Used Car Price Estimation

Team name: DS numba 1

Team member: *Zhe Huang (zh1087) Cong Liu (cl4055) Daoyang Shan (ds5471) Yulin Shen (ys2542)*

Business Understanding

Nowadays internet plays an increasingly important role in people's daily life, especially when we plan to buy something. From light bulb pack on Amazon to apartment on Zillow, people broadly use information on internet as reference before they make decision--the similar thing happens in pre-owned car market. Assume we are running a pre-owned car website and we want to attract more people to purchase used cars on our website. Unfortunately, we can expect that most people are not car experts and they don't have a reasonable sense of price. So we hope there is an automatic system to help people estimate the price of a certain car and avoid potential fraud, which means to avoid extremely low and high price. With such system, we can build a more reliable website and thus attract more users.

In Kaggle, we find a dataset including over 370000 used car instances scraped from Ebay Kleinanzeigen and each instance has 20 attributes including its price. Assume the information in this dataset is trustable, we can prepare the features of the cars at first, then some data mining algorithms, for example, linear regression, decision tree, random forest, could help us to estimate the target attribute precisely. Later, with the algorithms comparison, we can get a most precise model, and therefore utilize that model in our automatic system to estimate the price when the user inputs the specific information of the used car.

Data Understanding

The dataset has 20 attributes, and they are power, vehicle type, year of registration and so on, but not all of them are useful to estimate car price. Some of the features like the number of pictures on the website or information last seen on the website, are not really related to car price by intuition, so we simply delete them. For the rest, we can use domain knowledge to check their distributions to make sure the features are all related to the target variable.

Among the remaining categorical features, there was one feature we thought might be helpful in the prediction at the beginning which is the ‘offerType’. We assumed different price will differ significantly for different offer type. However, after plotting the bar chart of its number of different offer types, as shown in the figure 1-1, we find out that although the attribute has 2 different values, almost all of them are “Angebot”. Knowing the distribution of this attribute, we can say that this attribute is almost useless to predict the price. Set the remaining useful features as predictor variables, and set price as target variable.

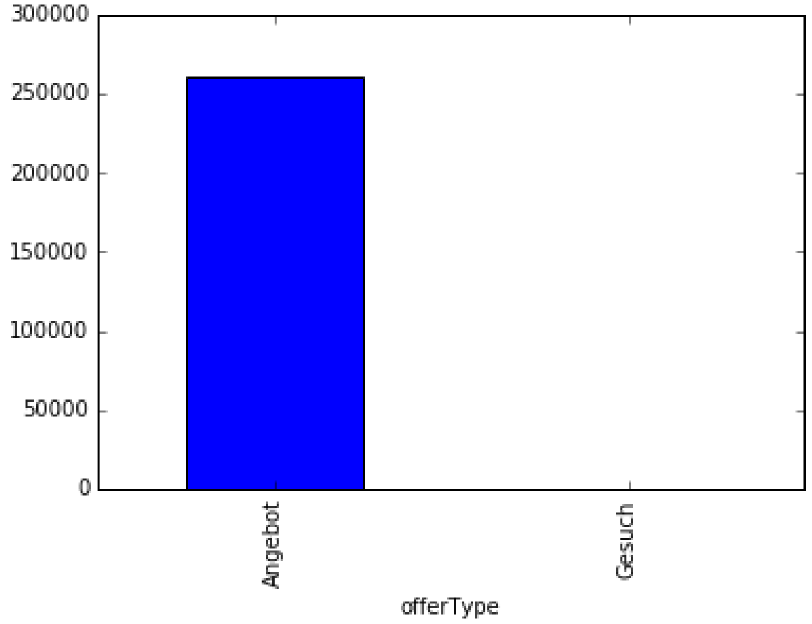

Figure 1-1. Angebot and Gesuch are two different values in predictor attribute “offerType”. The bar chart shows their quantities in the dataset respectively.

After dropping those unrelated information, we proceed to check relationships of each pair of remaining attributes. In our exploratory data analysis, we select target variable – car price, and also four numerical predictor variables, year of registration, month of registration, power, and kilometers, to make 5 times 5 scatter plots to see their relationships. In the figure 1-2, we can find that there are lots of outliers, which requires us to further clean the data set.

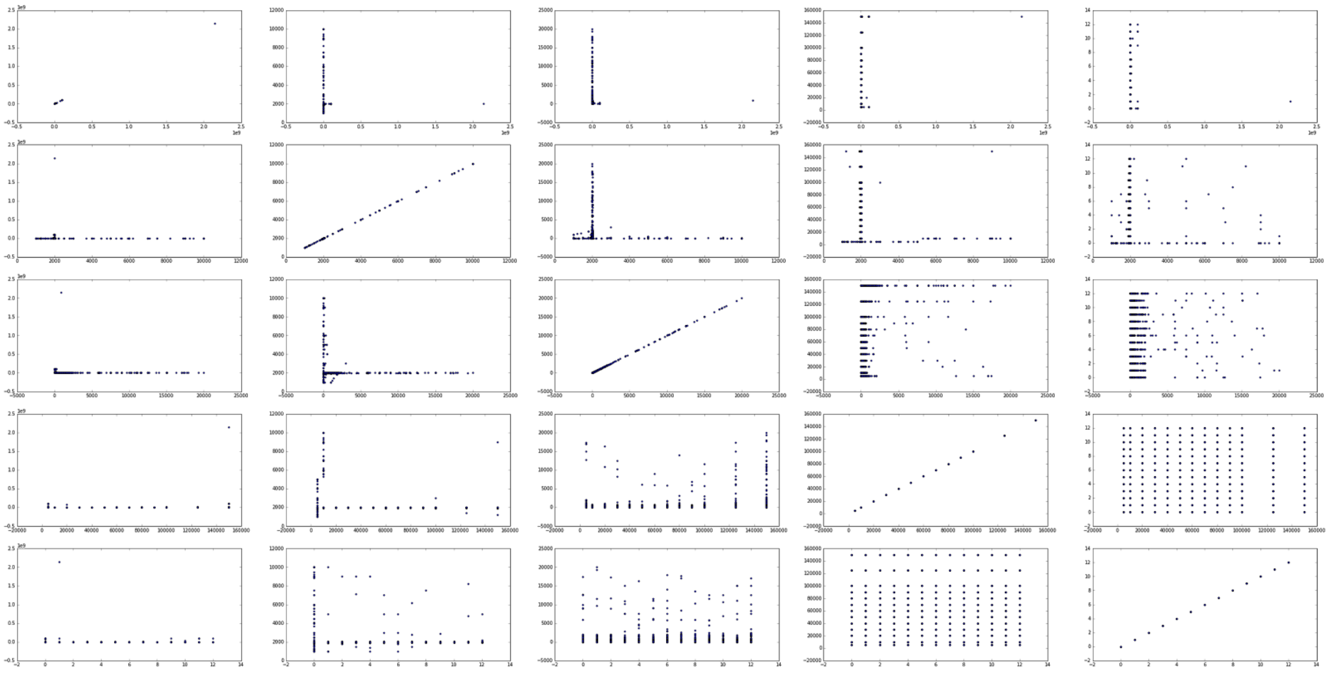

Figure 1-2. Scatter plots of price, year of registration, month of registration, power, and kilometers before cleaning the dataset showing lots of outliers.

We plot 3 histograms to see the range distribution of the variables above. The first one is price distribution, the second is for power, and the third one is for kilometer. In the figure 1-3, we can select the main part of their values. Then we can determine a criterion for getting rid of outliers. Limiting the range of the attributes' values could help us delete some "special" instances to clean the data. Based on the exploration, we limited car price in the range of 100 to 150000, year of registration in the range of 1950 to 2017, and power in the range of 10 to 500. After implementing it, we can see that about 60000 instances are removed, which are about 15% of the total amount of data.

After cleaning the dataset, we plot 5 times 5 scatter figures of 5 numerical variables again in the exploratory data analysis to check outliers. The figure 1-4 looks cleaner and better compared with raw data scatter plots.

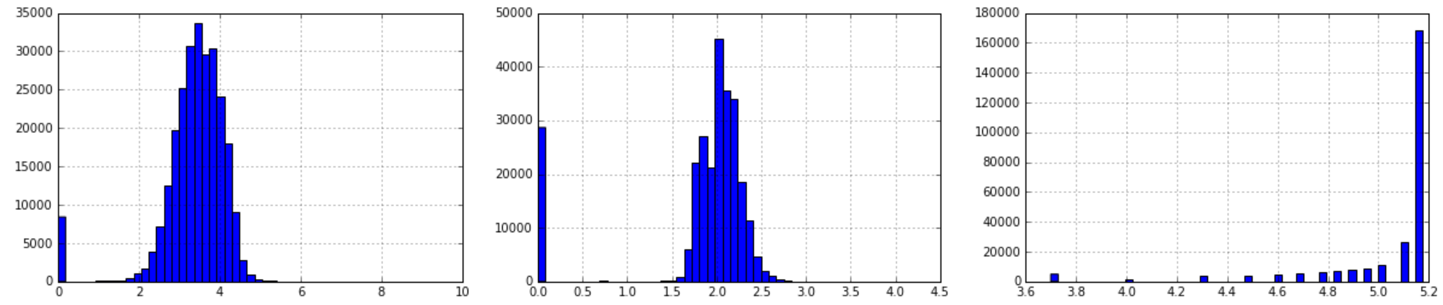

Figure 1-3. The range distributions of natural logarithm values of price, power, and kilometer respectively. They can show us the rough distribution of their values to do limitation parameters selection.

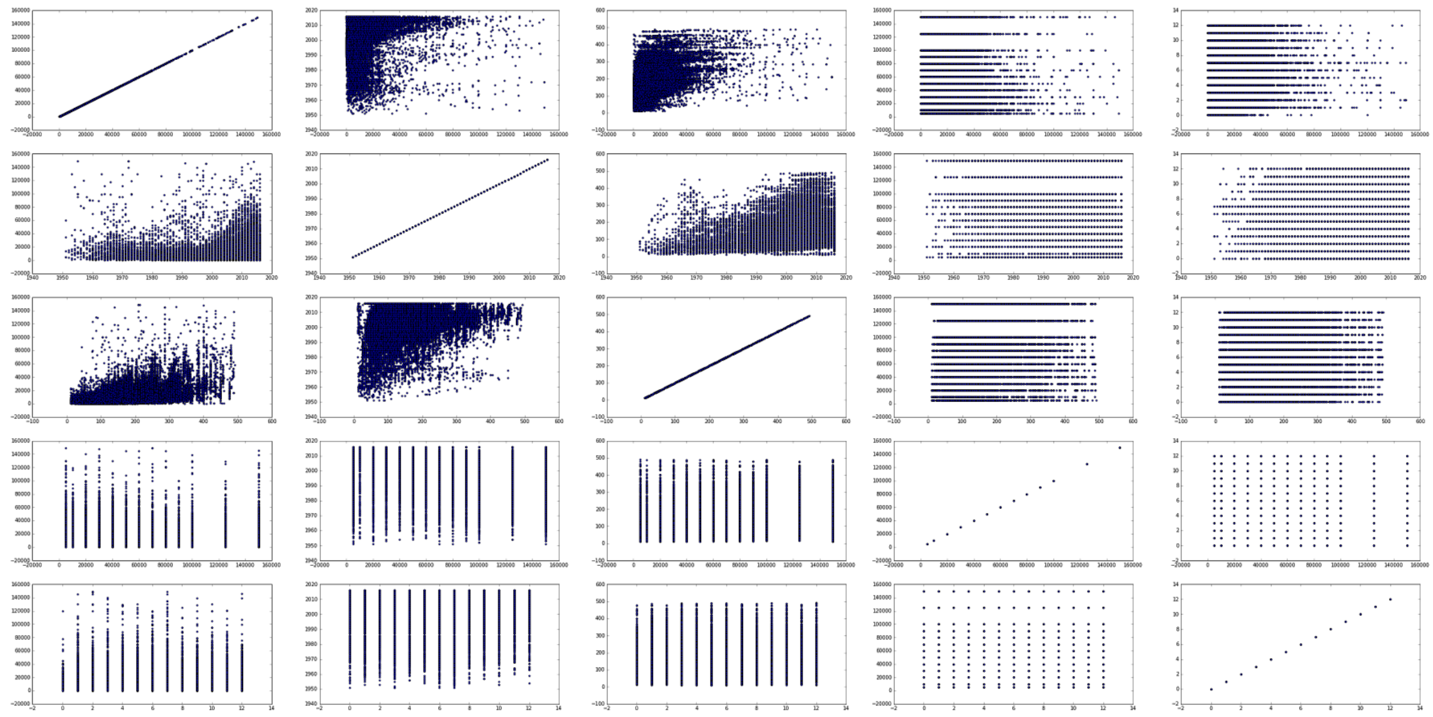

Figure 1-4. Scatter plots of price, year of registration, month of registration, power, and kilometers after cleaning the dataset showing less outliers.

Data Preparation

At first, we split the dataset to make 70% of them as the training set and 30% as testing set. Then we build a baseline model, in which we ignore the entire categorical variables, and use a linear regression with all default setting, since linear regression is very popular in this type of task. Then, we test the baseline model on the test set using mean-absolute-error as evaluation metric, since it is more straightforward in our task comparing to mean-square-error. We find that the mean absolute error is about 29426, which is very high. Based on plotting a graph to show predicted price compared with actual price, the figure 2-1 shows that the graph is skewed by outliers.

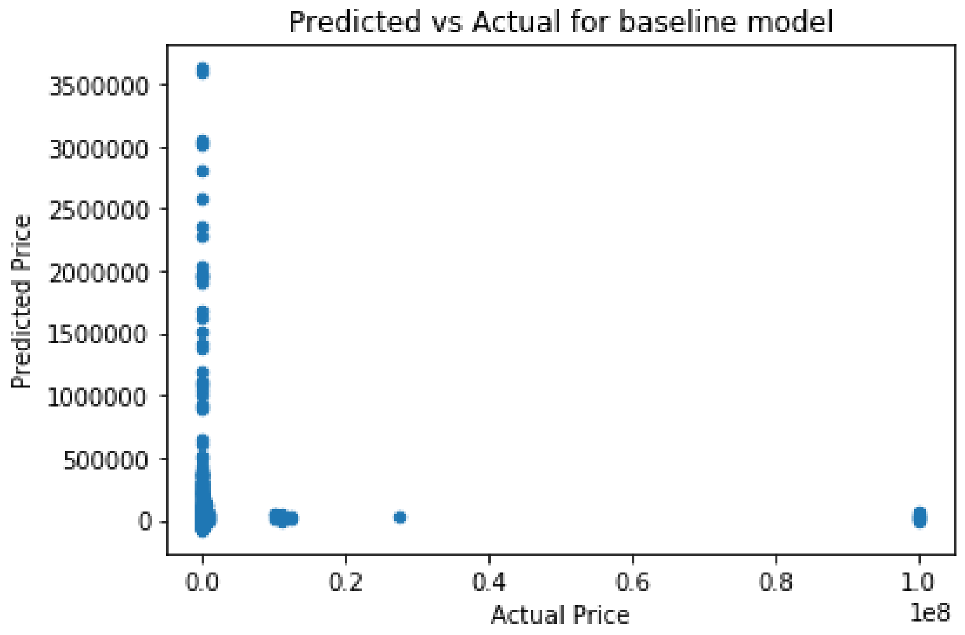

Figure 2-1. The scatter plot of predicted prices of baseline model and actual prices. The figure shows huge discrepancy between them.

Obviously, we should do feature engineering on the training set. In our exploratory data analysis, we have already concluded that we need to delete some unrelated attributes, because they are less informative to our target variable. So we delete variables like "dataCrawled", "name", "offerType", "abtest", "monthOfRegistration", "brand", "dateCreated", "nrOfPictures", "postalCode", and "lastSeen". The remaining attributes show much more information about estimating car price.

Although, we have already limited the value ranges of numerical variable in the dataset to avoid outliers, there are a lot of missing values, and we do not consider it in our exploratory data analysis. It is necessary to fill out all of them. The problem could be divided into 2 parts: numeric and category. We decide to treat all numerical missing values as the mean of other data in the column. Also, we treat all categorical missing values as "notDeclared".

As we known, there are so many kinds of car model types. In the dataset, this attribute shows us a variety of car model data, but some of them only emerge few times like some extremely luxury cars and unpopular models. We could combine all of them into one category – others.

In the dataset, variables “seller”, “vehicleType”, “gearbox”, “model”, “notRepairedDamage” and “fuelType” are all categories. The figure 2-2 shows that most of them have more than 2 categories in their columns. So, we use one-hot encoding to transform the categorical variables for the use of scikit-learn model. Now we are ready for modeling our training set.

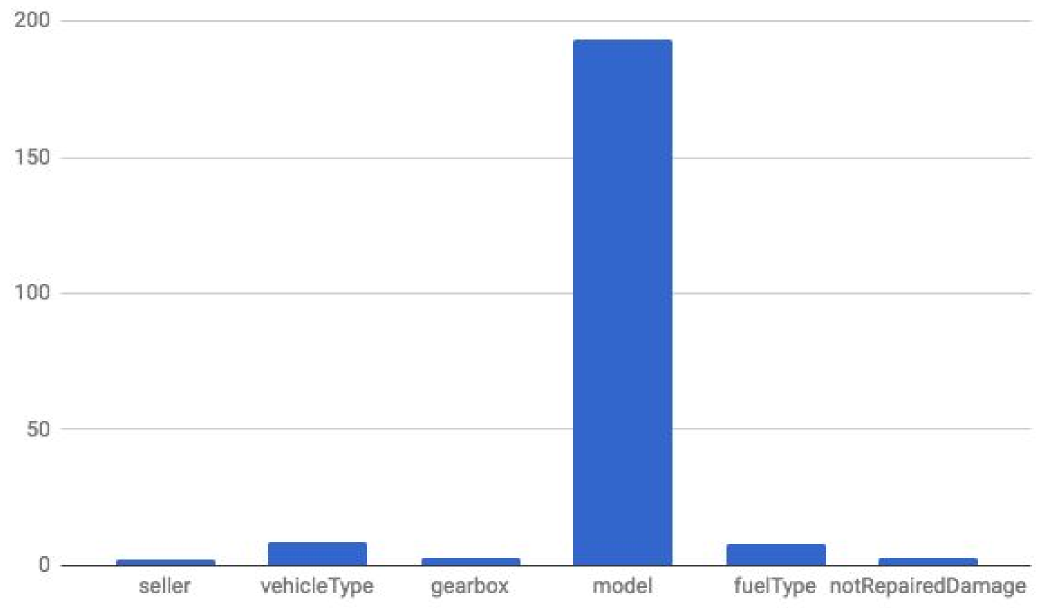

Figure 2-2. The bar chart shows the number of different kinds of categorical values in the attributes seller, vehicleType, gearbox, model, fuelType, and notRepairedDamage.

Modeling & Evaluation

We have chosen 5 possible algorithms as the model for our task: ridge regression, lasso regression, random forest regressor, decision tree regressor, and KNN regressor. To select the best algorithm, we perform a Gridsearch on each algorithm, to select the best parameters for them. We split the training set into 2 parts. 70% of the training data is used for the Gridsearch. The Gridsearch is implemented using sklearn GridSearchCV method with default 3 fold cross-validation. We are then going to compare these 5 algorithms in their best parameters, and select the best algorithm for our task.

We first try a large range of parameter, from 10^{-10} to 10^{10} , the result gives us a best parameter of 0, so we narrow the range and search it again. We then choose 10 numbers from 0 correspond to ordinary least square to 0.9 to be the candidates of ridge regression parameter alpha. As a result, parameter of 0.8 has the best test score.

We apply the similar approach to lasso regression. After searching the parameter from a wide range, we narrow parameter selection down to the range of 0.02 to 1.5. Figure 3-2 shows us the best parameter is in a narrower range. Because it is hard to see the best one in this figure, we also plotted figure 3-3, which shows us the best parameter of alpha is 0.04. Also, the Gridsearch infers that "normalize" parameter is false.

In random forest regressor, we let the tree to overfit and search for forest level parameters. We need to find parameter "n_estimators". So we choose 12 numbers from 10 to 500, and 200 performs best, and we can see that the scores converge after 200 trees in the figure 3-4.

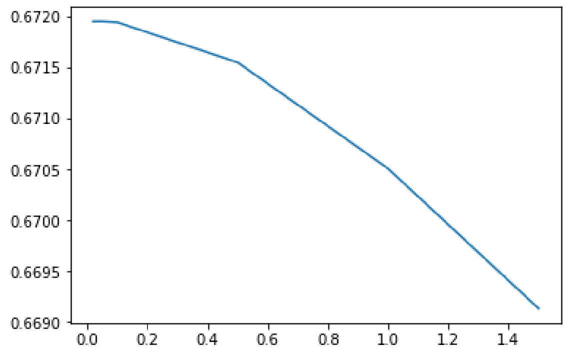

Figure 3-2. The learning curve of the lasso regression parameters selected from 0.02 to 1.5 and their corresponding test scores.

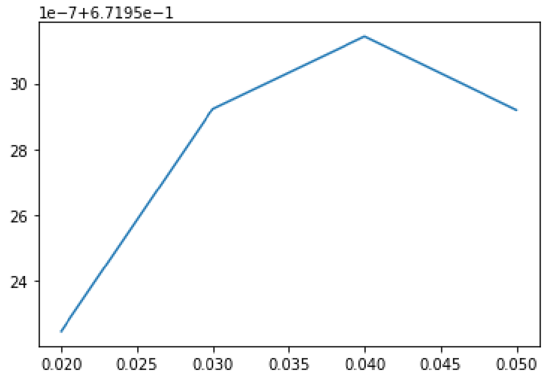

Figure 3-3. The learning curve of the lasso regression parameters selected from 0.02 to 0.05 and their corresponding test scores. The figure shows us that parameter of 0.04 has the best test score.

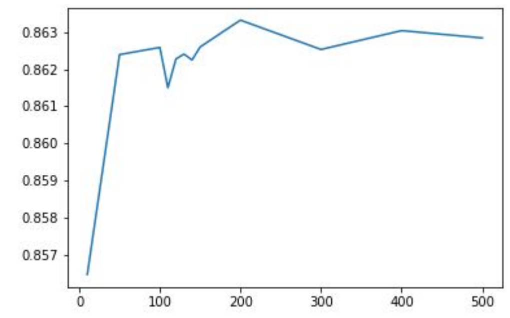

Figure 3-4. The learning curve of the random forest regressor parameters selected from 10 to 500 and their corresponding test scores. The figure shows us that parameter of 200 has the best test score.

Similarly, we search parameter for decision tree and KNN regressor, we find that parameters "max_depth" as none, "min_samples_leaf" as 1, and "min_samples_split" as 302 is the best combination for our decision tree model. Also, parameters "n_neighbors" as 5 and "p" as 1 is the best combination for our KNN regressor. The figure 3-5 shows the learning curve of different parameters of KNN regressor.

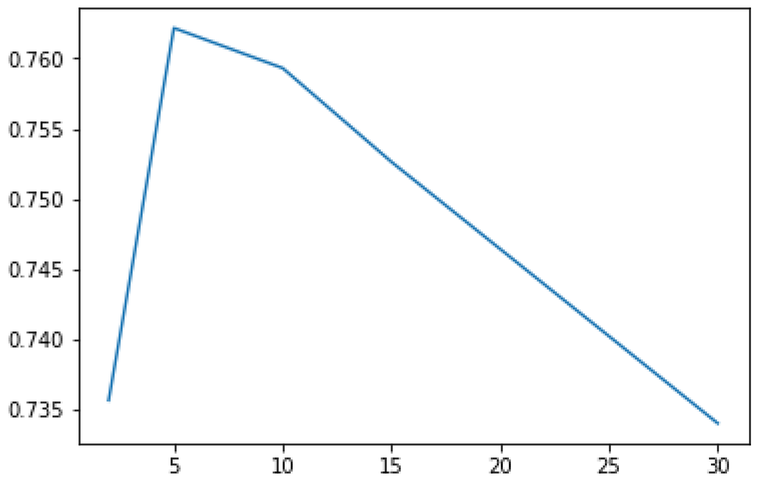

Figure 3-5. The learning curve of the KNN regressor parameter p selected from 2 to 30 and their corresponding test scores. The figure shows us that parameter of 5 has the biggest test score.

Now, we have successfully determined all parameters of each model. We then train the models on our 70% of the training set, and test the models on the remaining 30% of training set we left out. The mean-absolute-errors of each model are 3326, 2645, 1257, 1628, and 1537 respectively for ridge regression, lasso regression, random forest, decision tree, and KNN. From the result shown in the figure 3-6, we selected random forest regressor with setting “n_estimators” to be 200 as our best algorithms because it shows the least mean-absolute-error. Although random forest regressor is slower, it performs substantially better.

Then, We train the model on the whole training set, and applied the same transformation as the training set to the testing set. We run the model on the test set. The mean absolute error we get is only 1834 now, compared to 29426 in the baseline model, showing a huge improvement on the model in the figure 3-7.

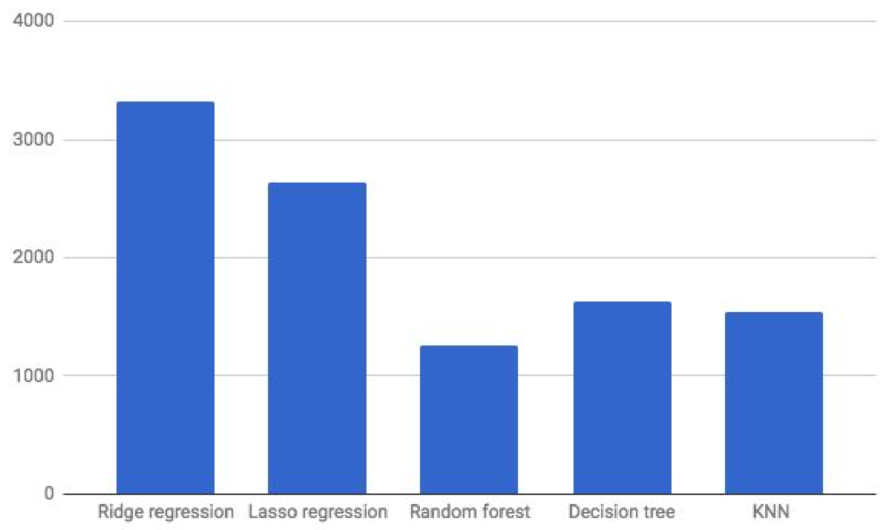

Figure 3-6. The bar chart shows the mean-absolute-errors of ridge regression, lasso regression, random forest, decision tree, and KNN based on the 70% of the training set.

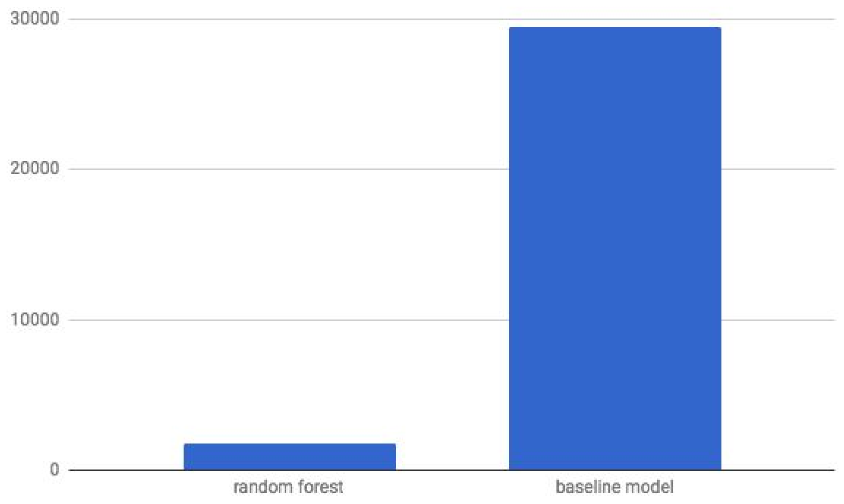

Figure 3-7. The bar chart shows the mean-absolute-errors of random forest and the baseline model based on the whole training set.

To visualize the final result, we plot predicted vs. actual price. In the figure 3-8, we see that most of the points are centered around the 45 degree line, showing high "precision" except few point at the far end are well off.

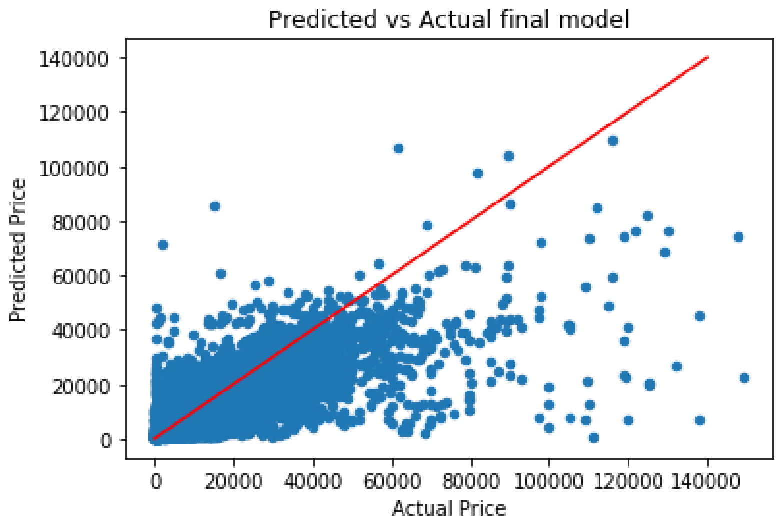

Figure 3-8. The scatter plot of actual price and predicted price based on the final model. The line in the figure shows that most of points are centered around the 45 degree which means 2 prices are matched.

After finishing modeling and evaluation steps, the CRISP data mining process shows that we should try to return to the steps business and data understanding. So, we plot the bar chart for each feature importance to check whether all attributes in the model are informative about the target variable. As shown in the figure 3-9, seller has very little importance compared to the other features. Then we try the subset of the features without attribute seller, and we get similar mean-absolute-error result of 1833.

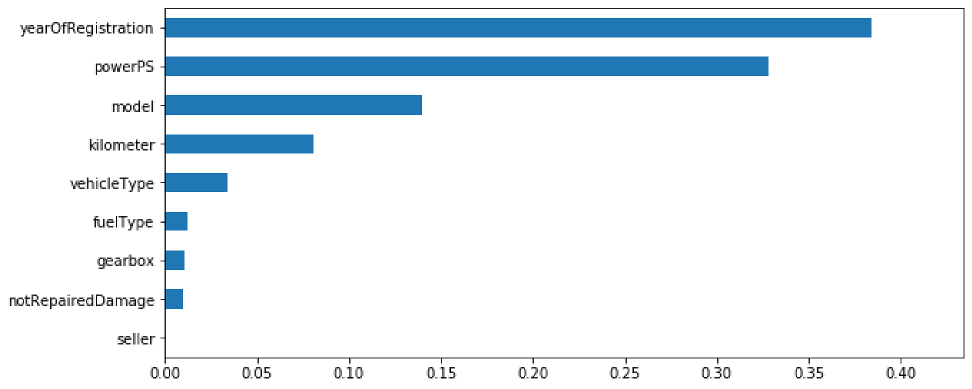

Figure 3-9. The bar plot of feature importance relating to the target variable price. It shows "seller" is less informative to the result.

Deployment and Simulation

Now we have trained a relatively good model, with which users could input necessary information to get car price estimation as reference. However in real-world business practice, what we need may not be just a static model because car industry and market are evolving constantly. Imagine the following scenario: lots of car owners and dealers want to sell their cars on our website, and our website receives tons of car information and associated price every day. Can we utilize the new information to get a better model of prediction? On the other hand, users who want to purchase a pre-owned car browse our website and send request for price estimation. Can we satisfy all their needs? These two questions motivate us to design a system that simulates the business challenges that our website may face and provides the ways to solve them.

Let’s formulate the problem in this way: if our website keeps receiving new data, how can we dynamically update our model to maintain a good level of prediction accuracy? The straightforward but practical way is: store a certain amount of new data in a batch, and add that batch to our original data and retrain our model. We divide the full data set into three parts, making 50% of the total amount as train set, 30% as batch set and 20% as test set. We further divide batch set into twelve equal-sized batches and add them to train set in succession. Each time we add a new batch to the train set, we combine two sets and retrain the model with default Random Forest. One thing we need to notice is the ‘rare model’ issue that we’ve discussed in the previous modeling. In a real-world scenario, the models that we categorize as ‘rare model’ may change with the size of train data increases. For example, sellers may post a huge amount of information of a model that we used to consider as rare. Also, new models that we never saw before may enter our data set, and they should be considered as rare models if their amount is small. The simplest way to handle this is to keep track of the amount of each model with a dictionary and update the list of rare models. This update will also influence our original train data as we mentioned before.

After we’ve finished the work of simulation, the next problems are: what should we expect from this simulation? How do we evaluate our work quantitatively? To test our model, we test it in two different ways whenever we retrain it: test it on a fixed static test set or on the new batch we’re going to add to be a dynamic test set. The result of static testing in the figure 4-1 confirms the intuition that more data brings better model, but this is rarely the case in real life. The intention of dynamic testing is normally closer to what may happen in real world, and we can observe that the loss curve in the figure 4-2 is not stably decreasing—instead, it shows an obvious fluctuation. In fact, if we cannot predict the pattern of incoming data, all data we already have could be highly biased, and model that works perfect for previous data may perform terribly on new data because of this bias. Although we can observe a decreasing trend in dynamic testing, we must be aware that all data in the original data set are recorded in a short time period. If we sample in a longer time period, we may experience a higher instability, which may even result in an increasing loss curve.

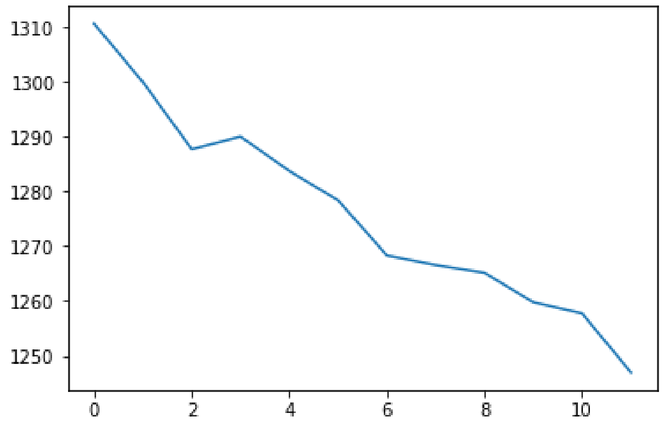

Figure 4-1. The loss curve shows the model based on the training set with numbers of equally-divided batch produces the mean-absolute-errors on the fixed testing set.

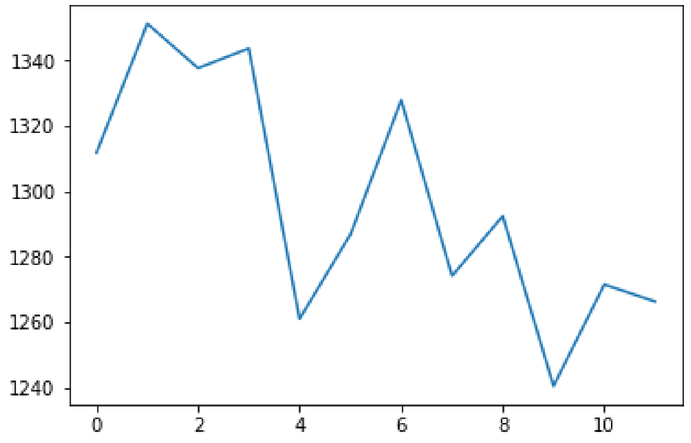

Figure 4-2. The loss curve shows the model based on the training set with numbers of equally-divided batch produces the mean-absolute-errors on the dynamic testing set.

We have to admit that even our simulation only reflects part of the challenges we may face in real world. In fact, more considerations of risk and potential problems, as well as the solutions to them, are needed before actual deployment.

Some challenges can be categorized as technical problems, which can be solved with more sophisticated design and modeling. One instance is the task of database update, which is highly simplified in our project. In reality, we must dynamically update our database and the associated model in more aspects to keep them robust and reliable. For example, the year of manufacturing is highly related to price, and if we work on a longer time period, we may need to use the time between current date the manufacturing time as a feature, and update this feature as the time flows. Another instance is the 'cold-start' problem--we don't have enough data at the beginning. One solution is to postpone the deployment of price estimation model until we acquire enough data, although this may harm user's experience at the beginning.

Some challenges, however, may not be easily controlled by us, and can cause some potential risk to our website. Think about the following two problems: First, as we discussed in the previous section, since we cannot predict the pattern of car market, how can our pre-trained model works well on the incoming but unknown data? How can our model capture the new market trend without experiencing it? Updating and retraining our model more frequently may partly solve this problem, but it can't eliminate the risk. Second, currently we use data input by our user as train set, but what if we have tons of unreliable data instances? What if 10% data in our train set are actually frauds? This could be extremely rare, but we can't exclude this possibility. Solutions like building auxiliary system that detect fraud based on sources like manufacturer price could be helpful, but still we should be aware of the potential untrustworthiness of users' data.

Appendix

Name: Zhe Huang (zh1087)

Contribution: Feature engineering, Algorithm search(random forest regressor, ridge regression, decision tree regressor), baseline model and final model construction and evaluation, writing final report.

Name: Daoyang Shan (ds5471)

Contribution: Data preparation, dynamic model improvement, deployment, writing final report.

Name: Cong Liu (cl4055)

Contribution: Exploratory data analysis, data preparation, writing final report.

Name: Yulin Shen (ys2542)

Contribution: Feature engineering, Algorithm search(KNN regressor and lasso regression), writing and arranging final report.