Churn Prediction - Report

Machine Learning

Eugenio Leoni - 3119306

May 2021

1. Preprocessing

1.1 Dataset Exploration and Missing Values

Upon inspecting the dataset, we can notice that there are 10 missing values for one of the predictors, *ChargesTotal*. Though, we can see that it is missing only for observations having time=0, indicating new customers. Hence we can just correctly set the total amount that they spent up to that point in time to zero.

Most of the predictors are categorical, so are to be encoded as dummy variables. To do this, I carefully joined the train and test datasets and used a simple Pandas function. I took some extra care to remove identical dummies resulting from some columns having the same third option "*no internet service*". The Python code can be found in the Jupyter Notebook[1] and it follows the same pipeline as in this report. From a quick inspection, as in Figure 1, we can see that the few continuos predictors appear to be related to our response variable.

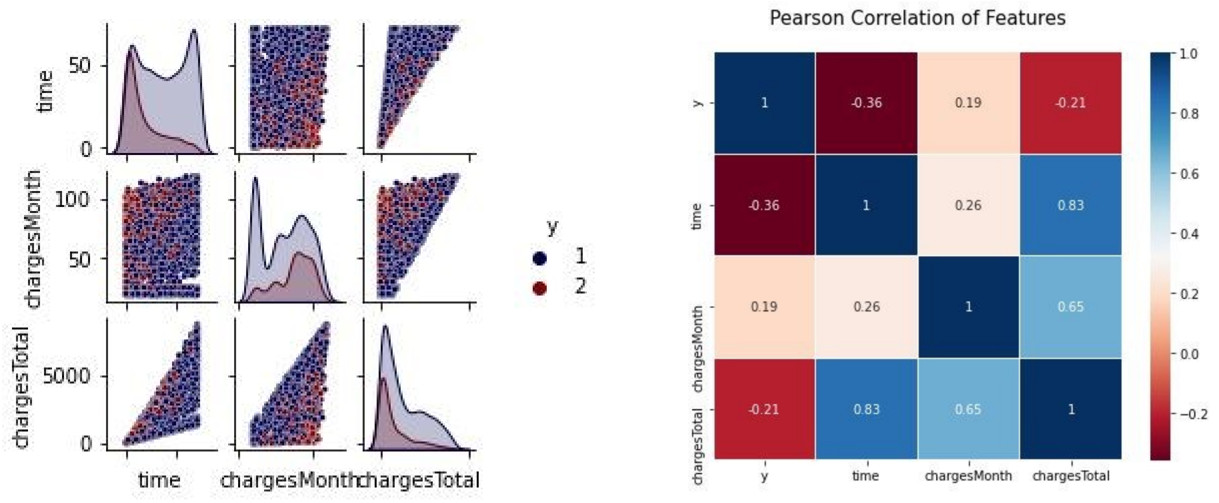

Figure 1. On the left are the distribution plots of the variables, with the colors indicating the classes. We can see that time and chargesMonth for non-unsubscribed users have very different peaks, while chargesTotal has a kinda similar distribution in the two classes, but with different absolute values. On the right is the correlation heatmap, with the first row being about the y. Although Pearson Correlation might not be best suited to a categorical variable, it allows us to have a first idea about the dataset at hand

1.2 Feature Engineering

Adding features which aren't too much correlated with the current ones and that capture signal in a different way improves performance.

[1] I cleaned most of the notebook so that it is easily runnable from start to end to get the final predictions. For example, I left just a cell for hyperparamter tuning, as it is almost the same script for all models. I also collected utility function in the homonymous .py script, to make relevant things more evident

Firstly I tried adding a measure of how much the subscription price changed relative to previous months, under the idea that a relevant price change may prompt some clients to go away, or it may signal an increase in the number of services subscribed. To do this, I created two variables:

- *avg_surprise*, which multiplies the current monthly expense for the number of months the customer has been subscribed. This returns a measure of how much the customer would have paid totally if he always paid as in the current month. Then I subtract the resulting amount by the total sum spent by the customer.
  If this quantity is positive, it tells us that, if the customer always paid as in the current month, he would have paid more overall, thus at some point in time prices must have increased.

- The second measure, *avg_surprise_2*, follows the opposite route. Firstly I calculate the *avg_expenditure* as the total expenditure divided by the time_steps. Then, I subtract it by the expenditure in the current month, and I divide the result by the absolute value of the average expenditure. Thus, when *avg_surprise_2* = 0, it means that the client's *avg_expenditure* is as in the current month, while positive or negative values indicate how much, in % terms, the current month's expense is greater or lesser than the average monthly one. When time=0, I use a caveat to make *avg_surprise_2* and *avg_expenditure* equal to zero.

- Lastly, I also included *avg_expenditure* as a feature

Except for the *avg_expenditure*, the two surprise metrics are very little correlated with other covariates, though, at first glance, they don't look very much related to the response as well, indeed I ended up removing *avg_surprise_2*.

Finally, I thought that some interaction terms could be very relevant in this dataset, more than quadratic terms. I tried to use the PolynomialFeatures Pandas function to create interaction among all the variables, to later pass them to a model with regularization. Though, this increased p and variance way more than it allowed to reduce bias, even with a strong regularization.

Thus, instead, I tried to add 'ad hoc' interactions. For example, I evaluated the interaction between 'backup' and 'time' under the idea that it is costly to abandon a service when one potentially has much data on its backup. I also figured that if payment methods are automatic, then *avg_surprise* plays less of a role, so I interacted these variables as well. I also tried to interact '*backup*', '*support*' and '*protection*', under the idea that if one is subscribed to more services, it is harder to unsubscribe and do without, and viceversa. Though, evaluating these features individually and all together, through a Logistic Regression, they didn't seem to reduce the Cross-Validated test error at all, so I ended up keeping things simple and avoiding interactions.

1.3 Data Trasformation

Some models (e.g. KNN, SVM, models with regularization etc.) need standardization of the input, so that the latter are reduced to mean zero and unit variance. Thus I created also a standardized version of the dataset, including dummy variables in the process, as suggested in a Piazza discussion. To standardize I fitted the standardization procedure on the train set and applied it to both the train and test set.

Since the classes are unbalanced, I also tried downsampling and upsampling techniques, taking care to upsample only the train set, and standardizing the cross-validated test error estimate by dividing for the test_set length, as, otherwise, downsampling would have often given smaller test error due to the test_set size being reduced. Unluckily, at their best they yielded a performance similar to the original one. I concluded that, since I was already not looking at accuracy, these techniques were somewhat less relevant. Furthermore, probably the decision boundary was 'simple' enough to learn, so that even with class disparity the model did well enough to capture signal from the minority class observations.

Assessing Models and Variables

2.1 Variable importance

Since n is small and the decision boundary of the task seemed linear judging by some baseline models, I decided to carry out a backward subset selection routine. I chose to remove, at each iteration k with k = p,…,1, the variable which yielded the worst model in terms of train weighted misclassification error. Looking at the train error in this case is fine, since we compare models with the same degree of complexity.
I calculated the train error on a LR since it fast to compute and is among the best scoring models. I then evaluated the best selected model at each k using a Logistic Regression and a Random Forest, so the latter could give some sort of feedback also on whether some variables were useful for the interactions, and at this stage it is relatively fast to run.

The most important variables, or the 4 survivors according to this procedure, are: '*time*', '*chargesTotal*' and '*paperless_billing*'\'*electronic_check*'.

It turns out that most categorical variables don't contain much signal which isn't found in the numerical variables, thus removing some of them doesn't even scratch the test error. To get an idea, removing all the categorical variables yields a worsening of the test error by 'just' 20-25 points, around 15%, and just a few categorical variables account for most of this increase in bias. Since less complex models are preferrable unless additional predictors allow for a meaningful decrease in bias, I ended up removing the noisiest variables, notably: *gender*, *married* and *phone*. These variables were non relevant also using other approaches henceforth described.

This procedure still carries some stochasticity, and, unlike best subset selection, it also isn't guaranteed to find the best combination of the predictors. Hence, I also looked at some feature importance metrics.

Firstly, I used Random Forest, evaluating features based on mean impurity decrease achieved by their splits: each variable is deemed more important the more its splits immediately lead to purer terminal nodes (a purer node is one containing a greater percentage of the majority class). This way we can also get a glimpse at whether some variables are important as interactions. From Figure 2 we can see the results of this procedure, and we obtain results in line with the previous approach: continuos features dominate, except for a handful of categorical features which appear to be meaningful enough.

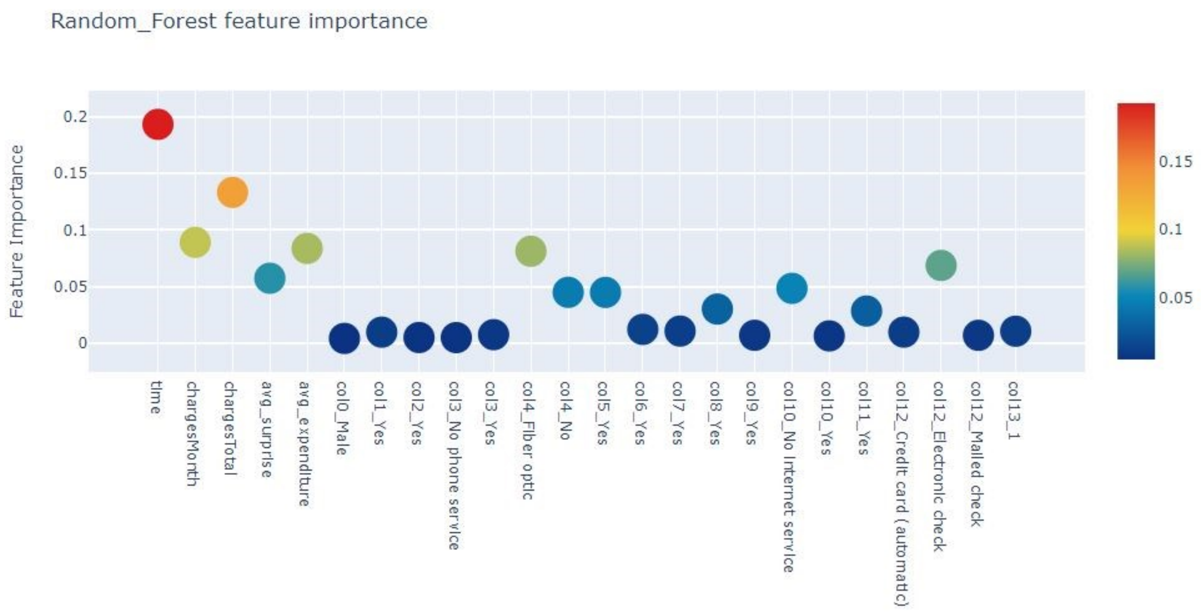

Figure 2, Random Forest Feature Importance based on Mean Impurity Decrease (MDI)

Still, we don't know the direction of their effect towards the response. To get this information, I decided to look at the coefficient of a LR fitted model. I used the standardized dataset so that the betas refer to variables on the same scale. Then, I fitted the model 10 times to account for stochasticity of the estimates, and I plotted the mean values of the coefficients. Since they contribute non-linearly to the prediction, we can't get an exact measure of their impact on the response, but we can evaluate their sign and magnitude. We obtain, once again, a confirmation of previous results, but the sign of the relationship may be surprising for some variables.

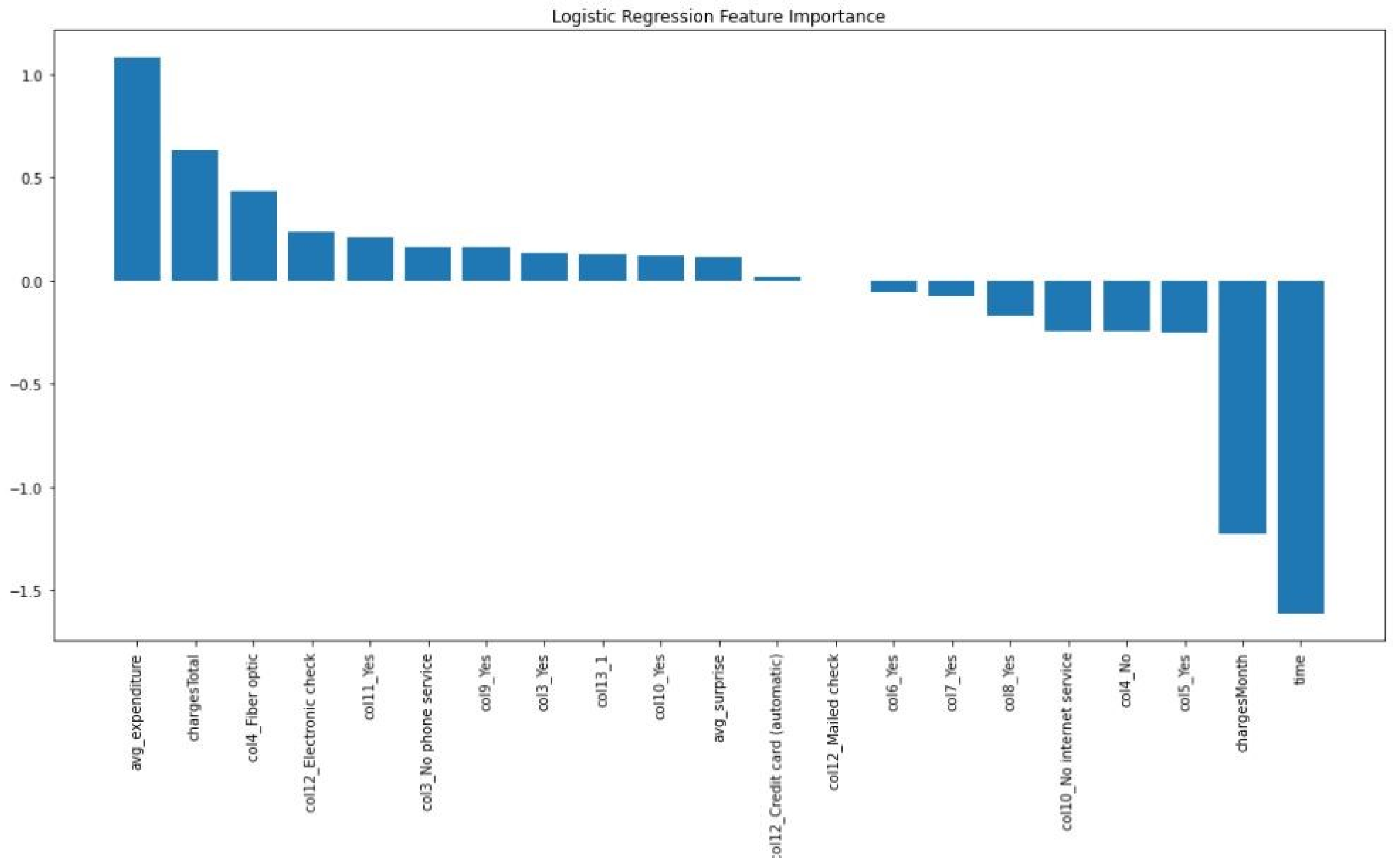

Figure 3, Logistic Regression Feature Importance. Coefficient values greater than 0 mean that as the feature increases so does the probability of churn

First of all, *avg_expenditure* is positive (increases chances of churn)[2] while *chargesMonth* is negative. Removing *avg_expenditure* causes *chargesMonth* to have a much smaller coefficient. Coherently, fitting the backward subset selection on a dataset without *avg_expenditure* leads to a more premature elimination of *chargesMonth*. A possible explanation is that these predictors can capture two opposite signals: a higher price means more services to which the customer is subscribed, thus a negative effect. At the same time, though, higher price tends to increase churn probability. Thus, *chargeMonth* may ended up capturing the former, while the latter is captured by *avg_expenditure.*

Furthermore, *chargesTotal* is positive, but it becomes negative if we remove *time*. This is easier to interpret: since *time* is the most important signal, if we remove it then *chargesTotal* tries to take its place, and we have confounding. Overall, in my opinion *chargesTotal* is sort of an interaction between *time* and *avg_expenditure*, meaning that, keeping other variables fixed, the more a customer has paid so far, the more he is likely to churn.

We can also notice that having a *streaming TV* or *electronic checks* or *paperless billing* tend to increase the chance of churn. I think that there is an omitted variable bias at play: these variables signal young customers, and young customers are probably more likely to churn. This may be due to several reasons: they may be offsite students, who eventually churn, they may be relatively poorer, hence 'tougher' clients to retain, or they may take advantage of offers and not stick afterwards (e.g. streaming services). It would be nice to collect the age, to verify this hypothesis, which may be relevant. Also having a *Fiber optic* increases churn probability, it may be for the same reason as above, or for customer unsatisfaction, or maybe people who are reached by the fiber are also more likely to be reached by other telecommunication companies, so there is more competition.

Finally, having more *phone lines* connected decreases the chances of unsubscribing, I wonder if this captures clients who are firms instead of people. Being subscribed to *security\backup\protection services*, as expected, also decreases the chances of churn. As we can see from figure 4, the proportion of people unsubscribing is much lessened when they are subscribed to a backup service, although it is even less when they aren't subscribed to an

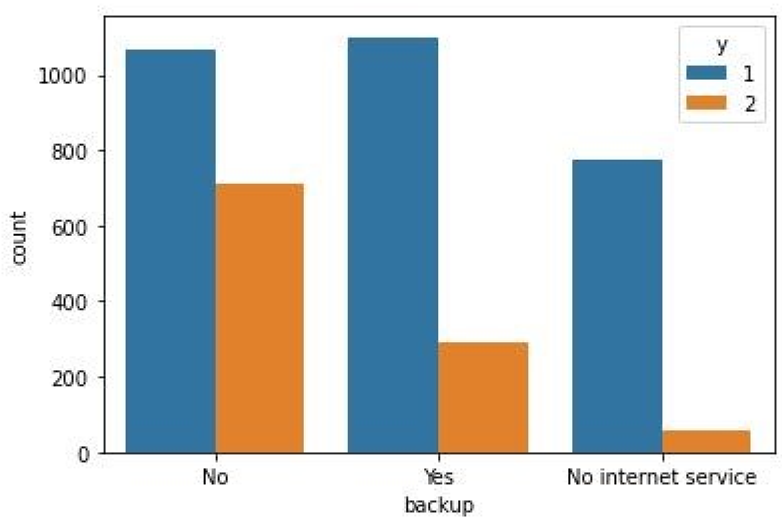

Figure 3, Distribution of y over the 'backup' variable

[2] I double-checked this by looking at predicted probabilities with just time as feature: it had a negative coefficient, and the more it increased, the more the probability of churn decreased

internet service at all (and thus they pay less)! This may mean that there are some customers who only care about the prices, while other deem highly the services offered.

Much more analysis is needed, but, also due to interactions not playing a relevant role, a good retention strategy can be a simple one, like keeping prices low for most customers while encouraging others to use more services. Most importantly, It is crucial to retain customers in the first months, so that they become more fidelized or accustomed to the status quo. It would be nice to collect data on the age of customers, or on lagged month expense.

2.2 Models and hyperparameter tuning

Given the relatively small sample size and the unequal class distribution, more flexible models are more at risk of overfitting. I used baselines to understand whether a linear decision boundary would fit better the task. By evaluating non-linear decision boundaries, as those produced by SVM, KNN, QDA, it seems that indeed linear boundaries are better in this case, such as those obtained from LR or LDA. I still optimized DecisionTree Ensembles as well, because I thought that interactions would have played a bigger role in this dataset.

To evaluate the models I used Cross-Validation. I initially used k=5, as, given the sample size, I was particularly scared by the variance of the test error estimate. Eventually, I resorted to k=7 or k=10, as it was fairer when used to compare more complex models with less complex ones, as the former tend to improve relatively more as new observations come in. Also, it provided test error estimates more comparable to the ones on the website, so they were easier to look at and to remember. As for the code, I used the GridSearch method provided by SKlearn, which allows to try out a model with several combinations of parameters, keeping the same random state.

After evaluating baseline models, tweaking some hyperparameters here and there, my main models of choice were: LDA, LR, Gaussian_NB, RF, DecisionTreeBoosting. I assigned class weights\priors, or, equivalently, set a threshold, for each model, to account for the asymmetric loss. Priors and class weights are set to give the second class 4-6 more times the importance of the first class, but I chose the exact value using CV, based on the model.

As for LR, a tiny regularization seemed to help, and I chose a liblinear solver since it is suggested for small datasets.

Same goes for LDA: a really tiny shrinkage of the variance-covariance matrix was useful.

Gaussian Naïve Bayes proved to be really good at predicting the second class, while a bit lacking on the first one. Indeed, curiously enough, I had to give the first class a very high prior in order to improve the cross-validated test_error performance! Giving a higher prior to the second class, instead, seemed to increase the false negatives much more than the decrease in false positives. This peculiarity of the G_NB seemed interesting when considering stacking.

As for RandomForest, I chose a number of estimators which seemed to provide convergence while not being too computationally expensive, and I played around with the depth of the trees and the m parameter. I ended up choosing 45 as the number of max_leaf_nodes, and m=16% (almost identical to m=log2(p) in this setting). Given that few variables dominate over the others, it seems reasonable to use a lower m than the default m=$\sqrt{p}$ one. To evaluate m, I used an initial range of values, and went more deeply at each iteration, centering the range around the best value according to the previous search.

As for DecisionTree Boosting, implemented through the AdaBoost algorithm, I evaluated it with d=1 and d=2, tweaking the learning rate and B to be lower in the second case, to avoid overfitting. This allows the two models to capture signal in a slightly different way, as the second one is more built around interactions. I optimized them with regard to these two hyperparameters, and they perform similarly, with the first one having a better score and the second one having a lower standard error.

Overall, except for the G_NB which performs much worse, the other models perform similarly, with the mean test error estimate being around 175, and the standard error being 15-20. Also in terms of AUC score, the models are very comparable, with an average score of 0.85. G_NB makes an exception, scoring 0.81, which is coherent with what noted above.

If I had to choose a model, I would have gone for LR, since it is not complex and also yields the lowest test error. Though, I decided to carry out stacking, as it seemed to me that the models captured signal in slightly different ways, making it worth to try it.

In order to do stacking, I used the pre-built StackingClassifier method of SKlearn, which passes the probability estimates made by the individual fitted models to a Logistic Regression, to which I fed the class weights. This LogisticRegression is optimized using an internal CV method, while the models are trained on the whole train_set. Effectively, this is comparable to weighting the probability estimates made by the individual models, choosing weights that minimize the weighted cross-validated misclassification error.

I evaluated several random combinations of the optimized versions of the aforementioned models laying them out both with class weights and without them, to let them capture different kinds of signal. I picked the best such stacking, which yielded a slight improvement over the simple LR. The winning stacking combination included: RF and AdaBoosting (d=2) with class_weights, combined with LR, LDA, G_NB, RF without class weights. In general, It seems that more models with no class weights are preferrable, as already the final estimator has its own class weights.