Craigslist Used Car Prices: Predictive Models

Catherine Tao, Sung-Lin Chang, and Arely Vasquez

University of California, San Diego

June 9, 2020

1 Introduction

With the enormous market that Craiglist offers to the car selling market, we decided to focus our project around that particular market. Craigslist is a very popular site where people buy refurbished items. One of the most popular things that Craiglist sells is cars. With that being said, we got our dataset from Kaggle[**Reference1**]. It consisted of a dataset with more than 500,000 used car sales. Each instance contains data of the car and its conditions, the location the car was sold in, and how much the car was sold for.

As our predictive task, we decided to use this dataset and make models in order to predict the price each car was sold for. We want to be able to look at all the conditions of the car and information of the transaction to be able to predict the cost of the car. On a larger scale, it is important to identify the importance of this task because with a build model we can predict the price of the car for a potential buyer or seller. For example if you were trying to buy a used car from craigslist and are unsure if the price is beyond the price of the car, or if the price is a good deal for the car, by building the model we can see how much every characteristic of the car and look at other similar previous transactions to determine whether or not you would purchase a car from Craigslist.

1.1 Data Comprehension

Before looking at our model, we need to understand the context of our data and that includes looking at the outliers. Figure 1 shows the wide range of Prices of cars, with a handful of outliers. Figure 2 shows the range of years cars were sold in our dataset, and Figure 3 shows the range of the odometer values of those cars. An outlier can dramatically skew a prediction far off depending on the model being used. These figures show our cleaned out values, and we thought these range of values were a good balance of keeping the data we need, without leaving out useful data to create models.

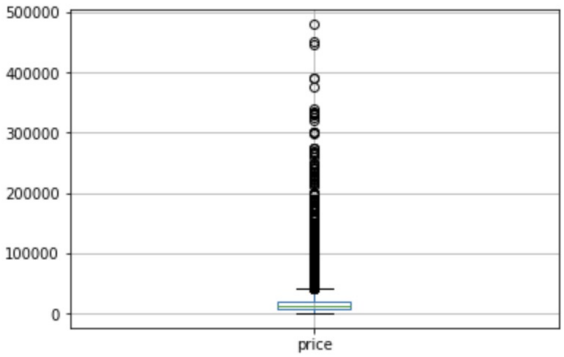

Figure 1: *Box Plot of Prices*

1.2 EDA and Feature Selection

In order to clean the data effectively, we first explored the data's overall integrity. We first considered the types of data in the features. Then, we considered the features useful to our project. We noticed around half of the features did not carry any meaning for our model to analyze. ID for example is a feature unique to each individual craigslist post, but this feature does not carry

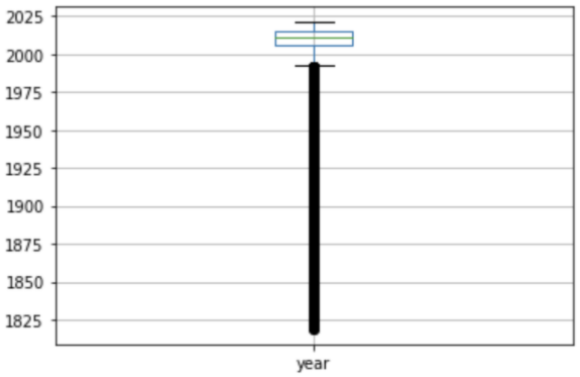

Figure 2: *Box Plot of Years*

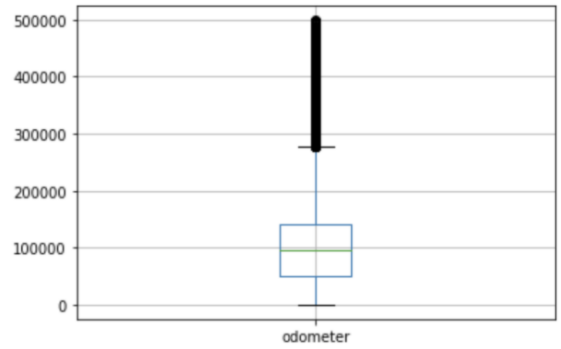

Figure 3: *Box Plot of Odometer*

any mathematical meaning to be calculated. We removed these features as our first step. Our second step was to account for the outliers in the dataset. We filtered the dataset based on prices and odometers of their extreme values. This ensures the prices are more representative of the features given.

1.3 Data Imputation (Novel Cleaning)

The final step was to impute and remove the missing values in our dataset. The description feature contains many useful information that may be used to impute the values missing. We imputed much of our features missing values based on targeted word matches in the description. For example, we match the words unique to a feature that is also found in a car's description. This method was not seen on any kaggle users. Other imputations just involved grouped distribution from relevant columns.

2 Baseline Model: Linear Regression

For the baseline model, we decided to use Linear Regression. We evaluated our model by comparing the R^2 score and the RMSE between a train and test dataset from the original dataset. A Linear Regression model is good for our baseline model since the target variable is numerical. The model is very simple and uses the correlated variables to be able to predict the price of the used car. Some of the drawbacks of this model include that the model is very sensitive to outliers. Initially a linear regression model was built with all of the columns that consisted of an R^2 score of roughly 0.316 and an average RMSE of 10,000. With a poor performance score, linear regression is not the best model to use. From there we built a variation of the initial linear regression model with the most correlated features. Figure 4 shows the correlation between each feature and the target variable, price. The features included in this new model included those who were most correlated such as "year", "fuel", "odometer", "drive", "transmission", and "cylinders". This new model showed a slight improvement with a new R^2 score of roughly 0.40 and a new RMSE of 9400.

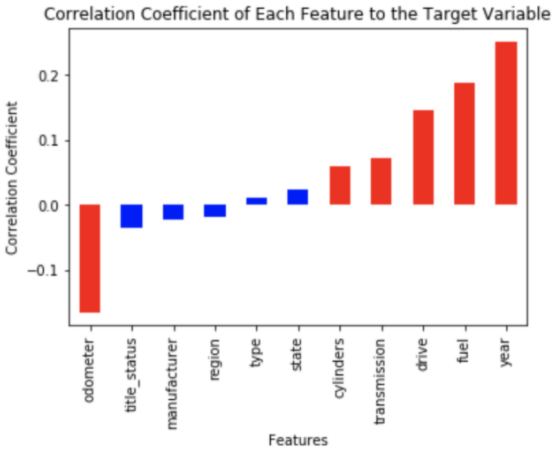

Figure 4: *The corresponding correlation coefficients between each feature to the target variable, Price. The bars in red represent the features that were used in the updated linear regression model with higher correlation.*

The equation for a Linear Regression shown below includes coefficients, inputs, and error values in order to find the predicted output. The inputs for our datasets would be the features we are using for the model (fuel, transmission, cylinders

etc.). The coefficients are weights for each input feature into our model. The error term is added at the end to include the noise within the model.

The equations below show and incorporate instances of both single and multi variable datasets:

$$\hat{y} = \beta_0 + \beta_1 x + \epsilon \tag{1}$$

$$\hat{y} = \beta_0 + \beta_1 x_1 + ... + \beta_p x_p + \epsilon \tag{2}$$

3 Models

3.1 Decision Tree

For our next model we decided to use the Decision Tree Regressor without any hyper parameters. We again get the R^2 score to see how much the price of the car was explained by the features. We figured a decision tree will fit the data much better than a standard linear regression. We noticed the tree was clearly overfitting without any hyper-parameters. We concluded it was over fitting for two reasons: we did not limit the depth of the tree and the prediction on the training set was at 0.99. Even with the overfitting, the decision tree performed better than the linear regression. The R^2 score went from around 0.40 in linear model to 0.60 for decision tree in our testing dataset, which was a significant boost. The RMSE went from 9400 in the linear model down to 7500 in the decision tree.

There are a few ways we can prevent overfitting in the easily overfitted decision tree model:

1. max_depth: This model can be improved by limiting the max depth of the tree to lower the likelihood of overfitting. Less depth means less specification on how the prices are determined.

2. min_samples_split: The typical decision tree has a default of 2 for splitting a sample. By increasing the min_samples_split we can potentially reduce the overfitting by heightening the bar for a feature to split a dataset.

3. We can switch out the Decision Tree Regressor all together with Random Forest Regressor from Ensemble Learning. The random forest regressor can much better tackle the overfitting problem by randomly generating the split.

Decision tree is a top down recursion following the process of:

1. Decide the best splitting rule based on criterion. In a regression tree, the best splitting point is typically the maximum reduction in the least sum of squares. (Equation 3)

$$\Sigma(y_i - \hat{y}_{R1})^2 + \Sigma(y_i - \hat{y}_{R2})^2 \tag{3}$$

2. Create children nodes based on the splitting criterion and assign training data. Apply the splitting rule on children.

3. The process will be repeated until stopping criteria is met. This can be limited by max depth, min samples split, or unsatisfactory splitting criterion.

3.2 Random Forest Regressor

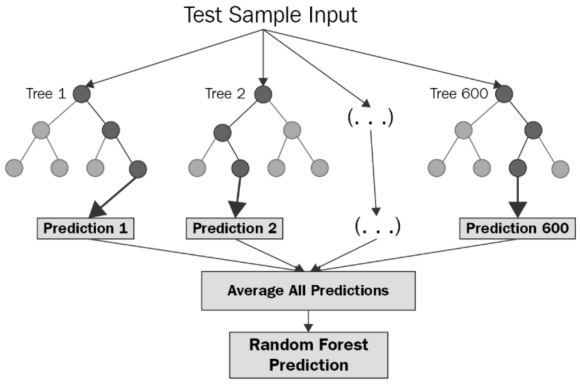

Figure 5: *Visually shows how the Random Forest Regressor determines a prediction [***Reference6***]*

Another model we chose to use was the Random Forest Regressor. This model utilizes ensemble learning which combines predictions of many machine learning algorithms. Bagging (bootstrap aggregating) is also involved for this algorithm which is a type of machine learning ensemble method to help average and stabilize our model. The method of bagging is also useful because it takes random samples within the data with replacement N number of times (each time we call it our "tree" in the forest). We once again used price as our target y value since we want our overall model to predict the price on each vehicle. The random forest regressor is used to reduce the amount of overfitting involved. Many of our columns are of categorical data type and this type of model is known for working well with this type of data. Although we thoroughly cleaned the data to ensure little to no null values are presented in the dataset, a benefit of the random forest regressor is that it can handle

missing values very well and is robust to outliers. As a result, all of these factors help to improve overall accuracy of our model.

The Random Forest Regressor works by randomly sampling from our dataset with replacement N number of times. Each time we sample, we consider this a "tree" in the forest. Once all of the trees are completed, the algorithm finds the most common majority amongst all of the trees and this is the final completed value output. As for hyperparameter tuning, we chose to tune the n_estimators parameter for our model. This parameter tells us the number of trees we want in the forest. We chose to use 1 for our number of n_estimators for our final model because this helps reduce the number of trees being utilized. The RMSE value for the random forest regressor resulted in 7759 for our test dataset.

The Random Forest Regressor algorithm utilizes the Mean Square Error (MSE) and Mean Absolute Error (MAE) depending on whether Scikit-learn is implemented. For our algorithm we utilized Scikit-learn. The algorithm runs N number of times, each time taking the absolute value summation of each y instance and mean value subtracted from one another.

Variance / Mean Square Error (MSE)	Regression	$\frac{1}{N}\sum_{i=1}^{N}(y_i-\mu)^2$	y_i is label for an instance, N is the number of instances and μ is the mean given by $\frac{1}{N}\sum_{i=1}^{N} y_i$
Variance / Mean Absolute Error (MAE) (Scikit-learn only)	Regression	$\frac{1}{N}\sum_{i=1}^{N}\lvert y_i-\mu\rvert$	y_i is label for an instance, N is the number of instances and μ is the mean given by $\frac{1}{N}\sum_{i=1}^{N} y_i$

Figure 6: *The figure shows the mathematical equations utilized within a Random Forest Regressor. Since our model is under Scikit-learn, we utilize the second common mathematical equation which implements regression analysis by use of Mean Absolute Error (MAE)* [**Reference7**]

3.3 AdaBoostRegressor

Another model was AdaBoostRegressor. With being known for being an adaptive regressor, we wanted to apply it to our model to see how good it would adapt to our dataset. This model was not successful compared to our other models and our final model. It had a large inconsistency of evaluation measures of score and RMSE. The average R^2 score ranged from about .15 to .33, with a range of RMSE of 10000 to 12000. We see a very large range of values because this model's algorithm is based on random adjusted values. The algorithm initially fits a regressor to the Craigslist dataset, and from there it repeatedly adjusts the weights. This adaptable algorithm is a variation of a Decision Tree Regressor, with that being a default parameter of the AdaBoostRegressor. With that being said, the randomness of the algorithm does not work effectively for this dataset as we can see from the RMSE and R^2 score. One con of the model is it's very prone to being overfitting. With that the hyper-parameters can be adjusted in order to improve the overfitting. Those parameters include:

1. Base_estimator: as mentioned above, the model is a variation of the DecisionTreeRegressor, with a default parameter of DecisionTreeRegressor(max_depth = 3). We found the default parameter to be a fitting value in order to prevent overfitting.

2. N_estimators: having this value be too large can lead to overfitting since it reiterates boosting until the perfect fit is found. If the N_estimators value is an enormous number, the model can be overfitted. Although if it is too small, it doesn't have enough attempts to capture the perfect fit. We found the right balance to be set at its default.

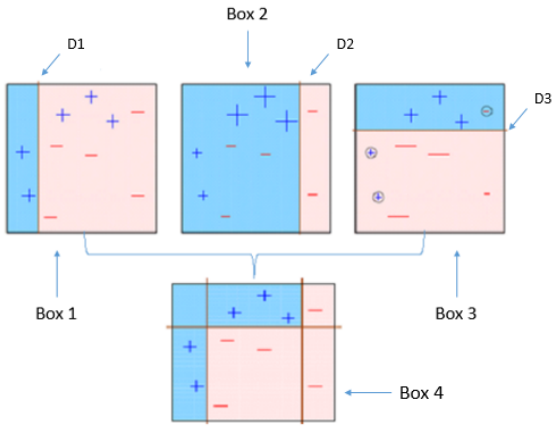

Figure 7: *Visual of how AdaBoostRegressor works and how the decision stumps group the data* [**Reference4**]

In the figure above, we can see how the number of stumps is determined by the parameter of Base_estimator. Like in the example in the figure, there are 3 stumps like our model's default parameter. This creates the lines that ultimately are used as a regressor line to predict the target variable. From there the algorithm works to determine which weights are more/less weighted and the number of times this is determined is by the

other parameter, N_estimators.

3.4 XGBoost Regressor

An additional model we chose to use was XGBoost Regressor to view the accuracy and to compare this model to our other models. We chose this regressor as one of our models because this model utilizes regularization. There are two types of regularization: L1 Lasso Regression and L2 Ridge Regression. XGBoost Regressor incorporates both types of regularizations that can be tuned in their respective parameters. This helps prevent our regressor from overfitting. This model also uses cross validation which is beneficial because the model finds the most optimum boosting iterations in one single run. Another reason we opted to use XGBoost Regressor is the fact that this model handles missing values very well. The algorithm chooses a left or right hand split on the training data and learns by finding the higher loss. This regressor had an RMSE of 6641 on its test data set.. The hyperparameters which were tuned include colsample_bytree, gamma, learning_rate, max_depth, n_estimators and random_state. The colsample_bytree subsamples a specific amount of columns when creating a tree. The gamma parameter minimizes the amount of loss created with each separation of a leaf node from a given tree. The learning_rate parameter helps to prevent overfitting while minimizing the feature weights in order to preserve the boosting algorithm. The max_depth parameter provides the maximum depth for a given tree. The n_estimators parameter specifies the number of trees present. The random_state parameter represents a random number seed.

The XGBoost objective function analysis [**Reference5**]:

$$L^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t) \qquad (4)$$

Can see that f(x+$\triangle x) where$:

$$x = \hat{y}_i^{(t-1)} \qquad (5)$$

The XGBoost Regressor's math utilizes many functions within functions. This algorithm basically finds the minimized summation of loss in order to acquire the most accurate predictions for the price of cars for our dataset. This equation incorporates regularization in order to minimize the loss for each iteration.

4 Final Model: LGBM Regressor

For the final model, we decided to use Light Gradient Boost Regressor because it is one of the fastest and best fitting regressors in the kaggle community. We discovered gradient boost is similar to Adaboost described above, but instead of stumps (depth of 1), gradient boost utilizes trees (depth >= 1). Both regressors take the gradient errors from the previous tree and calculate the errors to form a new tree. Since the LGBM Regressor is a more advanced version of Adaboost, there are also more hyper-parameters to tune. Our model was heavily overfitted in the beginning until we derived four of which are the most important for the best fit and accuracy.

1. min_data_in_leaf : When you have a huge dataset this parameter should be adjusted up to heighten the bar for the minimum number of data needed for a leaf, vice versa for the smaller dataset.

2. num_leaves : This parameter is to be adjusted less than the $2^{(max\_depth)}$ because it can lead to overfitting, but can be adjusted to more or less to increase the accuracy.

3. max_depth: This specifies how deep the tree can grow, so the tree cannot be too fitted into the training set.

4. max_bin : The larger the bin size will increase the fit of the data, but can too much can lead to overfitting.

After these adjustments our model's rmse gap between training and testing dataset shrank significantly.

LGBM Regressor Equation:

Light Gradient Boost Machine is an implementation of the gradient boosting, so we will explain the steps and mathematics of gradient boosting below. Pre-processing:

1. Get the minimal loss function for a constant value:

2. For 1 to M:

a. Compute pseudo-residuals/gradients:

$$r_{im} = \frac{-\partial L(y_i, F(x_i))}{\partial F(x_i)}_{F(x)=F_{m-1}(x)} \qquad (6)$$

for i = 1, ..., n.

b. Fit a regression tree to pseudo-residual to show the adjustment rate.

c. Calculate the minimized pseudo-residual for each leaf of the tree.

$$\gamma_m = argmin\Sigma L(y_i, F_{m-1}(x_i) + \gamma h_m(x_i)) \quad (7)$$

d. Update the model for each predicted y using corresponding x from each leaf.

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x) \quad (8)$$

3. Output Fm(x)

5 Literature

With the dataset being from Kaggle, there are a handful of discussion and data analysis already implemented on this dataset. With that being said, some of those publishings have the same task as us, predicting the price of the car.[**Reference3**] Although we have also found other tasks that people have done with the same Craigslist dataset. These other tasks include finding the best time to sell or buy a car, car accidents and their sales, and the estimation of the price of selling specifically diesel cars.

For those who have the same predictive task, after looking at others' work and findings, each publisher presents their information in a unique way with focusing on certain features and columns of the data. Similar to our data cleaning process, a lot of others decided to use EDA to clean out a handful of the missing value in the original data. People also used the Random Forest Tree and Decision Tree as models to predict the price. Although other kaggle users had a better R^2 score on their models, our final model of LGBMRegressor performed better than those on kaggle when compared by the RMSE.

Our successful RMSE score for our final model could have been from our unique and effective method to data scrape from the "description" column to fill missing data; that was the major novelty of our work. We also took into account the hyperparameters to make sure not to overfit our model which was beneficial to us in the end. Along with our data scraping from other columns and finding missing by design data, we used data imputation, and EDA to have a dataset to start building our models with. As seen in Table 1, you can see the steps of cleaning our data, and how the percent of null values progressively decreased.

The table below shows the percentage of null values for each column are shown above during each part of the data cleaning process:

Column	Inital	Data Scraping	Data Imputation	EDA
cylinders	.39	.31	0.0	0.0
description	0.0	0.0	0.0	0.0
drive	.29	.25	0.0	0.0
fuel	.006	.006	0.0	0.0
manuf	.04	.03	0.0	0.0
model	.01	.01	0.1	0.1
odometer	.18	.18	.18	0.0
paint color	.32	.21	.21	.21
price	0.0	0.0	0.0	0.0
region	0.0	0.0	0.0	0.0
state	0.0	0.0	0.0	0.0
title_status	.005	.005	0.0	0.0
transmission	.007	.007	0.0	0.0
type	.27	.18	0.0	0.0
vin	.42	.42	.42	.42
year	.002	.002	0.0	0.0

With that being said, our conclusion to our most successful model was different than those to some kaggle users. Other Kaggle users found Decision Tree Regressor to be the best model with the best score, but we found LGBMRegressor to be the best. Aside from those models, they also worked with other baseline models such as Linear Regression, but our models seemed to perform better in terms of RMSE score because of our effective data cleaning.

6 Result

Our proposed final model is the LGBM Regressor model. This model is the most appropriate to use in predicting the price of used cars from

our Craigslist dataset because we were able to efficiently predict the cost of each car. The LGBM Regressor worked the best out of our other models due to its resulting accuracies. This model was able to outperform our other models due to the fact that the model uses higher efficiency and the algorithm runs much faster than our other models. This model works especially well with larger datasets in terms of runtime. Although this model is prone to overfitting, the hyperparameter of max_depth is used in order to decrease the amount of overfitting.

The gap is significant because our other models have many aspects deemed detrimental to our goal of predicting the price of used cars from Craigslist. The linear regression baseline model did not work as well as the LGBM regressor because the algorithm for linear regression is extremely sensitive to outliers. Our data had some outliers we had to clean, however this may have affected the accuracy and performance for the linear regression baseline model. The decision tree machine learning model did not perform as well as our final model because decision tree regression is known for overfitting on its data. Random forest did not perform as well because it is considered an expensive algorithm. This algorithm does not run efficiently on larger datasets, which is why it did not outperform. The Adaboost Regressor did not outperform because the model does not work well with noise within the data. There were many data imbalances which hindered the success of accuracy for Adaboost. XGBoost performed fairly well with this dataset, however the only disadvantage to XGBoost is that it does not work as well with larger datasets in comparison to LGBM. The main reason that LGBM outperformed all other models is the fact that it can run a large dataset more efficiently and less costly. This allowed our final model to more accurately predict the cost of each Craigslist car presented.

All features we designed were effective in achieving accuracy for our final model. We used OneHotEncoder and FunctionTransformer as our main features to help achieve accuracies in predicting Craigslist car prices. OneHotEncoder was used for categorical data types while FunctionTransformer was used to pass in our cleaned columns into the model.

The hyperparameters used to tune our LGBM regression model were the num_leaves, max_bin, max_depth, and min_data_in_leaf. It is important to hypertune our model in order to achieve higher accuracy for our results. The num_leaves value signifies $2^{(max\_depth)}$. Although it can lead to overfitting, this parameter worked well for our model in approving its overall accuracy. The parameter for max_bin is used to increase accuracy and to improve the speed in which the algorithm runs. The max_depth parameter sets a specific number for the depth of the tree and reduces overfitting. The min_data_in_leaf parameter prevents the tree from growing too deep.

The major takeaways overall is that each model is unique and significant for their own purposes. There are advantages and disadvantages to all models. The LGBM Regressor model resulted in our final, most accurate model because it works efficiently with large datasets. The other models used did not perform as well due to the size of this dataset. From our findings, execution time for our final model is extremely faster than other models in comparison. Our hyperparameters for this model also helped to achieve more accurate results.

Model	RMSE(training) USD	RMSE(testing) USD	R^2(training)	R^2(testing)
Linear	9479.40	8952.62	0.40	0.40
DecisionTree	221.52	7417.98	0.60	0.99
RandomForest	4840.58	7759.23	0.83	0.57
Adaboost	10911.32	10863.09	0.16	0.15
XGBoost	5243.03	6641.02	0.80	0.69
LGBMRegressor	5401.72	6085.21	0.79	0.73

Figure 8: *The final performance scores of each of our models*

References

[1] Kaggle.com (2019) *LaTeX, Craigslist Cars Dataset*

[2] Youtube.com/latex (2011) *LaTeX: a video:How to write a conference paper (IEEE, ACM) using Latex*

[3] Kaggle Competitors (2019) *LaTeX: other submissions to kaggle dataset*

[4] Towards data science *LaTeX: Understanding Adaboost*

[5] Wikipedia *LaTeX: Gradient boosting*

[6] Towards data science LaTeX: *Random Forest Regression*

[7] Towards data science LaTeX: *The Mathematics of Decision Trees, Random Forest and Feature Importance in Scikit-learn and Spark*