HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

MACHINE LEARNING REPORT
Prediction customer churn

Nguyen Huu Duan	20214951
Nguyen Dai Minh	20214965
Nguyen Duc Tam	20210767
Vu Ngoc Quyen	20214971

HANOI, 2023

Prediction customer churn

School of Information and Communication Technology – Hanoi University of Science and Technology

1. Introduction

Churn prediction is one of the most popular Big Data use cases in business. It consists of detecting customers who are likely to cancel a subscription to a service.

This project is a binary classification problem. If we focus on company goals, actually our problem is losing customers. In this situation the company needs to find churned customer in other words the customers that we will lose. Thus, the company will avoid losing profit and revenue.

In the recent years, churn prediction is becoming very important issue in the telecommunications industry. In order to deal with this problem, the telecom operators must recognize these customers before they churn. Therefore, developing a unique classifier that will predict future churns is vital. This classifier must be able to recognize users who have a tendency to churn in the near future, so the operator will be able to react promptly with appropriate discounts and promotions. Many models can handle the problem well. We recommend the very popular model of supervised learning - decision tree, random forest and some models based on gradient boosting with a huge improvement in algorithm optimization - Extreme Gradient Boosting (XGBoost) and Adaptive Boosting (AdaBoost).

In our project, we will use dataset from the open source Cell2Cell data prepared by the Teradata center for customer relationship management at Duke University. This dataset has 51047 rows, 58 columns in cell2celltrain.csv and Churn is the target variable. This column is Yes if the customer churns and No if customer doesn't churn. Through testing, we want to evaluate the effectiveness of the models and their optimization and suitability for churn prediction.

2. Telecom customers churn

The dataset comprises the activity related to monthly telecom communication such as minutes, the percentage change in minutes, peak call traffic, off-peak call traffic, overage minutes, and received calls. Moreover, the dataset provides records of the occupation, age of the household, and some information related to user conditions.

As with any Machine Learning approach, a processing procedure is required to prepare raw data for use in model training and testing processes. The pre-processing procedure carried out had as its objective the encoding of categorical data, the deletion or replenishment of missing data, the handling of outliers, and the scaling of features. Specifically, the pre-processing procedure performed is described below:

2.1 Data preprocessing

1. Handle missing data

After seeing data in dataset, we recognize that AgeHH (Age of Head of Household) can not have a value of 0. So if "0" appears in features AgeHH1, AgeHH2, it could potentially indicate a null or missing value. We exchange "0" for null value.

Figure 1 represents the number of missing values for features having null values. The list of features with missing values includes only two categorical variables, which are MaritalStatus and ServiceArea, while the remaining features are numerical.

- Handle missing values for categorical variables. Because the proportion of missing value of MaritalStatus is quite large (>30%), so we eliminated this feature. For ServiceArea, we filled this with the mode.
- Handle missing values for numerical variables. Our data has many columns with missing values, and our plan is to handle these columns depending on the number of missing values. But the proportion of missing values of numerical variables are very small (<1%) except AgeHH1 (<30%), so we filled these variables with the median. Since the percentage of null values for AgeHH2 and HandsetPrice is more than 50%, we have eliminated these features from the dataset.

2. Handle outlier for numerical variables

An outlier is an observation that is unlike the other observations, a sample that is exceptionally far from the mainstream of the data. Figure 2 shows that our data has many outliers, and our solution is to use Interquartile Range (IQR). The interquartile range is the difference between the $25th$ percentile ($Q1$) and the $75th$ percentile ($Q3$) in a dataset. It measures the spread of the middle 50% of values. The method is to declare an observation to be an outlier if it has a value 1.5 times greater than the IQR ($Q3 + 1.5\ IQR$) or 1.5 times less than the IQR ($Q1 - 1.5\ IQR$). And then, we will handle these outliers by capping the maximum value at $Q3 + 1.5\ IQR$ and the minimum value at $Q1 - 1.5\ IQR$.

In fact, our initial intention was to utilize the z-score as a means to eliminate outliers. However, upon examining the KDE (Kernel Density Estimation) of the characteristics, we discovered that some of the attributes were not normal distribution. As a result, we decided to switch to using the IQR to filter out the outliers.

3. Encoder categorical variables

Since the machine learning models only work with numerical variables, preprocessing the categorical variables becomes a necessary step. We need to convert these categorical variables to numbers such that the model is able to understand and extract valuable information. In this process, we encode all categorical variables except "Occupation" by Label encoding. For "Occupation", we encode it by One - hot encoding. In Label encoding, each unique category value is assigned an integer value. This kind of encoding is suitable for almost variables, because in these columns, there are only two unique values: "Yes" and "No", and Label encoding will be good to distinguish them. For "Occupation", this has many unique values, so Label encoding is not suitable. In this case, One – hot encoding is the best method, we add a new binary variable for each unique value in the categorical variable.

4. Feature scaling

In this step, Standardisation method was carried out so that all the variables would take values between 0 and 1. In this way, we can avoid the situation where variables taking values of great magnitudes having a greater influence on the application of machine learning algorithms. This method can apply for all situation, not like Normalisation which just applies when the all the features in the data set have the normal distribution

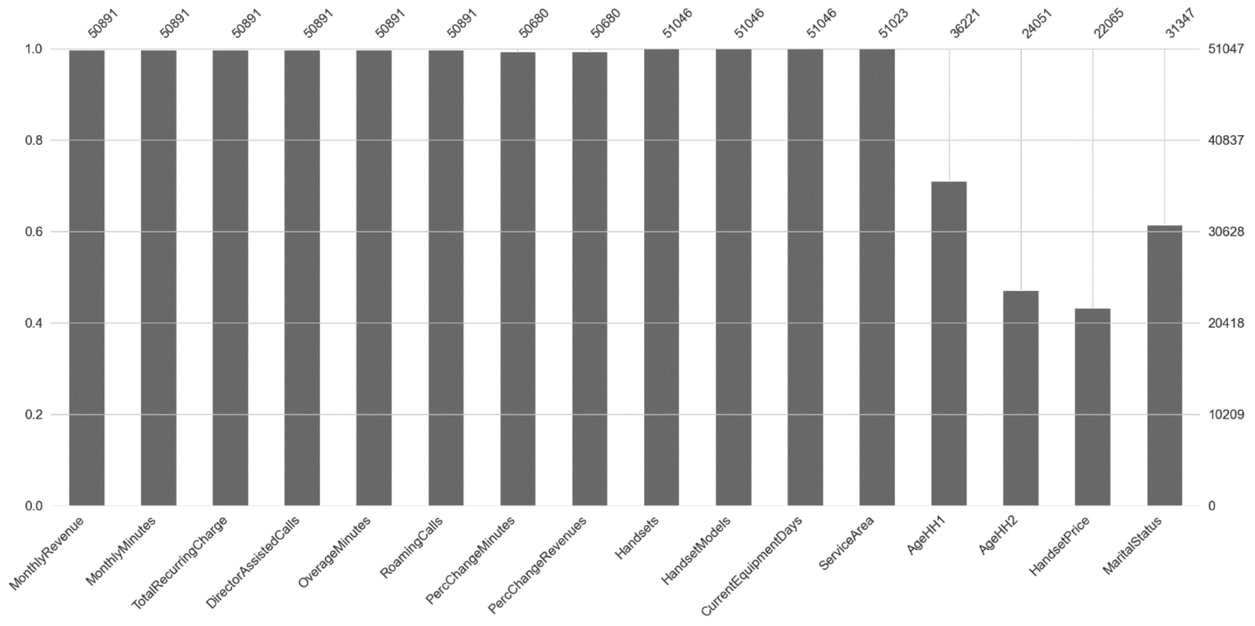


Figure 1: The number of non-null data for each variable

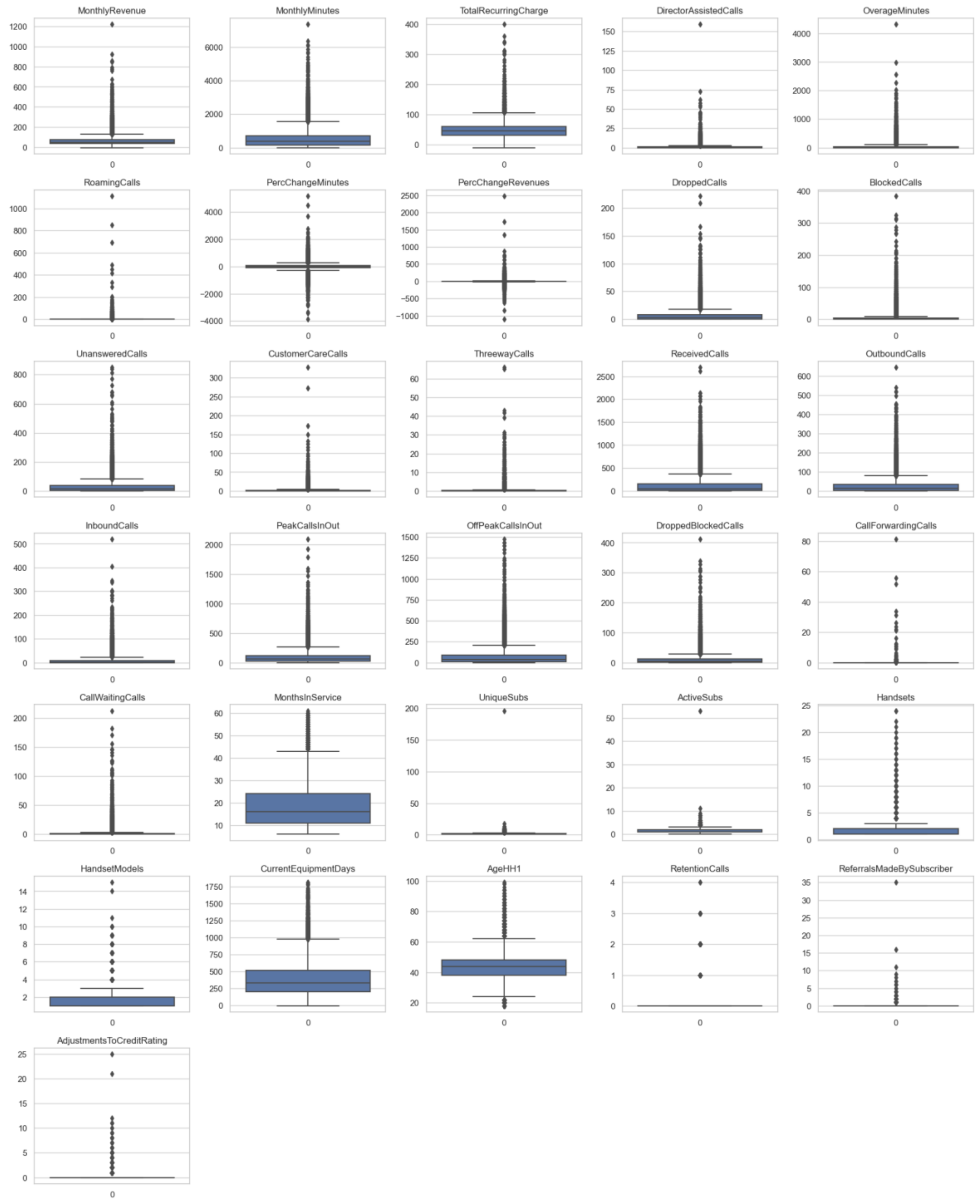


Figure 2. Box plots of numerical variables

2.2 Feature selection

We have a large dataset with 57 features, and we need to reduce its size. Initially, we eliminate features that are not directly relevant to the topic, such as TruckOwner, RVOwner, OwnsMotorcycle, and NonUSTravel. These features provide information about the user's situation, but they are not highly relevant to the telecommunications service topic. We already have access to other information, such as IncomeGroup and MonthlyRevenue, which enable us to assess the user's current situation.

After removing the outliers, we noticed that some features have sparse data. One possible solution is to delete those features from the dataset

3. Methodology

3.1 Loss and metrics

3.1.1 Loss function

The effectiveness of models is heavily influenced by the selection of hyperparameters and data preprocessing... Therefore, it is necessary to have a metric to assess the performance after each modification of the model, and to compare the performance of different models.

3.1.2 Metrics

We define:

- TP_i (true positive) is the number of instances that are assigned correctly to class c_i
- TN_i (true negative) is the number of instances inside c_i that are assigned correctly to another class
- FP_i (false positive) is the number of instances that are assigned incorrectly to class c_i
- FN_i (false negative) is the number of instances inside c_i that are assigned incorrectly to another class

a. Accuracy

Percentage of correct predictions on testing data:

$$Accuracy = \frac{TP_i + TN_i}{TP_i + FP_i + TN_i + FN_i}$$

b. Precision

Percentage of correct instances, among all that are assigned to c_i:

$$Precision(c_i) = \frac{TP_i}{TP_i + FP_i}$$

c. Recall

The ratio of the number of true positive cases to the total number of actual positive cases

$$Recall(c_i) = \frac{TP_i}{TP_i + FN_i}$$

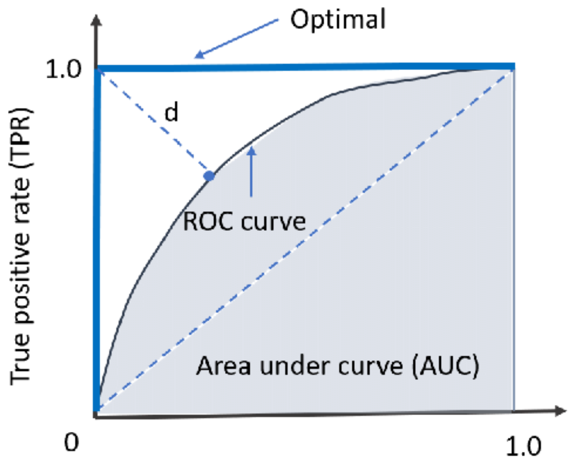
d. F1 – score

F1 is the harmonic mean of precision and recall. It can provide a unified view on the performance of a classifier and is computed as:

$$F_1 = \frac{2.Precision.Recall}{Precision + Recall} = \frac{2}{\frac{1}{Precision} + \frac{1}{Recall}}$$

e. ROC curve and ROC - AUC

The ROC (Receiver Operating Characteristics) curve is plotted with True Positive Rate (TPR) – as known as Recall against the False Positive Rate $\frac{FP_i}{TN_i + FP_i}$ (FPR), where TPR is on the y-axis and FPR is on the x-axis.



While ROC is a probability curve, AUC (Area under the curve) represents the degree or measure of separability. It tells how much the model is capable of distinguishing between classes. Higher the AUC, the better the model is at predicting 0 classes as 0 and 1 classes as 1.

3.2 Methods

3.2.1 Decision tree

A decision tree is a type of supervised machine learning used to categorize or make predictions based on how a previous set of questions were answered. The model is a form of supervised learning, meaning that the model is trained and tested on a set of data that contains the desired categorization. Decision trees imitate human thinking, so it's generally easy for data scientists to understand and interpret the results. A decision tree resembles, well, a tree. . The time complexity of decision trees is a function of the number of records and number of attributes in the given data. The decision tree is a distribution-free or non-parametric method, which does not depend upon probability distribution assumptions. Decision trees can handle high dimensional data with good accuracy.

Attribute selection measure (ASM) is a heuristic for selecting the splitting criterion that partition

data into the best possible manner. ASM methods evaluate the importance of attributes by computing some metrics based on the information and correlation between the attributes and the data classes. Most popular selection measures are Information Gain, Gain Ratio, and Gini Index

a. Information Gain

Information Gain (IG) is a popular attribute selection measure used in decision tree algorithms. It compares the entropy of the dataset before and after the split to determine the importance of the attribute in the classification task. The attribute with the highest IG is selected as the splitting criterion. High IG indicates that the attribute can effectively separate the classes in the dataset, making it a good choice for feature selection.

Information gain of attribute A in S is defined as:

$$Gain(S,A) = Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$

where $Entropy(S) = -\sum_{i=1}^{c} p_i \log_2 p_i \ (p_i \ is \ probability)$

b. Gain Ratio

Besides information gain, we can use other measures, one of which is Gain Ratio. It is an extension of the Information Gain measure that takes into account the intrinsic information of an attribute and the potential information gain. Gain Ratio is a popular attribute selection measure because it balances the trade-off between information gain and overfitting.

$$GainRatio(S,A) = \frac{Gain(S,A)}{SplitInformation(S,A)}$$

where $SplitInformation(S,A) = -\sum_{v \in Values(A)} \frac{|S_v|}{|S|} \log_2 \frac{|S_v|}{|S|}$

c. Gini Index

Another decision tree algorithm CART (Classification and Regression Tree) uses the Gini method to create split points

$$Gini(D) = 1 - \sum_{i=1}^{m} p_i^2$$

where p_i is the probability that a tuple in D belongs to class C_i. The Gini Index considers a binary split for each attribute. You can compute a weighted sum of the impurity of each partition. If a binary split on attribute A partitions data D into $D1$ and $D2$, the Gini index of D is:

$$Gini_A(D) = \frac{|D_1|}{|D|} Gini(D_1) + \frac{|D_2|}{|D|} Gini(D_2)$$

In case of a discrete-valued attribute, the subset that gives the minimum gini index for that chosen is selected as a splitting attribute. In the case of continuous-valued attributes, the strategy is to select each pair of adjacent values as a possible split-point and point with smaller gini index chosen as the splitting point.

$$\Delta Gini(A) = Gini(D) - Gini_A(D)$$

3.2.2 Random forest

As the result of Decision Tree for classification is not good enough, we decided to use the Random Forest algorithm to improve performance. Random Forest is an ensemble learning algorithm that builds a set of decision trees from randomly selected subsets of the training data. It then aggregates the predictions of the individual trees to determine the final class for each test object.

Ensemble Learning is a machine learning technique that combines multiple algorithms, either of the same type or different types, to improve the accuracy and robustness of the model. For example, we could use AdaBoost, and Decision Tree algorithms to make predictions, and then combine their results to make a final prediction for each test object. Overall, Random Forest is a powerful ensemble learning algorithm that can help improve the accuracy and generalization of classification models.

3.2.3 AdaBoost

Boosting is a general strategy for learning classifiers by combining simpler ones. The idea of boosting is to take a “weak classifier” - that is, any classifier that will do at least slightly better than chance - and $\{(x_i, y_i)\}_{i=1}^{N}$ use it to build a much better classifier, thereby boosting the performance of the weak classification algorithm. This boosting is done by averaging the outputs of a collection of weak classifiers. The most popular boosting algorithm is AdaBoost, so-called because it is “adaptive.” AdaBoost is extremely simple to use and implement, and often gives very effective results. There is tremendous flexibility in the choice of weak classifier as well. Boosting is a specific example of a general class of learning algorithms called ensemble methods, which attempt to build better learning algorithms by combining multiple simpler algorithms.

Suppose we are given training data where $x_i \in R^K$and $y_i \in [-1,1]$. And suppose we are given a (potentially large) number of weak classifiers, denoted$f_m(x) \in [-1,1]$, and loss function defined as:

$$I(f_m(x), y) = \begin{cases} 0, if\ f_m(x) = y_i \\ 1, if\ f_m(x) \neq y_i \end{cases}$$

After learning, the final classifier is based on a linear combination of the weak classifier:

$$g(x) = sign(\sum_{m=1}^{M} \alpha_m f_m(x))$$

Loss function. Here we discuss the loss function interpretation of AdaBoost. AdaBoost can be viewed as greedy optimization of a particular loss function. We define $f(x) = \frac{1}{2}\sum_m \alpha_m f_m(x)$ and rewrite the classifier as $g(x) = sign\big(f(x)\big)$ (the factor of $\frac{1}{2}$ has no effect on the classifier output). AdaBoost can then be viewed as optimizing the exponential loss:

$$\mathcal{L}_{exp}(X, y) = e^{-yf(x)}$$

so that the full learning objective function, given training data $\{(x_i, y_i)\}_{i=1}^{N}$ is:

$$E = \sum_i e^{-\frac{1}{2}y_i \sum_{m=1}^{M} \alpha_m f_m(x)}$$

which must be optimized with respect to the weights and the parameters of the weak classifiers. The optimization process is greedy and sequential: we add one weak classifier at a time, choosing it and its to be optimal with respect to E, and then never change it again. Note that the exponential loss is an upper-bound on the 0-1 loss function:

$$\mathcal{L}_{exp}(X, y) \geq \ \mathcal{L}_{0-1}(X, y)$$

Therefore, if exponential loss of zero is achieved, then the 0-1 loss is zero as well, and all training points are correctly classified. Consider the weak classifier fm to be added at step m. The entire objective function can be written to separate out the contribution of this classifier:

$$E = \sum_i e^{-\frac{1}{2}y_i \sum_{j=1}^{m-1} \alpha_j f_j(x_j) - \frac{1}{2}y_j \alpha_m f_m(x_i)}$$

Since we are holding constant the first m1 terms, we can replace them with a single constant:

$$w_i^{(m)} = e^{-\frac{1}{2}y_i \sum_{j=1}^{m-1} \alpha_j f_j(x_j)}$$

Hence, we have:

$$E = \sum_i w_i^{(m)} e^{-\frac{1}{2}y_i \alpha_m f_m(x_i)}$$

We can split this into two summations, one for data correctly classified by fm, and one for those misclassified:

$$E = \sum_{i:f_m(x_i)=y_i} w_i^{(m)} e^{-\frac{\alpha_m}{2}} + \sum_{i:f_m(x_i)\neq y_i} w_i^{(m)} e^{\frac{\alpha_m}{2}}$$

Rearranging terms, we have:

$$E = \left(e^{\frac{\alpha_m}{2}} - e^{-\frac{\alpha_m}{2}}\right) \sum_i w_i^{(m)} I(f_m(x_i) \ \neq y_i) + e^{-\frac{\alpha_m}{2}} \sum_i w_i^{(m)}$$

Optimizing this with respect to f_m is equivalent to optimizing $\sum_i w_i^{(m)} I(f_m(x_i) \ \neq y_i)$, which is what AdaBoost does. The optimal value for m can be derived by solving $\frac{dE}{d\alpha_m} = 0$. After that, we divide both side by $\frac{\alpha_m}{2} \sum_i w_i^{(m)}$, we have

$$\frac{(e^{\frac{\alpha_m}{2}} + e^{-\frac{\alpha_m}{2}}) \sum_i w_i^{(m)} I(f_m(x_i) \ \neq y_i)}{\sum_i w_i^{(m)}} - e^{-\frac{\alpha_m}{2}} = 0$$

Replacing $\frac{\sum_i w_i^{(m)} I(f_m(x_i) \neq y_i)}{\sum_i w_i^{(m)}} = \varepsilon_m$:

$$(e^{\frac{\alpha_m}{2}} + e^{-\frac{\alpha_m}{2}})\varepsilon_m - e^{-\frac{\alpha_m}{2}} = 0$$

$$e^{\frac{\alpha_m}{2}}\varepsilon_m = e^{-\frac{\alpha_m}{2}}(1 - \varepsilon_m)$$

$$\frac{\alpha_m}{2} + ln\varepsilon_m = -\frac{\alpha_m}{2} + \boldsymbol{ln\,(1 - \varepsilon_m)}$$

$$\alpha_m = \ln\,(\frac{1 - \varepsilon_m}{\varepsilon_m})$$

3.2.4 XGBoost

XGBoost is an efficient implementation of the Gradient Boosted Trees algorithm. XGBoost is expressed as a function of functions. By assembling many weak learners, XGBoost avoid the variance – bias tradeoff problem.

To train the XGBoost model, we minimize the following regularized objective:

$$L = \sum_i l(\widehat{y_i}, y_i) + \sum_k \Omega\,(f_k)$$

Here l is a loss function that calculates the difference between the target y$_i$ and the prediction $\widehat{y_i}$, **Ω** is a regularization term that serves as a penalty for the model complexity.

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda\|w\|^2$$

With T is the number of leaves and w is the weight of leaves

In the case of binary classification, each tree likes a piecewise function and each leaf has its weight, to make a prediction, we use the sigmoid function. Let p denote instance set that x map to and w present the real predicted value of each leaf. A decision tree is expressed as: $f_k = w_{p(x)}$

Since the tree ensemble model is a function of function, we cannot optimize in Euclidean space. So the XGBoost is trained in an additive manner. The new loss function at iteration t is the following:

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

To transform the loss function to Euclidean space and use traditional optimization techniques such as newton's method, we use the Taylor series to approximate it. The second-order approximation is used:

$$L^{(t)} \approx \sum_{i=1}^{n}(l\big(y_i, \widehat{y_i}^{t-1}\big) + g_i f_t\,(x_i) + \frac{1}{2}h_i f_t^2(x_i)) + \Omega(f_t)$$

where $g_i = \partial_{\widehat{y_i}^{t-1}} l\big(y_i, \widehat{y_i}^{t-1}\big)$, $h_i = \partial^2_{\widehat{y_i}^{t-1}} l\big(y_i, \widehat{y_i}^{t-1}\big)$

Since $l\big(y_i, \widehat{y_i}^{t-1}\big)$ are constant, the above equation is equivalent to:

$$L^{(t)} \approx \sum_{i=1}^{n}(g_i f_t\,(x_i) + \frac{1}{2}h_i f_t^2(x_i)) + \Omega(f_t)$$

Therefore, the subsequent objective is to discover a learner that minimizes the loss function. We are already familiar with the optimal solution for a quadratic function with a single variable. To determine the optimal solution for a fixed tree q, we can do the following approach:

$$w_j^* = -\frac{G_j}{H_j+\lambda} \text{ and } \min\left(L^{(t)}\right) = -\frac{1}{2}\sum_{j=1}^{T}\frac{G_j^2}{H_j+\lambda} + \gamma T$$

We start from a tree with depth 0 and try to make a split to reduce the gain (the change of loss function):

$$Gain = \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{(G_L+G_R)^2}{H_L+H_R+\lambda} - \gamma$$

where G_L and H_L are the G and H of the left child, G_R and H_R are the G and H of the right child.

The final prediction is the sum of all trees. In practice, we use a learning rate to reduce the effect of a tree and have more chances for future learning. Each tree is a weak learner, they are trained on different datasets, different columns, and limited depth. From that, we will get a more generalizable model and reduce the overfitting and the effect of outliers. A small learning rate is also necessary to keep the validity of Taylor's approximation

XGBoost is a powerful model that is well-suited for tabular data, and it is capable of handling missing data. In addition to the traditional splitting method, XGBoost also supports histogram aggregation, which is based on the histogram graph of a set of data instances. Furthermore, XGBoost offers a CPU version of the tree_method, which allows for fast model training. However, because XGBoost has a large number of hyperparameters, searching for the optimal set of hyperparameters can be time-consuming. To address this issue, this project will consider hyperparameters one by one, selecting the best value for each parameter before moving on to the next parameter.

4. Results

Because we use an imbalanced dataset ('Churn' = 28.8%, 'Non Churn' = 71.2%), we use UnderSampling to balance our data. Balanced data is important to ensure that the model is able to accurately classify all classes and is not biased towards the majority class. This, in turn, leads to a more reliable and accurate model that can be used to make better predictions on new data

4.1 Decision tree

This algorithm is implemented using ***DecisionTreeClassifier*** model in ***tree*** module defined in scikit-learn library. The model contains some modified parameters:

- criterion: it is a function used to measure the quality of a split at each node of the tree
    - 'gini'
    - 'entropy'
    - 'log_loss'
- max_depth: the maximum depth of the tree
- min_samples_split: the minimum number of samples required to split an internal node
- min_samples_leaf: the minimum number of samples required to be at a leaf node

We analyzed the effect of some factors to the model and choose the best values for them. We use K-fold and calculate average to find the best value. First, we need to find the best value for max_depth to avoid overfitting.

max_depth	3	5	6	7	9
F1 – score	0.5524475756932864	0.5718318261484854	***0.5743468916154895***	0.5706112995088188	0.5662575346757406

For max_depth, the most optimal result obtained for max_depth = 6

After finding the value of max_depth, we fix its value and then search for the best values of other variables to avoid overfitting.

criterion	gini	entropy	log_loss
F1 – score	***0.5743468916154895***	0.5732498917065352	0.5732498917065352

For criterion, the most optimal result obtained for criterion = 'gini'

min_samples_split	2	4	6	8	10
F1 – score	0.5743468916154895	0.5743393533176582	0.5745014051682763	0.5745014051682763	***0.5745437779752628***

For min_samples_split, the most optimal result obtained for min_samples_split = 10

min_samples_leaf	1	3	5	7	9
F1 – score	0.5743468916154895	0.5745766369280234	0.5746478770870819	***0.5747608122754088***	0.5745011549863364

For min_samples_leaf, the most optimal result obtained for min_samples_leaf = 7

The most optimal combination of parameters for the model:

max_depth = 6
criterion = 'gini'
min_samples_split = 10
min_samples_leaf = 7

4.2 Random forest

This algorithm is implemented using ***RandomForestClassifier*** model in ***ensemble*** module defined in scikit-learn library. The model contains some modified parameters:

- n_estimator: the number of trees in the forest.
- criterion: it is a function used to measure the quality of a split at each node of the tree
  - 'gini'
  - 'entropy'
  - 'log_loss'
- max_depth: the maximum depth of the tree
- min_samples_split: the minimum number of samples required to split an internal node
- min_samples_leaf: the minimum number of samples required to be at a leaf node
- bootstrap:

Like decision tree, we analyzed the effect of some factors to the model and choose the best values for them. First, we need to find the best value for max_depth to avoid overfitting.

max_depth	3	5	7	9
F1 – score	0.5605630582063128	0.5783551086755578	0.5862431500796854	***0.5911346736268291***

For max_depth, the most optimal result obtained for max_depth = 9

After finding the value of max_depth, we fix its value and then search for the best values of other variables to avoid overfitting.

n_estimators	50	100	150	200
F1 – score	0.5901333329471881	0.5911346736268291	***0.5927146217345363***	0.5922821831994716

For n_estimators, the most optimal result obtained for n_estimators = 200

criterion	gini	entropy	log_loss
F1 – score	0.5911346736268291	***0.5937113767722237***	0.5937113767722237

For criterion, the most optimal result obtained for criterion = ‘entropy’

min_samples_split	2	4	6	8	10
F1 – score	0.5911346736268291	0.5915071153916625	0.5914554213184616	***0.5925219222039896***	0.5900010912904683

For min_samples_split, the most optimal result obtained for min_samples_split = 8

min_samples_leaf	1	3	5	7	9
F1 – score	0.5911346736268291	0.5924132549905026	0.5912949557505767	***0.5926662816639393***	0.5913881883634552

For min_samples_leaf, the most optimal result obtained for min_samples_leaf = 7

bootstrap	True	False
F1 – score	0.5911346736268291	***0.5919055446797229***

The most optimal combination of parameters for the model:

max_depth = 9
n_estimators = 150
criterion = ‘entropy’
min_samples_split = 8
min_samples_leaf = 7
bootstrap = False

4.3 AdaBoost

This algorithm is implemented using ***AdaBoostClassifier*** model in ***ensemble*** module defined in scikit-learn library. The model contains some modified parameters:

- n_estimators: the maximum number of estimators at which boosting is terminated. In case of perfect fit, the learning procedure is stopped early
- learning_rate: weight applied to each classifier at each boosting iteration

Because of the few parameters, we decided to use GridSearch to find the best setting for AdaBoost. We obtained the most optimal combination of parameters for the model:

n_estimators = 300

learning_rate = 0.5

4.4 XGBoost

This algorithm is implemented using ***XGBClassifier*** model defined in ***xgboost*** library. The model contains some modifed parameters:

- objective: This is defined the loss function to be minimized.
    - 'binary:logistic' : logistic regression for binary classification, return predicted probability.
    - 'multi:softmax' : multiclass classification using the softmax objective, return predicted class.
    - 'multi:softprob': same as 'multi:softmax', but return predicted probability.
- booster: The model has two types of bosster which are tree bosster and linear booster but tree booster always outperforms linear booster, so that almost models use tree booster ('gbtree')
- eval_metric: The metric to be used for validation data
    - 'rmse' : root mean squre error
    - 'mae' : mean absolute error
    - 'logloss' : negative log - likelihood
    - 'error' : binary classification error rate (0.5 threshold)
    - 'auc' : area under the curve
- tree_method: The tree construction algorithm (default = 'auto')
    - 'auto' : use heuristic to choose the fastes method
    - 'exact' : exact greedy algorithm. Enumerates all split candidates.
    - 'approx' : Approximate greedy algorithm using quantile sketch and gradient histogram.
    - 'hist' : Faster histogram optimized approximate greedy algorithm.
    - 'gpu_hist' : GPU implementation of hist algorithm.
- eta: This determines the impact of each tree on the outcome. This makes the model more robust by shrinking the weights on each step
- gamma: Minimum loss reduction required to make a further partition on a leaf node of the tree. The larger gamma is, the more conservative the algorithm will be
- max_depth: The maximum depth of a tree. It is used to control overfitting as higher depth will allow model to learn relations very specific to a particular sample.
- n_estimators: The number of sequential trees to be modeled
- subsample: the fraction of observations to be randomly selected for each tree.
- scale_pos_weight: A value greater than 0 should be used in case of high class imbalance as it helps in faster convergence.

We analyzed the effect of colsample_bytree, eta, gamma, max_depth, min_child_weight, n_estimator, subsample to the model and choose the best values for them:

For max_depth, we build and test model for 5 values: 4, 5, 6, 7, 8. From that, the most optimal result obtained for max_depth = 4

max_depth	4	5	6	7	8
F1 – score	***0.6038927204502155***	0.5970128483572947	0.5917730367427021	0.5933731756662558	0.5847089359612568

For eta, we build and test model for 7 values: 0.001, 0.01, 0.05, 0.1, 0.2, 0.5, 1. From that, the most optimal result obtained for eta = 0.1

eta	0.001	0.01	0.05	0.1	0.2	0.5	1
F1 – score	0.5768764291134596	0.5822818003314097	0.6006434364146821	***0.606034709245316***	0.601161334678061	0.5815190181357666	0.566260515959284

For colsample_bytree, we build and test model for 6 values: 0.4, 0.5, 0.6, 0.7, 0.9,. From that, the most optimal result obtained for colsample_bytree = 0.4

colsample_ bytree	0.4	0.5	0.6	0.7	0.9
F1 – score	***0.6048115302884914***	0.6008401159189367	0.6013635036678457	0.6022843560831718	0.6020941335344302

For gamma, we build and test model for 6 values: 0, 0.1, 0.2, 0.5, 0.8, 1. From that, the most optimal result obtained for gamma = 0

gamma	0	0.1	0.2	0.5	0.8	1
F1 – score	***0.6038927204502155***	0.6019927610758158	0.6038663864886933	0.6016826396386519	0.6011978493994743	0.5992680942163029

For n_estimators, we build and test model for 6 values: 50, 100, 150, 200, 250, 300. From that, the most optimal result obtained for n_estimators = 100

n_estimators	50	100	150	200	250	300
F1 – score	0.6018401084176025	***0.6038927204502155***	0.5996138650026431	0.5986705966320003	0.5993508605091837	0.5975278961101911

For min_child_weight, we build and test model for 4 values: 0.5, 1, 1.5, 2. From that, the most optimal result obtained for min_child_weight = 1

min_child_weight	0.5	1	1.5	2
F1 – score	0.601758622297764	***0.6038927204502155***	0.602266185626998	0.6008687144901927

For subsample, we build and test model for 6 values: 0.5, 0.6, 0.7, 0.8, 0.9, 1. From that, the most optimal result obtained for subsample = 1

subsample	0.5	0.6	0.7	0.8	0.9	1
F1 – score	0.594205305155613	0.5974030664444652	0.5986987010254412	0.5997629552084129	0.6037785687654871	***0.6038927204502155***

The most optimal combination of parameters for the model:

max_depth = 4

eta = 0.1

colsample_bytree = 0.4

gamma = 0

min_child_weight = 1

n_estimator = 100

subsample = 1

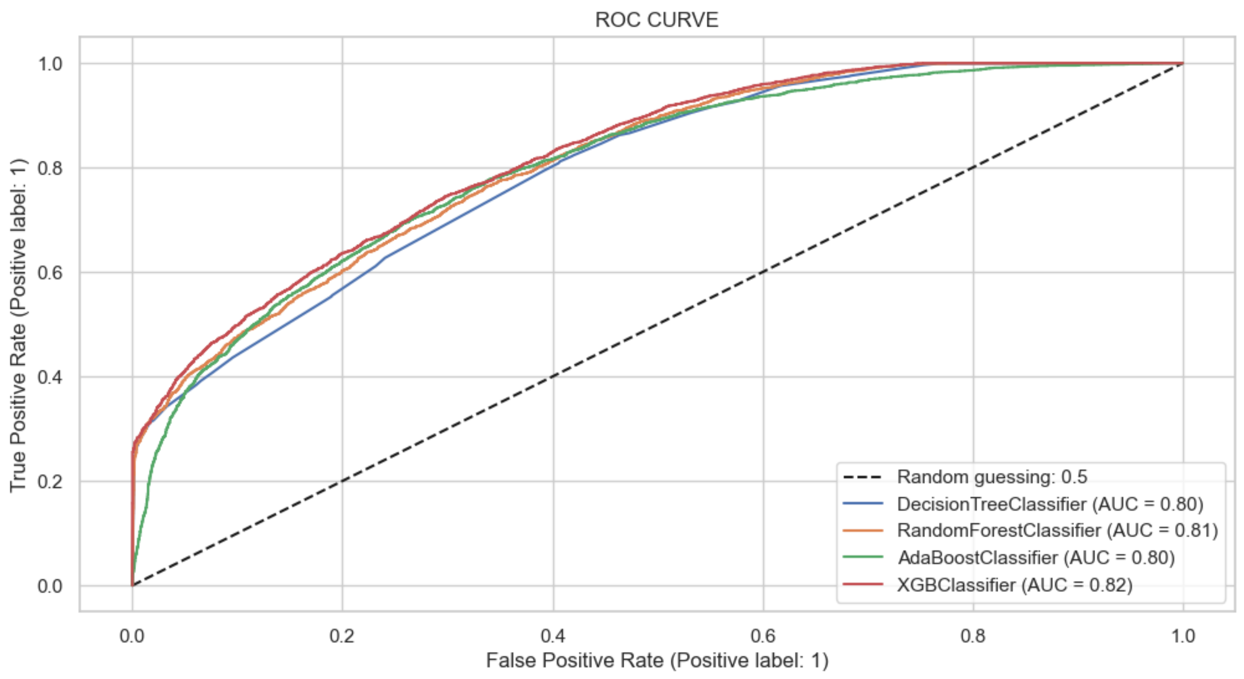
4.5 Comparative analysis

The metric values on test data

Algorithm	Decision tree	Random forest	AdaBoost	XGBoost
Accuracy	0.66	0.70	0.72	0.72
ROC-AUC	0.797	0.810	0.804	0.821
Precision	0.45	0.48	0.51	0.51
Recall	0.81	0.75	0.72	0.74
F1 – score	0.58	0.59	0.60	0.60

To make a comparative analysis, we calculate some metrics: accuracy, ROC-AUC score, precision score, recall score, F1_Score. Overall, XGBoost performs best in the 4 models. XGBoost has the highest ROC – AUC, it means that XGBoost distinguishes better and returns more relevant results than other models.

If a telecom company aims to identify customers who are likely to stop using their service, they should prioritize recall. To achieve this, the Decision Tree model is the most effective option since it can get a recall rate as high as 0.81.



5. Conclusion and future work

In the Telecom Prediction Customer Churn problem, the main goal is to predict whether customers are likely to churn or not. However, we want to ensure that the model only predicts churn for customers who are highly likely to churn, avoiding false predictions that customers who do not intend to churn are incorrectly evaluated.

F1-score is the harmonic mean of precision and recall, which helps to evaluate the overall performance of the classification model. If the model has a high F1-score, it means that the model achieves high accuracy in both precision and recall, avoiding false predictions.

Therefore, F1-score is a suitable evaluation metric for the Telecom Prediction Customer Churn problem, helping to evaluate the performance of the classification model and ensuring that the model only predicts churn for highly likely customers.

This study set out to compare the prediction performance based on Decision Tree, Random Forest, AdaBoost, and XGBoost. To achieve this objective, we implement all of those and have the result: XGBoost (with 72 % accuracy in test set) has the same performance as AdaBoost but the F1-score (recall – score) result is slightly better than the other models.

Future work would be directed at fine-tuning the hyperparameters of the prediction models with the aim of higher accurate prediction rate.

Appendix:

Data preprocessing	Numpy, pandas, sklearn
Decision Tree	tree (sklearn)
AdaBoost, Random Forest	ensemble (sklearn)
XGBoost	xgboost
Hyperparameter tuning	GridsearchCV, Stratified K- fold (sklearn)
Visualization	Seaborn, matplotlib

Table 1: List of libraries were used

No.	Attributes	Description
1	*Churn*	Yes = the customer left the company this quarter. No = the customer remained with the company. Directly related to Churn Value.
2	*MonthlyRevenue*	a financial metric that represents the total revenue generated by a user in a given month.
3	*MonthlyMinutes*	the total number of minutes used by a customer in a given month
4	*TotalRecurringCharge*	the total amount of charges billed to a customer on a recurring basis for telecommunications services
5	*DirectorAssistedCalls*	the number of calls made by customers that require assistance from a live operator or customer service representative.
6	*OverageMinutes*	the total number of minutes used by a customer in excess of their monthly allotted minutes for activities
7	*RoamingCalls*	the number of calls made by a customer while they are outside of their home network and using a different network provider
8	*PercChangeMinutes*	the percentage change in the total number of minutes used by a customer compared to a previous period
9	*PercChangeRevenues*	a financial metric that represents the percentage change in total revenues generated by a customer compared to a previous period.
10	*DroppedCalls*	the number of calls that are terminated or disconnected before they are completed
11	*BlockedCalls*	the number of calls that are prevented from being completed due to technical issues or limitations
12	*UnansweredCalls*	the number of incoming calls that are not answered by the called party
13	*CustomerCareCalls*	mean number of customer care calls
14	*ThreewayCalls*	mean number of threeway calls
15	*ReceivedCalls*	the total number of incoming calls received by a customer or a company in a given period of time
16	*OutboundCalls*	the total number of outgoing calls made by a customer in a given period of time
17	*InboundCalls*	the total number of incoming calls received by a customer in a given period of time
18	*PeakCallsInOut*	the highest number of simultaneous incoming and outgoing calls during a particular time period
19	*OffPeakCallsInOut*	the number of simultaneous incoming and outgoing calls during the off-peak hours of the day

20	*DroppedBlockedCalls*	the total number of calls that are either dropped or blocked before they can be completed
21	*CallForwardingCalls*	the number of incoming calls that are redirected to another number
22	*CallWaitingCalls*	the number of incoming calls that are received while the customer is already on a call
23	*MonthsInService*	months in Telecom service
24	*UniqueSubs*	the total number of unique subscribers
25	*ActiveSubs*	the total number of subscribers or customers who are actively using a particular service or product during a specific period
26	*Handsets*	the total number of physical devices, such as mobile phones or landline phones, that are in use by customers
27	*HandsetModels*	the total number of unique models of physical devices, such as smartphones or landline phones, that are in use by customers.
28	*CurrentEquipmentDays*	the average age of the physical devices
29	*AgeHH1*	Age of first household member
30	*AgeHH2*	Age of second household member
31	*ChildrenInHH*	whether or not children in household
32	*HandsetRefurbished*	whether or not Handset refurbished
33	*HandsetWebCapable*	whether or not Handset web capable
34	*TruckOwner*	whether or not owning truck
35	*RVOwner*	whether a household owns or rents their residence
36	*Homeownership*	whether a household owns their home or not
37	*BuysViaMailOrder*	whether customer buy through mail order catalogs, online shopping or not
38	*RespondsToMailOffers*	whether customers respond to mail offers or not
39	*OptOutMailings*	whether customers choose to opt-out of receiving mailings or not
40	*NonUSTravel*	whether customers travel outside of the United States or not
41	*OwnsComputer*	whether a household owns a computer or not
42	*HasCreditCard*	whether a customer has a credit card or not
43	*RetentionCalls*	the number of calls made by a company's retention team to customers
44	*RetentionOffersAccepted*	whether customers accept retention offers or not
45	*NewCellphoneUser*	whether a customer is a new cellphone user or not
46	*NotNewCellphoneUser*	whether a customer is an existing cellphone user or not

47	*ReferralsMadeBySubscriber*	the number of customers who were referred to the company's services by an existing subscriber
48	*IncomeGroup*	the level of income of an individual or household
49	*OwnsMotorcycle*	whether a customer owns a motorcycle or not
50	*AdjustmentsToCreditRating*	changes made to an individual or business' CreditRating
51	*HandsetPrice*	the price of a physical device
52	*MadeCallToRetentionTeam*	whether a customer has contacted the company's retention team or not
53	*CreditRating*	a numerical score assigned to a customer to assess their creditworthiness
54	*PrizmCode*	classify users
55	*Occupation*	the type of work or profession that a customer is engaged in
56	*MaritalStatus*	the current marital status of an individual
57	*ServiceArea*	the geographic area where a telecommunications company provides its services
58	*CustomerID*	a unique identifier assigned to each individual customer in a database or system

Table 2: Description for each attribute in Telecom customer churn dataset

References:

[1] Telecom customer churn dataset, Kaggle.
Available from: https://www.kaggle.com/datasets/jpacse/datasets-for-churn-telecom

[2] scikit-learn Package. Available from: https://scikit-learn.org/

[3] seaborn Package. Available from: https://seaborn.pydata.org/

[4] xgboost Package. Available from: https://xgboost.readthedocs.io/

[5] Understanding AdaBoost. Available from: https://mbernste.github.io/files/notes/AdaBoost.pdf