Don’t Get Kicked: Predict if a Car Purchased at Auction is Lemon

Reza Karimi, Zelalem Gero
Emory University

CS526: Machine Learning Project Report
Spring 2017

Abstract

One of the biggest challenges of an auto dealership purchasing a used car at an auto auction is the risk that the vehicle might have serious issues that prevent it from being sold to customers. The auto community calls these unfortunate purchases "kicks". Kick cars can be very costly to dealers after transportation cost, throw-away repair work, and market losses in reselling the vehicle. The goal of this project is to predict if the car purchased at the Auction is a Kick (bad buy). So, this is a binary classification problem. We preprocessed the data, expanded the feature space and develop models, then compared against each other. The best performing classifier was RandomForest achieving 0.916990 of accuracy and 0.882512 of AUC.

1. Problem Overview

Buying and selling used cars is a common practice all around the world. Purchasing a used vehicle has advantages like lower price than a comparable new car, lower continuing ownership expenses such as collision insurance and taxes, and mostly a used vehicle has already taken its biggest depreciation hit. In general, buying used is a way to get a nicer car than you would be able to afford buying a new one. While this market can be very profitable for buyers of used cars since they are often sold for below retail prices, there is the large risk that buyers face in purchasing a faulty vehicle. Commonly referred to as “kicks”, these purchases can severely hurt the finances of a car dealership business, as the money spent on them ultimately becomes a wasted cost. Seemingly functional used cars that end up having no utility value, “lemons”, pose a significant risk to auto dealerships because they may be very difficult to detect at an auction.

Kicked cars often result when there are tampered odometers, mechanical issues the dealer is not able to address, or some other unforeseen problem. Kick cars can be very costly to dealers after transportation cost, throw-away repair work, and market losses in reselling the vehicle. Given the high stakes involved for auto dealerships, they have to ensure every car they purchase at an auction is not a lemon and will be sold to a customer. It would be extremely useful to find a better way to predict whether a car is a lemon or not at time of the auction. Greater predictability will reduce the likelihood of bad, costly purchases.

Hence, the goal of this project is to predict if a car purchased at an Auction is a lemon (bad buy). So, this is a binary classification problem. We develop predictive models that can predict beforehand whether a given vehicle in an auction is good buy or not so that the buyers can avoid the bad ones. The target variable is “isBadBuy”, expressed by a probability of being a lemon. This problem particularly needs to be wary of the high cost of a false negative, falsely predicting that a lemon has a higher probability of being a good buy. This would result in a dealership buying the car thinking it would be sellable, incurring transportation/repair costs, and then realizing it is left with a defective car and unsellable inventory. There is also an opportunity cost associated with a false positive, in this case falsely predicting that a good car has a higher likelihood of being a lemon. In this case the dealership would refrain from purchasing a car that would have otherwise generated profit for the company after being successfully sold to a customer.

2. Data Description

The dataset we used was downloaded from kaggle.com [1] provided by Carvana [2]. The training set contains 72,983 instances, with 32 independent variables and a dependent variable. Another dataset of 48,000 instances is also set aside for test. However, the test set does not have the target class labels and cannot be used to measure the performance of our models locally. Therefore, we split the training set further into train and holdout sets to measure our performance. But, the original test set was used to submit our best performing

model to the competition.

The dataset has features that are categorical, numeric and date type. Each record has a unique RefID assigned as an identifier. A feature of date type called “purchDate” describes the actual date, month and year the vehicle is purchased. “VehYear” is a related feature that describes the year the vehicle is made. From the above two features, we get “VehicleAge” which is totally an extension of the previous two. We will talk more about this later. A categorical attribute “Auction” describes the auction names where the car was purchased.

The features “Make”, “Model”, “Trim” and “SubModel” represent the vehicle’s brand characteristics. We also have more categorical features like “Color”, “Transmission”, “Nationality”, “Size”,“WheelType”, “WheelTypeID”. The last two are dependent on each other and taken care of while preprocessing. While “VehOdo” describes the odometer reading of the car, “TopThreeAmerican” is whether the vehicle is from one of the top three American manufacturers.

Another closely related set of features describe the different types of costs associated with each vehicle.

MMRAcquisitionAuctionAveragePrice,

MMRAcquisitionAuctionCleanPrice,

MMRAcquisitionRetailAveragePrice,,

MMRAcquisitonRetailCleanPrice,

MMRCurrentAuctionAveragePrice,

MMRCurrentAuctionCleanPrice,

MMRCurrentRetailAveragePrice

MMRCurrentRetailCleanPrice.

"Acquisition" means the price of the vehicle MMR at which it was sold at auction. "Clean" refers to the price of the vehicle in good condition. These vehicles are usually more expensive than the ones with “Average”.

"Auction" refers to the expected price of the vehicle at the auction

"Retail" refers to the expected price of the vehicle to which the customer is willing to pay at the dealership.

“VNST” describes Zip code while “VNZIP1” tells the state where the vehicle is purchased. Therefore this two are highly correlated.

In the case of feature AUCGUART, PRIMEUNIT these are related because “PRIMEUNIT” describes the level of demand with respect to a standard purchase “AUCGUART” describes the risk that can be run with the vehicle, meaning how much guarantee the seller is willing to give.

Finally, we have “BYRNO”, which is a code assigned to the purchaser of the vehicle. “VehCost” is the price of the vehicle. “IsOnlineSale” is whether the sale was done online. And “WarrantyCost” is the price of the warranty.

3. Preprocessing

In this section we perform data visualization and exploration to better understand our dataset.

3.1 Univariate analysis

The first thing we did is a univariate analysis of each feature to see the distribution.

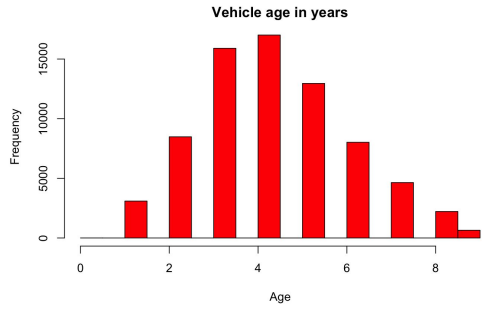

Fig 1. Distribution of vehicle age

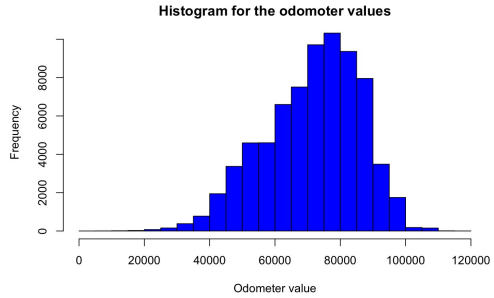

Fig 2. Distribution of odometer readings

As we can see from fig1 and fig2, some features have a Gaussian like distributions. Such features include VehAge, VehOdo, Vehyear and Nationality. Some other have a skewed distribution as shown in fig3 and fig4. All the MMR features behave similar to the

acquisition cost.

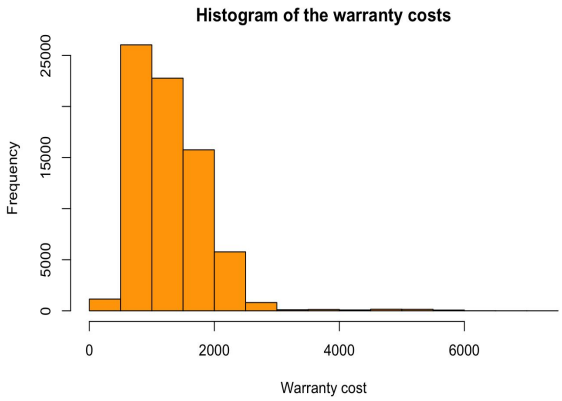

Fig3. Distribution of warranty cost

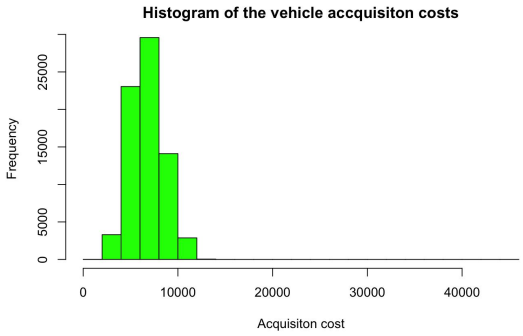

Fig4. Distribution of acquisition cost

The remaining features do not exhibit any interesting distribution.

3.2 Bivariate Analysis

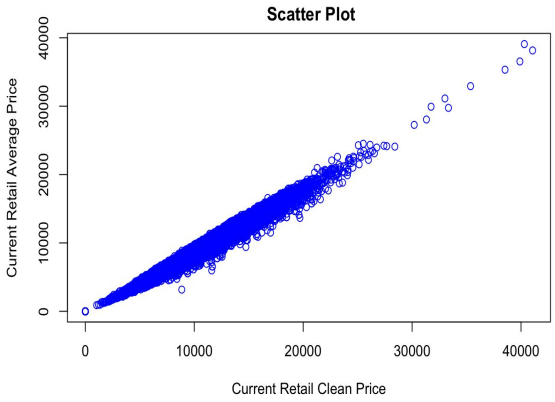

Fig5. Scatter CRCP vs CRAP

Next we try to identify features that are highly correlated using bivariate analysis of features. We used scatter plots to see any potential correlation. We can see from fig 5 and fig 6 that the MMR cost features are linearly correlated and we should be careful about this when we use linear models.

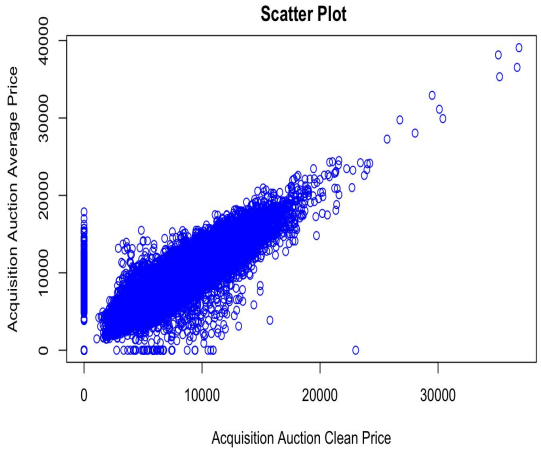

Fig6. Scatter AAAP vs AACP

3.3 Missing value imputation

Most of the features in the dataset contain missing values. Two especially problematic features (“PRIMEUNIT”, “AUCGUARD”) are observed with missing values of 96% and 98%. Except for this two features, we performed mean value imputation for all numerical features and frequent occurring value for categorical features. This worked well as there were no features with much missing data except for the

aforementioned 2 features.

3.4 Feature Engineering

Based on the above data exploration, we performed feature elimination and transformation tasks. We dropped the two features with almost all missing value ("PRIMEUNIT", "AUCGUARD"). "RefId" is also dropped as it is just a unique number with no relation to prediction. "VehYear" which tells the year a vehicle is made is dropped because we have "VehicleAge" which captures the same information. The same scenario happens with "WheelTypeId" and "WheelType", therefore we keep just "WheelTypeID". The features "VNZIP1" and "VNST" both describe the location of the auction where the car is bout, so we kept just "VNST" this tells us a state instead of a very specific ZIP.

Next, we created new features extracted from existing ones. The features "Model" and "SubModel" have a lot of excellent properties that can be used to differentiate car types even within same models. Therefore, we created new features based on the number of cylinders, types of injection the vehicle uses, the driving system of the vehicle, number of injectors, body type, and engine properties in general. This has given us 20 more features.

We have also split the purchase date ("purchDate") into three separate features based on date, month and year. We also did transformation to the MMR price features to make the market price of a car a relative descriptor not an absolute one as the price of different cars is inherently different. So we used the vehicle age and warranty cost to transform the cost features into relative terms so that comparisons can be easily made. We have done this to the odometer reading as well. These gave as another dozen features to add on our original set. Finally, we dummy coded the categorical features.

3.5 Data Transformation

We performed Standardizing the features so that they are centered around 0 with a standard deviation of 1 so that comparing measurements that have different units won't be biased.

Normalization is also used to get smaller standard deviations, which can suppress the effect of outliers.

3.6 Imbalanced Class Handling

Our dataset was 88% negative, 12% positive class. With such huge class imbalance, any naive classifier who guesses everything to be negative would be 88% accurate. Therefore, we used SMOTE oversampling technique with 40% for the positive class and increased our overall dataset to 96,013.

3.7 Feature Selection

Now that we have more features, we need to find out if there are features that are not important. We trained a decision tree based on all all features and evaluated the importance of each feature based on Gini impurity. Turns out that all the features are important for the classification and we didn't eliminate any feature.

4. Modeling and Evaluation

Our problem definition and preliminary data analysis showed that using generalization error alone as a performance metric could be misleading since we have a bigger portion of data from the negative class. So, in the context of our problem, it is more reasonable to evaluate the performance based on accuracy and one of f1 or f2 scores. between f1 and f2 we have chosen the f2 because in this kind of detection problems having less false negatives is more desirable.

we also got help from AUC as a good indicator of one classifier's ability for correct prediction over another to get a sense of precision and recall of the model.

4.1 Selected Models

We tried a large set of classifiers including Logistic Regression, NaiveBayes, LDA, QDA, Decision Tree, AdaBoost, GradientBoosting, XGBoost, RandomForest, KNN, MLP and SVM which among them we took out the linear ones: Logistic Regression, NaiveBayes, LDA and also QDA due to their poor performance on the dataset.

4.2 Building Models

We built models using the primary feature set. Four best performing models were RandomForest, MLP, XGBoost and GradientBoosting.

Model	**Random Forest**	**MLP**	**XGBoost**	**GDBoost**
Accuracy	0.889178	0.868104	0.849355	0.847689
Recall	0.742839	0.669721	0.624309	0.625872
F2	0.770877	0.669721	0.664081	0.664594
ROC AUC	0.852593	0.818508	0.793094	0.792235

After expanding the feature space we trained our classifiers again, performing better than before. The best three models were RandomForest, XGBoost and GradientBoosting.

Next step we Ensembled RandomForest, XGBoost and KNN models, calculated weights using cross validation. then we got slightly better results than the best performing one which was RandomForest.

Model	**Random Forest**	**XGBoost**	**GDB**	**Ensemble**
Accuracy	0.916990	0.916331	0.913935	0.918934
Recall	0.779085	0.776273	0.913935	0.796687
F2	0.810330	0.808018	0.805857	0.814386
AUC	0.882512	0.881314	0.879179	0.8850644

In all the above experiments, we used cross validation for feature selection and parameter tuning wherever appropriate.

4.3 Comparison to the Competition Winner

After training our model with all the training data we had from kaggle, we used the competition test set which has 48,000 records with no class labels. We tried submitting two different types of models to see how much there is an improvement. First step which we built models using the primary feature set got score 0.11654, After building models using the extended feature set, our Ensemble model got score 0.24245.

Kaggle uses Gini coefficient [3] to rank the submitted solutions. It is calculated by 2*AUC-1.The first place winner has score 0.26720.

5. Conclusion

In this project we dealt with a huge amount of work on data preprocessing. Addressing missing values and having class imbalance were the first issues with data. we also normalized and scaled data in the preprocessing phase. So after training a set of classifiers RandomForest, MLP, XGBoost and GradientBoosting were the best performing models among all. After that we expanded our feature space using dummy coding categorical features and making combinations of primary numerical features. Which gave us a great boost in performance, having RandomForest, XGBoost, GradientBoost as the best performing models. Next step we made an ensemble of RandomForest, XGBoost and KNN resulting a better performance than the best single model, demonstrating 0.918934 of accuracy and 0.8850644 of AUC.

6. References

[1] https://www.kaggle.com/c/DontGetKicked
[2] https://en.wikipedia.org/wiki/Carvana
[3] https://en.wikipedia.org/wiki/Gini_coefficient