Machine Learning
EMAp FGV

Churn Prediction Project

Rener Oliveira
June 24, 2021

Contents

1 Introduction

Customer churn (loss of customers) is a problem for telecommunication companies, considering an environment of increasing competition. When a company loses customers, it not only loses the future revenue, but also the investment made to get those customers. According to [5], some studies show that acquiring new clients is five to six times more expensive than retaining existing ones.

Telecom companies have two approaches to deal with churners: the reactive one, which tries to convince customers who wants to cancel to stay; and the proactive one, which predicts who is more likely to churn before they explicitly decide to churn and send them suitable offers to avoid their loss.

The aim of this project is to study the churn problem using the CrowdAnalytix dataset, which has information regarding usage patterns from customers of a telecom company, which the real name was anonymized. This data was part of a Machine Learning Challenge issued by CrowdAnalytix in 2012.

The raw data contains 3333 observations of 20 variables which are described in Table 1. The Kaggle copy of the data was already divided into 80% train and 20% test. All code is on this Github.

2 Data Cleaning and EDA

The State variable was transformed into regions (Northeast, South, North Central, and West), reducing the number of dummy variables from 50 to just 4. We also removed four columns from the data, which can be explained by the Figure 1 continuous variables correlation plot[1].

As we can see above, the vast majority of the features are uncorrelated. But the pairs like total minutes/total charge are perfectly correlated, which makes sense if users are charged by minutes used. We choose to remove the charge variables to avoid multicollinearity.

Visualizing the distribution of some variables we can find interesting patterns. In Figure 2a we see that among churners, we have fewer customers with voice mail plan (approximately 16.75%) comparing with non-churners subscribed to this plan (approximately 29.32%). On the other hand, we have much more churners with an international plan, comparing with non-churners, as shown in Figure 2b. A possible explanation would be that international plan subscribers are finding better offers in this service with competitors.

Another insightful plot is the one in Figure 3. We can see that churners make more calls to customer services than non-churners; these calls may be related to complaints about the services and unsatisfactory manifestations.

Taking a look at the regional distribution in Figure 4, although Northeast and South re-

[1]We generated all plots in this section from the training set.

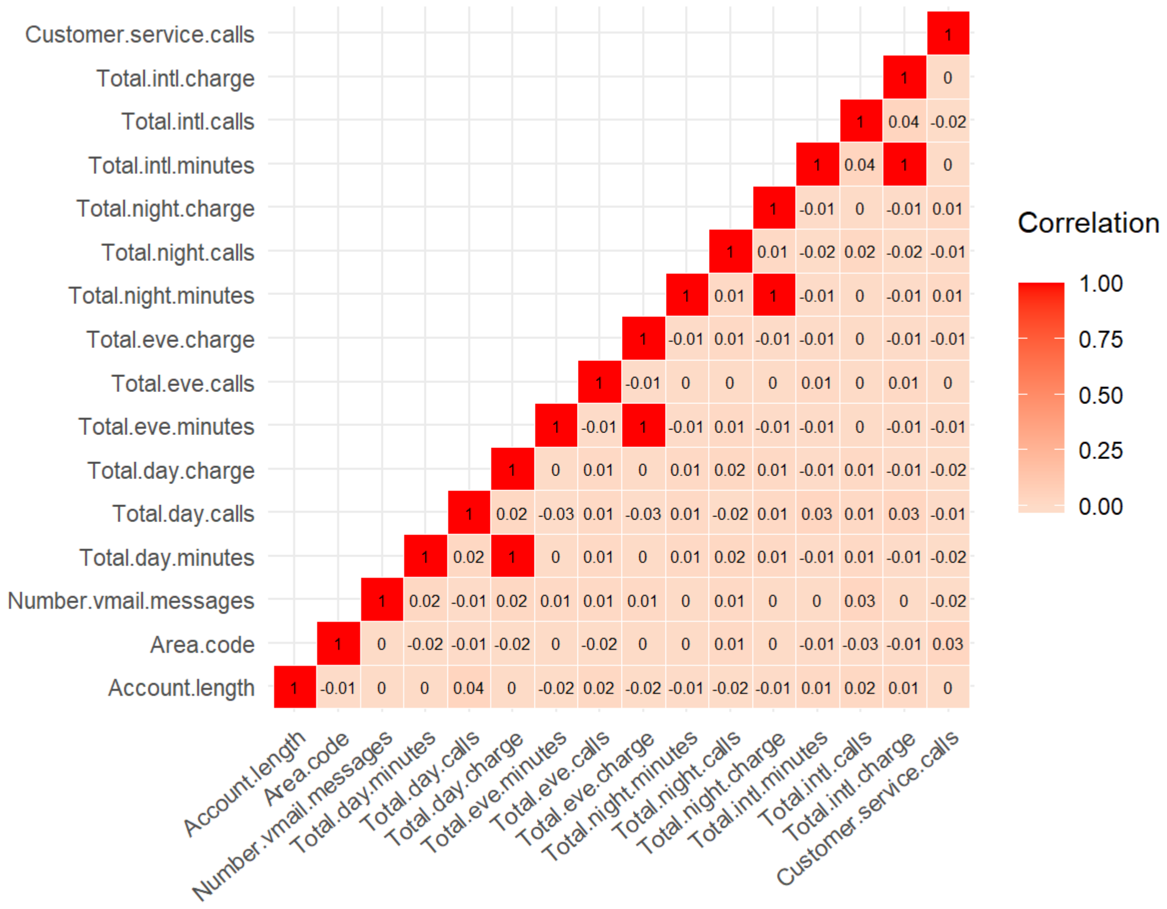
Table 1: Features Description

Variable	**Description**
State	Customer's state
Account.length	Time since subscription
Area.code	Phone number area code
International.plan	Has an international plan? (Yes=1,No=0)
Voice.mail.plan	Has a voicemail plan? (Yes=1,No=0)
Number.vmail.messages	Number of voicemail messages
Total.day.minutes	Total minutes used during the day
Total.day.calls	Total calls made during days
Total.day.charge	Total charge during days
Total.eve.minutes	Total minutes used during evenings
Total.eve.calls	Total calls made during evenings
Total.eve.charge	Total charge during evenings
Total.night.minutes	Total minutes used during nights
Total.night.calls	Total calls made during nights
Total.night.charge	Total charge during nights
Total.intl.minutes	Total international minutes used
Total.intl.calls	Total international calls made
Total.intl.charge	Total international charge
Customer.service.calls	Number of calls to customer services
Churn	Has the customer churned? (Yes=1,No=0)

gions have more churners, the churn percentage in all of them is around 14.6% which is the percentage in the training set as a whole.

About the percentage of churners, an approximate proportion of 14% in the training and test set (see Figure 5) might be a problem for classifier algorithms because the sample will bias the learning models to the majority class (non-churners). These models may behave poorly in the prediction power of the minority class (churners). In the next section, we discuss some ways to handle this potential problem.

Figure 1: Continuous variables correlation



3 Classification Methods

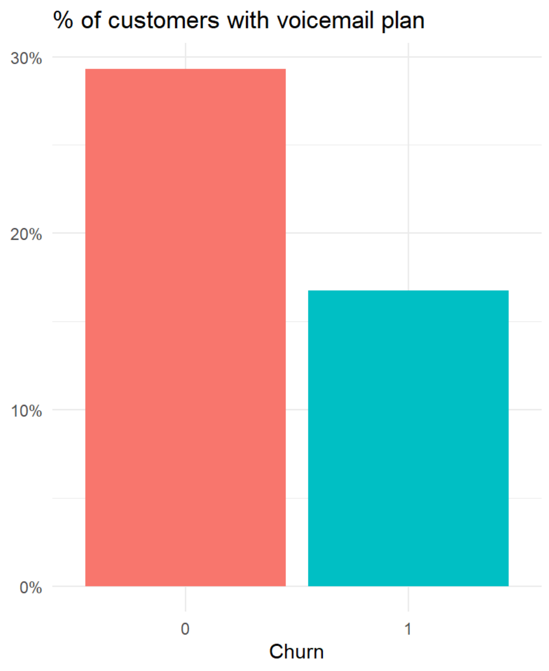
3.1 Evaluation Metrics

The task of churn prediction is a binary classification problem, i.e we have a $n \times p$ features matrix $\boldsymbol{X}$, and a target column vector $\boldsymbol{y} \in \{0, 1\}^n$, which represents churners as stated in the introduction. We want to learn functions that receive an p-dimensional feature input and outputs a binary value, 0 or 1.

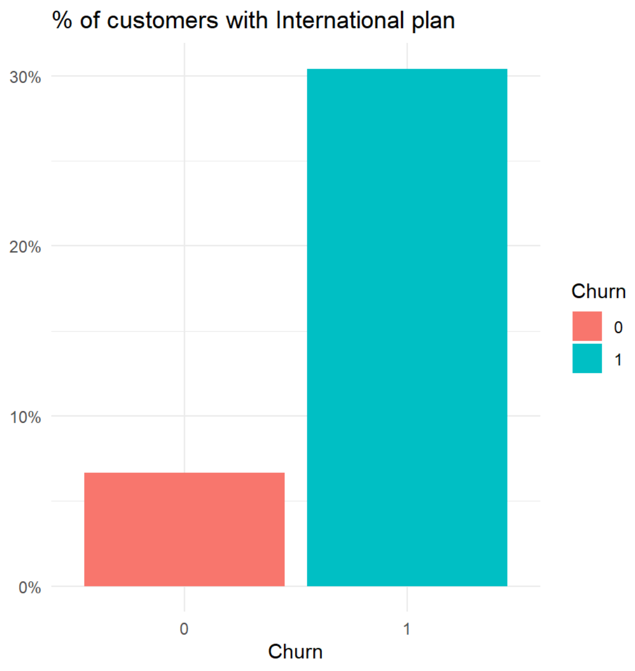
With a learned function/model in hands we can apply it in our test set and build the Confusion Matrix which summarizes how the model has performed.

From this table we can derive the following metrics:

- Accuracy: (TP+TN)/N;
- Recall: TP/(TP+FN),



(a) Voicemail clients proportion



(b) International plan clients proportion

Figure 2: Categorical Variables

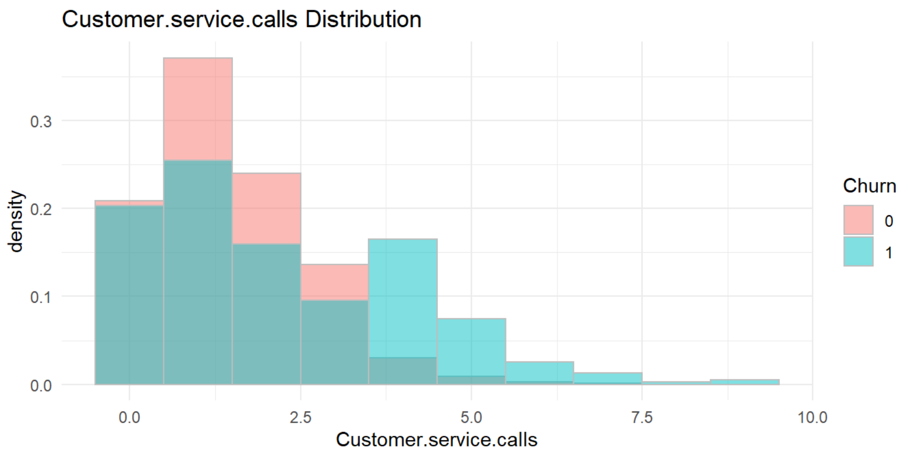


Figure 3: Customer services calls distribution

where N is the total number of test observations (TP+TN+FP+FN). Accuracy measures what proportion of the test set our model predicted correctly. When dealing with class imbalance, a model which assigns the majority class to everyone will have high accuracy but will perform poorly in predicting positive instances (churners).

The goal of our prediction algorithm is to correctly identify churners to proactively prevent

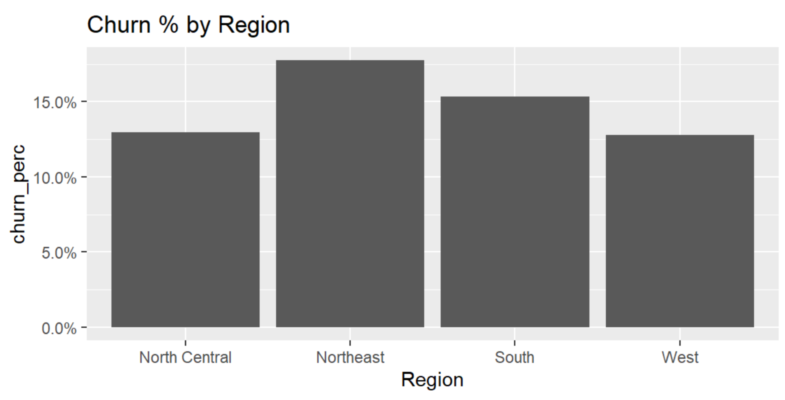


Figure 4: Churners distribution by region

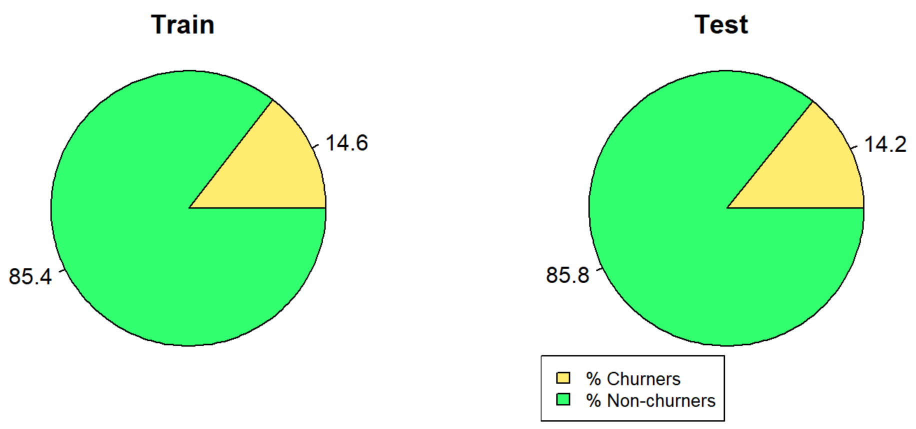


Figure 5: Churners percentage by dataset

		Model Outcome	
		1	**0**
Actual Value	**1**	(TP) True Positive	(FN) False Negative
	0	(FP) False Positive	(TN) True Negative

Table 2: Confusion Matrix

customer loss. So, in this case, a False Negative has much more impact than a False positive, because the company wants to predict as many churners as possible before they churn. A False Negative may lead to a customer loss that would be avoidable if predicted correctly, but targeting an offer/discount to a False Positive that wouldn't churn is not too harmful, because this client will stay loyal to the company.

Recall aims to measure prediction power among positive instances, i.e the proportion of actual churners we're capturing by the algorithm.

We're also gonna use the **AUC-ROC** metric, a number between 0 and 1 with measures the area under ROC curve, which is generated plotting false positive rate (FP/(FP+TN)) against true positive rate (TP/(TP+FN)) using different probability/odds thresholds outputted from models. A random classifier will have an AUC equal to 0.5.

Another important metric, widely used in literature and industry is the **lift**[5]. If the model yields probabilities, we sort the test predictions by these probabilities in a descending order, then we define the lift as the ratio between the percentage of positive entries in the top 10% lines ($\beta_{10\%}$), and the percentage of churners (positives) in the entire test set (β_0). We then define the **top-decile-lift** as the fraction:

$$\frac{\beta_{10\%}}{\beta_0}$$

We can use other percentile values instead of 10%, but we'll use the decile version since it's widely used performance metric for this kind of problem. A top-decile lift of 2 means that the model classifies two times more churners in the top 10% group than a random classifier. The lower bound is 0, but a non-random classifier should have a lift at least greater than 1. The upper bound occurs when $\beta_{10\%} = 100\%$, so in our case with $\beta_0 \approx 14.2\%$, the lift upper bound will be ≈ 7.042254. Such a maximum lift means that all entries in the top decile are churners.

We'll devote special attention to recall and top-decile lift and recall because of the imbalance problem. If a model predicts 0 to every test observation, it would be approximately 85% accurate, but recall would be zero, and lift undefined.

3.2 Resampling methods

The reference[5] presents a lot of data-level solutions to deal with class imbalance. The goal is to resample our training set to create a rebalanced new one. We'll focus on the three techniques mentioned below:

- (ROS) Random Oversampling: randomly replicates churners instances;
- (RUS) Random Undersampling: randomly eliminates the non-churners instances;
- (SMOTE) Synthetic minority oversampling technique[1]

With undersampling, we might be removing valuable information from the dataset. Oversampling may cause biased observations since we're going to replicate minority points several times to rebalance the data.

Smote oversamples the minority class in a smarter way, creating synthetic new data. To do that, for each existing (churner) observation SMOTE selects a random point out of the k-nearest-neighbors (we'll use the default k=5) and creates a new point by performing a random linear interpolation between the point in consideration and the selected neighbor. The article [1] shows details of this algorithm. We'll use the smote R implementation of performanceEstimation package[4]. For ROS and RUS we created our own code, available in the functions.R GitHub file.

3.3 Classification Algorithms and Results

K-Nearest-Neighbors

KNN may be the simplest classification algorithm that predicts the class by the majority of the K nearest neighbors classes of the point in consideration. The probability of a given observation belongs to the class j $(j = 0, 1)$ is the proportion of neighbors of the class j.

We performed 5-fold cross-validation and selected K that maximized the sensitivity (recall), which was $K = 8$. Then, we standardized all variables and trained the model using resampled training sets (by RUS, ROS, and SMOTE) and calculated performance metrics in the test set. The results are in Table 3.

	Accuracy	Recall	Lift	AUC
KNN	**0.8711**	0.1789	0.5240	0.1963
KNN + ROS	0.6987	0.6842	0.8383	0.4056
KNN + RUS	0.8066	0.6211	0.6288	0.4448
KNN + SMOTE	0.7196	**0.7368**	**1.1527**	**0.5124**

Table 3: KNN results

The first line labeled as just "KNN" stands for the metrics obtained training the model directly in the imbalanced original dataset. We bolded the best value of each metric. Although the original model got a high accuracy, it got a pretty bad recall (0.1789), which makes sense since we don't have many positive instances in the training set for the model learn by neighbors. In general, KNN performed badly, but the smoted dataset yielded acceptable metrics (AUC>0.5, lift>1), and it's better than a random classifier.

Logistic Regression

Logistic Regression models the log-odds of being positive (churner) as a linear function of the predictors.

$$\log\left(\frac{p(y_i=1)}{1-p(y_i=1)}\right)=\beta_0+\sum_{j=1}^{p}\beta_j X_{ij}$$

We estimate the coefficients by maximizing the likelihood of a binomial distribution model for the target. As before, we trained this model using the balanced resampled training sets and calculated the performance in the test set. Results follows in Table 4.

	Accuracy	Recall	Lift	AUC
LogisticReg	**0.8546**	0.1789	3.1438	0.8244
LogisticReg + ROS	0.7661	0.7684	3.1438	**0.8303**
LogisticReg + RUS	0.7631	0.7368	3.1438	0.8213
LogisticReg + SMOTE	0.7391	**0.8211**	**3.3533**	0.8292

Table 4: Logistic Regression results

Again, training in the imbalanced data yielded a poor recall. Although we lost accuracy with resampling, we got substantially higher recall which is a crucial metric to this problem, as we stated before.

Let's review the coefficients of SMOTE version of the regression, which was the best one in terms of recall and lift. We got more statistically significant features than in the model trained in the original data. In both cases the "strongest" predictor (in terms of z-value) was Customer.service.calls, followed by International.plan and Total.day.minutes.

```
## Coefficients:
##                          Estimate Std. Error z value Pr(>|z|)
## (Intercept)            -7.2338220  0.6458328 -11.201  < 2e-16 ***
## Account.length          0.0021405  0.0009932   2.155  0.03116 *
## Area.code              -0.0006780  0.0008921  -0.760  0.44727
## International.plan      2.5649186  0.1242593  20.642  < 2e-16 ***
## Voice.mail.plan        -2.1678067  0.3801978  -5.702 1.19e-08 ***
## Number.vmail.messages   0.0452635  0.0120397   3.760  0.00017 ***
## Total.day.minutes       0.0133785  0.0007205  18.568  < 2e-16 ***
## Total.day.calls         0.0051677  0.0019800   2.610  0.00905 **
## Total.eve.minutes       0.0072255  0.0008227   8.783  < 2e-16 ***
## Total.eve.calls        -0.0008711  0.0020065  -0.434  0.66420
## Total.night.minutes     0.0032962  0.0008402   3.923 8.74e-05 ***
## Total.night.calls       0.0031070  0.0020616   1.507  0.13178
## Total.intl.minutes      0.0767535  0.0146975   5.222 1.77e-07 ***
## Total.intl.calls       -0.0983638  0.0176142  -5.584 2.35e-08 ***
## Customer.service.calls  0.6687630  0.0298297  22.419  < 2e-16 ***
## RegionNortheast         0.4678842  0.1194881   3.916 9.01e-05 ***
## RegionSouth             0.2496615  0.1024976   2.436  0.01486 *
## RegionWest              0.0861546  0.1143103   0.754  0.45103
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Bagging & Random Forest

Bagging multiples decision trees in the dataset selecting different bootstrapped samples each time. The final prediction is given by the majority vote aggregating the results of the trained trees. Random Forest is like bagging, but each time a new split is made in a decision tree, a random subset of m predictor out of p are selected, and the split uses only these variables. We can draw a probabilistic output by taking the proportion of positive/negative votes among the trees.

For bagging, we used R's default hyperparameter of 500 trees. Again, we trained the model in the different training sets and compared performance in the test set. See results in Table 5.

	Accuracy	Recall	Lift	AUC
Bagging	**0.9610**	0.7789	**6.9163**	**0.9303**
Bagging + ROS	0.9505	0.7579	**6.9163**	0.9192
Bagging + RUS	0.9025	**0.8632**	5.8683	0.9165
Bagging + SMOTE	0.9265	0.8526	6.4971	0.9247

Table 5: Bagging Results

Good results in general, but RUS and SMOTE yielded better recall without much prejudice to other metrics, specially SMOTE, with a higher lift than RUS. Bagging trained in the original dataset was the best, except for recall, our key focus. Nevertheless, it produced a 0.7789 measure which is not too bad compared to what we have seen with logistic regression and KNN.

For Random Forest, we selected the value of $m = 6$, which optimized OOB error. We also did 5-fold cross-validation to choose the number of trees, but all tested values gave similar results, so we used the default value of 500. Results follow in Table 6. Again, resampled training sets enhanced recall, but the original dataset got a higher lift, very close to the upper bound of 7.0422.

Bagging-based ensembles The way we're overcoming imbalanced in the training set until now is by the data-level solution of resampling it to rebalance it. From now on, we'll focus on algorithm-level solutions, which use the original dataset and deal with imbalance internally. This part focuses on modifications in bagging in the bootstrap step; In the next part, we present a modified Adaboost which deals with imbalance by weighting a minority misclassification.

	Accuracy	Recall	Lift	AUC
Random Forest	0.9565	0.7368	**7.0211**	0.9281
Random Forest + ROS	**0.9655**	0.8105	6.7067	0.9216
Random Forest + RUS	0.8936	**0.8526**	5.9731	0.9197
Random Forest + SMOTE	0.9325	0.8421	6.3923	**0.9298**

Table 6: Random Forest results

Bagging-based ensembles as proposed in [3] are original bagging, but instead of bootstrapping the entire dataset, for each bag, one uses RUS, ROS, or SMOTE to create a balanced sample. This produces brand new models that we can call RUSBagging, ROSBagging, and SMOTEBagging, depending on the technique used. We used R implementations from [6]. Results (comparing with original bagging) are shown in Table 7.

	Accuracy	Recall	Lift	AUC
Bagging	**0.9610**	0.7789	**6.9163**	**0.9303**
RUSBagging	0.8906	**0.8632**	6.2875	0.9176
ROSBagging	0.8831	**0.8632**	6.6019	0.9070
SMOTEBagging	0.8846	0.8526	5.6588	0.9052

Table 7: Bagged-based ensembles results

All 3 methods yielded better recall measures than original bagging, but ROSBagging got a higher lift among them (although original bagging was better). The other metrics were pretty close among the 3 models.

AdaBoost and Cost-sensitive learning

AdaBoost[2] is a boosting algorithm for classification problems. It uses weak learners to fit the data and then iteratively uses the error to retrain the model and improve the overall learner. As the final set of algorithms, we used the implementation of [6] of the modified version of AdaBoost proposed in [2] to deal with class imbalance. The goal is to introduce a higher cost for misclassifying the minority class (churners). The article proposes three different modified formulas with costs (AdaC1, AdaC2, and AdaC3), but we just tested the AdaC2, which was the only one implemented in IRIc package[6].

To adjust the cost parameter we created a validation set using 1/8 of the training set with the same churn proportion. Figure 6 shows the Recall/Accuracy measures for different costs values. This cost will penalize the algorithm for misclassifying a churner, so if the weight is too high we're gonna end with a model that only yield positive classes and has a poor overall accuracy in spite of recall close to 1.

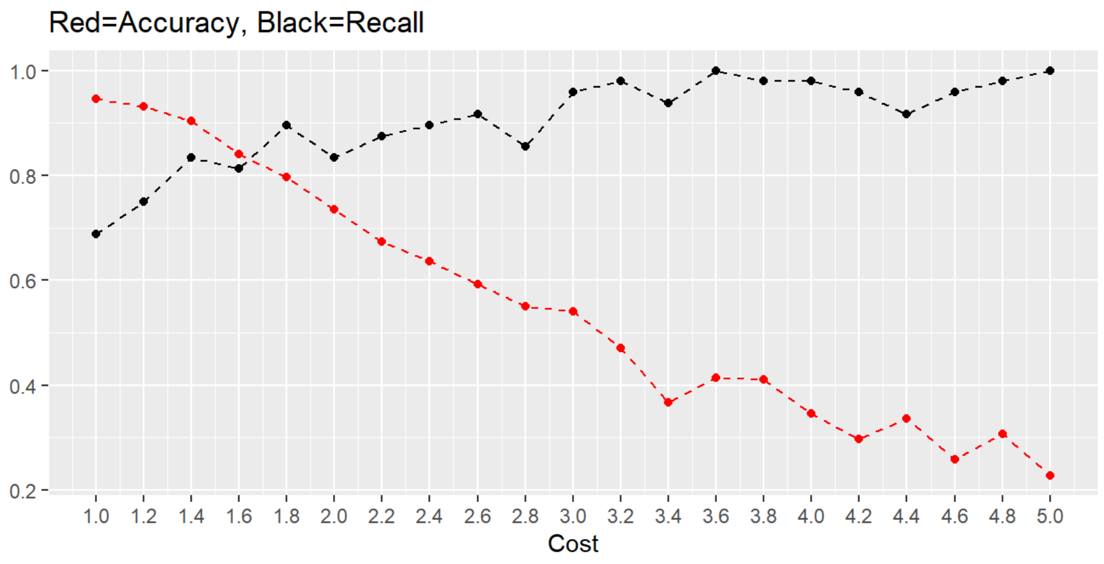


Figure 6: Cost parameter tuning

Although we argued that Recall is a crucial measure in our case, we don't want a low accuracy, so we choose 1.4 as the cost, which seems to be a good equilibrium point in the validation set.

Then we applied the original AdaBoost, and compared to the AdaC2, the version with higher cost to false negatives. The results are shown in Table 8.

	Accuracy	Recall	Lift	AUC
AdaBoost	**0.9505**	0.7158	**6.7067**	**0.9270**
AdaC2	0.8816	**0.8316**	6.4971	0.9257

Table 8: Boosting results

Original Adaboost performed better in all metrics except Recall. Losing some accuracy, we can get a better recall using AdaC2, without too much prejudice to the other measures.

4 Final Results and Comments

Table 9 shows the metrics for all models tested in this project. Although we have these champions for each metric, excluding KNN and the basic Logistic Regression all the models performed very well. A recall lower bound of 0.71 (AdaBoost) means we're identifying 71% of the real churners.

A lift lower bound of 3.1 (Logistic + RUS/ROS) means we have 3.1 times more churners in the top decile than a random classifier would have. This is approximately 44%, but excluding logistic, all lifts were higher than 5, which corresponds to around 71% of churners in the top decile. Our lift champion, Random Forest almost reached the maximum lift, so using these models to target the top decile with a suitable offer would be a good idea to avoid churn.

	Accuracy	Recall	Lift	AUC
LogisticReg	0.8546	0.1789	3.1438	0.8244
LogisticReg + ROS	0.7661	0.7684	3.1438	0.8303
LogisticReg + RUS	0.7631	0.7368	3.1438	0.8213
LogisticReg + SMOTE	0.7391	0.8211	3.3533	0.8292
KNN	0.8711	0.1789	0.5240	0.1963
KNN + ROS	0.6987	0.6842	0.8383	0.4056
KNN + RUS	0.8066	0.6211	0.6288	0.4448
KNN + SMOTE	0.7196	0.7368	1.1527	0.5124
Bagging	0.9610	0.7789	6.9163	**0.9303**
Bagging + ROS	0.9505	0.7579	6.9163	0.9192
Bagging + RUS	0.9025	**0.8632**	5.8683	0.9165
Bagging + SMOTE	0.9265	0.8526	6.4971	0.9247
RForest	0.9565	0.7368	**7.0211**	0.9281
RForest + ROS	**0.9655**	0.8105	6.7067	0.9216
RForest + RUS	0.8936	0.8526	5.9731	0.9197
RForest + SMOTE	0.9325	0.8421	6.3923	0.9298
RUSBagging	0.8906	**0.8632**	6.2875	0.9176
ROSBagging	0.8831	**0.8632**	6.6019	0.9070
SMOTEBagging	0.8846	0.8526	5.6588	0.9052
AdaBoost	0.9505	0.7158	6.7067	0.9270
AdaC2	0.8816	0.8316	6.4971	0.9257

Table 9: Final Results

References

[1] N. V. Chawla et al. "SMOTE: Synthetic Minority Over-sampling Technique". In: *Journal of Artificial Intelligence Research* 321-357 (2002). DOI: https://doi.org/10.1613/jair.953. URL: https://www.jair.org/index.php/jair/article/view/10302.

[2] Yanmin Sun et al. "Cost-sensitive boosting for classification of imbalanced data". In: *Pattern Recognition* 40.12 (2007), pp. 3358–3378. ISSN: 0031-3203. DOI: https://doi.org/10.1016/j.patcog.2007.04.009. URL: https://www.sciencedirect.com/science/article/pii/S0031320307001835.

[3] Mikel Galar et al. "A Review on Ensembles for the Class Imbalance Problem: Bagging-, Boosting-, and Hybrid-Based Approaches". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 42.4 (2012), pp. 463–484. DOI: 10.1109/TSMCC.2011.2161285.

[4] L. Torgo. "An Infra-Structure for Performance Estimation and Experimental Comparison of Predictive Models in R". In: *CoRR* abs/1412.0436 (2014). URL: http://arxiv.org/abs/1412.0436.

[5] Bing Zhu, Bart Baesens, and Seppe K.L.M. vanden Broucke. "An empirical comparison of techniques for the class imbalance problem in churn prediction". In: *Information Sciences* 408 (2017), pp. 84–99. ISSN: 0020-0255. DOI: https://doi.org/10.1016/j.ins.2017.04.015.

[6] Bing Zhu et al. "IRIC: An R library for binary imbalanced classification". In: *SoftwareX* 10 (2019), p. 100341. ISSN: 2352-7110. DOI: https://doi.org/10.1016/j.softx.2019.100341. URL: https://www.sciencedirect.com/science/article/pii/S2352711019301700.