Kimberley's Data Mining Report on Resale Cars

Cai Runze
e0505538@u.nus.edu

Zhang Junzhe
e0792460@u.nus.edu

Zhang Shuo
e0724275@u.nus.edu

Huang Wenzheng
e0792455@u.nus.edu

***Abstract*—In this project, we accomplish three data mining tasks including Prediction of Car Resale Prices, Car Recommendation, and Prediction of Actual Depreciation Rate of Used Cars. The goal of our project is to give valuable information on data patterns for both car resale companies and the users who want to buy or sell cars. To achieve this goal, we apply data mining skills including Exploratory Data & Preprocessing (EDA), model selection, and result evaluation and interpretation. Our goal is achieved and get a good result by applying the data mining methods we propose in this report.**

Keywords—Data Mining, EDA, Tree-Based Model, Deep Learning, K-Fold

I. Motivation

In the real car business, car resale prices and car recommendations play a significant role in a good sale. Setting the car resale price to a proper position can improve profit and enlarge the group of customers at the same time. So does a good car recommender. So given the data related to these two aspects, our goal is to build models as a good regressor and recommender for the two tasks respectively. To achieve the goal, we focus on data cleaning and preprocessing, feature selection, and optimizing our regression and recommendation model. And we handle the data with EDA as well as our common sense. And we tend to solve the problems with tree-based models and deep learning models for the resale price regression because of their good ability to explore the relationship between data and generalization. As for the recommendation task, we design two tracks to solve the problem. One is based on pairwise item-item similarity and the other is based on user-item similarity, which both are very explainable.

In addition to the price regression and car recommendation, we also explore the depreciation rate of used cars to tell buyers the longtime values of those cars if they want to sell their cars in the future. This sub-task can be seen as a regression task. Highly explainable tree-based methods are leveraged to do the prediction task. The idea is very realistic and can satisfy customers to a considerable extent.

II. Exploratory Data Analysis & Preprocessing

We apply EDA to get basic insights into the data pattern and we clean the data based on our understanding of the data. We also select different features based on different targets of different sub-tasks. The details of EDA and data preprocessing for the three different sub-tasks are given as follows.

A. Prediction of Car Resale Prices

We do some initial investigations on data to find some data patterns, data quality, etc. In this part, we will illustrate the results of data analysis and the way to preprocess. And we will discuss categorical and numerical data separately.

1) Categorical Data

a) Title and Make: We find that there are missing values in *make* and the first part of the *title* is the *make* of the car. So we extract the first part of the *title* and we leverage the values to fill in the missing values in *make* and convert them to lowercase. For example, for the car with *listing_id* 1021510, the *make* of this car is missing. However, we can fill the missing value with the *title* of "Toyota Hiace 3.0M". We extract "Toyota" from the *title*, covert it into lowercase, and use it to fill the missing data of *make*.

b) Eco_category: The *eco_category* of all vehicles is a constant value, so we delete this feature.

c) Category: We separate the *category* with commas and only keep the first part based on the finding that different listing cars have different numbers of categories and the first category may be the most important category. Then we replace the first item of category with the original data.

d) Model: Each brand of car has a large number of different models. If we keep the feature, it will result in a sparse matrix of features after we use the one-hot encoding on the categorical data. Therefore, we delete this feature to reduce the dimension of the final data.

e) Categorical Data with Many N.A. values: The *opc_scheme* and *fuel_type* have too many missing values (above 75% of the data), so these two features are deleted.

f) Other Text Features: description, features, accessories are unstructured long text features and it is hard to retrieve the important information from these data and convert the information into numerical features. To reduce the complexity of this sub-task, we delete these features.

2) Numerical Data

Firstly, we draw the correlation heatmap (Fig. 1) of the dataset to analyze the numerical features.

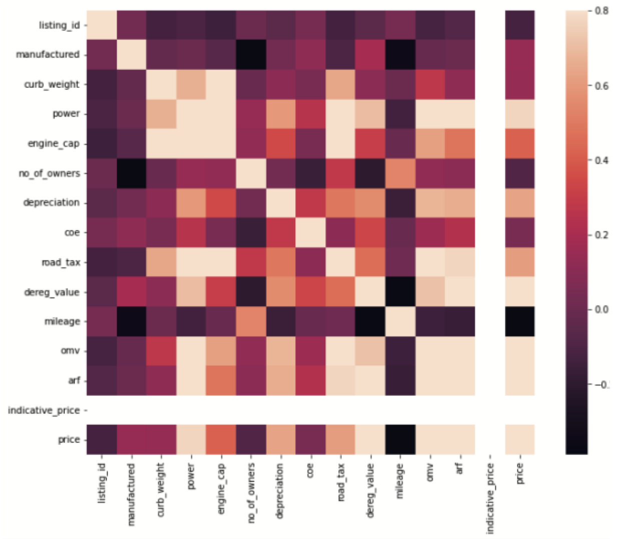


Fig. 1. Heatmap of correlation between original features

a) no_of_owners: The heatmap in Fig. 1 shows that the correlation between *no_of_owners* and *price* is close to zero. And the relationship between *no_of_owners* and *price* shown in Fig. 2 also tells us that the *price* ranges for different *no_of_owners* are all very large. Therefore, we delete the *no_of_owners*.

b) Manufactured: We analyze the histogram of the manufacturing year and find that the largest manufacturing year is 2925, so we delete all data with a manufacturing year greater than 2021.

c) Registration Date: The *Registration Date* is converted into the standard date format, and only the *year* value is selected because we consider the month and day could only have a slight influence on the prediction.

d) Listing Id: Listing Id keeps the identity of each listing car, but this feature is not useful in the prediction of car resale prices. So, this feature is deleted as well.

e) Other Numerical Features with Many N.A. values: we find that all *indications_price* are empty. Besides, *original_reg_date* and *fuel_type* have too many missing values (above 90% of the data). Therefore, these three features are deleted.

Meanwhile, for all numerical features in the training dataset and test dataset, we fill in the mean value of this feature for each N.A. value and do normalization on them to help us train the regression model better.

Finally, we perform one-hot encoding on all categorical data, which will bring about the problem of high feature dimensionality and sparseness. However, there is no alternative solution to this problem. The factorization encoding converts all categorical data into categorical numbers, which will be meaningless if we input these numbers into the regression models. Hence, the trade-off solution is still using the one-hot encoding.

In conclusion, after analyzing the data, we decide to keep the features, including *make*, *manufactured*, *reg_date*, *type_of_vehicle*, *category*, *transmission*, *curb_weight*, *power*, *engine_cap*, *depreciation*, *coe*, *road_tax*, *dereg_value*, *mileage*, *omv*, *arf*.

And the deleted features are *listing_id*, *title*, *indication_price*, *model*, *description*, *features*, *accessories*, *original_reg_date*, *fuel_type*, *no_of_owners*, *opc_scheme*, *lifespan*, *eco_category*.

After preprocessing the data, we draw the final heat map as shown in Fig. 3 The light color represents the positive correlation between the two features. From Fig. 3, we can see

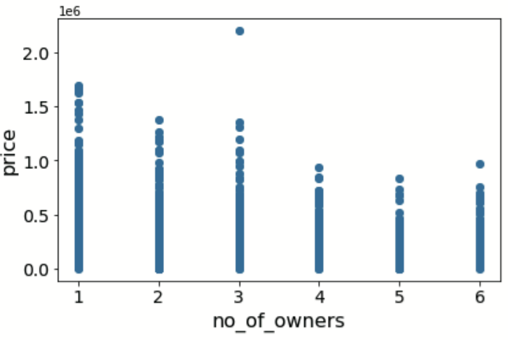


Fig. 2. Relationship between *no_of_owners* and *price*

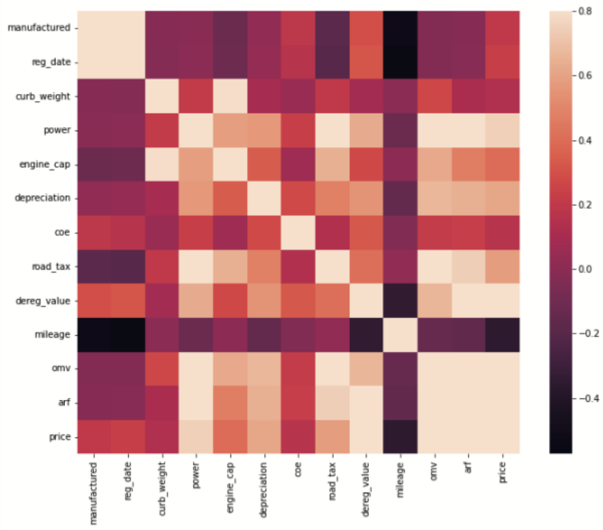


Fig. 3. Heatmap of the correlation between preprocessed features

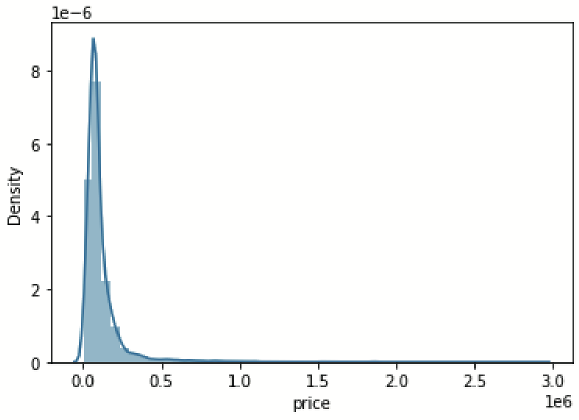


Fig. 4. Distribution of *price*

that the attributes that have an obvious positive correlation with price are *power*, *dereg_value*, *omv*, *arf*. In other words, the higher the values of these attributes, the higher the price of the used car. On the contrary, we can also see that the *mileage* has an obvious negative correlation with price.

In addition to analyzing the features above, we also analyze the price distribution and draw the curve (Fig. 4). We find that some listing cars are much more expensive than other cars. This distribution would have a bad effect on our prediction because we use the Root Mean Squared Error (RMSE) as an evaluation indicator, and the precision of prediction is more affected by high-priced cars, while the number of high-priced cars is very small.

B. *Car Recommendation*

In the scenario of car recommendation, we suppose that the users are viewing the website of resale car and they want to get a recommendation from what they like, or they can give scores to the viewed cars. With this idea, we hope to present complete, understandable, and unmodified data to users.

In the part of features selections, we apply some strategies used in the EDA of sub-task 1. We delete the features with only a few valid values and the features in unstructured text format.

Compared to what we apply in EDA and data preprocessing of sub-task 1, we deal with the N.A. data in selected features by dropping them directly instead of filling the N.A. values with the mean value of certain features because of the goal of presenting complete data to users. Using the mean value of certain features to fill the N.A. data is not a good approach here because the modified data would interfere with the users' judgment. And we also do not normalize the data in the EDA part because of the goal of presenting understandable data to the users. We leave the normalization work in the latter part after the users select their favorite listing cars or give scores to some listing cars.

C. *Prediction of Actual Depreciation Rate of Used Cars*

In this task, we want to give suggestions to the users who want to buy a new car and plan to sell their cars in the future by helping them estimate the actual depreciation rate of used cars. To regress the depreciation rate of the used cars, the target value is set as *(omv + coe + arf – price) / (omv + coe + arf)*. We notice that there is a feature named *depreciation* in the dataset. However, this feature means *(resale price – deregistration value) / left year of COE,* which is different from our goal. In this sub-task, we select *make, manufactured, mileage, power,* and *reg_date* as regression causing factors based on our experience of the most important factors to influence the depreciation rate of cars. And we only select the cars with *no_of_owners =1* because the depreciation rate here is only meaningful for buying a new car and then selling it. After selecting the features, we drop the data with N.A. values since the number of our selected features is small and the reality of the data is important to guarantee the regression quality. We transform *reg_date* into numerical values and then do normalization for each numerical data to make them suitable for the regression task. As for the only string feature *make,* we use the one-hot encoding for it.

III. Data Mining Methods

We leverage different models in different sub-tasks based on the specific target and the need for interpretation of the result. We apply three major types of models in total, including the Tree-Based Model, Deep Learning Model, and Ensemble Model. The details of these data mining methods are given as follows.

A. *Prediction of Car Resale Prices*

This sub-task is a regression task, thus we build several regression models to solve the task. In the EDA part, we have shown a rough relationship between different features and the final resale prices. In this section, we build the regression models as follows to combine different features and get a good fitting of the resale prices.

1) *Tree-Based Model:*

Given the information about the used car and the target to regress the resale prices of cars, the first type of model we leverage is the Tree-Based Model. This type of model has a good interpretation for the result and it is easy to apply.

a) Decision Tree Regressor: The Decision Tree Regressor is the first model we apply in this sub-task because the model is easy to leverage and the training speed of this model is fast. We import the Decision Tree Regressor from sklearn and use the Grid Search to find the best hyperparameters. The parameter tuned here is the *max_depth.* We tune this parameter because we not only want to increase the value to get a good fitting but also need to ensure it is not so large that leads to overfitting. After the Grid Search, we select *max_depth = 60* and keep other parameters as the default settings to avoid unnecessary complexity in the parameter searching.

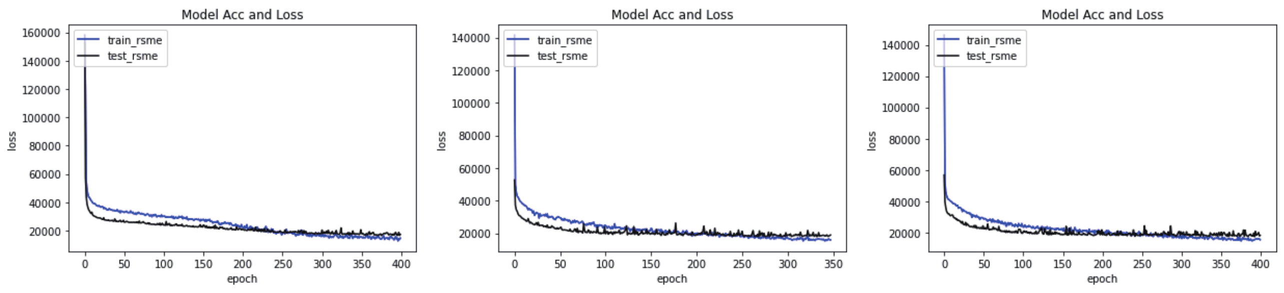
b) Random Forest Regressor: The Random Forest Regressor is the second model we apply in this sub-task because compared to the Decision Tree Regressor, this model usually has higher accuracy. We import the *Random Forest Regressor* from sklearn and use the Grid Search to find the best hyperparameters. The parameters tuned here are the *n_estimators* and *max_depth.* A higher value for the two parameters can fit the pattern of data better but a large value for the two parameters could also lead to overfitting. After Grid Search, we select *n_estimators = 200* and *max_depth = 20*. We keep other parameters as the default settings to avoid unnecessary complexity in the parameter searching as well. There are some limitations of applying the model, including less interpretation and a slow training speed. But the RMSE score of the Random Forest Regressor is reduced by 18.7% compared to the Decision Tree Regressor.

c) Gradient Boosted Tree: The Gradient Boosted Tree is the third model we leveraged in this sub-task. Compared to the Random Forest Regressor, this model usually has higher accuracy. We import the Gradient Boosting Regressor from sklearn and use the Grid Search to find the best hyperparameters. The parameters tuned here are the *n_estimators* and *learning_rate.* A higher value for the *n_estimators* can fit the pattern of data better but a large value could also lead to overfitting. The *learning_rate* also is important to build a good Gradient Boosted Tree with less error. After Grid Search, we select *n_estimators = 400* and *learning_rate = 0.15*. We keep other parameters as the default settings to avoid unnecessary complexity in the parameter searching as well. There are some limitations of applying the model, including less interpretation than the Decision Tree and a slow training speed. But the RMSE score of the Gradient Boosted Tree is reduced by 1.3% compared to the Random Forest Regressor.

2) *Deep Learning Model:*

we leverage one-hot encoding in the data preprocessing part and that results in a high feature dimensionality, i.e. 123 dimensions. It is not good to feed the data with such high dimensionality to the Tree-Based Model because it influences the height and complexity of the tree. To solve the problem, we apply the Deep Learning Model.

a) Baseline Model: The Baseline Model is a simple Multi-layer Perceptron [1]. This model is built on Keras and contains 6 *Dense* layers. To avoid overfitting, the *Dropout* layers [2] and *l2* regularization are leveraged. And *Random Normal* [3] is used to initialize the weights of hidden layers. We choose the *relu* and *selu* [4] as the layers' activation. To train the model appropriately, we try to use Stochastic Gradient Descent (SGD) [5], Adam [6], and RMSprop [7] to be the optimizer. And after several trials, we find that the RMSprop achieves the highest performance among the three optimizers. Besides, we also use the Early Stop [8] method to avoid overfitting. One of the problems of using the deep learning model is that we need to deal with a great number of hyperparameters. To find a good set of hyperparameters, we leverage the K-Fold method (5-Fold in our implementation) to test the performance of each combination of hyperparameters on different parts of data. After selecting the hyperparameters, we train the model. The RMSE of training



(a) (b) (c)

Fig. 5. The RMSE of Training Data and Validation Data on Different Models

data and validation data on the Baseline Model is shown in Fig. 5 (a).

b) Embedding Model: The baseline model could achieve a good result on the leaderboard, which could beat the classic tree-based models and proves the effectiveness of the representation from MLP and its generalization. But the basic MLP takes one-hot encodings as some of the input and this may cause the input to be too sparse and avoid the network to model the relation between input features well. So we propose to transform spare input features into dense format [9], with a simple embedding layer. The numerical features are combined as a single feature vector and each categorical feature is passed through a fully-connected layer to get a dense vector whose dimension is much smaller. Then all the dense features are combined into one as the new input for basic MLP. This can make the model "discover" the relation between input features easier. To find a good set of hyperparameters, we also leverage the K-Fold method (5-Fold in our implementation) to test the performance of each combination of hyperparameters on different parts of data. After selecting the hyperparameters, we train the model. The RMSE of training data and validation data on the Embedding Model is shown in Fig. 5 (b).

c) Wide & Deep Model: We also apply an idea from one of the state-of-the-art deep learning models called *Wide & Deep* [10], which is often seen in the industry. The idea is based on the traditional deep learning model or MLP. The deep model could gain an abstract representation for the input features and learn how to make use of the representation for the prediction. However, the knowledge inside the original input is largely forgotten by the deep model after the input is very high-dimensional and abstract. Wide & Deep's idea is to add a wide part into the whole model:

$$\textit{wide\_feature} = \textit{FC\_layer}(\textit{input}) \tag{1}$$
$$\textit{embedding\_feature} = \textit{embedding\_layer}(\textit{input}) \tag{2}$$
$$\textit{deep\_feature} = \textit{MLP}(\textit{embedding\_feature}) \tag{3}$$
$$\textit{prediction} = \textit{output}(\textit{cat}(\textit{wide\_feature}, \textit{deep\_feature})) \tag{4}$$

The wide part could be seen as the traditional logistic regression, with input passing through a simple *fc* layer and then combined with the deep feature for the final layer of output. The wide part is a memory unit that maintains knowledge from the original input. To find a good set of hyperparameters, we also leverage the K-Fold method (5-Fold in our implementation) to test the performance of each combination of hyperparameters on different parts of data. After selecting the hyperparameters, we train the model. The RMSE of training data and validation data on the Wide & Deep Model is shown in Fig. 5 (c).

3) Ensemble Model: We have trained different types of models and find that all the models that we apply above have their advantages and limitations. To overcome the shortcomings of each model and combine the advantages of different models, we design an ensemble model that returns the weighted average of the predictions of different models. We select the Random Forest Regressor, Gradient Boosted Tree, and the Deep Learning Model (Baseline Model and Embedding Model) to combine. The weights of different models are set based on the RMSE scores. The single model which has a lower RMSE score will be assigned a higher weight. If we combine the predictions of Random Forest Regressor, Gradient Boosted Tree, and the Baseline Model, The RMSE score on the test dataset is reduced by 12.2% compared to the average RMSE score of the selected models. If we combine the prediction of Random Forest Regressor, Gradient Boosted Tree, and the Embedding Model, The RMSE score on the test dataset is reduced by 12.7% compared to the average RMSE score of the selected models. The good result on test data shows the Ensemble Model has a better performance than each single model.

B. Car Recommendation

This sub-task is a recommendation task and we designed two tracks to solve this task in two different scenarios.

1) Track 1 - Using Pairwise Item-Item Similarity: We start with a simple scenario to solve the sub-task. The first track focus on the scenario that when the users are viewing the listing cars, they may find one car that satisfies their need. In this case, they hope that the website can give more recommendations of listing cars that are highly similar to the desired car.

Since the data after EDA and preprocessing are user-friendly but not appropriate for calculation, we normalize the data first. We calculate each car's cosine similarity with the desired car after the normalization. The cosine similarity scores are ranked and the cars with the top-k similarity are recommended to the users.

One of the limitations of this method is that there are too many features after using one-hot encoding for the feature of *make*. The total dimension of features before one-hot encoding is 17 (including the *listing_id*), however, the total number of dimensions after using one-hot encoding on *make* is 79 (excluding *listing_id* here). Only applying Pairwise Item-Item Similarity to solve the problem may not always get a good recommendation because only one data of car with limited features are calculated and this method cannot reflect preference degree of users to certain cars. Therefore, we need to leverage extra values to overcome the limitation.

2) Track 2 - Using User-Item Similarity:

In this track, we design a recommendation system in a more complex scenario. Because the recommendation system based on collaborative filtering requires a lot of users' ratings, we still choose the content-based recommendation system here. The difference with track 1 is that we choose user-item similarity to design the recommendation system here. We only need some features of each used car to construct the item profile, as well as the user profile. When we construct the user profile, we add a special prerequisite that users can initially filter the characteristics of the used car they want to buy. For example, in our code, the user first specifies that they want to buy a BMW with a price of less than 80,000 SGD. Then, we will randomly return 10 used cars that meet the conditions and ask users to rate them. The full score is 10.

Meanwhile, we still use the clean data after EDA and preprocessing for this sub-task. After doing normalization for numerical data and one-hot encoding for categorical data, we can construct the user profile. By calculating the cosine similarity between the user profile and the items under the specified conditions except for the 10 used cars used for rating, we can get the top-k used cars based on the similarity and recommend them to the user.

Similarly, we are still facing the problem of high feature dimensionality brought to us by one-hot encoding. We will try to solve this problem through embedding encoding in future work. And we can also try to get more user ratings to adjust the recommendation algorithm.

C. Prediction of Actual Depreciation Rate of Used Car

As for this open task, we choose to do the regression for the depreciation rate of used cars, which is very interesting and practical, leading to a more satisfied purchase experience of customers. Given the precise prediction of this attribute, customers could evaluate the long-term value of those cars in advance and would not be regretful after they buy the excellent cars home in the next following years.

Similar to sub-task 1, and considering the interpretability of the result, we choose Tree-Based models as our regression models. The nodes in the tree model could give an obvious reason why the model tends to get this prediction. We also try Tree-Based ensemble models like random forest and gradient boosted tree. Due to their powerful learning ability, which could exploit deeper knowledge of the data, they even get a better regression result.

For all the Tree-Based models we apply in this task, we use the Grid Search to select the hyperparameters as what we do in sub-task 1. After the Grid Search, we select the *max_depth = 100* for the Decision Tree Regressor; *max_depth = 10, n_estimators = 50* for the Random Forest Regressor, and *learning_rate = 0.15, n_estimators = 300* for the Gradient Boosted Tree.

Besides, from the prediction results on the test dataset, the experiments prove the ensemble Tree-Based models are better and more effective. We think our models can be used in a real car store, helping customers foresee the value of the car and make a valuable investment.

IV. Evaluation & Interpretation

In this section, we evaluate different models used in different tasks.

A. Evaluation of Task 1

In task 1, we randomly assign data to a training dataset and a test dataset. The training set consists of 80% of data and the test dataset consists of 20% of data. Using K-fold cross validation method with $k = 5$, we obtain 6 regression models including three tree models (Decision Tree Regressor, Gradient Boosted Tree, and Random Forest Regressor) and three deep learning models (baseline model, embedding model, and Wide & Deep Model). Moreover, we combine some of the tree-based models and deep learning models to get an ensemble model. Next, we focus on the other 20% of data on the test dataset to evaluate how these models work.

The metric we use is RMSE, i.e., root mean square error. It is defined as follow:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(\hat{y}_i - y_i)^2}{n}}$$

where n is the number of predictions, $\hat{y}_i$ is predicted value and y_i is observed value. RMSE serves to aggregate the magnitudes of the errors in predictions for various data points into a single measure of predictive power. It is a measure of accuracy, to compare forecasting errors of different models for a particular dataset. Therefore, it is a good metric for the car resale price prediction.

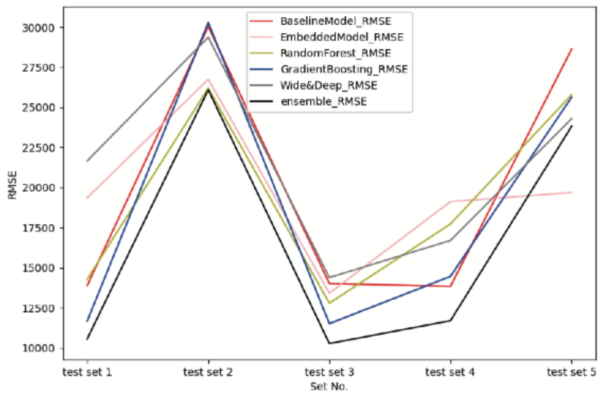


Fig. 6. RMSE of models derived from different dataset parts in Task 1.

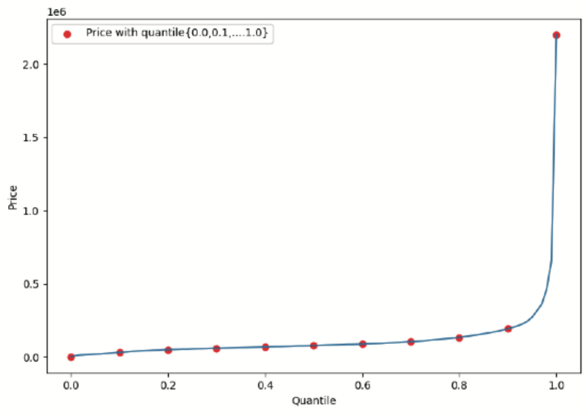


Fig. 7. Prices of different quantiles.

All the models that we leverage can make precise predictions of prices to a certain extent. Both tree models and deep learning models achieve overall RMSE of around 20000 on our test dataset. The best model is the ensemble model, which is a combination of tree-based models and the deep learning model. Furthermore, we do a fine-grained analysis of RMSE scores on the test dataset.

1) *RMSE in Different Test Parts*

Inspired by the idea of K-Fold method, we first divide the test dataset into 5 parts and examine how the models work on different parts. Fig. 6 shows the RMSE of different models on different parts of the test dataset. The worst model is the baseline model, of which the highest RMSE reaches over 30,000 and the average RMSE is also the highest. The best model is the ensemble model with the lowest RMSE in most of the test dataset parts. Other models get similar outputs with the highest score around 25,000 on test dataset 2 and the lowest score around 14,000 on test dataset 3.

In Fig. 6, we can find a great variance between these 5 test datasets. Particularly for test datasets 2 and 5, most of the models result in very large RMSE values, which are much larger than the RMSE obtained over the whole test dataset. But in other test datasets (1, 3, and 4), most models achieve much smaller RMSE. Therefore, data from these two test datasets (2 and 5) may have a great impact on a worse result of RMSE.

In the EDA process, we find some cars with low performance but still get high resale prices. Then we assume that there may exist some outliers like vintage cars that are very old or bad but valuable for some special reasons (e.g., very collectible). So, we need to check if the prices from different ranges will affect the prediction.

2) *Insight of Test Dataset*

We examine the resale price on the test dataset. Fig. 7 plots the price at each quantile, where the quantile ranges from 0 to 1 with *step = 0.01*. The red points are quantile points with *step = 0.1*. From Fig. 7, we know that before quantile reaches to 0.9, the price increases at a slow pace, indicating that most car prices are lower than the price at 0.9 quantile. Quite different from the previous trend, the price increases sharply after 0.9 quantile. This abnormal steep rise may account for the high variance of RMSE.

3) *Evaluation on different quantile ranges*

We then divide the whole test dataset into 10 parts according to the prices in ascending order. Each part consists of 10% of the test dataset. All models are applied to predict all parts of the test dataset, and we obtain Fig. 8. In the figure, all RMSE values in the last interval rise to around 50,000, which coincides with our previous assumption. Our models perform badly on those data with high prices. We can conclude that there are many outliers in the high-price interval and the trained models do not learn the data pattern of these outliers because the total number of outliers is small compared to the whole training dataset.

In addition to the last interval, there are tips in the interval [0.3, 0.4). Except for random forest regressor, all other models obtain high RMSE values from around 25,000 to 35,000. The difference between random forest regressor and other models may account for this phenomenon.

From the previous analysis, we can conclude that all models we use cannot handle the high-price data perfectly. Two possible reasons and solutions are listed below:

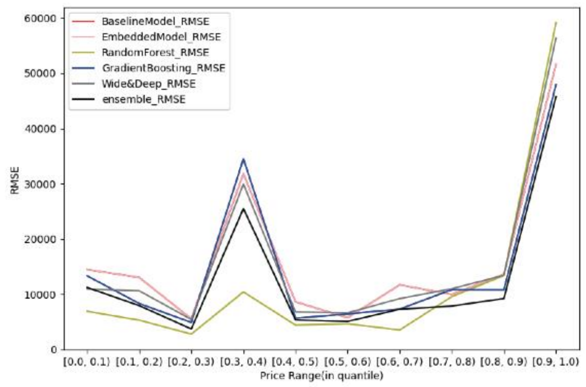


Fig. 8. RMSE of models in different price ranges.

a) High-price data are error points. Models cannot and should not learn a good pattern to predict the price of the part of data. The simplest way to deal with this kind of data in the training dataset is to delete them.

b) High-price data are considered noise, which means the given dataset consists of too few of them but these data are correct and meaningful in the real world. If more data are inclusive, the prediction will be better.

Another way to mitigate the bad effects that the high-price cars bring is using another evaluation metric, like Root Mean Squared Logarithmic Error (RMSLE). RMSLE emphasizes relative prices more than absolute prices because it applies a log function on the prices.

Apart from high-price data, data with price in [0.3, 0.4) interval also indicate that the models have certain shortcomings leading to a bad test score. For the moment, we can only say that the characteristics of different models might be the reason.

B. *Evaluation of Task 2*

1) *Track 1*

The recommendation system using item-item similarity is easy to check. If the user inputs a specific car, then the system will recommend the car with the highest cosine similarity, i.e., all features of the recommended car are similar to that of the input one. Choose the car with *row id = 4353* as an example input. With *k = 3*, we get recommendations with similar features, as shown in Table. I. The make of recommended car with *row id = 1460* is the same as our input. The types of the other two cars are also "mid-sized sedan". And the power and price of these recommended cars are similar to the desired one.

TABLE I. RECOMMENDED CARS

Row id	Make	Type	Power	…	Price
4353	hyundai	mid-sized sedan	93.8	…	82400
1460	hyundai	suv	130.0	…	109800
4755	kia	mid-sized sedan	93.8	…	76800
4880	kia	mid-sized sedan	93.8	…	79000

But if the user only cares about some of the features, then we should limit the number of features used to calculate cosine similarity. Otherwise, the user only gets results with every feature being similar.

2) *Track 2*

To evaluate the performance of the user-item similarity, we design a specific user who has scored many cars. Given a dataset with *make = "bmw"* and *price < 80,000*, the user gives high scores (7…10) to those with *price > 70,000* and low scores (1…4) to those with *price < 70,000*. We create the user profile using these scores and finally get the top 10 recommendation cars. The prices of most of the resulted cars are greater than 70,000. The system works well in the single numerical feature.

We also try to give scores according to multiple conditions, e.g., given a dataset with *make* ∈ {*"bmw", "nissan", "toyota"*} and *price < 80,000*, cars with *power < 80* and *price > 70,000* receive high scores, and cars with *80 < power < 120* and *price < 70,000* receive low scores. The recommendation system gives results that satisfy the price constraint but ignore the power constraint. There may be two reasons:

a) Cars satisfying both constraints are rare. So the system recommends cars with the closest power.

b) Other features of the cars have a greater impact on the user profile. The system recommends by the patterns of other features rather than power.

We can impose different weights on features to avoid the second situation. What should be done is to tune good weights by analyzing the importance of each feature.

C. *Evaluation of Task 3*

Like task 1, we use RMSE as the metric and split test dataset into five parts. As shown in Fig. 9, the decision tree regressor achieves the highest RMSE in all dataset parts. Its overall RMSE on the test dataset is 0.2065. Random forest regressor and gradient boosted tree achieve much lower RMSE, which are 0.1520 and 0.1220 respectively. Gradient boosted tree performs better than random forest regressor on almost all parts of the test dataset. In summary, gradient boosted tree makes the best predictions. The precision of depreciation rate prediction of gradient boosted tree could provide meaningful suggestions to the users who plan to buy a new car and have the plan to sell it in the future.

Next, we analyze the different performances of the three tree-based models. Decision tree is a single-tree model. Naturally, its accuracy should be lower than the other two tree-based ensemble models, and we obtain the expected same result. The RMSE scores of the other two regressors show that tree-based ensemble model with gradient descent is more capable to solve this problem. In addition, the low variance in RMSE shows that gradient boosted tree has learned good enough patterns to predict the Depreciation Rate.

V. Conclusion

In this report, we have demonstrated how we analyze and preprocess the data, how we choose proper data mining methods, and how we evaluate the results based on the different goals of the three different data mining tasks. The goal of our project is to give valuable information on data patterns for both car resale companies and the users who want to buy or sell cars. This goal is achieved in our project.

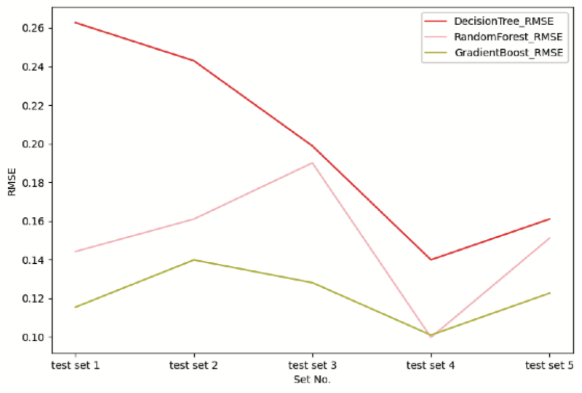


Fig. 9. RMSE of models derived from different dataset parts in Task 3.

However, there are still some limitations in our project that we can improve in future work. For example, we will try to balance the data because we find the number of different brands of cars is imbalanced. And we can also analyze the text data to retrieve some important information. The data mining methods can be also improved by using state-of-art models.

Acknowledgment

Thanks Professor Chris and Teaching Assistants of CS5228 for their feedbacks on our project. And thanks all members of Group Kimberley for their extraordinary efforts on this project.

References

[1] Rosenblatt, F. (1961). Principles of neurodynamics. perceptrons and the theory of brain mechanisms. Cornell Aeronautical Lab Inc Buffalo NY.

[2] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, *15*(1), 1929-1958.

[3] Peebles Jr, P. Z. (2001). Probability, random variables, and random signal principles. McGraw-Hill.

[4] Klambauer, G., Unterthiner, T., Mayr, A., & Hochreiter, S. (2017, December). Self-normalizing neural networks. In *Proceedings of the 31st international conference on neural information processing systems* (pp. 972-981).

[5] Bottou, L. (2012). Stochastic gradient descent tricks. In *Neural networks: Tricks of the trade* (pp. 421-436). Springer, Berlin, Heidelberg.

[6] Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

[7] Tieleman, T., & Hinton, G. (2012). Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, *4*(2), 26-31.

[8] Yao, Y., Rosasco, L., & Caponnetto, A. (2007). On early stopping in gradient descent learning. *Constructive Approximation*, *26*(2), 289-315.

[9] Wan, Shengxian, et al. "Next Basket Recommendation with Neural Networks." *RecSys Posters*. 2015.

[10] Cheng, Heng-Tze, et al. "Wide & deep learning for recommender systems." *Proceedings of the 1st workshop on deep learning for recommender systems*. 2016.