Predicting Churn: Milestone Report II

Ashton Reed

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.