-prediction/ctr-prediction-using-hashing-trick-logistic-regression-sgd-and-only-simple-python/

[11] Deep Learning :
https://machinelearningmastery.com/binary-classification-tutorial-with-the-keras-deep-learning-library/