Car selling price model evaluation

Tianjian Wang

May 9, 2017

1 Introduction

Car dealers specialize at giving you the lowest price possible when you trade in your car, and demanding from you the highest price possible when you would like to buy from them. They want to maximize their profit, and you pay the extra for lack of understanding of the car market. This must be put to a stop! In this paper, I designed a Random Forest model to evaluate the car price based on some key factors of the car and I believe this model could be readily used for self-evaluation when selling or purchasing cars.

This problem is undoubtably a regression problem, with the results as continuous values. The dataset used to train the regressor comes from Kaggle, with over 370,000 used cars scraped from Ebay Kleinanzeigen, the Ebay Classifieds of Germany. The link to the dataset is given `https://www.kaggle.com/orgesleka/used-cars-database`. There are 20 fields in the dataset, and I used 10 most relevant fields in my regressor.

After careful consideration, I decided to predict the car's price based on these parameters:

- the nature of the car: vehicle type, model, brand, gearbox, power, fuel type
- the characteristic of the car: year, kilometer, any damage not repaired

I've planned to evaluate the regressor by the mean squared error (MSE). The definition of MSE is as below:

For the held-out test set P, with p_i as the ith real price in the set, $\hat{p}_i$ as the predicted price in the set, and a total number of k examples, the error e_P would be like this: $e_P = \sum (p_i - \hat{p}_i)^2/k$

But in the real process, since the price ranges from 0 to 2,147,483,647, the mean squared error is too large a number for me to remember and compare. So I switched to R^2 score.

The definition of R^2 score is like this: $R_P^2 = 1 - \frac{SSE}{TSS}$, where $SSE = \sum(p_i - \hat{p}_i)^2 = k \times e_P$, $TSS = \sum(p_i - \bar{p})^2$ and $\bar{p} = \frac{1}{k}\sum p_i$. We can then convert the definition into $R_P^2 = 1 - \frac{k \times e_P}{\sum(p_i - \bar{p})^2}$. Notice that for a fixed test set P, it's easy to see that TSS will remain fixed for different regressors. So if a regressor R_1 has MSE larger than another regressor R_2 on the test set P, then the R_P^2 score of R_1 will be smaller than R_2 because both TSS and k will remain the same for P and it's a monotonically decreasing transformation.

From the derivation above, we can see that R^2 score will be smaller than 1 and the larger it is, the smaller the MSE, therefore the better the regressor. So in reality I'll be using the R^2 score as the evaluation metric. There also exists another reason. I would like to see the effect of centering the price distribution. It is often achieved by making a log transformation. Then the mean square error would also be affected by the transformation, and it's hard to compare between different models. But as R^2 score is a ratio, it scales well and is consistent. Therefore this evaluating procedure will be the same for each and every model I'll be using.

But while choosing the regressor criterion of different regressors, I will still choose mean squared error function because I want to penalize more heavily the predicted results that have a lot of difference from the label value. So the bigger the difference, the much bigger it will be after the square, which makes this criterion ideal.

2 Datasets and Inputs

There are a total of 20 fields for the dataset, some are discrete features and some are continuous.

The discrete features include:

- dateCrawled : when this ad was first crawled, all field-values are taken from this date
- name : "name" of the car, including the brand and sometimes the sequence number
- seller : either private or dealer
- offerType : either offer or request
- abtest : only two values 'test' and 'control', meaning unclear
- vehicleType : one of the total eight categories

- gearbox : either manual or automatic
- model : the car's model
- fuelType : one of the total seven fuel categories
- brand: the car's brand
- notRepairedDamage : if the car has a damage which is not repaired yet, either 'yes' or 'no'
- postalCode : indicating the region the car is from

The continuous features include:

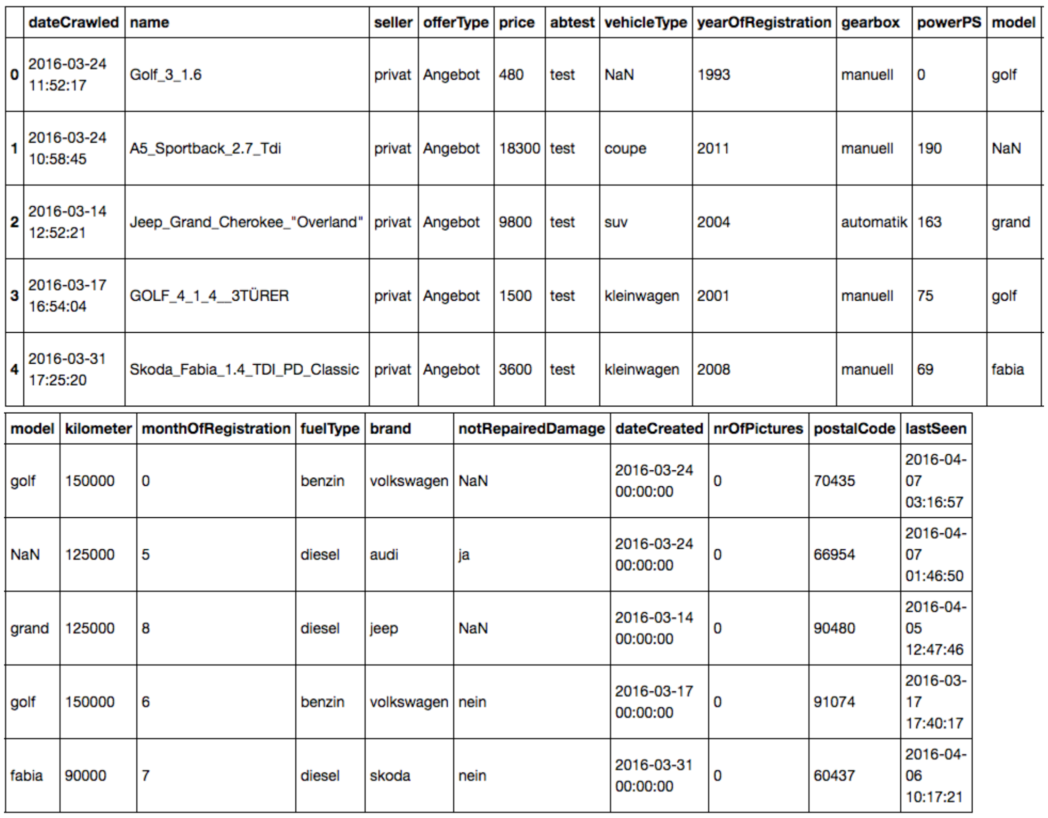
- price : the price on the ad to sell the car
- yearOfRegistration : at which year the car was first registered
- powerPS : power of the car in PS
- kilometer : how many kilometers the car has driven
- monthOfRegistration : at which month the car was first registered
- dateCreated : the date for which the ad at ebay was created
- nrOfPictures : number of pictures in the ad
- lastSeenOnline : when the crawler saw this ad last online

It's interesting to see that some of the features, although in themselves does not make much sense, could be combined together to produce valuable information. For example, 'lastSeenOnline' and 'dateCreated' could be combined to estimate the time it takes for the car to be sold. The first five rows of the database is shown as figure 1.

2.1 Basic Analysis of Dataset

After a brief look through the data, it seems that "nrOfPictures" in the first five rows are all 0. Calculating the sum of that column in the whole dataset also gives me 0. Therefore that column has no values other than 0. So that column is dropped.

The column "seller" has only 3 values of "gewerblich", and all others are "privat". So after deleting the rows of "gewerblich", the column will only have value "privat", and could



	dateCrawled	name	seller	offerType	price	abtest	vehicleType	yearOfRegistration	gearbox	powerPS	model
0	2016-03-24 11:52:17	Golf_3_1.6	privat	Angebot	480	test	NaN	1993	manuell	0	golf
1	2016-03-24 10:58:45	A5_Sportback_2.7_Tdi	privat	Angebot	18300	test	coupe	2011	manuell	190	NaN
2	2016-03-14 12:52:21	Jeep_Grand_Cherokee_"Overland"	privat	Angebot	9800	test	suv	2004	automatik	163	grand
3	2016-03-17 16:54:04	GOLF_4_1_4__3TÜRER	privat	Angebot	1500	test	kleinwagen	2001	manuell	75	golf
4	2016-03-31 17:25:20	Skoda_Fabia_1.4_TDI_PD_Classic	privat	Angebot	3600	test	kleinwagen	2008	manuell	69	fabia

model	kilometer	monthOfRegistration	fuelType	brand	notRepairedDamage	dateCreated	nrOfPictures	postalCode	lastSeen
golf	150000	0	benzin	volkswagen	NaN	2016-03-24 00:00:00	0	70435	2016-04-07 03:16:57
NaN	125000	5	diesel	audi	ja	2016-03-24 00:00:00	0	66954	2016-04-07 01:46:50
grand	125000	8	diesel	jeep	NaN	2016-03-14 00:00:00	0	90480	2016-04-05 12:47:46
golf	150000	6	benzin	volkswagen	nein	2016-03-17 00:00:00	0	91074	2016-03-17 17:40:17
fabia	90000	7	diesel	skoda	nein	2016-03-31 00:00:00	0	60437	2016-04-06 10:17:21

Figure 1: Head rows of original data

also be dropped. The same situation occurs with the column "offerType". It has only 12 values of "Gesuch". So I deleted the outlier rows and dropped the column.

For the column "name", it has a total of 233517 values, and each sample is named randomly by separate writer of the post, therefore containing different information, so it is impossible to parse. I just dropped that column as a whole.

2.2 Analysis of Columns Values

Let's start with the reality of the data. In the column "yearOfRegistration", from figure 2 , we can see that the year of registration has the highest value of 10,000, which is, of course, unrealistic. Also, cars below the year 1950 should be antiques, not vehicles. So for that column I deleted those values below 1950 and larger than the year of the dataset, that is 2015, making the values denser, like figure 3 shows.

Then, by looking at the price of cars registered before 1985, it is shown that there are a total of 5953 cars and the mean price of those all is 71,745, much larger than the mean

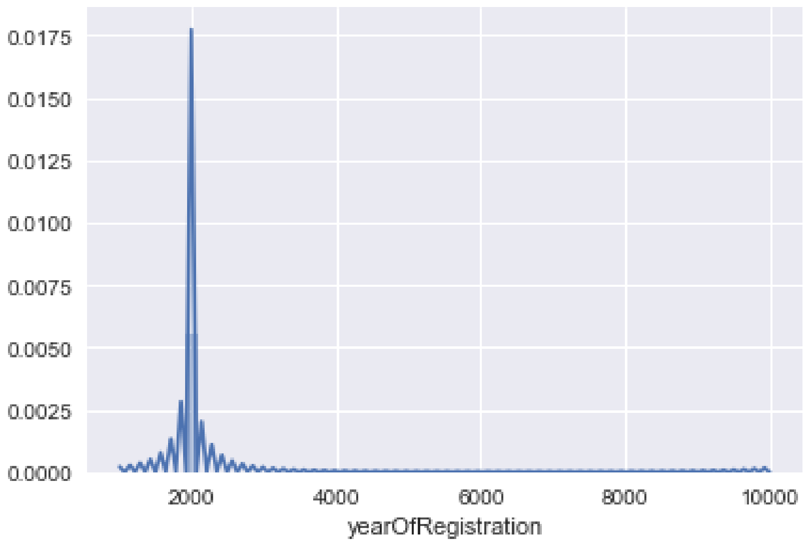


Figure 2: Orginal year data

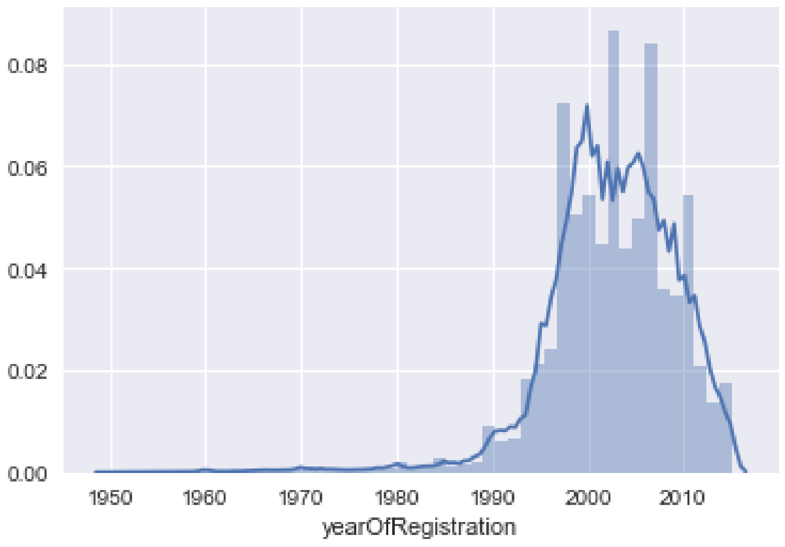


Figure 3: Transformed year data

price for cars after year 1985, which is 16,816. This shows the older cars are much more valued by their year rather then anything else, which is certainly not good for training the regressor. So I deleted the values before 1985, too. Figure 4 shows the relationship between yearOfRegistration and price after preprocessing.

The reality consideration is also applied on column ”monthOfRegistration”, where I deleted all those months that are 0, and on columns ”vehicleType”, ”gearbox”, ”model”,

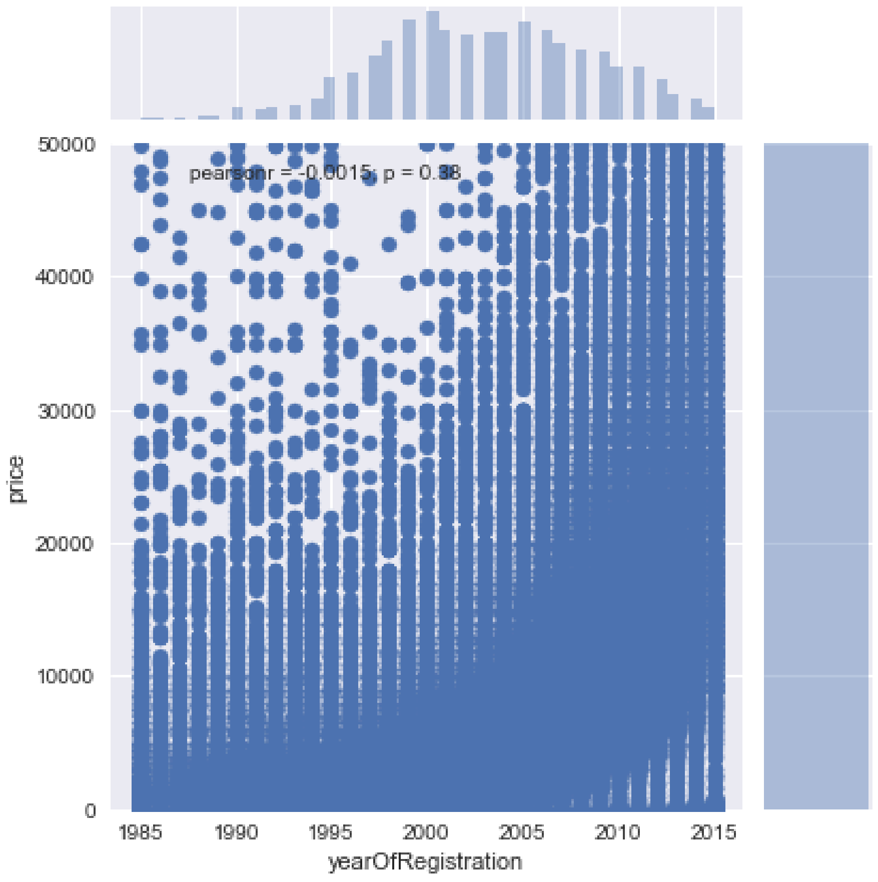


Figure 4: Preprocessed year data

"fuelType", where I deleted all those "Nan"s.

Then I deleted the rows with one field having rarely seen values. Like in column "fuelType", ['cng', 'electro', 'hybrid', 'lpg', 'other'] adding together make up less than 5% of the whole dataset, so I deleted all extra values.

2.3 Algorithms and Techniques

For this regression problem with a middle-size dataset, I've looked through several regression models including Linear Regression model, KNN model, Decision Tree model, Random Forest model, SVM model and feed-forward Neural Network model.

- Linear Regression model: aims at finding a linear decision hyperplane that minimizes the squared error of the training set. It's lighting fast, and scales well on big datasets with little or no overfitting. However most of the times it could have underfitting because of the simple nature it has.

- Decision Tree model: Build the regressor by asking questions and looking for maximum information separability. It produces a tree structured regressor that creates a set of hyperplanes in the sample space. This model has the danger of overfitting the training data and could be controlled by limiting the depth of the tree and keeping more sample points within the leaves of the tree.

- KNN model: finds K sample points that have least distance to the test point and makes an average of them. This is a soothing of the sample space and the results depend on how well to calculate the distance and how to put the weights on nearby points.

- Random Forest model: Randomly sample the data and grow decision trees based on them, preserve all decision trees and decide the results based on a maximum-vote procedure. This model makes randomness out of a fixed sample set, and is good at controlling overfitting even if it's having deeper trees and less sample points within the leaves of the tree.

- SVM model: uses kernels and margins to better fit the data points to a linear or non-linear decision boundary. SVM models are trying to find points nearest to the classifier to maximize the margin of the decision hyperplane. The nearest points are called "support vector"s and have a dominating effect on the shape of the boundary. SVM models have a penalty parameter to indicate its tolerance towards outlier points.

- feed-forward Neural Network model: with the power given by a couple of hidden layers and large size dataset, can simulate any form of 'true' decision boundary which lies under the sample points. Figure 5 shows the structure of a simple Neural Network. Multi-layer feed-forward Neural Network models were hard to train in the past, but are relatively easy to train at present by using backpropagation and gradient descent algorithm. However, the training algorithm cannot guarantee a global optimal solution. The model have randomness from the initial weight set and the learning rate, and may get stuck in a not-so-good local solution, making it hard to switch to different sets. But it's quite easy to do dynamic training as we use batch training anyway.

- Extreme Gradient Boosting model: A popular model at present, sequentially grows a weak learner into a strong one. With every iteration, the tree is giving more weight to previously misclassified points. This model has the possibility of overfitting if the tree structure is too detailed, so tuning the model is a hard problem. DART booster is not tried here because we manually tried to control the overfitting by limiting the number of iteration.

These models have different underlying assumption, and are naturally suited for different kinds of problems. There should be one kind of model that is best suited for this problem,

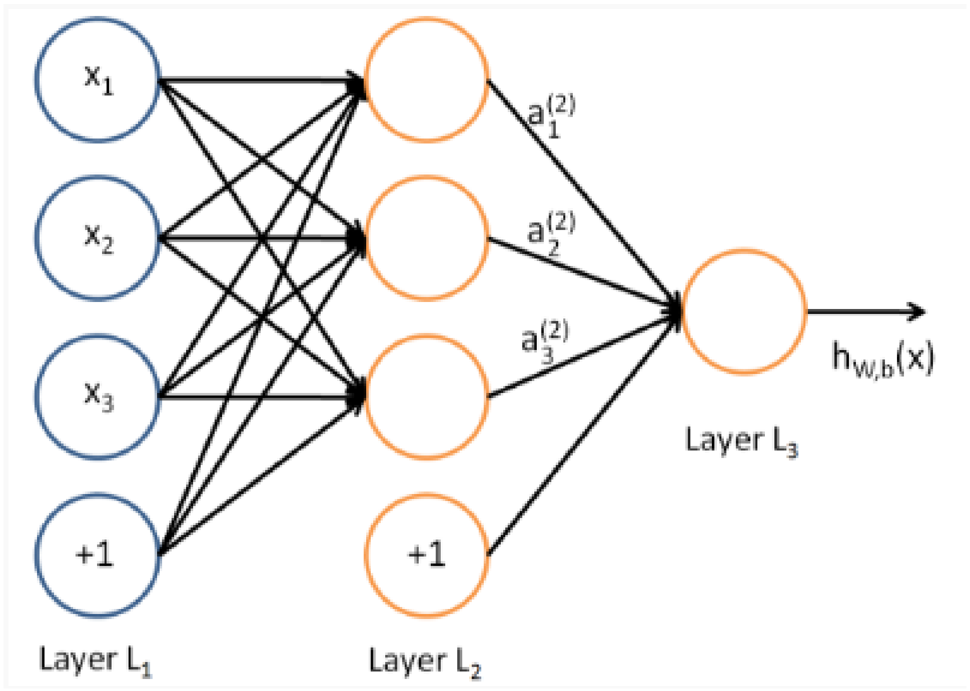


Figure 5: Structure of Neural Network

and from the analysis in section 5.1 we can see that, from the importance of categorical values in this dataset, we may guess that decision-tree structured models would achieve the best results, and in fact it did.

3 Benchmark model

Frankly speaking, this is a not so complicated problem, as experienced dealers could easily determine the approximated price of a car. They would calculate the price using the following procedure. First, from the model, brand, and year, a basic price of the car would be put up. The price may vary a little because of small features such as the gearbox is manual or automatic, and whether the car has a larger powerPS. Then the dealer would calculate the time period the car was driven, and took off the depreciation cost. It's a similar procedure for the mileage of the car, and any possible damages of the car. The dealer's "formula" for calculating car price would be as listed below:

$$P_{car} = P_{base} - C_{time} - C_{mileage} - C_{damage}$$

In here P_{car} would be the dealer's price for selling. P_{base} is the base price of the car only determined by its nature. $C_{time}, C_{mileage}, C_{damage}$ are respectively, the cost taken off base price from the perspective of time length, mileage, and damage.

It's not a linear function as we could conclude $C_{time} = K \times time$. In fact, most cars take a great decrease in price once they are bought for the first time. Similar for $C_{mileage}$, where the price often decrease dramatically when the engine is about to reach its designed mileage. But we could try to estimate the factors as linear functions as those would be the majority of cars sold.

From the analysis above, the intuitive machine learning method to use would be Linear Regression, to try and directly find a linear function that does the trick. So this will be the benchmark model to use.

The resulting R2 score achieved with this benchmark model after preprocessing the data in the way mentioned in section 4.1 is 0.6608, together with a Mean Squared Error of 1.47×10^7. The other models would be trying to continue developing on top of this benchmark model.

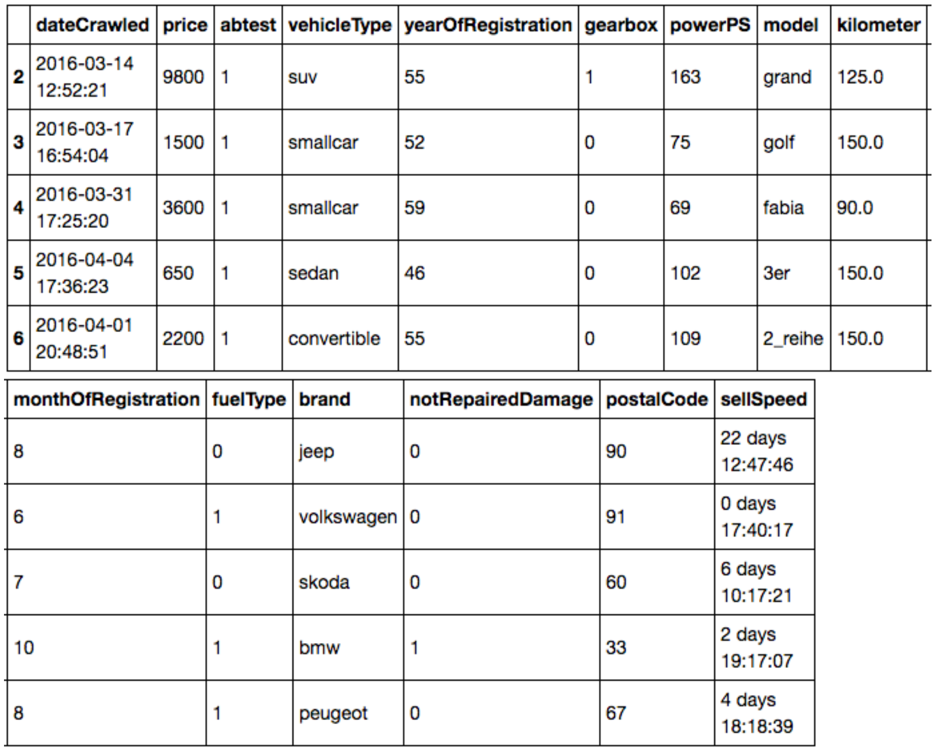
4 Methdology

4.1 Data Preprocessing

Like I mentioned in the Analysis of Columns subsection, I filtered out some unrealistic rows in the dataset. Now we are having all columns except for "nrOfPictures", "seller", "offerType", "name". For feature "postalCode", I decided to only keep the largest two bits, because in Germany, first two bits of postal Code is a region code.

As we all know, outliers could affect the performance of a regressor at a great deal. So I deleted all the outliers in continuous fields. The metric I used was "Tukey's Method for identfying outliers", by calculating the outlier step that is 1.5 times the interquartile range, and any sample with a field that is beyond an outlier step outside of the IQR for that feature is considered abnormal. This method is mainly out of intuition that data way too away from the center must be outliers and the parameter 1.5 could be changed readily. If data is heavily biased to the small or big part, it may not be so efficient, so it would be better to do a 'log' centering first in that case. However I only applied the method to columns "powerPS" and "yearOfRegistration" since those were continuous and are what I believe to be important.

For fields that already have numeric values, I did a linear transformation to avoid overflow of integer (if possible). Like I subtracted 1949 from "yearOfRegistration". I also changed those feature with only two possible values into '0' and '1'. After preprocessing, the data is like figure 6.

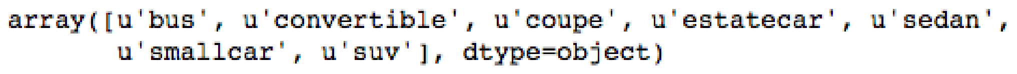


	dateCrawled	price	abtest	vehicleType	yearOfRegistration	gearbox	powerPS	model	kilometer	monthOfRegistration	fuelType	brand	notRepairedDamage	postalCode	sellSpeed
2	2016-03-14 12:52:21	9800	1	suv	55	1	163	grand	125.0	8	0	jeep	0	90	22 days 12:47:46
3	2016-03-17 16:54:04	1500	1	smallcar	52	0	75	golf	150.0	6	1	volkswagen	0	91	0 days 17:40:17
4	2016-03-31 17:25:20	3600	1	smallcar	59	0	69	fabia	90.0	7	0	skoda	0	60	6 days 10:17:21
5	2016-04-04 17:36:23	650	1	sedan	46	0	102	3er	150.0	10	1	bmw	1	33	2 days 19:17:07
6	2016-04-01 20:48:51	2200	1	convertible	55	0	109	2_reihe	150.0	8	1	peugeot	0	67	4 days 18:18:39

Figure 6: Head rows of clean data

There's only 15 columns that make sense. And all columns are either numerical data, categorical data or boolean data.

As sklearn only accepts numerical values, I encoded all categorical values into numerical values by using LabelEncoder in sklearn.preprocessing, which constructs a dictionary for different categorical values and change them into a sequence of numbers. Like in figure 7, it's the classes the column 'vehicleType' has been reduced into. So 'bus' will be encoded into 0, 'convertible' will be encoded into 1 and so on. It could also be transformed back to original categorical value if needed.



```
array([u'bus', u'convertible', u'coupe', u'estatecar', u'sedan',
       u'smallcar', u'suv'], dtype=object)
```

Figure 7: Encoder Example

4.2 Data Regression

After the preprocessing, we have a vector 'price' of length $249,371$, and a matrix 'data' of size $(249,371,12)$, which are all floating point numbers. In order to make the estimated error of regressor as accurate as possible, I used cross-validation on all models. The dataset is divided into 10 folds and I left one fold out each time for testing and use the other 9 folds to train.

- Linear Regression model. It's a simple model and do not have many structure variations. So for this model I tried switching different forms of input. I used both the original data, the data after principle component analysis, and the data after Standard Scaling, and both the original price and the price after centering by log function.
- Decision Tree model. Decision trees have two very important parameters, max_depth and min_samples_split. "max_depth" determines the maximum depth or height the tree would reach. "min_samples_split" shows at the leaf node at least how many samples should be kept. It's default parameters are max_depth = None and min_samples_split = 2. I did brute-force search to find the best parameters for this model. I also tested the model performance for data transformed by principle component analysis and centered prices.
- K-Nearest-Neighbor model. I tried from $K = 1$ to 11 nearest neighbors, and average the K nearest neighbor prices for the final result.
- Random Forest model. It's a more complicated model, so I did brute-force search to find best parameters on max_depth, min_samples_split and n_estimators. As in structure the data which gives the best performance in Decision Tree model would also give the best performance in Random Forest model, the type of data and label I used was the same to data and label which gives out the best performance in Decision Tree model.
- Support Vector Machine model. It's a more normalized version of linear regression, when using linear kernel. So I used the type of data and label which gives out the best performance in Linear Regression model. Then I tried different penalty values and different kernel functions.
- Feed-Forward Neural Network model. Neural Networks are somewhat more unstable and requires carefully chosen structural parameters. Therefore I tried different number of hidden layers, different number of nodes in layers, different learning rate, and different activation methods.
- Extreme Gradient Boosting model. Like any other tree-structured models, we tried exhaustive search for max_depth and min_samples_split, except here min_samples_split

is called min_child_weight. We also tuned the parameter 'eta' which acts similarly to learning rate in neural networks.

4.3 Model Refinement

In the beginning, I focused mostly on the accuracy of the data, and deleted only values without reality, like a year after 2016 or a car powerPS more than 1000. After the pre-processing, with the whole range of prices, I trained a basic Decision Tree Regressor and a Linear Regressor, and they gave out a score of -16.57 and 0.0108, respectively. I tried to use PCA for Linear Regressor but only achieved worse results.

However when I limited the price to a maximum of 75000, and a minimum of 100, the rows of data decreased from $252,951$ to $249,303$, taking away less than 1.5% of the data, and the scores on the Decision Tree Regressor increased tremendously to 0.8470, and the scores on Linear Regressor increased to 0.6609. So that's a significant improvement by removing outliers from labels.

The score of Linear Regression model is largely depending on the structure of data since it's a so simple model. It achieves best result when both the data and the label are centered.

With the use of brute-force search on Decision Tree Regressor's parameters max_depth and min_samples_split, the best I could achieve is with max_depth = 18 and min_samples_split = 28.

I used the Euclidean distance as the distance metric to use, and after brute force search on a range of K's values, it turned out that $K = 5$ really gives out the best performance for K Nearest Neighbor Regressor. Since some of its columns are categorical values, using 'distance' rather than 'uniform' as weight parameter would be a better fit for the nature of the problem.

Random Forest models takes up much more time to compute than simple ones. As the number of estimators it creates gets larger, the resulting score gets larger at a slow rate, and the time it takes to train increases almost linearly. So I only picked a reasonable number of estimators. The optimal tree structural parameters is different from Decision Tree Regressor, with max_depth=27 and min_samples_split = 7. It is quite interesting to see that random forest models are accepting much more complicated trees than decision trees, and is thus much more powerful while the random shuffling of data made it robust to overfitting.

Support Vector Machine models takes up so much time to train if not using linear kernels, as its training time complexity is between $O(n^2)$ and $O(n^3)$, and I have a somewhat large dataset. So I have to terminate early, by setting the max_iter parameter. After trying through all the types of kernels, linear kernel gives out only a little better than Linear

Regressor. The best performance is given by rbf kernels which maps the feature space into a infinite one, using already centered data and label.

Neural Network models are more like a blackbox, any random change of hidden layer size, initialization weight, batch size, or learning rate could make dramatic change on complex problems. But as it takes so much time to train, I can only try some possible parameter sets, and this may not be the best performance this model could achieve. The best current results are given by a neural network with two hidden layers, both containing 200 nodes, with a tanh unit activation function, initial learning rate of 0.01 and the early stopping technique.

Extreme Gradient Boosting models are also quite like a blackbox, and in the reality showed sensitivity to the size of the dataset. A brief search showed the best results are given by a tree with maximum depth of 9 and a minimum number of 5 samples to perform a split. In each boosting a step size shrinkage of 0.3 is used in updating.

5 Results

For each model the best result is shown in table 1, where the first column is different models used, the second column denotes the R2 score, the third column shows the mean-squared error and the fourth model shows the mean absolute error for the corresponding models.

models	R2 score	MSE	MAE
Benchmark model	0.6608	1.46×10^7	2568
Linear Regression	0.7568	1.05×10^7	1781
Decision Tree	0.8839	5.02×10^6	1203
K-Nearest Neighbor	0.8336	7.20×10^6	1376
Random Forest	0.9111	3.56×10^6	1008
SVM(linear kernel)	0.7648	1.01×10^7	1728
SVM(rbf kernel)	0.8706	5.59×10^6	1288
Neural Network	0.8566	6.19×10^6	1334
Extreme Gradient Boosting	0.9200	3.46×10^6	1031

Table 1: Results of different models

The benchmark model, which is just linear regression performed on original data, gives out the worst results. By far, Extreme Gradient Boosting models are giving out the best individual results. Table 1 showed the results for 10-fold cross validation, where we just randomly choose 10% of the whole dataset as test set, and use the other 90% as training set. We repeat this for 10 times so that all the rows have been in both training set and test set.

Then we try to estimate the effect that the randomness of the data split has on the

models	10-Fold	First (0)	Second (5)	Third (10)
Linear Regression	0.7568	0.7623	0.7549	0.7531
Decision Tree	0.8839	0.8721	0.8719	0.8724
K-Nearest Neighbor	0.8336	0.8100	0.8132	0.8123
Random Forest	0.9111	0.9112	0.9112	0.9087
SVM(rbf kernel)	0.8706	0.8684	0.8673	0.8652
Neural Network	0.8566	0.8536	0.8403	0.8541
Extreme Gradient Boosting	0.9200	0.8861	0.8847	0.8810

Table 2: Results of Random Data Split

models. We split the data into 60% training set and 40% test set, and tried three different random split of data. This can also show the ability of the models to capture the underlying distribution, and their scalability, as the dataset is 33% smaller. The results are shown in table 2. The second column are the resulting R2 scores from 10-fold cross validation, same as the table 1. The third to fifth columns are the R2 scores on different data splits with random_state = 0, 5, 10, respectively.

From table 2, we can see that several models are doing worse than when using 10-fold cross validation. It's mainly because they are having more than 30% less data to train with. The model that is most affected would be Extreme Gradient Boosting model where the tree structure is highly sensitive to the size of the dataset, and with less training data should have a simpler structure, showing it is unable to scale well with the specific structures. K-Nearest Neighbor model is performing worse as it relies heavily on the sparseness of the sample space. Linear Regression, Decision Tree, Random Forest, SVM, and Neural Network models are remaining almost the same number of R2 score, which indicates they have good scalability and robustness. Therefore, taking scalability into account, we can see that the best performance model is not Extreme Gradient Boosting but Random Forest model.

5.1 Result Analysis

The biggest problem for this project is the categorical values. We have a total amount of 8 vehicle types, 39 different brands, and 248 different models. Those data cannot be easily converted to numerical values. There's a way to extend features and make dummy columns, but there's simply so many different values to extend on. If we extend the feature space into about 300 columns, the data points would be too sparse and there's not a good way to do dimensionality reduction on that.

So my decision was to use integers to act as classes. The first seen feature would be assigned 1 for that column, the second would be assigned 2 and so on. This transformation is adding new information into the relation of features, because it gives models an impression that a point with brand 1 and brand 2 are more similar than a point with brand 1 and brand 39, but that's not the case, the categories are more like orthogonal to each other. I believe that's why decision-tree structured models are giving out best individual model results.

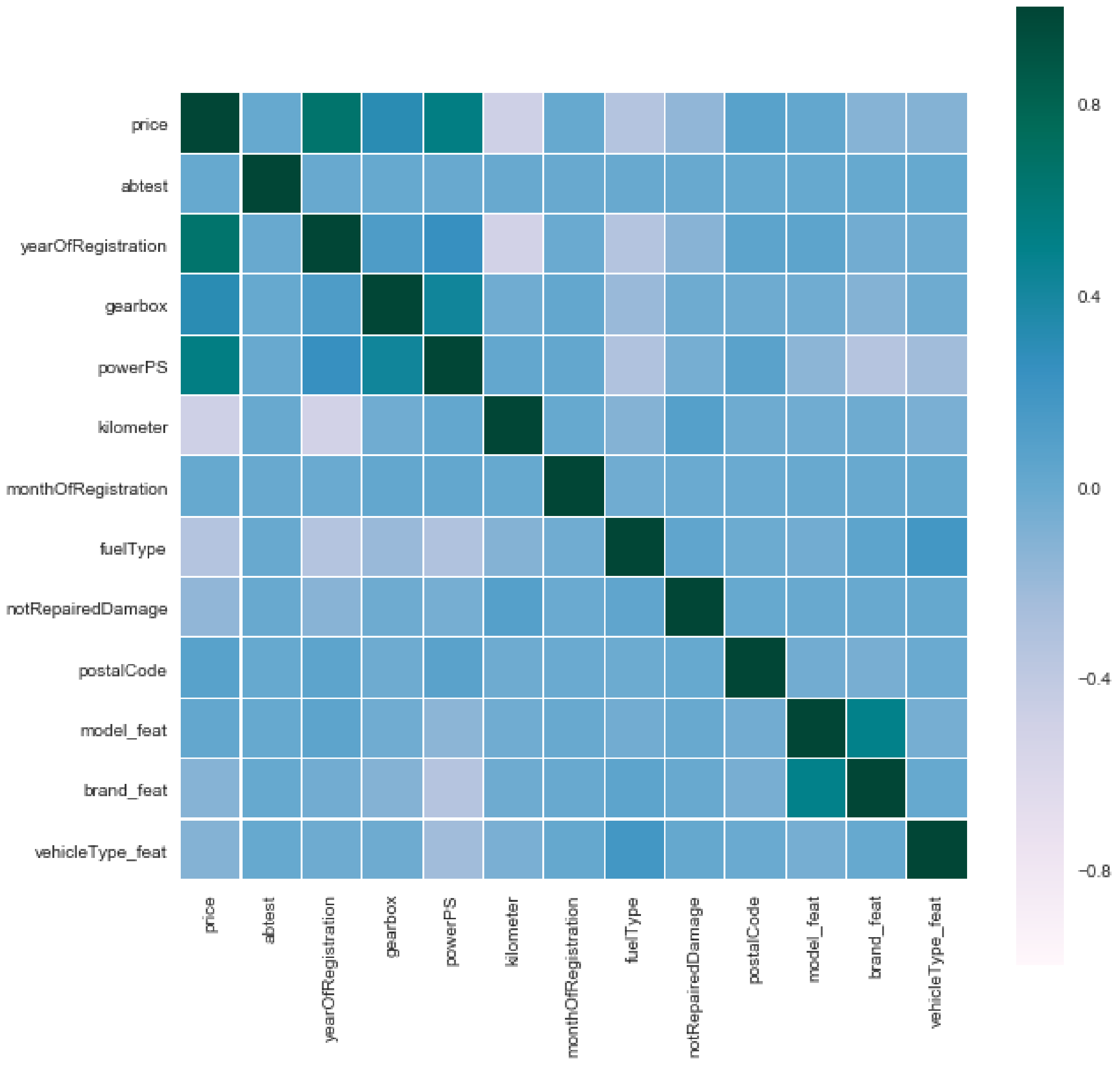


Figure 8: Feature Correlation

In the correlation figure 8, with the color darker for more positive correlations, and color lighter for more negative correlations, we can see 'yearOfRegistration' and 'powerPS' are having a very strong correlation with price on its own, and 'kilometer' is having a very strong negative correlation with the price, which is quite understandable. Also, 'kilometer' and 'yearOfRegistration' have a very strong negative correlation within themselves, meaning if a car is very young it's quite impossible for it to have driven a lot of miles.

It's also easily seen that although model and brand columns are transformed into numerical values, they have strong correlation with each other, showing the information is reserved quite well.

In Linear Regression model, 'yearOfRegistration' is a key feature for price prediction, with the largest positive coefficient. 'fuelType' and 'notRepairedDamage' are two key negative influencing features. So that's why it's not giving out very good results. Decision Tree model place more influence on 'yearOfRegistration', 'powerPS' and 'kilometers', which are the most influencing individual features. And it is more able to generate information from 'model', 'brand' and 'vehicleType', so it's a much better model.

In more advanced models like SVM and neural networks, it's more difficult to have impression on the model, but they are somehow doing feature transformation first and they are giving out good results because they can find the true features during training and make better use of them. However, both SVM and Neural Networks require more preprocessing of the data, namely standardize and centerizing the values.

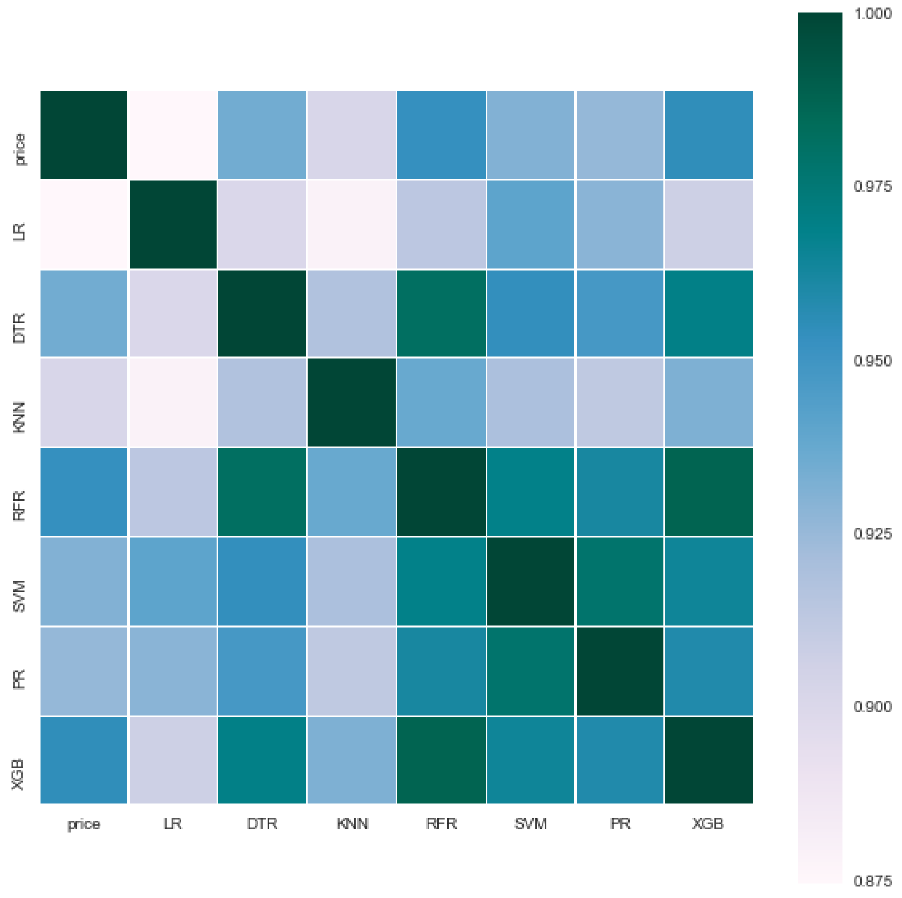
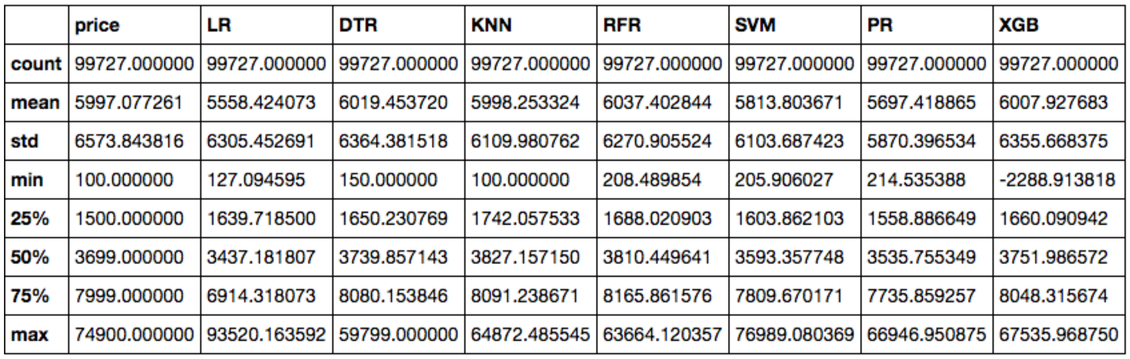


Figure 9: Predicted Value Correlation

Figure 9 is an correlation between the predicted price and the real price.

All these models are doing a reasonable job in correlating with the real price, with the worst one Linear Regression model have a correlation of more than 0.85. In here, we can see which models have similar assumptions. Linear Regression have correlation with SVM and Neural Network, which is reasonable because they perform a kernel transformation first. For Decision Tree models, we can see it correlates quite heavily with Random Forest models and Extreme Gradient Boosting models, as they have the same underlying assumption and structure. Random Forest models perform better because it adds randomness to the model and therefore decreases overfitting. KNN do not have much correlation with any other method, and especially less with Linear Regression model because they follow totally different assumptions. It's interesting to see Random Forest model has some correlation with SVM model, which maybe because they are both on the right track.

Here's a description of the predicted values of all models, with left to right being Linear Regression model, Decision Tree model, K-Nearest Neighbor model, Random Forest model, Support Vector Machine model, Neural Network model, and Extreme Gradient Boosting model.



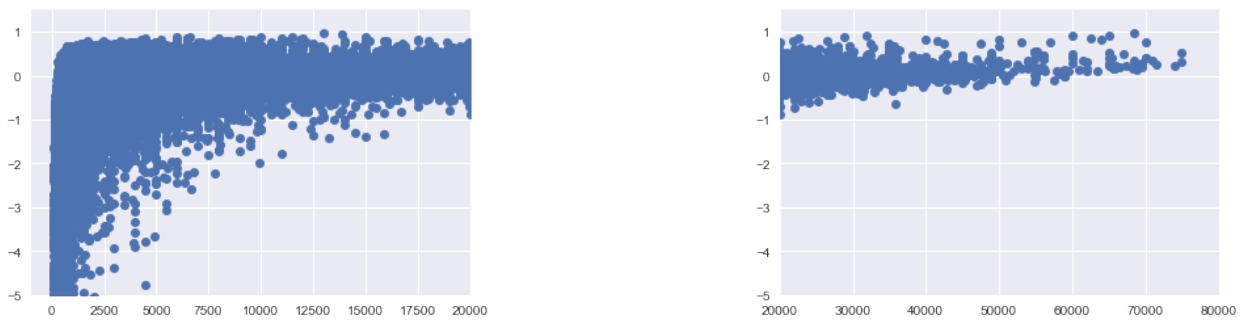
	price	LR	DTR	KNN	RFR	SVM	PR	XGB
count	99727.000000	99727.000000	99727.000000	99727.000000	99727.000000	99727.000000	99727.000000	99727.000000
mean	5997.077261	5558.424073	6019.453720	5998.253324	6037.402844	5813.803671	5697.418865	6007.927683
std	6573.843816	6305.452691	6364.381518	6109.980762	6270.905524	6103.687423	5870.396534	6355.668375
min	100.000000	127.094595	150.000000	100.000000	208.489854	205.906027	214.535388	-2288.913818
25%	1500.000000	1639.718500	1650.230769	1742.057533	1688.020903	1603.862103	1558.886649	1660.090942
50%	3699.000000	3437.181807	3739.857143	3827.157150	3810.449641	3593.357748	3535.755349	3751.986572
75%	7999.000000	6914.318073	8080.153846	8091.238671	8165.861576	7809.670171	7735.859257	8048.315674
max	74900.000000	93520.163592	59799.000000	64872.485545	63664.120357	76989.080369	66946.950875	67535.968750

Figure 10: Brief Description of Predicted Values

We could see from the figure 10, most models tend to centerize the data, for the maximum of all the predicting models are smaller than the real price except for Linear Regression model and SVM model. Let's pick the universally best performance model, Random Forest model for example and see the difference it has with the corresponding prices.

In figure 11, where the X-axis is the price of cars and the Y-axis is $1 - predictedPrice/price$, which is the error ratio of predicted price, we can clearly see that the most error comes from the cheapest cars. Let's consider the car with the worst error, which did not appear in the figure.

The car with the worst error has all original its parameters shown in table 3. We can see it's a 2013 Volkswagen Transporter with only $30,000$ miles. This was only sold for 220, which is very weird. My predicted price would appear more reasonable, as I searched for



(a) Cheaper Cars.

(b) More Expensive Cars.

Figure 11: Ratio of Random Forest Model

used car in Germany and found the majority of 2014 Volkswagen Transporters are about 30,000.

abtest	yearOfRegistration	gearbox	powerPS	kilometer
control	2013	manual	143	30000
monthOfRegistration	fuelType	notRepairedDamage	postalCode	brand
11	diesel	no	95	volkswagen
model	vehicleType	price	predictedPrice	
transporter	bus	220	31418	

Table 3: Worst Car Predicted

Therefore the model could predict some useful results, and could be applied to self evaluate used cars. Also, it maybe used to help companies identify suspicious selling prices, and could improve the safety of transactions.

6 Future Improvement

If categorical values are handled more precisely, perhaps we can get better results. For example, we can use the brand, model and vehicle type of the car to get the original selling price of the car when it first appeared on the market, and delete the three columns. In this way the model is used to predict the difference by the usage of the car.

The prediction of price on used cars is not only depended on the parameters of the cars themselves, but also on the circumstance of sellers. In the worst case above in table 3, it maybe that the seller is in desperate of money, and would accept a lowest price fast (which is still quite unreasonable). This could not be spotted anyhow by the model, thus increased random noise in the data. So even a most precise model would still have its limits.