Kaggles Competition "Dont Get kicked!" by Random Forest

Shi Cao

October 2017

Abstract

In this report, we build a Random Forest model to predict if a car purchased at auctions is a "Kick" (Bad buy). The dataset is imbalanced with about 87.7% good cars and the rest are bad cars. We perform a series of data prepossessing techniques such as data cleaning and data engineering. After the data prepossessing steps, we test the Random Forest model on the Kaggle website and the test result is ranked top 100 among all 570 teams. The performance of our Random Forest model is further improved by manually balancing the dataset to build a balanced Random Forest model. With the ensemble of a Random Forest model obtained with imbalanced dataset and with manually balanced dataset, our test result is ranked within top 50.

1 Introduction

Purchasing a car at auctions takes more risks comparing with purchases made from dealerships or private owners. At a dealership, the buyers always have opportunities to drive the car to test. In addition, there may be warranties for buyers to choose. However, at the auctions, usually it is prohibited to drive the car to test. The goal of this project it to predict that "is the car a bad buy or not?"

The available information for a car includes the following main categories:

- **The car brand-related and outlook information:** including make, year, model, submodel, trim, transmission type, wheel, color, size, nationality information, and so on.
- **The car usage information:** including purchase data, odometer reading, and so on.
- **Buyers information:** including the state where the buyers purchased cars, buyer purchase ID, online order or not, and so on.
- **Other financial information:** including the auction price and retial price, the warranty cost, original vehicle cost, and so on.

We choose to build a Random Forest model to make use of all these available information to make predictions. The challenge is that the dataset is seriously skewed, which means a naive guess may result in a 87.7% prediction accuracy. We would like to build a classifier with higher accuracy than this naive guess. However, the accuracy is not the metric we rely on since we may try to predict the bad buys more aggressively to prevent potential loss. The objective function we want to optimize can be described as follow:

$$Total\_Gain = \Sigma PF_{GC} - \Sigma LS_{BC} \tag{1}$$

where PF_{GC} means the profit from a good car and LS_{GC} is the loss for a bad car. Clearly, predictions have to make a trade-off between the terms PF_{GC} and LS_{BC}. A miss classification of good cars will decrease the gain from the first term in eq. 1 and a miss classification of bad cars will increase the loss. If the term LS_{BC} is very large, which means a false negative (where we identify a bad buy as a good one) will lead to a large loss in the total gain. We may trade off prediction accuracy with lower false negative rates. We evaluate our model by a combination of accuracy and a F1 score. The metric can be describe as follow:

$$Metric = argmax\ F1\_Score = \{F1|if\ accuracy > CF\} \tag{2}$$

where CF is the cutoff for accuracy. In this project, since we have 87.7% labels of 0s so CF is a number higher than that.

2 Experimental details and data preprocessing

We use software WEKA to do data exploring and visualization. We can easily see the attribute distributions in figure 1. In WEKA, we can also identify the missing values. It turns out the correct filling of missing data will help to improve test results. The data preprocessing and model building are done by Python programming, where the sklearn, pandas, numpy, scipy packages are used. More details can be found in the code and we brifly introduce some key steps below.

2.1 Filling missing data and data cleaning

The color attribute has two types of missing values. One is "not avail" and the other one is blank. We filled only the blank one with "UnknowColor". We fill the missing wheel type as "Unkown". We filled other numerical missing data with means.

There are some redundant features such as the state and zip code, wheel type and wheel type ID, and so on. We delete the redundant features for our Random Forest model.

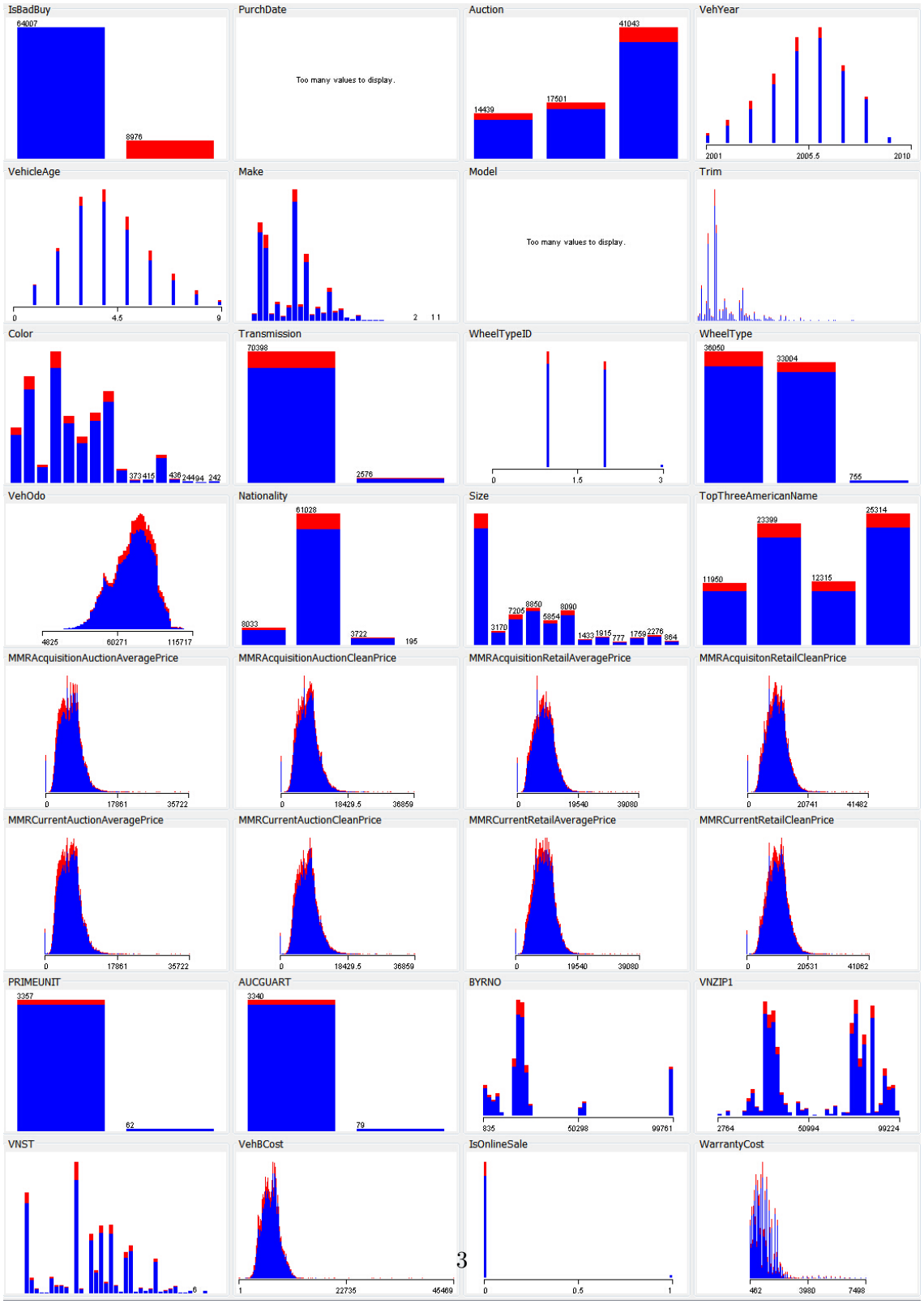

Figure 1: Data visualization

Training and test on raw training dataset			
Model Name	Training Accuracy	Dev. Accuracy	Dev. F1 score
Imbalanced Model	98.913 %	89.231 %	0.425
Balanced Model	93.644 %	73.453 %	0.529

Table 1: Selected results from training and development dataset.

2.2 Feature Engineering

After we explore the data, it turns out there are thousand models of a car and hundreds of sub-models, and so on. It is very unfeasible to do one-hot-encoding (OHE) to all these features. We combine some of the make inforamtion of the car to reduce the OHE dimension space.

We also notice that the price information of a car plays a key role. We create more features from the original price attributes. Basically, we want to find out the difference betweem some price features.

3 Results and discussions

With the data preprocessing and data engineering, we first build a Random Forest model with imbalanced raw dataset. We randomly divide the raw training dataset (72983 records with 87.7 % labelled as "1"s) into two parts: 90% as a training set and 10% as a development set. The orginal test set without ground truth label as the test set.

One of the advantage of tree model is that we can know the feature importance. We use the feature importance score to do the feature engineering and feature selections. The top 10 features in the model used in this project is: Unknow wheel type, four features created from feature engineering, vehicle odometer reading and some price information of cars. It also shows the effects of feature engineering since 4 out of 10 most important features are created manually.

The training accuracy is very high showing lower bias of our Random Forest model. The development set accuracy is higher than 87.7 %, which means our classifier is better than a random guess. The results are shown in table 1 first line.

4 Conclusion and future work

we apply this Random Forest model to the test set where there is no ground truth available. However, the Kaggle website can test the prediction based on Gini coefficient. Our score ranking is within top 100, as shown in a screen shot in figure 2.2 second line.

Private Score	Public Score
0.24914	0.24964
0.24428	0.24595

Figure 2: Top line are the best score with ensembel two Random forest classifier. The second line shows the score from one Ramdom Forest classifier.

In this dataset, the data are seriously skewed. We will just simply try down-sample on the original training dataset. We randomly select 40% out of the data with labels of "0" and keep all the data with labels of "1". We consider it as a balanced dataset. Then we use 90% data from this balanced dataset as a training set and 10% as a new development dataset. The training accuracy is slightly lower comparing with banlanced Random Forest model. However, the F1 score is higher. The results are shown in table 1 second line.

The classifier built from this manually balanced dataset (Balanced Model) is expected to have better performance on data with labels of "1"s. Another classifier (Imbalanced model) built from the entire dataset may have better performance on data with labels of "0"s. We just ensemble the predictions from these two classifiers and a higher ranking is achieved, as shown in figure 2.2 first line.

The goal of this project is to build a working pipeline for a Kaggle competition within a short time period (several hours of work). We choose to implement Random Forest, which is a tree based method and it can be trained fast. The bootstrap strategy with random selection of features in Ramdom Forest make it an ideal classifier for our problem. The feature importance can also be obtained and it can be use to further do featuer selection and engineering. We can build a single Random Foreste model to obtain a top 100 ranking and with data balancing skills, a top 50 ranking is achieved.

In this project, we show how to clean the data, do the feature engineering and feature selection to improve the classification results. In the future, the feature selection and parameter tuning are expected and non-tree-based model such as support vector machines (SVMs) or other methods can be used to ensemble a more powerful classifier with Random Forest for improving performances