Binary Classification using Naïve Bayes & XGBoost

Patrick O'Brien, Matthew Ondash, Dante Razo[1]

Abstract
Porto Seguro, a Brazilian auto insurance company, is looking for the probability of a customer filing an insurance claim in the next year. They provided anonymized training and testing datasets for Kaggle users to analyze. After preprocessing the data, and selecting the most important features using an extra trees classifier, Naïve Bayes and XGBoost models were created for binary classification. The accuracy of both classifiers were compared, and XGBoost outperformed Naïve Bayes significantly for this specific use case.

Keywords
naïve bayes — xgboost — binary classification — k-fold — porto seguro — kaggle

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

Contents

1 Problem and Data Description 1
2 Data Preprocessing & Exploratory Data Analysis 1
2.1 Handling Missing Values 1
2.2 Exploratory Data Analysis 2
2.3 Dimension Reduction 2
3 Algorithms and Methodology 2
3.1 Extra Trees Classifier 2
3.2 Naïve Bayes 2
3.3 K-Fold Cross-validation 3
3.4 XGBoost 3
4 Experiments and Results 3
4.1 Experiment 1: Naïve Bayes 3
4.2 Experiment 2: XGBoost 3
4.3 Results 3
5 Summary and Conclusions 3
6 Libraries 3
6.1 Python 3
6.2 R 3
References 3

1. Problem and Data Description

The raw training data has 892,816 points with 58 features. The data is separated into four different groups of related features: ind, reg, car, calc. The datasets includes binary, categorical, ordinal, and continuous data types. Missing values in the dataset were represented by -1. There are 767,885 points in the training dataset with at least one missing attribute.

This is a binary classification problem, which can be solved using a myriad of classification models. Before using any model, the data should be preprocessed to increase model accuracy. The first step in preprocessing is deciding how missing values will be represented in the models. Two popular methods are ignoring missing values and imputing with the mean, median, or mode.

The second step of preprocessing is reducing the dataset to its most meaningful members. This could entail removing columns with low variance, high correlation with other features, and/or with too many missing values. The normalization of features is also advisable to prevent the skewing of feature importance.

A final thought to consider before preprocessing is overfitting of the model. This is important because even if a model performs well on the training data, it may not generalize well to new data.

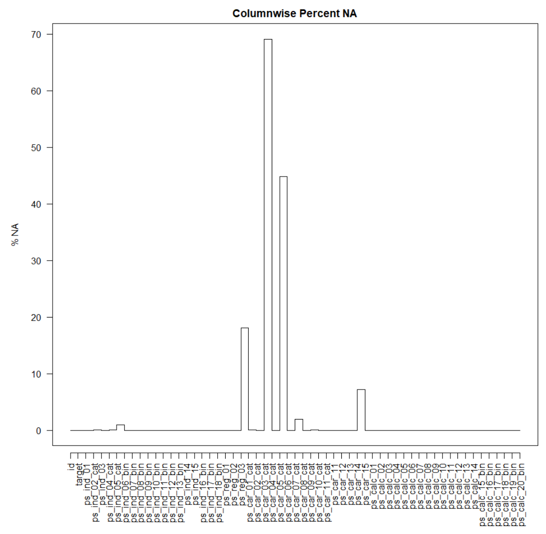
2. Data Preprocessing & Exploratory Data Analysis

2.1 Handling Missing Values

All of the missing values in the provided training dataset were converted from -1 to NA for readability and debugging.

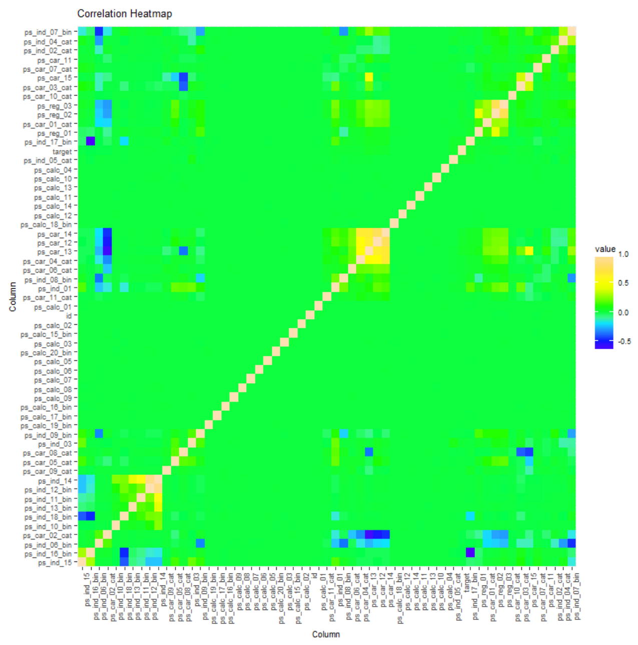
The first step in working with missing values is recording the percentage of missing values for each feature and then removing features with a significant missing percentage. Over 40% of the values in ps_car_03_cat and ps_car_05_cat were missing, as shown by the graph below titled "Columnwise Percent NA". The columns were subsequently removed.

It was decided that for categorical features, missing values would be encoded as a separate category instead of imputing with the mode. The remaining missing values were imputed based off their feature type. Continuous and ordinal data were imputed with the column mean. Binary data was imputed with the column mode.



2.2 Exploratory Data Analysis

A heatmap of the correlation matrix of the training data was created for easy visualization.



In this heatmap, highly-correlated data is clearly visible in groups. A large portion of the high-correlated data is binary, whose correlation does not carry as much weight as continuous or ordinal correlation.

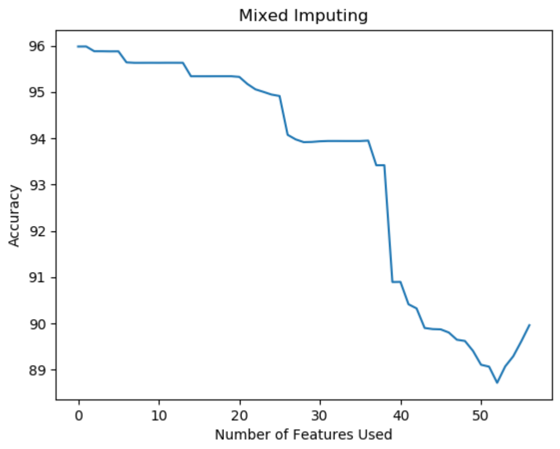
The two features with the highest correlation were ps_ind_12_bin and ps_ind_14, which were positively correlated.

One-hot encoding was briefly considered, but was not used to go through with the technique due to the extra dimensions it added.

2.3 Dimension Reduction

Dimension reduction was achieved by implementing feature selection. The Extra Trees classifier algorithm was used for the latter task.

The Extra Trees classifier returns the datasets' feature labels in order of importance. Models were created using the first n important features. Their accuracy was plotted versus the value of n, illustrated in the graph below titled "Mixed Imputing".



Rank	Feature
1	ps_car_13
2	ps_calc_14
3	ps_reg_03
4	ps_calc_10

The graph shows that as n increases, the overall accuracy of the model decreases. Therefore, it was decided that 4 is the optimal number of features to maximize accuracy. The top 4 most important features are shown in the table above.

3. Algorithms and Methodology

3.1 Extra Trees Classifier

"Extra Trees" stands for **Ext**remely **Ra**ndomized Trees. The algorithm works in a similar fashion to decision trees, but picks a random cut point instead of picking based on information gain. This allows the algorithm to rank features based on their importance to the model.

3.2 Naïve Bayes

The Naïve Bayes classifier is a simple probabilistic classifier which utilizes Bayes' theorem and assumes feature independence. Naïve Bayes was chosen for its versatility and simplicity.

3.3 K-Fold Cross-validation

The best Naïve Bayes classifier was selected and run through 5-Fold cross-validation. This is important in determining whether or not the model will generalize to new data or is overfitted to the training data. The results are shown in the following table:

Fold	% Accuracy
1	18.030
2	17.997
3	17.983
4	17.886
5	18.031

The extremely low accuracy of the Naïve Bayes models indicated that they were indeed overfitted and unsuitable for this dataset.

3.4 XGBoost

XGBoost stands for **Ex**treme **G**radient **Boost**ing. Gradient boosting is a machine-learning algorithm where many weak learners — typically decision trees — are combined to build a stronger model. This library was chosen as it is considered one of the most powerful classifiers.

4. Experiments and Results

4.1 Experiment 1: Naïve Bayes

With $n = 4$, the predetermined optimal number of features, the accuracy of the best Naïve Bayes classifier on the training set was 96% and found to be overfitted. When applied to new data, the accuracy dipped to 18%.

4.2 Experiment 2: XGBoost

Using the *XGBoost* library, the models were 5.4 times more accurate at predicting new data than the Naïve Bayes models. The library uses gradient boosting to predict a target variable using multi-dimensional training data. The *XGBoost* classifier used a max depth of 2 and ran for 10 rounds.

4.3 Results

Submitting the labels generated by Naïve Bayes classifier to Kaggle returned a normalized Gini coefficient of 0.18605.

Using *XGBoost* for the submission, Kaggle returned a normalized Gini coefficient of 0.24039.

5. Summary and Conclusions

After preparing the data and removing unimportant features using an Extra Trees classifier, two models were created for the purpose of binary classification. Both classifiers were compared based on accuracy, and *XGBoost* outperformed Naïve Bayes significantly for Porto Seguro's specific use case.

In conclusion, XGBoost is much more effective in this use case at classifying data than Naïve Bayes. The accuracy of 96% for the former was 5 times greater than the 18% accuracy of the latter.

6. Libraries

6.1 Python

- NumPy
- pandas
- scikit-learn
- matplotlib

6.2 R

- xgboost
- ramify
- FSelector
- zoo
- caret
- DiagrammeR
- reshape2
- mlbench
- gower

References

1. *Dimension Reduction Methods* by Analytics Vidhya
2. *How the Naïve Bayes Classifier Works in Machine Learning* by Rahul Saxena
3. *Extremely Randomized Trees* by Pierre Geurts, Damien Ernst & Louis Wehenkel
4. *Introduction to Boosted Trees* by DMLC