Predicting Churn: Final Report

Ashton Reed

Problem Statement:

Churn is a concern for most companies and even a crisis for some. High churn rates can cost companies tens of thousands and can be an issue both internally (employees) and externally (customers). There are plenty of articles out there to inform companies of the danger churn presents to their business. However, I'm sure most companies following a recurring revenue model have already figured this out for themselves and recognized what churn can do to their monthly/quarterly/yearly expected revenue. There are several operational changes that can be made to reduce churn, and I am working on a LinkedIn post about it. However, if you read my post and determine that you have made all the adjustments possible to better accommodate your customers and still would like to learn more about predicting churn, please read on.

It's important to remember that there is no one-size-fits-all churn model. The generalized model for this project is built around a publicly available sample dataset of a new telecom company's client base, but it is also possible to build a model based on data from a CRM, such as Salesforce. Please keep in mind that in order to build and model and fit it to data, you must first have data, and while it does not necessarily have to be the cleanest, it does at least need to be fairly accurate. You cannot build a good model on bad data. No matter how impressive the model may seem with its pipeline and hyperparameter tuning, it will not be very accurate if it is trained on inaccurate data. For example, if you are not regularly updating the information in your CRM with a client's final payment date once they have churned, it will be difficult to add that customer to the "Churned" column when building the model for your data. A churn model will never be 100% accurate, but the more [pertinent and accurate] the data with which you can build and train a model, the better your chances are that you have more accurate predictions.

Despite how it may appear at first glance, churn analysis is not a simple classification problem but can be considered, more specifically, an issue of survival. Using publicly-available sample data, I will build a deep learning model that utilizes survival analysis to predict whether or not a customer will churn. By predicting churn, this model will help companies, such as telecom companies or SaaS companies, pinpoint what changes need to be made in order to retain customers. The deliverables for this project are a series of Jupyter notebooks containing the code for the model, two milestone reports, a final paper, and a slide deck.

Data source: https://www.kaggle.com/raumonsa11/churn-telco-europa

Explanation and Initial Analysis of the Dataset:

With a dataset like this, we can only see how many days a customer was with the company. For customers that have not yet churned, we can assume that their signup/join date was “DAYS_LIFE” number of days before the cutoff date. A survival analysis takes into account the fact that the data is right-censored, meaning that everyone will churn at some point, but for customers who have not churned yet, the event (churn) did not happen during the observed period of time. For customers who have already churned during the observed period, without a date, we have no way of knowing how long ago they left the company. Was it early on in the company’s history? Does the product have new features or improved usability? These are reasons that it is important to keep track of exact dates for later use if possible. It could also be helpful to reach out to customers who leave in order to request they fill out a survey of their experience with the company and reason for leaving. This might reveal a feature not contained in the specific dataset but one that could have a high predictive power if documented and subsequently used to build the model.

For several of the articles that I read, the assumption was that most customers in the dataset have not yet churned, leading to a class imbalance. However, this company seems to have the opposite issue--nearly 8 out of every 9 customers have already left the company, which also leads to a class imbalance. This should cause us to question why so many customers have left a telecom provider so quickly.

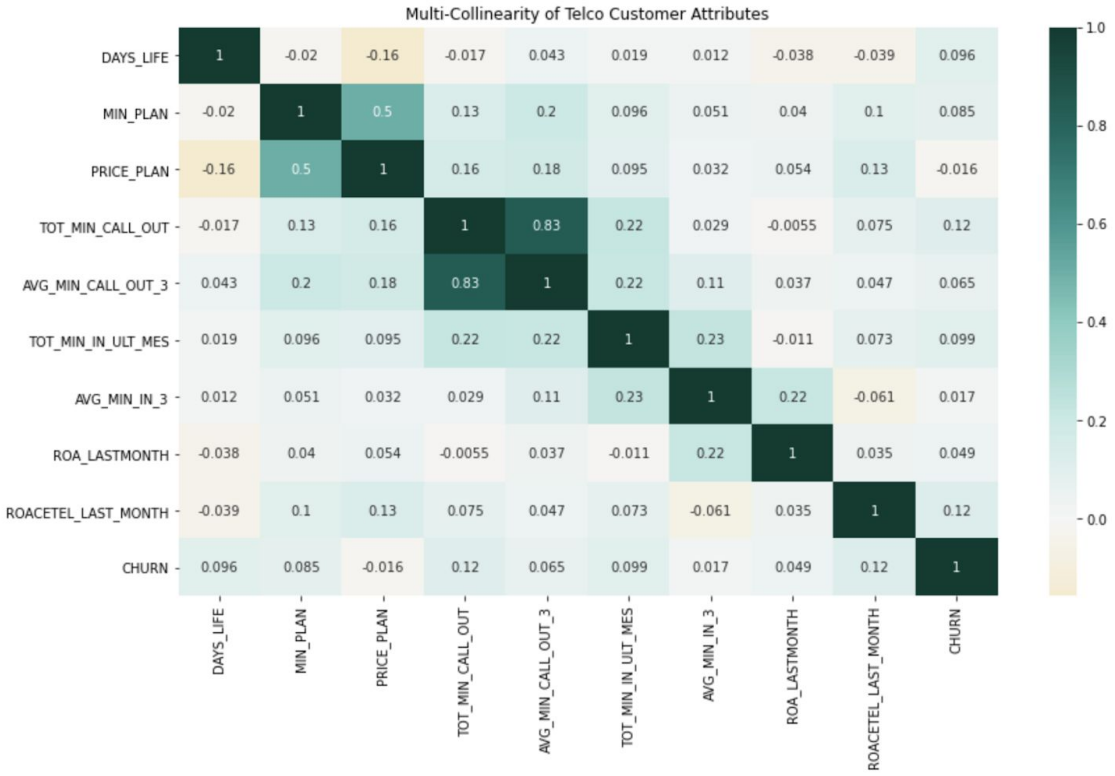
It is important to keep in mind that there can also be voluntary and involuntary churn. Voluntary churn is when the customer chooses to leave the company, oftentimes because they found a better deal with a rival or because they no longer need the product. Involuntary churn happens when the company cuts ties with the customer--for example, if the customer has overdue bills to the company. If the different types of churn are known for each observation, it could be useful to have a separate model for each type when possible. However, this dataset does not contain the customer’s reason for leaving/churning, so a further analysis such as that is currently out of the scope of this project.

Whether or not a customer churns is an important piece of churn analysis, but it is not the only one--time is also an important part of churn analysis. In order to predict churn, it is necessary to know how long each customer has been with the company. In the dataset for this project, both of these feature columns were well-populated. Two observations had negative numbers for tenure (“DAYS_LIFE”), so those two were dropped. In some cases, state or city values were not consistent, and neither was antenna technology. (In several thousand instances, “STATE_VOICE” and “STATE_DATA”, “CITY_VOICE” and “CITY_DATA”, or “TEC_ANT_DATA and

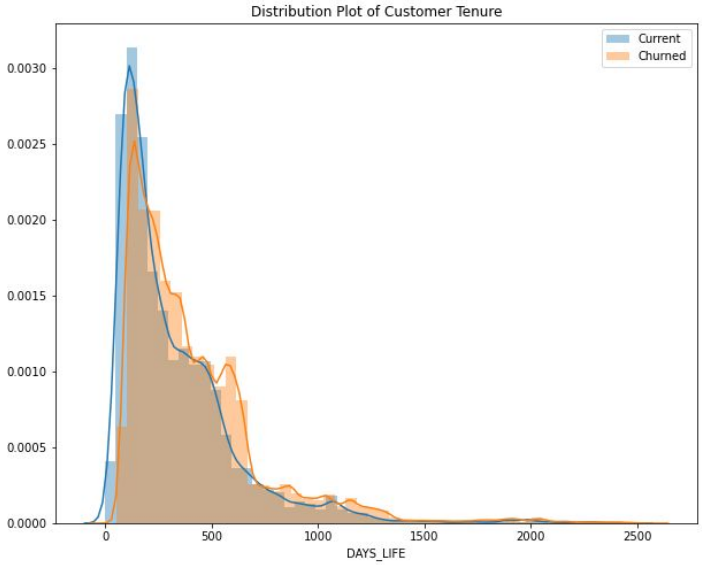
“TEC_ANT_VOICE” were different.) Only the city/state data had missing values. Those were filled in the machine learning pipeline notebook with the value of “9999” before being one-hot-encoded, but they were left as-is in the first Jupyter Notebook. Several categorical features had values that were outside the bounds laid out in the “Data Dictionary” portion of this dataset. Those were left as-is with comments regarding them in the first notebook. There were more than 1000 occurrences of this for several different features. As those will be one-hot-encoded, I did not attempt to figure out a better value to substitute. As mentioned in the first notebook also, the categorical features should not have been assigned numerical values except as a means of ranking. Realistically, this assigning numerical values to categorical features does not make sense because these are not necessarily ordinal, despite being converted to numerical values. For example, with DEVICE, it does not make sense to assign Sony a value of 1, Apple a value of 4, and "Other" a value of 18. Despite knowing the appropriate string to which it was possible to map DEVICE or DEVICE_TECHNOLOGY, I refrained. This is because there were several DEVICE_TECHNOLOGY entries greater than 5, and city/state data that also fell outside the bounds. DEVICE would have therefore been the only feature I could have mapped to a specific set of words, but I chose to leave the entire set of categorical features as numerical until one-hot-encoding them. This proposed mapping can be done for readability after one-hot-encoding if desired--for DEVICE and for most (but not all) observations for DEVICE_TECHNOLOGY. At the end of Part 1 (“Cleaning the Data”) in the first notebook, I converted the categorical features to the “object” data type.

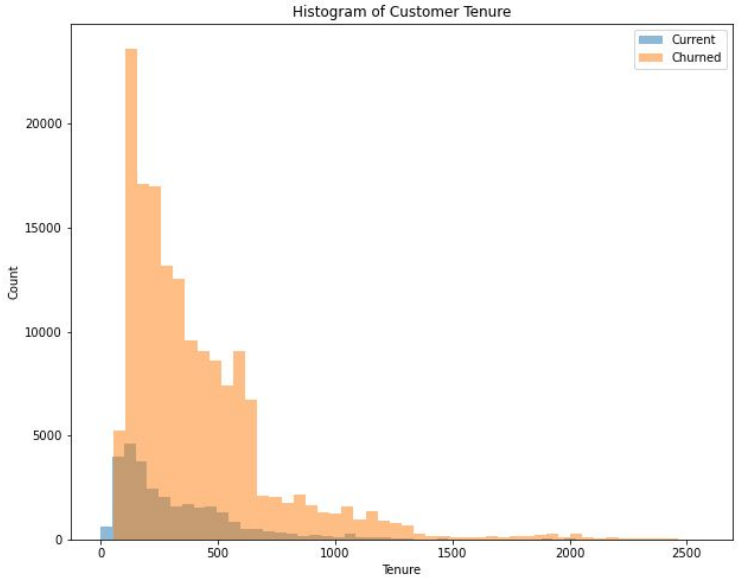
Further Exploration and Storytelling:

To further explore the dataset, I began by creating a heatmap of the continuous variables in order to check for collinearity. The highest (0.83) was found to be between the variable for the average number of minutes of outgoing calls over the last three months (“AVG_MIN_CALL_OUT_3”) and the total number of minutes of outgoing calls (“TOT_MIN_CALL_OUT”). This is not very surprising as a user's behavior over the last 3 months contributes to their behavior over their total time with the company. The second-highest correlation (0.5) was found to be between the price of a plan (“PRICE_PLAN”) and the number of minutes contained in the plan (“MIN_PLAN”). We would typically expect some correlation here, but the fact that the correlation is not higher and because we did not see the exact same spikes in the histogram leads us to believe that price changes were made to the existing plans, and perhaps even new plans were added. An important question to ask would be when were these changes made, and how many customers churned after that. If a customer is not satisfied with and not using much of their current plan, it may be possible to retain them as a customer by downgrading their plan to something that is a better fit for them.

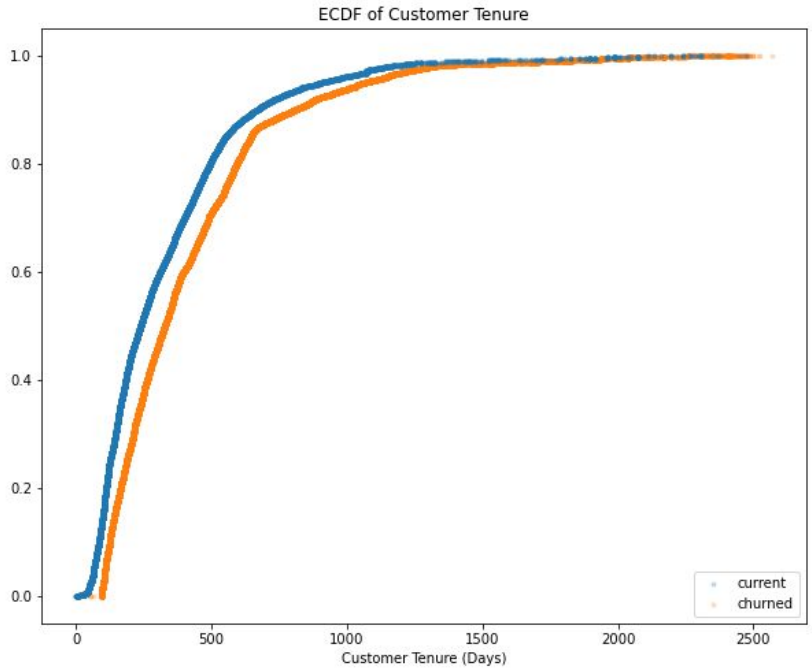


The next thing I looked at was customer tenure ("DAYS_LIFE"). I created both a kernel density/distribution plot and a histogram, as seen below.



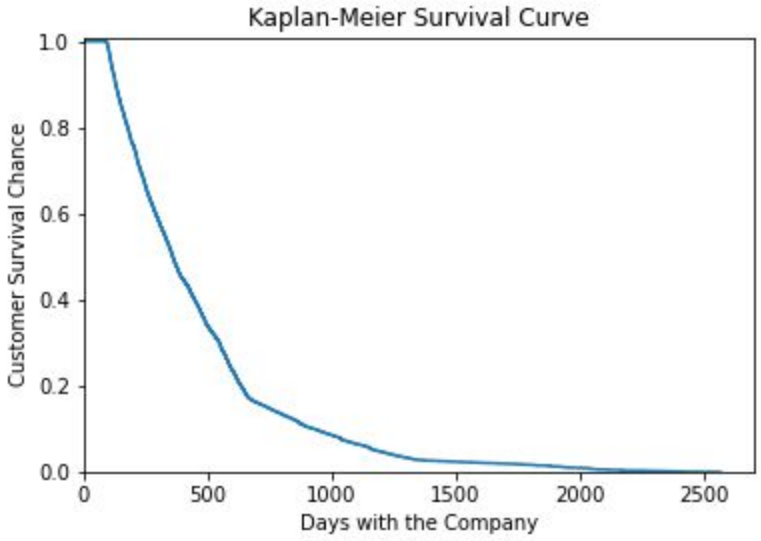


The two groups have similar density distributions overall with “Current” customer distribution slightly to the left of “Churned”, but as mentioned previously, there are many more observations that fall into the “Churned” group instead of “Current”, as demonstrated in the histogram above. The two cumulative density functions plotted below also show “Current” to the left of “Churned”.



We can also see that observations for “Current” begin to become a bit more sparse around 1250 days or so, while the “Churned” observations remain tightly clustered. On the high end the customer with the longest tenure was with the company for more than 2,500 days, and of the customers that churned, there is a small cluster slightly greater than zero that does not connect to the rest of the “Churned” curve. For the most part, there are two distinct curves--one for each group.

When looking at the Kaplan-Meier Survival Curve below, we see the opposite of the CDF. By Day 2500, there is a very low chance of survival--a very low chance that the customer is still with the company. We also notice that there is a plateau and then a sharp drop a few days into tenure. This is probably due to customers subscribing for the minimum number of days necessary to sign up and then deciding to cancel before the next billing period.



The next section will go into more depth on the Kaplan-Meier Survival Curve and the Cox Proportional Hazards Model.

Machine Learning:

Churn analysis often utilizes statistical methods such as using a Kaplan-Meier Estimator or a Cox Regression Model (aka Cox Proportional Hazards Model). The Kaplan-Meier is the most popular non-parametric approach, but its biggest limitation is that it does not take into account covariates when estimating survival. It only factors in the event (churned or not) and the time to the event.

The Cox Proportional Hazards Model, on the other hand, is a semi-parametric approach that does take into account covariates. It is composed of two parts. The first is the baseline hazard function. This is the non-parametric portion that shows the probability of an event occurring over a period of time in the future. It estimates the risk

at “baseline” feature levels. The second part of the Cox PH Model is the covariates’ relationship with said baseline hazard function. This parametric part is the time-invariant factor that directly affects the baseline hazard. In addition, this model contains the proportional hazards assumption, stating that while a covariate’s associated hazard may change over time, the hazard ratio remains constant.

In creating this model, I began by trying to change the data type of all categorical variables to “category” and dropping the specific customer-identifying information (“CNI_CUSTOMER” and “'CETEL_NUMBER”). I then attempted to use a pipeline and imputed the value 9999 for the missing values in the “STATE_DATA”, “STATE_VOICE”, “CITY_DATA”, AND “CITY_VOICE” columns that were missing roughly 2,000 entries.The maximum value for city data was less than 400 and for state was 100, so I decided to use this number 9999 for simplicity. I could have also converted all four columns to strings and then imputed a word like “missing”, but since I was simply imputing before one-hot-encoding, I chose a large, repeating number like 9999 that I knew would stand out when looking at the columns. The value -1 should also stand out, as this was a number outside the lower bounds outlined in the Data Dictionary, but nonetheless it appeared over 1,000 for each of the four previously mentioned columns. After imputing 9999 for the missing values, the next step in the pipeline was to one-hot-encode the categorical data. I attempted to use column transformers for the imputation and encoding then put those as sequential steps in a pipeline. When I used them together in the same column transformer, I ran into issues because the transformations were completed in parallel, and I needed missing values (in category-type columns) imputed before they were one-hot-encoded.

Because I could not get this to work resulting in a format that I wanted, I instead opted to use the “fillna” function and the pandas’ function “get_dummies” function for imputing and one-hot-encoding. In the process, I also dropped the “*_VOICE” columns, which had a high correlation to the “*_DATA” columns. Aside from the two customer identifier columns (“CETEL_NUMBER” and “CNI_CUSTOMER”), I ended up dropping “TEC_ANT_VOICE”, “STATE_VOICE”, and “CITY_VOICE”. I kept “TEC_ANT_DATA”, “STATE_DATA”, and “CITY_DATA”. In order to set a baseline for the baseline hazard function, I dropped the most popular one-hot-encoded column for each feature. I ultimately ended up dropping “DEVICE_TECHNOLOGY_2”, “'AVG_MIN_CALL_OUT_3”, “DEVICE_13”, “TEC_ANT_DATA_4.0”, “STATE_DATA_100.0”, and “CITY_DATA_345.0”.

Once I had dropped the necessary columns and had the dataset in a form that I could use, I created X by copying the dataframe and dropping the “CHURN” and “DAYS_LIFE” columns. These columns I used to create y, using “Surv.from_dataframe()”. Afterwards, I attempted to use the “cross_validate” function to split the datasets into testing and training portions and test the score of each. Unfortunately, this failed, with errors of “Matrix is singular” and “"Estimator fit failed”.

Using the “train_test_split” function, I was able to successfully separate the data into a testing group and a training group. However, fitting CoxPHSurvivalAnalysis() to the training data also resulted in an error of “Matrix is singular”. When using a pipeline (with “StandardScaler()” and “CoxPHSurvivalAnalysis()”) or “CoxPHSurvivalAnalysis()” by itself, I received errors that the matrix was singular and that the estimator fit had failed. When using the test-train split for the pipeline or the model by itself, I only received the error message that the matrix was singular. I am not sure why I cannot use “cross_validate” or “train_test_split” but using the whole of X and y is successful. However, when I attempted to fit it on the full X and y sets instead of splitting them into separate test/train groups, it was successful--after running for approximately 8 minutes and producing multiple errors warning about an ill-conditioned matrix, which could lead to inaccurate results.

After further experimentation, I discovered that “CoxPHSurvivalAnalysis()” was much more flexible when the majority of the dataset was not one-hot-encoded data. In the second section of the notebook, I created a smaller dataset by dropping all one-hot-encoded column features. I was then able to successfully use the “cross_val_score” function on this data, in addition to creating a successful pipeline (using “StandardScaler” and the CPH model).

The score of the Cox Proportional Hazard model based on the entire dataset (including one-hot-encoded features) received a score of 0.63. When using cross-validation on the smaller [numerical-only] dataset, the model scored a low of 0.5852 and a high of only 0.6105. This demonstrates that the model loses some of its predictive power if it does not take into account the categorical variables. However, with a score of only 0.63, it appears that we should look more closely at our “ill-conditioned matrix” or that we might be missing some important predictors in our dataset.

For future work, I could spend more time trying to figure out if it is possible to use “CoxPHSurvivalAnalysis()” on a dataset that is mostly composed of one-hot-encoded variables. I could also test out a dataset such as this on the “lifelines” package for Python that several other data practitioners have posted articles about.

Presentation:
https://docs.google.com/presentation/d/1-1G7j45ZnSmG4zCaXNownRhFrwLStjCZmFzwfU9QfB0/edit?usp=sharing

Code:
https://github.com/ashtonreed/Springboard/tree/master/Capstone2