Semester Project

William Huibregtse[1], Joshua Baker[1], Chris East[1]

Abstract
Include abstract here – A summary of your work

Keywords
Keyword1 — Synergy — Keyword3

[1] *Computer Science, School of Informatics , Computing and Engineering, Indiana University, Bloomington, IN, USA*

Contents

1. Problem and Data Description

First we want to get a general idea of our data set and get a deeper understanding of the underlying structure.
There are 59 named features or variables for our data set.
With 892816 observations for training and 595212 for test
There are no duplicate observations.
Features that belong to similar groupings are given certain feature names.
- Ind: related to individual or driver
- Reg: related to geographical region
- Car: related to car being insured
- Calc: are calculated features done by Proto themselves
Postfix descriptors describes the features data type.
- Bin: Binary (1 or 0)
- Cat: Categorical *Note: the dataset has the categorical data already convert into factors and then integers
- All other variables are either integer or numeric
As stated the Data Types are numeric and integer, with integer being the predominant type 49 to 10.
Missing values are represented by -1.
In total, there are 13 variables with missing values.
There is Target feature which denotes the binary classification for that observation. This feature is the feature we are trying to learn/predict for the test data.
There is an ID feature which is an anonymized identities of insured drivers.

Porto Seguro's Safe Driver Prediction has 59 variables and 1.3 million observations, which qualifies as a good candidate for reducing overall dimensions of the data to significantly increase the speed of analysis techniques at the cost of more upfront data processing.

There are only 21694 cases of classification 1, which is 3.64 percent of the observations in the training data set, showing significant skew in the expected class towards a "0" prediction.

2. Data Preprocessing & Exploratory Data Analysis

2.1 Feature Engineering

As even missing data can be significant, a new feature was added to the data set. This feature was the count of missing values for each entry before these missing values were processed. This technique allows the retention of the potentially useful information provided by the missing values.

2.2 Handling Missing Values

After observing the summaries of each variable in our data sets, it was clear that variables ps-car-03-cat and ps-car-05-cat contained mostly missing values for each data set. Because we later used a missing value replacement techique to process the data, applying this techique to variables with mostly missing values may have overfitted and affected the outcome of future analysis, thus both because of this possible overfitting and that useful information may be captured in the engineered feature of missing value counts, these columns were removed before proceeding with our NA replacement technique.

After the columns with mostly missing were removed, we replaced missing values with the mean of it's variable using the R package "mice".

2.3 Exploratory Data Analysis

Originally, after looking through a few useful posts about PCA on the Kaggle discussion boards about the number of principal components to include in dimensionality reduction, one popular post used PCA without scaling to form new components. This is likely a poor decision as the variables in the data have vastly different variances and ranges, even though all are numerical. To perform PCA on this data, the built in scaling parameter was used to force all variables into unit variances so that each variable was treated with equal weight in PCA. The discussion post in question found that 95 percent of variance was captured within the first 15 principle components when using unscaled data, however our findings show that 95 percent of variance is not captured within the new data until the first 45 are considered, thus PCA would only be reducing the variable size from 57 to 45, which is only a 20 percent reduction in overall dimensions. When moving on to later steps in analysis, the change in variables to unit variances was kept, however no components were dropped as the majority of components needed to be considered before an appropriately large proportion of overall variance was achieved.

To reduce the complexity of future analysis techniques, we used the prcomp function from base R to perform PCA dimensionality reduction on our data. After the transformation, the first 16 principle components represented over 99 percent of the the data's variance. Using this as a reasonable cutoff, we proceeded forward using only these first 16 principle components.

Note: To perform this transformation correctly, the test and train data must be transformed together and split after the dimensions have been reduced.

3. Algorithm and Methodology

3.1 Naive Bayes

Once the data was processed, we used the Naive Bayes algorithm to form our model. We tested it with both the implementation from the e1071 package and the caret package. The caret package implementation ended up being more powerful since it allowed for easy cross validation.

Our analysis with the Naive Bayes algorithm uses the m-estimate smoothing technique to avoid conditional probabilities of 0, however none of the conditional probabilities found were 0, so this added layer of protection was not necessary in this case, likely due to the very large data size. The formula for Naive Bayes is as follows:

$$P(\theta|\mathbf{D}) = P(\theta)\frac{P(\mathbf{D}|\theta)}{P(\mathbf{D})}\,||I, \tag{1}$$

The algorithm gives the probability of an outcome based on a set of conditions from each condition's probability of individually implying the outcome.

3.2 Gradient Boosting

We wanted to test a second classifier, so we chose gradient boosting, since it had been used successfully by a number of people previously. The algorithm tries to minimize a loss function with gradient descent, using progressively smaller learning rates. In order to do this it uses multiple decision trees trained on the data set, each with a different weight that can be updated in the gradient descent step. By using multiple decision trees, the chance of over- fitting to the training data is reduced significantly. By sequentially updating them with gradient descent, the trees further learn from the mistakes of previous trees, so this helps further increase accuracy.

We used the xgboost R package, which is a popular and fast R package for gradient boosting. The binary logistic regression objective automatically outputs the probability, which worked well for our purposes. It ran significantly faster than Naive Bayes, meaning we were able to test a number of different learning rates, tree depths, and max iterations. The function has built- in error evaluation, which was helpful for deciding what parameters to use.

4. Experiments and Results

Since we had access to multiple submissions on Kaggle, we were able to use their fitness measurements to test our predictions. When we did PCA, we originally kept 95% of the variance, and used that to run Naive Bayes. When we submitted this, we only got a GINI score of .125, but when we increased the total variance to 99%, the score jumped up to .20566.

5. Summary and Conclusions

Acknowledgments