credit card type/scheme

REPORT

Chatzimoschou angeliki

JANuary 2017

# TABLE OF CONTENTS

[TABLE OF CONTENTS 2](#_Toc467074496)

[PART I – DATA REPRESANTATION AND EXPLORATION 3](#_Toc467074497)

[PART II – DATA PREPROCESSING AND TRANFORMATION 3](#_Toc467074498)

[PART III – MACHINE LEARNING TECHNIQUES 7](#_Toc467074499)

[CLASSIFICATION METHODS 7](#_Toc467074500)

[PART IV – CONCLUSION & FUTURE WORK 12](#_Toc467074503)

# PART I – DATA REPRESANTATION AND EXPLORATION

In this project, we are assigned to deploy several Machine Learning Techniques on the “DataScientistPricing” dataset provided by -XYZ-.SA. This dataset summarizes a heterogeneous set of features about historical transactions with various field columns collected in a period of 3 weeks.

Specifically, this dataset contains 23 attributes (with numerical and categorical values) describing different aspects of each customer’s reservation, from a total of 153.033 reservations. It is composed by 23 independent variables, such as: transaction date, client type, affiliate id, currency type, language, origin airport, destination airport, etc. and one dependent variable: the type/scheme of the credit card that was used.

The goal is to predict the credit card type/scheme each customer is going to use in order to provide a more correct price, leading to more sales, better profitability or even both.

# PART II – DATA PREPROCESSING AND TRANFORMATION

To accomplish our analysis, we used the following tools:

* R Studio version 0.99.891
* R version 3.2.3

For the stage of preprocessing, we deployed various techniques using the tool R Studio. More specifically, we performed data preprocessing techniques in order to examine at a deeper level the dataset. Some missing values were found and were omitted, as they were wrong entries of the form “ - “, “ -,- “, or empty rows at the last part of the dataset and levels with no values in some categorical features.

In addition, we decided to omit several attributes: 3 DATE variables (“Date\_at”, “Departure\_date\_at”, “return\_date\_at” and the “ul\_event\_id” which contained only unique values, so there was no statistical information included. After handling the missing values issue and deleting the aforementioned variables, we are left with 113.940 observations.

Our dataset is mainly consisted of categorical features and 4 numerical ones. By observing the histogram of each numerical feature (Figure 2), we decided to operate a log base 10 transformation on variables with skewness > 3 (“children”, “infants”, “Sales”). Unfortunately, only “Sales” variable got closer to normal distribution.

Several categorical variables contain a lot of levels (25000 plus) and some of them are of low frequency. In order to diminish the number of levels to make computation faster and also avoid the risk of a classifier not being able to run (i.e. Random Forests run only with categorical features which contain until 53 levels), we combined the levels with less than 1% value. After combining them, we created a new category level named as ‘Other’. Since we have a different train and test dataset, it is only expected that same categorical variables will contain different levels. To address this issue, as classifiers would fail to apply on test dataset, we merged the levels of each categorical variable needed. Finally, we checked our newly formed datasets for near 0 variance attributes and discovered that we should delete the factor “full\_route” since it contained only 1 level, thus was suffering from 0 variance and no statistical information.

Furthermore, we examined a core objective of Machine Learning process, the inter-attribute correlations. Correlations are not only necessary for predictions and classifications – since rules would fail in the absence of pattern – but also for the identification of groups of mutually correlated attributes. Actually, this dataset is well designed and correlated information between features is limited, as shown in Figure 2, where dark blue color indicates high positive correlation among a pair of features and red, high negative correlation respectively. Only a very small percentage of the dataset is lightly correlated, thus there is no need to perform any dimension reduction technique, like PCA or SVD.

Our class variable, the credit card scheme/type, is a metric that defines how often each type of credit card is used for ticket reservations in the platform of -XYZ-. It is a categorical variable with 7 classes with discrete values ranging from “164” to “170”.

Judging by table 1, we may conclude to the fact that we have to deal with an imbalanced dataset, since classes 6 and 7 are represented by a much smaller number of cases compared to classes 1 and 3. When the classification categories are not approximately equally represented, standard classifiers are biased towards the larger classes, which means that they tend to focus more on the large classes and ignore the small ones. Several solutions have been proposed at a data level and algorithmic level. From the data point of view, there are methods of re-sampling such as over-sampling, where replication of examples or generation of new instances is performed on the minority classes or under-sampling, where elimination of instances is performed on the majority classes. From the algorithmic point of view, an adjust of the operation of the algorithm is carried out to treat more effectively the unbalanced data.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Card Type | 164 | 165 | 166 | 167 | 168 | 169 | 170 |
|  | 35.143 | 8.935 | 51.264 | 13.374 | 4.658 | 46 | 520 |

*Table 1 – Class variable Information*

In order to bring the response classes at a better balancing state, we implement the Synthetic Minority Over-sampling Technique (SMOTE). The main target of this method is to artificially generate new instances of the minority class using the nearest neighbors of these cases. In addition, the majority classes are getting under-sampled, leading to a more balanced dataset.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Smoted Card Type | 164 | 165 | 166 | 167 | 168 | 169 | 170 |
|  | 29.918 | 7.690 | 44.169 | 11.472 | 4.054 | 16.606 | 401 |

*Table 2 – smoted Class variable Information*

As last step, for simplicity sake and faster time computing we renamed the response variable into “card\_scheme” and recoded its classes by applying a sequence of values from 1 to 7.

We work in the same way with the test dataset.

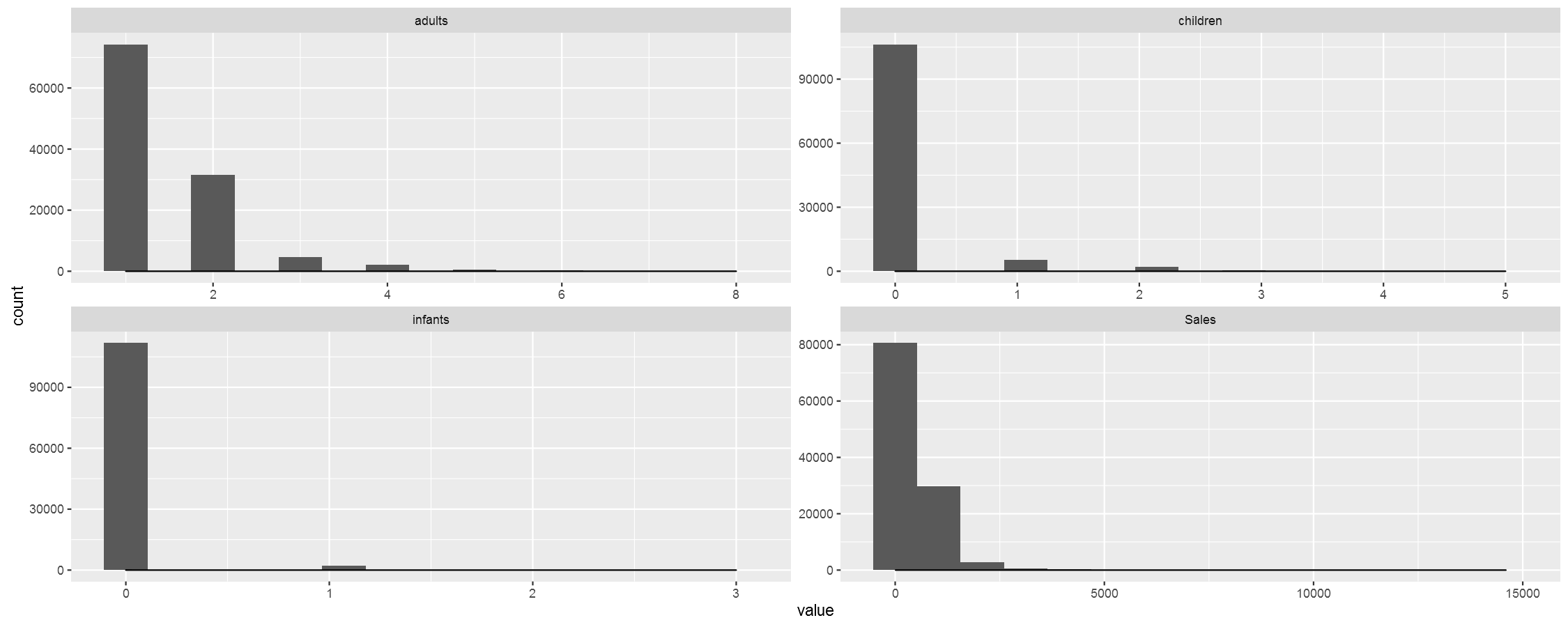
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Card Type | 164 | 165 | 166 | 167 | 168 | 169 | 170 |
|  | 11.140 | 2.932 | 15.981 | 3.441 | 1.597 | 13 | 156 |

*Table 3 – test Class variable Information*

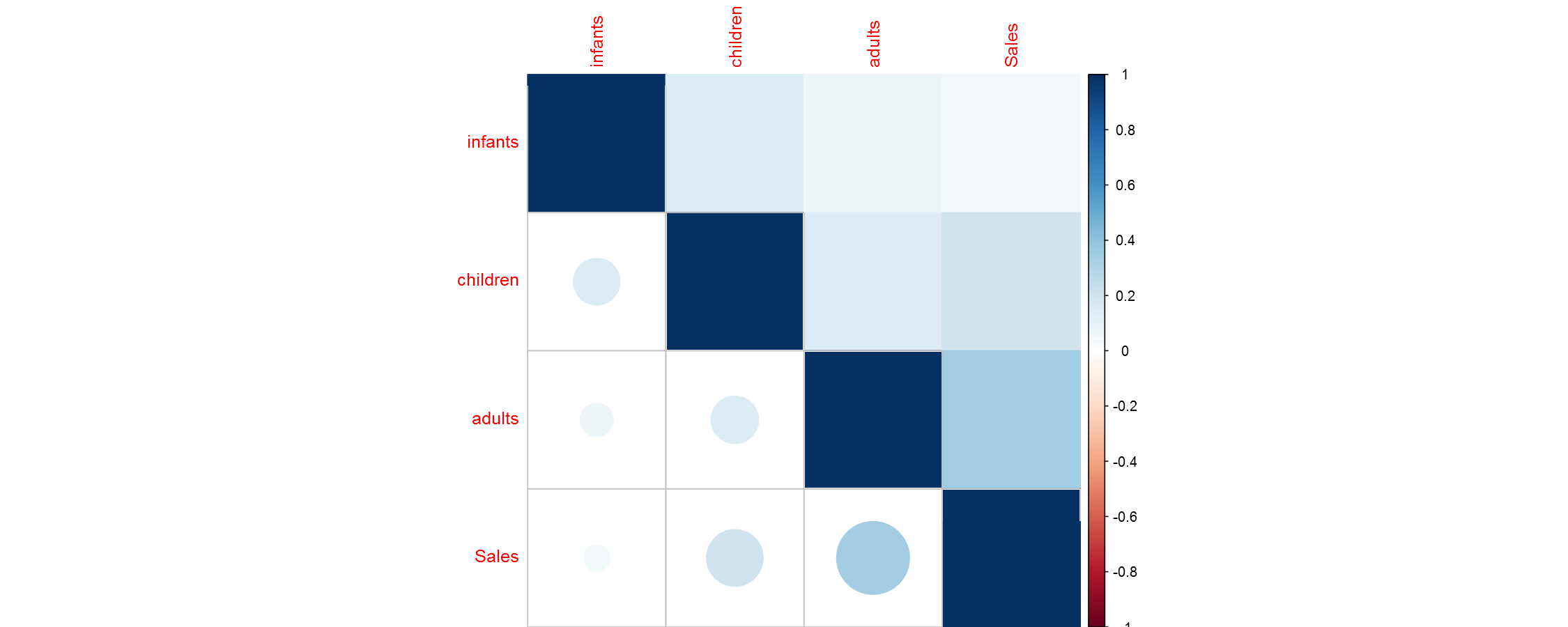
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Smoted Card Type | 164 | 165 | 166 | 167 | 168 | 169 | 170 |
|  | 9.215 | 2.379 | 13.417 | 2.807 | 1.305 | 6.513 | 127 |

*Table 4 – smoted test Class variable Information*

Observing the initial and smoted datasets, we may conclude that we get more balanced datasets, however there are still imbalances among the classes. This time the imbalances are not so high as in the initial datasets. This is because in multiclassification SMOTE doesn’t do a perfect job as in binary classification (even tried re-balancing the smoted dataset).

*Figure 1 – Multiple histograms of raw dataset*

*Figure 2 – Multiple histograms after log10 transformation*



*Figure 3 – Correlation plot of numerical features*

# PART III – Machine Learning TECHNIQUES

## CLASSIFICATION METHODS

When we have to work on imbalanced problems, accuracy (even balanced accuracy) is considered to be a poor evaluation metric because:

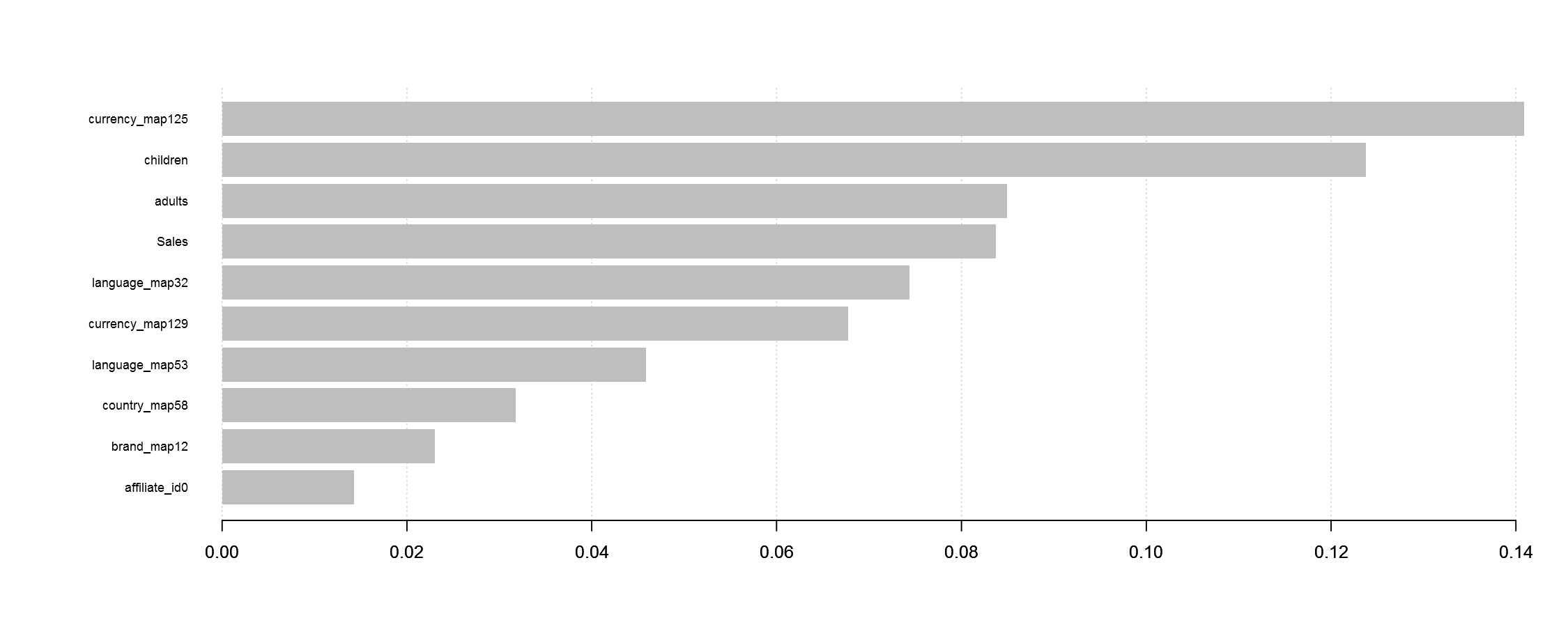
1. accuracy is calculated by ratio of correct classifications / incorrect classifications,
2. this metric would largely tell us how accurate our predictions are on the majority class. However, we need also to know how we are doing in minority class too.

The metrics we used to evaluate our predictive performance are coming from the elements of the confusion matrix and are the following:

* Sensitivity (Recall) = True Positive Rate (TP/TP+FN), how many positive values of each class have been predicted correctly,
* Specificity = True Negative Rate (TN/TN +FP), how many negative values of each class have been predicted correctly,
* Precision = Positive Predictive Rate(TP/TP+FP), the fraction of correct predictions for a certain class,
* F score = 2 \* (Precision \* Recall)/ (Precision + Recall), the harmonic mean of precision and recall. The higher the better.

In this project, we are using xGBoost. It’s a highly sophisticated algorithm, powerful enough to deal with all sorts of irregularities of data. xGBoost (eXtreme Gradient Boosting) is an advanced implementation of gradient boosted decision trees designed for speed and performance. Gradient boosted trees have to be built in series, so that a step of gradient descent can be taken in order to minimize the loss function. Unlike Random Forests, that can’t simply build the trees in parallel. XGBoost, however, builds the tree itself in a parallel fashion. Also, in return for this speed, we get a much faster grid search for optimizing hyper parameters in model tuning.

More specifically, after making the transformations needed and getting the best parameters from the tuning process, we built the model and we got an overall balanced accuracy of 0.57. Figure 4 indicates the importance of the Top 10 features. In other words, we discovered which features weigh more to predict which card type will be used.



*Figure 4 – Feature importance plot*

In addition, we experimented with several classifiers, trying a variety of values tuning their parameters and ended up with the best scores respectively, given at the Table 4 that follows:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Classifiers | Classes | Balanced Accuracy | Sensitivity | Specificity | F - Score | | Parameters |
| xGBoost | 1 | 57% | 36% | 77% | | 0.34 | nrounds = 100,   objective = multi: softmax,  eval\_metric = merror |
| 2 | 60% | 26% | 93% | | 0.06 |
| 3 | 63% | 50% | 76% | | 0.58 |
| 4 | 73% | 51% | 95% | | 0.48 |
| 5 | 70% | 44% | 97% | | 0.11 |
| 6 | 96% | 95% | 96% | | 0.87 |
| 7 | 50% | 0% | 100% | | 0 |
| SVM | 1 | 56% | 38% | 75% | | 0.16 | kernel=radial,  c=1,  gamma=0.0045 |
| 2 | NA | NA | 93% | | NA |
| 3 | 65% | 45% | 85% | | 0.60 |
| 4 | 77% | 60% | 95% | | 0.48 |
| 5 | NA | NA | 96% | | NA |
| 6 | 82% | 75% | 90% | | 0.61 |
| 7 | NA | NA | 100% | | NA |
| Random Forest | 1 | 48% | 23% | 74% | | 0.20 | ntree=500,  mtry= rounded square root of the number of features of train dataset \* |
| 2 | NA | NA | 93% | | NA |
| 3 | 59% | 41% | 76% | | 0.56 |
| 4 | 60% | 28% | 92% | | 0.03 |
| 5 | 98% | 100% | 96% | | 0% |
| 6 | 91% | 98% | 84% | | 0.23 |
| 7 | NA | NA | 100% | | NA |
| Neural Networks | 1 | 55% | 33% | 77% | | 0.33 | single hidden layer,  size=10,  maxit=1000,  MaxNWts=2400 |
| 2 | 51% | 8% | 93% | | 0.1 |
| 3 | 54% | 42% | 67% | | 0.47 |
| 4 | 68% | 41% | 94% | | 0.38 |
| 5 | 54% | 11% | 97% | | 0.12 |
| 6 | 83% | 80% | 86% | | 0.42 |
| 7 | 50% | 0% | 100% | | 0 |
| Multinomial Logistic Regression | 1 | 53% | 32% | 75% | | 0.22 | maxit=200,  MaxNWts=1600 |
| 2 | 55% | 17% | 94% | | 0.08 |
| 3 | 60% | 46% | 74% | | 0.56 |
| 4 | 8% | 38% | 95% | | 0.40 |
| 5 | 4% | 10% | 97% | | 0.13 |
| 6 | 81% | 74% | 88% | | 0.50 |
| 7 | 51% | 2% | 99% | | 0.02 |
| Naïve Bayes | 1 | 55% | 35% | 75% | | 0.19 | default settings |
| 2 | 55% | 15% | 95% | | 0.20 |
| 3 | 64% | 53% | 74% | | 0.56 |
| 4 | 62% | 28% | 96% | | 0.37 |
| 5 | 60% | 24% | 97% | | 0.24 |
| 6 | 79% | 67.5% | 90% | | 0.58 |
| 7 | 51% | 2% | 99% | | 0.03 |
| Linear Discriminant Analysis | 1 | NA | NA | 74% | | NA | we use only numerical features |
| 2 | NA | NA | 93% | | NA |
| 3 | 50% | 37% | 62% | | 0.53 |
| 4 | 57% | 22% | 92% | | 0.01 |
| 5 | NA | NA | 96% | | NA |
| 6 | 47% | 13% | 81% | | 0.07 |
| 7 | NA | NA | 100% | | NA |
| K – Nearest Neighbors (K-NN) | 1 | 53% | 30% | 76% | | 0.31 | K=rounded square root of the number of rows of train dataset \* |
| 2 | 53% | 12% | 94% | | 0.12 |
| 3 | 58% | 47% | 69% | | 0.48 |
| 4 | 60% | 26% | 94% | | 0.28 |
| 5 | 55% | 13% | 97% | | 0.13 |
| 6 | 85% | 78% | 91% | | 0.67 |
| 7 | 50% | 0% | 100% | | NA |

*Table 5 – Classification  
  
\* as performance is not an issue, in order to reduce training time, we used mtry=4 and k=3 respectively*

At this point, it is necessary to raise the disparity between the 1st classifier applied and the 2nd. Both train several decision trees for one dataset. The main difference is that in Random Forests, trees are independent and in boosting, the tree n+1 focuses its learning on the loss (what has not been well modeled by the tree n). This difference has an impact in a part of the feature importance analysis: the correlated features. Luckily, in our data we cannot talk about correlated features (except a very small percentage, which is mentioned in Part II).

In the rest of this Part, we are providing a few more details about the classification methods used:

**Random Forests** or random decision forests are an [ensemble learning](https://en.wikipedia.org/wiki/Ensemble_learning) method for [classification](https://en.wikipedia.org/wiki/Statistical_classification) and [regression](https://en.wikipedia.org/wiki/Regression_analysis) that operate by constructing a multitude of [decision trees](https://en.wikipedia.org/wiki/Decision_tree_learning) at training time and outputting the class that is the [mode](https://en.wikipedia.org/wiki/Mode_(statistics)) of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees’ habit of [over fitting](https://en.wikipedia.org/wiki/Overfitting) to their training set. This method increases the diversity and hence can reduce the variance.

We trained the Random Forests classifier on the balanced and unbalanced training set using different parameters such as the number of random features to be selected at each node, the number of decision trees the ensemble model would contain. We used an mtry parameter value of 4 based on the guideline of selecting a value that is the square root of the number of attributes, in our case 19. We chose the number of decision trees: numIterations (150) that minimized the out-of-bag error and the number of trees, which helped us to follow the law of parsimony – keeping the model simple.

**Neural Networks** are a computational approach which is based on a large collection of neural units loosely modeling the way the brain solves problems with large clusters of biological neurons connected by axons. Each neural unit is connected with many others, and links can be enforcing or inhibitory in their effect on the activation state of connected neural units. Each individual neural unit may have a summation function which combines the values of all its inputs together. There may be a threshold function or limiting function on each connection and on the unit itself such that it must surpass it before it can propagate to other neurons. These systems are self-learning and trained rather than explicitly programmed and excel in areas where the solution or feature detection is difficult to express in a traditional computer program.

Neural networks typically consist of multiple layers or a cube design. Back propagation is where the forward stimulation is used to reset weights on the "front" neural units and this is sometimes done in combination with training where the correct result is known. More modern networks are a bit freer flowing in terms of stimulation and inhibition with connections interacting in a much more chaotic and complex fashion. Dynamic neural networks are the most advanced and can, based on rules, form new connections and even new neural units while disabling others. The goal of the neural network is to solve problems in the same way that the human brain would. Neural networks are based on real numbers, with the value of the core and of the axon typically being a representation between 0 and 1.

**Support Vector Machines** (**SVMs**) analyze data used for classification and regression analysis. An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap (maximum margin hyperplane) that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall on. In addition to performing linear classification, SVMs can efficiently perform a non-linear classification using what a different kernel (Radial Basis Function, Polynomial and Hyperbolic Tangent) implicitly mapping their inputs into high-dimensional feature spaces.

**Multinomial logistic regression** is a [classification](https://en.wikipedia.org/wiki/Statistical_classification) method that generalizes [logistic regression](https://en.wikipedia.org/wiki/Logistic_regression) to [multiclass problems](https://en.wikipedia.org/wiki/Multiclass_classification). It is used to model nominal outcome variables, in which the log odds of the outcomes are modeled as a linear combination of the predictor variables (which can be real- valued, binary-valued, categorical-valued, etc. ). [Collinearity](https://en.wikipedia.org/wiki/Multicollinearity) is assumed to be relatively low.

**Naive Bayes** classifiers are a family of simple [probabilistic classifiers](https://en.wikipedia.org/wiki/Probabilistic_classifier) based on applying Bayes’ theorem with strong (naive) [independence](https://en.wikipedia.org/wiki/Statistical_independence) assumptions between the features. They all assume that the value of a particular feature is [independent](https://en.wikipedia.org/wiki/Independence_(probability_theory)) of the value of any other feature, given the class variable. Some types of Naive Bayes classifiers are:

a) Gaussian Naive Bayes: when dealing with continuous data, a typical assumption is that the continuous values associated with each class are distributed according to a [Gaussian](https://en.wikipedia.org/wiki/Normal_distribution) distribution.   
  
b) Multinomial Naive Bayes: with a multinomial event model, samples (feature vectors) represent the frequencies with which certain events have been generated by a multinomial (p1, ….., pn) where pi is the probability that event *i* occurs.

c) Bernoulli Naïve Bayes: in the multivariate [Bernoulli](https://en.wikipedia.org/wiki/Bernoulli_distribution) event model, features are independent binary variables. Like the multinomial model, this model is popular for document classification tasks, where binary term occurrence features are used rather than term frequencies.

**Linear discriminant analysis** (**LDA**) a method used [machine learning](https://en.wikipedia.org/wiki/Machine_learning) to find a [linear combination](https://en.wikipedia.org/wiki/Linear_combination) of [features](https://en.wikipedia.org/wiki/Features_(pattern_recognition)) that characterizes or separates two or more classes. The resulting combination may be used also for [dimensionality reduction](https://en.wikipedia.org/wiki/Dimensionality_reduction), in addition to [classification](https://en.wikipedia.org/wiki/Statistical_classification).   
LDA attempts to express a categorical dependent variable as a linear combination of continuous [independent variables](https://en.wikipedia.org/wiki/Independent_variables). In the case where there are more than two classes, the analysis used in the derivation of the Fisher discriminant can be extended to find a [subspace](https://en.wikipedia.org/wiki/Linear_subspace) which appears to contain all of the class variability.

# PART V – CONCLUSION & future work

In this project, we tried to handle the multi-class issue directly. So, multi-classification models were trained for predicting the card type of each trip reservation, using a large and imbalanced dataset, with approximately 114.000 instances collected during a 3-week period from the database of the popular company -XYZ-. Overall, the work done was important in understanding the main reasons that affect a user’s decision concerning which card type they would use to book a trip.

As expected, the best result was reached by xGBoost algorithm. xGBoost achieves high speed and better accuracy from any other Classification model. But it seems that no algorithm can reach 70 % accuracy, an acceptable predictive performance, given the provided datasets. Furthermore, we noticed that SVM, Random Forests and LDA classifiers weren’t able to predict all of the response classes. We tried in general to tune the classifiers’ parameters (wherever it was possible) with utter purpose to be able to predict all 7 classes, as it was known from the beginning that the datasets didn’t contain features with significant statistical information. In addition, this is the reason why we didn’t create more plots to discover any hidden patterns or didn’t implement any k-fold cross validation or deeper tuning techniques to get the optimal hyper-parameters and class weights.

We also demonstrated the first 10 significant features influencing the chosen card type each time.

In future work, we could suggest some extra pre-processing and attempt to construct new features (unfortunately in a black box form), derived from the initial ones by using the method of t-SNE or Multiple Component Analysis (a generalization of Principal Component Analysis, that works also with the categorical features). Alternatively, we could try and create some stacked models (train simple classifiers and use their predictions as meta-features for the 2nd level models) or take a subset of the most important features and re-train our classifiers. Another approach would be to decompose the problem into several binary classification tasks. There a few techniques used for this, such as: one versus- all, all-versus-all and error-correcting output coding.

Ps: Even tried some multi-classification algorithms (random forests and neural networks) in Microsoft Azure but didn’t succeed better balancing or accuracy results.