Data Preparation Final Report

Parker Lutz | STA 6714 Fall 2017

Data Background

In this project we were tasked with preparing a large raw dataset of 710,201 observations and 64 feature variables for use in a classification task.

This data could be representative of both categorical and numeric descriptors of the demographic and economic status of a customer purchasing a vehicle. This type of dataset is incredibly useful for creating models that allow us to classify a new customer walking through the door by their likelihood of good or bad payment activity based on the performance of customers from the training data. If we know a customer's classification, we can make an informed decision about what down payment they should be assessed and what kind of payment terms they should be given. In this dataset, we can use the binary variable firstterm_survival to measure performance. A customer with a 1 does survive the first payment term and a customer with a 0 does not survive the first term. This can be owed to a variety of factors, but we are most concerned about their risk for not making payments on the asset, whether they walked in the door with fraudulent intentions or were simply unable to afford their payments.

Our goal in this project is to understand and prepare the feature variables that will be used to create the model which can classify our new customers as accurately and completely as possible.

Data Exploration

We are going to complete this project inside Jupyter Notebooks using a Python kernel. Jupyter Notebooks are a great tool for combining live code with explanatory text and inline graphics. Python provides us with a couple of convenient libraries which we begin with loading, including numpy for mathematical computing, seaborn for data visualization, and pandas for working with data in data frame format. Many Python programmers either love or hate pandas, but we see great value in the simple indexing, string operations, and aggregation and shaping operations available with the data frame structure for this data preparation task.

We want to take time to explore the data and pay careful attention to possible data anomalies like miscoding errors, outliers, skewed distributions, and missingness. Most datasets are observational, and in large data we are very likely to have these anomalies present. We need to recognize them and handle them, so they don't negatively impact model performance. Categorical variables are most susceptible to the data anomalies of having low frequency categories, too many categories, missing data, or coding errors. Numerical variables are most susceptible to the anomalies of missing data, extreme outliers, skewness, and coding errors. We will address each of these in this dataset, and step 1 is exploring the data to identify them. While it is impossible to detect every possible data anomaly, we can review a basic checklist. For categorical variables, we want to verify cardinality by checking that there are as many categories in the data as we expect there to be. We also want to review the frequency counts for each level. For numerical variables, we want to check for skewness and outliers by visually examining the distribution and reviewing the summary statistics, extreme values, and data range.

One of the most important tasks of data exploration is visualization, or presenting data graphically in order to give the analyst a qualitative understanding of the information and

relationships in the data. Understanding the data allows us to detect, measure, and compare data points. We can use bar charts, histograms, and boxplots to visually review most of the anomalies on our checklist.

Our data exploration consists of reading in the csv file and viewing the first few lines, after which we check out the frequency of values in the categories of affin_grp. We create a new variable called grp_affin_grp for handling the low frequency values, and we visually inspect it by looking at a bar chart of each group on the x-axis and the percentage of values falling into that group on the y-axis. We can see that more than 70% of the data falls into the AGNCY category, with AFFRV, BDMKT, and Other falling around 8-10% each. We also view a histogram of total_fee together with a kernel density estimation of the probability density function of the variable. We can see that this variable is skewed very far to the right. The histogram peaks between fees of 0-100, but the graph has a tail out to 600.

Categorical Variables

After taking time to better understand our data, we will tackle the various data anomalies in our categorical variables.

Missing Values

Missing values are very problematic because many data mining algorithms require complete cases for analysis. One way of handling missing data is by creating missing value indicators. A missing value indicator is a variable which has a 1 indicating that a value is missing from the original categorical variable and a 0 indicating that the original variable is not missing.

Missing value indicators are valuable because the missingness itself can be informative. Missing data has a missing value pattern that is useful for predictive modeling. In our example, someone who is missing a credit score may not have data available because they have no credit history, or because they have provided a misspelled name for their credit check, or because their credit file is not available with the agency used by the business. The missingness carries information that would be lost if we chose to remove observations with missing values or otherwise impute the values without creating a missing indicator first.

In this project, we create missing value indicators called MI_PIP_Limit and MI_CP_Limit for both PIP_limit and CP_limit, and then we create a single missing indicator called M_Others for all other categorical variables with any missing values, including BI_Limit, CreditActionCode, DistributionChannelName, GoverningStateCode, HomeownerInd, MarketingPlanCode, PaymentMethodName, PaymentPlanDesc, PreferredMailDocumentsCode, PriorCarrierTypeCode, ProductVersionName, StateGroup, StateRegion, affin_grp, assoc_grp, paymentmethodlong, and premcat.

We can also handle missing values through complete case analysis. With complete case analysis, we would remove any observation with missing values. This is a feasible option when few cases are missing but that is not the case in our data. Removing incomplete cases results in ignoring a large portion of the population—we would only have $(1-\alpha)^k$ complete cases for k variables missing with probability α. This means that if our 64 variables have a 1% chance of missing, we only have 53% complete cases and our dataset is reduce to 373K observations. If we remove all observations with any missing values, we also make it difficult to evaluate new cases presented to our model with missing values because there is no precedent in the training data.

If we want to heed the drawbacks of complete case analysis, there are some data mining algorithms we can employ that will handle missingness inherently. Decision trees, for example, classify data by assigning association rules to an outcome that can be pictured as tree with various nodes and splits. They can inherently handle missing data by creating surrogate splits in the tree.

In our project, we need to be able to use algorithms like regression that cannot inherently handle missing values, so we want to find a way to impute the values. When we replace missing values, it is important to consider if the values are missing completely at random (MCAR), so that their missingness is completely independent of the variables, missing at random (MAR), so that their missingness is likely dependent on observed data but not on the explanatory variables, or informative, so that their missingness is driven by an explainable cause. Most of the data we use is observational, so we assume we are working with data that is MAR.

When we replace missing values, we want to consider four requirements: we need to produce an unbiased estimation, retain the relationship between variables, retain information about missing values, and choose a method with lower computational cost. There are many choices for numerical imputation which we will consider later in the project. For categorical variables, we can impute using count, a default constant, or the modal category. We use mode imputation on BIGroupNum, HomeownerInd, PreferredMailDocumentsCode, and PriorCarrierTypeCode.

Clustering

Categorical variables with too many levels can be a problem because there are likely to be categories with low frequency which are unreliable for two reasons. Most obviously, fewer

observations in a single level of a category provide less data to draw conclusions about and outliers or edge cases may have large influence on the sample data. This is also a challenge when we build models in case extremely low frequency levels are not represented in both training and test data. For example, if we look at zip code as a feature variable, we may have some zip codes with very few to zero current customers in them. In that case, how do we handle it if zip code is a required feature for the model and a new customer lives in a rural zip code that is not represented in the training data? It is better to either look at a less granular feature, like county or state, or in this example we could combine all very low frequency zip codes into a single Rural_Zip level.

We encounter this situation in our data with affin_grp. We see that the levels AFFRV, AGNCY, and BDMKT are significantly more frequent than AFFAUxRLC, ALLY, FF, GMREL, INTER, and RLC, so we combine the low frequency groups into one level called Other. We also cluster all levels of StateGroup and ProductVersionName with less than 25 observations into another level called Other.

Coding Errors

Coding errors or cross-functional differences in data representation are important to pay attention to. In our dataset, each customer is assigned a PaymentPlanDesc which describes their payment plan. In most cases they are assigned something like “20% down, 4 payments,” meaning that they would pay a down payment of 20% of the price of the vehicle and they pay the other 80% over 4 payments. We want to split the plan description into two new variables, DownPayPct and N_Payment to describe these two pieces of information separately. We also must consider the special cases of the Pay in Full plan, the Better Budget plan, and customers with missing plan information. We start by redefining the special cases so that Pay in Full is

"100% down, 0 payments", Better Budget is "0% down, 1 payments", and missing plans are replaced with "100% down, 0 payments." Then we can use pandas built-in string operations to split the data into new variables.

We also find that the CreditActionCode contains both a letter and a number, like T1, when we only need to know the first letter of the code. Again, we use string operations to create a new variable FL_CreditActionCode which has just the first letter of the CreditActionCode. It is always important to look out for these types of recoding challenges when faced with a new dataset and it is best practice to consult a subject matter expert because a new data analyst may not understand the nuances required to clean the data appropriately.

Smoothing

A categorical variable with n levels needs n-1 dummy variables to represent it, where a 1 indicates the presence of the category and a 0 indicates its absence. As a result, a categorical variable with significantly many levels should not be recoded with dummy variables because of high dimension inflation. If we consider a level with only two observations in our dataset of 700,000, we can also see that we exacerbate the inaccuracy problem inherent in low frequency levels by dummy coding a new variable that would be less than .001% non-zero values. The question becomes, how do we handle situations with these types of categorical variables?

In cases where the variable has a clear relationship to the target variable, we can do enumeration using smoothing. That is, to represent each level of a categorical variable with a smoothed number calculated using the proportion of the target variable present in that category. We perform logit smoothing on ProductVersionName, FL_CreditActionCode, and StateGroup. The first step, which we have already completed, is to cluster low frequency categories of all

three variables together. The next step is to obtain the proportion of the response variable at each level of the variable, which we can do by using a combination of aggregation by count and grouping by the two firsterm_survival responses, 0 and 1. This gives us the frequency of 0s and 1s in each level of the category, from which we can calculate overall frequency (number of observations in each level), churn frequency (number of "yes" or 1 observations in each level), and churn rate (churn frequency divided by overall frequency). We can use all of this to calculate the smoothed logit:

$$logit_i = \log(\frac{n_{1i} + \rho_1 * smoothing\ factor}{n_{0i} + \rho_0 * smoothing\ factor})$$

where n_{1i} is the number of 1s in the ith category, ρ_1 is the proportion of 1s in the total population, n_{0i} is the number of 0s in the ith category, ρ_0 is the proportion of 0s in the total population, and the smoothing factor is chosen. We can then join the smoothed logit values back to the original data set as Logit_ProductVersionName, Logit_FL_CreditActionCode, and Logit_StateGroup.

Numerical Variables

Finally, we will address the data anomalies in our numerical variables.

Missing Values

Like with our categorical variables, we want to handle missing numeric values in a way that produces an unbiased estimation and retains the relationship between variables. An imputation method is unbiased if the distribution is maintained after imputation, so the mean and

standard deviation of the data are not altered. One way to do this with numerical data that is MAR is to impute using the unconditional mean. The relationship between variables should also be maintained, though, and the unconditional mean could destroy it if variables are dependent on each other. There are conditional imputation methods, however, that use the statistics of several variables to impute values. Some conditional methods include regression, tree, and clustering imputation. Regression imputation is used when a variable has a linear changing relationship to another variable. Therefore, we can fit a regression line to the available known data and use the line to find the missing values. It is very accurate and considers the relationship between variables, but it has high computational cost, so it is best to use when there is a very low portion of missing values on only a few variables. Tree imputation is likewise favorable because it considers the relationship of variables, but it is lower cost and easier to implement than regression, so it should be used when there is no linear relationship between variables to be imputed. With clustering imputation, we cluster numerical data and assign missing values to their clusters.

The variables in our dataset are mostly independent and missing at random, so we can replace the missing values with simple unconditional methods without being concerned about preserving the distribution. We could use mean, maximum, minimum, median, or many other statistical values that represent the dataset. For this project, we choose to use the median to impute CL_Limit, PD_Limit, MP_Limit, CreditScoreNum, InsuranceExperienceDaysNum, PreCreditTierNum, PriorSwitchesCount, RateManualNum, and uwtiergroup.

Extreme Values

We are careful to handle extreme values that skew our data. We know that our results can be significantly influenced by the presence of unusually high or low values as well as anomalies that violate our expectations of possible values in a field. If we don't handle these outliers, we could have regression coefficients shifted in our model build, variance inflation, or a shift in Pearson correlation. All of these influence how we build our model and its accuracy.

In the project, we limit the field PNIAge to the range [16, 100]. We want to eliminate records we can't make sense of like purchases made by babies or people with nonsensical ages like -12 or 999.

Coding Errors

Like our categorical variables, we are careful to handle coding errors in our numerical data. We split vehxdrv into two variables, N_Vehicles and N_Drivers.

Transformation

At this stage in our data preparation, our data is mostly cleaned. We now want to consider transforming some of our numeric values to make them better features for a model.
We use transformation to reduce the impact of extreme values in data without removing observations. There are several reasons we want to handle extreme values. We know that the relationship between a target value like firstterm_survival and any other input variable usually taper at an asymptote, but most regression models will fit a line that increases (or decreases)

without bound. We also know that values have more influence on model fit the farther they are away from the median, and models built on untransformed data will fit the extreme values at the cost of fitting the data clustered near the center. We don't want to address our concerns by abandoning interpretable models like regression, so it is better to transform data to accommodate extreme values. The transformation methods for reducing the impact of extreme values on a model include truncation, power transformation, and rank transformation.

We can truncate values by assigning all values of X to some value less than an upper limit. It is easy to use and doesn't impact interpretability, but finding the optimal upper limit is difficult. It may also not entirely solve the problem of high values having high leverage on models if the value of A is still far from the center of the data.

Power and log transformation are simply taking x^r or log(x). This does make data less interpretable, but it is easy to implement, easy to automatically find the optimal value of r, and it solves the challenge of extreme values and model leverage.

Rank transformation is as easy as ordering and ranking values. It is interpretable, easy to implement, solves the problems at hand, and can be better than power transformation on skewed data.

When deciding how to transform data, we want to start by examining the distribution of data. If the data has a positive skew, we want to perform a log transformation. If the data is still positive skewed, we can do a negative power transformation. If after log transformation, the data is negative skewed, we can do a power transformation using a power between 0 and 1. If the data has a negative skew, we can do a square transformation. If the data is still positive skewed, we can do a power transformation of 1 or 2. If the data is negative skewed after square transformation, we can do a power transformation of a power greater than 2. We should do rank

transformation on any variables that we can't transform suitably with a power transformation, and use bucket transformation on all the other numerical variables.

In this project, we perform rank transformation on RateManualNum, BI_Limit, FirmCode, agege75_lt30_lt21_pointed, PD_Limit, MP_Limit, maxvehvalue, PIP_Limit, nextpremch, DaysLapseNum, Logit_ProductVersionName, and lastendmt. We perform log transformation on uwtiergroup, noncancelendmts, and incurred_loss. We perform square transformation on Logit_FL_CreditActionCode and Logit_StateGroup. Finally, we perform a power transformation of powers between 0 and 1 on total_fee, total_ann_prem, CL_Limit, npchcat, InsuranceExperienceDaysNum, and CP_Limit.

Conclusion

We have thoroughly considered the data anomalies in the dataset provided and handled missing values, coding errors, categorical variables with too many levels, and numerical variables with extreme values. We have also performed logit smoothing on categorical variables and rank, log, and power transformation on numerical variables to make them better input features for a model. This data should now be prepared to input into a Decision Tree, Random Forest, Gradient Boosting, or Regression model.