

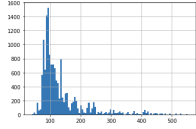
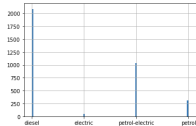
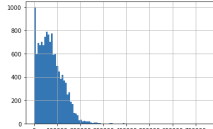
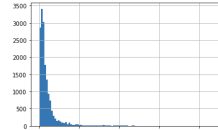
CS5228 Final Project

Project Title: Data Mining on Car Resale

Liu Ziming (A0236156E)
Zhen Hao (A0228573X)
Gao TianYou (A0243830L)
Zhan Zhengdao (A0229934U)

1 Exploratory Data Analysis (EDA)

1.1 Get to Know the Data

| | |
|------------------------|---|
| make | There are 44 unique 'make', many of them contains the information about 'model'. |
| model | There are 745 unique 'make', 'mode' and 'make' together can strongly indicate the 'price'. |
| description | description may contain important information of car's quality and condition, such as 'well maintained', and 'low fuel consumption'. |
| power |  <p>The distribution of 'power' is unbalance, for cars in the same 'make', the 'power' can be a indicator to difference car's price.(High/Low configuration).</p> |
| fuel_type |  <p>there are few 'electronic' cars, and most cars are diesel. 'Electronic' cars' price are more balanced than diesel.</p> |
| coe | coe means the right to own and use cars. |
| dereg_value | It is the money one can get back from the government when deregistering the car. It is related to 'coe'. |
| mileage |  <p>Mileage may be a strong indicator of 'price', most of the cars have mileage below 100000.</p> |
| omv |  <p>The OMV of a vehicle is the price paid when a car is imported, with a positive relationship with 'price'.</p> |
| arf | Arf is a tax imposed on registration of a car, it is a strong indicator of 'price'. Most of the 'arf' are below 10000 SGD. |
| Features & Accessories | Features and accessories may provide more detailed information about the car, such as 'bluetooth', 'paddle shifter',etc. |

1.2 Data Reduction

The data reduction process mainly include reducing number of features and dropping some of the training data. How this is done would differ according to the method we use, so we will discuss about this in the specific sections below.

1.3 Data Pre-processing

- For feature 'reg_date', outliers such as 18xx or 21xx years have been removed, and all 'reg_date' are subtracted by the earliest date.
- For feature "mode", regular expression is used to extract "model" from corresponding text of "make" because we found feature "make" often contains information of "model".
- For features "power", "engine_cap", "mileage", "curb_weight" and "no_of_owners", we simply used the average value of all cars.
- For features "depreciation", "coe", "dereg_value", "omv" and "road_tax", missing values are filled according to the average value of corresponding "make" and "mode". We believe these features are highly related to the brand and model.
- For feature "arf", we filled the missing data by using "omv", and by using the following formula:

| Vehicle OMV (\$75,000) | ARF Rate | ARF Payable |
|------------------------|----------|---------------------------|
| First \$20,000 | 100% | 100% x \$20,000= \$20,000 |
| Next \$30,000 | 140% | 140% x \$30,000= \$42,000 |
| Above \$50,000 | 180% | 180% x \$25,000= \$45,000 |

However, we found that by comparing with LightGBM's default method of dealing missing values, our method get worse result on Kaggle. It is because LightGBM uses NaN to replace missing values and when training, during a split, the missing numerical values are allocated to one side which reduce the loss the most, compared with our method, lightGBM can better dealing with the over-fitting problem.

- For accessories, we first made a list of accessories that we believe to have a higher effect on the price of the cars. For example, we included feather seats, parking assistant system, air-cons and so on. We first collected a set of different expressions of the accessories and then used Bert[1] as the embedding layer to get the meaning of these expressions. Than we used cosine similarity to check whether these accessories is included in each data. Finally we would get a vector for each car, containing all the accessories it has.

2 Task 1: Prediction of Car Resale Prices

We mainly tried out two methods in this task: XGBoost and LightGBM. We will separately discuss about the two models in the following sections.

2.1 XGBoost

2.1.1 algorithm

XGBoost [2] is a kind of boosting algorithm, whose basic idea is to use multiple weak learners as a whole strong learner so as to do jobs as classification or regression. The target function of XGBoost is defined as:

$$obj(\theta) = \sum_i^n l(y_i, \hat{y}_i) + \sum_{k=1}^K \Omega(f_k) \quad (1)$$

The first half of the function is the sum of loss between our prediction and the true value and according to the score rule of the competition we should use RMSE loss here, which is defined as:

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

The second half of the equation is the regularization part, which can be written as:

$$\gamma T + \frac{1}{2} \lambda \sum_{j=1}^T w_j^2 \quad (3)$$

In this equation, λ and γ are hyper-parameters that we are supposed to set, which will give a corresponding extend of penalty to the trees with complex structures(e.g. too many leaf nodes) and reduce over-fitting.

With the target function, we can define how good a split is by:

$$Gain = \frac{1}{2} \left[\frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda} \right] - \gamma \quad (4)$$

where G is the total gain on the sub-tree and H is the sum of the second derivatives of the loss function. γ is also a parameter that we can adjust, which is used to determine how good a split has to be so as to be accepted by us.

2.1.2 experiment

As id discussed in the first part, we used the data set whose empty values have been filled according to the models and manufacturers. We first included all the features in the data set, including the artificial features we extracted from the description column and the accessory column. We normally used grid-search CV to adjust the hyper-parameters of XGBoost. The parameters are as follows:

| parameter name | description | final value |
|------------------|--|---------------|
| learning rate | Define how fast our model would converge and also how accurate the model would be at finding the best point | 0.1 |
| max_depth | Define how deep the learners can become. Helps with reducing over-fitting | 5 |
| min_child_weight | Define the minimum sum of instance weight needed in a child. Prevent the devision of a node with too little samples and reduce over-fitting. | 3(suspicious) |
| gamma | As is explained above, defines how good a split has to be for us to accept. | 0 |
| subsample | Subsample ratio of the training instances. Also helps with reducing over-fitting. | 0.6 |
| colsample_bytree | The subsample ratio of columns when constructing each tree | 0.6 |
| n_estimators | The number of round of iterations, which also means the number of weak learners. The higher the more accurate, but also the more complex the model is. | 600 |
| reg_alpha | L1 regularization term on weights as explained above. | 0.1 |

With the hyper-parameters defined as above and features selected according to their importance, we get the final importance, with features being the 11 accessories, low fuel consumption, well maintained, better loan offer,reg_date,power,engine_cap,mileage,n_dereg_value,omv,arf,model_price and is_new.

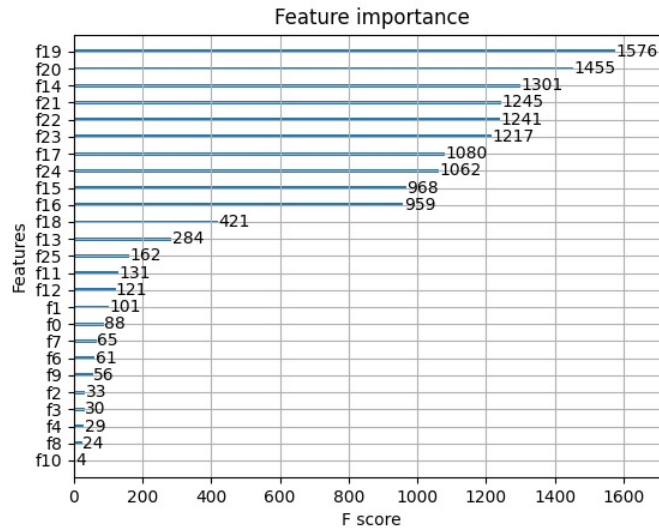


Figure 1: XGBoost feature importance

2.1.3 result

With the selection of features and parameters above, we reached a average rmse score of about 22000 on the training set with 5-fold cross validation, which seemed to be pretty acceptable. But the performance on the training set was quite the opposite and reached as high as 36000. With reference to the lightGBM model we later on implement, this was mainly caused by the empty values we filled in by ourselves. They strongly affected so estimated rows and caused the result to be not satisfying.

2.2 CatBoost

2.2.1 algorithm

Catboost is an oblivious trees based GBDT framework, which has fewer parameters and high accuracy. CatBoost can better deal with categorical features by embedding category features to numerical features based on some statistics features (frequency, etc.) , it has advantages compared to one-hot encoding.

Catboost can combine different category features to get new features, which can take advantage of the relationship between features to get high-order dependencies, and greatly enriches the feature dimension. For example, in a movie recommendation system, categorical features like User ID and Movie type may have some relationship, which can be encoded by CatBoost.

2.2.2 experiment

We found an over-fitting problem during training, and we tuned following parameters to address the problem:

max_depth: We reduced the maximum depth of a tree from default value **6** to **4**, to make the model less complex.

min_child_samples: We increased the min_child_samples from default value **1** to **3**. Therefore the model will stop further partitioning when the the sum of instance weight of a child is equal or smaller than 3. It will make the model more conservative.

subsample: We set subsample **0.8**, therefore the sample rate for bagging is **0.8**.

sampling_frequency: We set sampling_frequency to **PerTreeLevel**, CatBoost sample weights and objects for each time before choosing each new split of a tree.

For data we used, all features are incorporate, with our own missing value filling methods.

2.2.3 Visualization

This figure shows each features' impact on the model, color red means the feature pushing the prediction price higher, color blue means the feature pushing the prediction price lower.

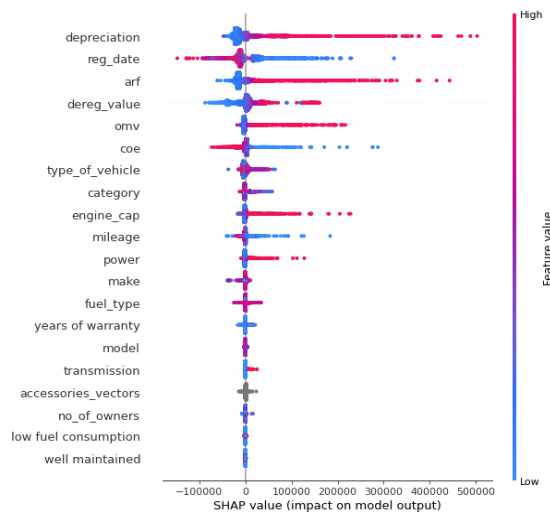


Figure 2: CatBoost feature analysis

We can find that "depreciation", "reg_date" and "arf" have great impact on model. The below figure shows more details about the relationship between "price" and "omv".

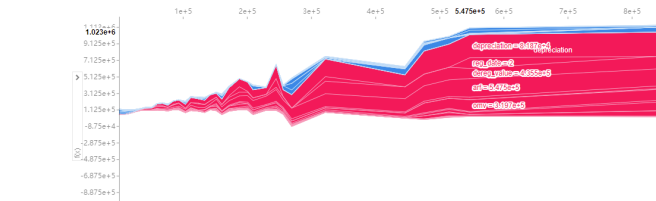


Figure 3: omv-price relationship 1

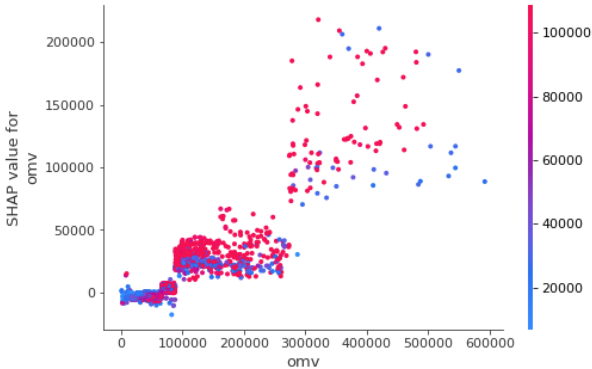


Figure 4: omv-price relationship 2

We can see that the "price" has a positive relationship with "omv", also "depreciation", "arf", etc. However, the "mileage" has a negative impact on "price".

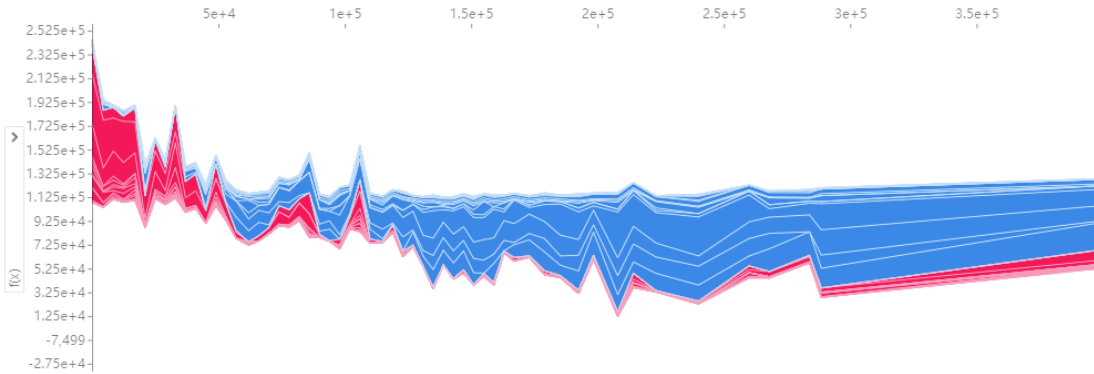


Figure 5: mileage-price relationship

For categorical features such like "make" which are automated embed by CatBoost, there are no explicit relationship between them and the "price".

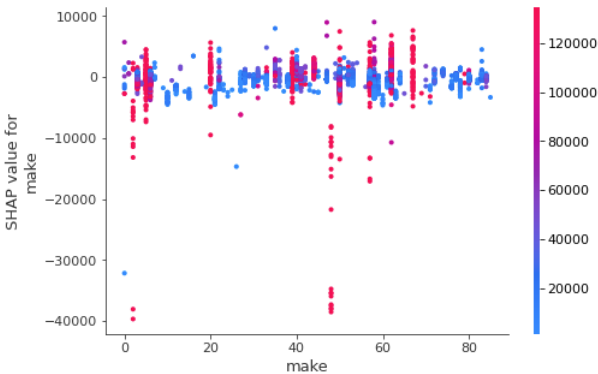


Figure 6: make-price relationship 1

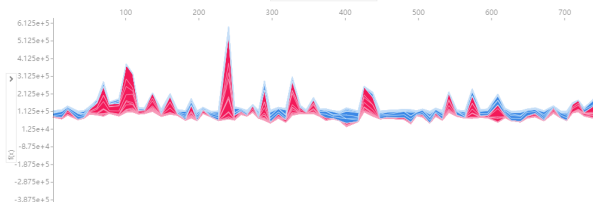


Figure 7: make-price relationship 2

These findings indicates that we should be more focus on numerical features which can reflect car price in the real world, and categorical features embedded by CatBoost didn't has a explicit advantages.

2.2.4 result

With the selection of features and parameters above, we reached a average rmse score of about 20000 on the training set with 5-fold cross validation, But the model got a score of 37000 on test set, the missing value we filled will lead to serious over-fitting problems.

2.3 LightGBM

2.3.1 algorithm

LightGBM [3] mainly has two unique algorithms compared with others, which are GOSS and EFB. GOSS(Gradient based One Side Sampling) would assign weight to samples according to its gradient, which would enable the model to focus on less used samples and balance the training. EFB(Exclusive Feature Bundling) would look for the mutually exclusive features and bundle them together, so as to reduce the number of features and increase the training speed. The algorithm description taken from the paper is listed below.

Algorithm 3: Greedy Bundling

Input: F : features, K : max conflict count
Construct graph G
 $searchOrder \leftarrow G.sortByDegree()$
 $bundles \leftarrow \{\}$, $bundlesConflict \leftarrow \{\}$
for i **in** $searchOrder$ **do**
 $needNew \leftarrow True$
 for $j = 1$ **to** $len(bundles)$ **do**
 $cnt \leftarrow ConflictCnt(bundles[j], F[i])$
 if $cnt + bundlesConflict[i] \leq K$ **then**
 $bundles[j].add(F[i])$, $needNew \leftarrow False$
 break
 if $needNew$ **then**
 Add $F[i]$ as a new bundle to $bundles$
Output: $bundles$

Algorithm 4: Merge Exclusive Features

Input: $numData$: number of data
Input: F : One bundle of exclusive features
 $binRanges \leftarrow \{0\}$, $totalBin \leftarrow 0$
for f **in** F **do**
 $totalBin += f.numBin$
 $binRanges.append(totalBin)$
 $newBin \leftarrow new\ Bin(numData)$
for $i = 1$ **to** $numData$ **do**
 $newBin[i] \leftarrow 0$
 for $j = 1$ **to** $len(F)$ **do**
 if $F[j].bin[i] \neq 0$ **then**
 $newBin[i] \leftarrow F[j].bin[i] + binRanges[j]$
Output: $newBin$, $binRanges$

Figure 8: EFB algorithm

2.3.2 experiment

As the result with the data set whose missing value is filled with our strategy did not turn out well, we tried to use the original data set instead as the lightgbm library provides functions to deal with missing values. When facing a missing value in a sample, lightgbm would try putting the sample on both sides, compare the gain value and finally choose the higher one as the accepted split. In this part we also used the GridSearchCV library. After many rounds of parameter adjustments, the parameters we chose and the structure of our model can be shown as below:

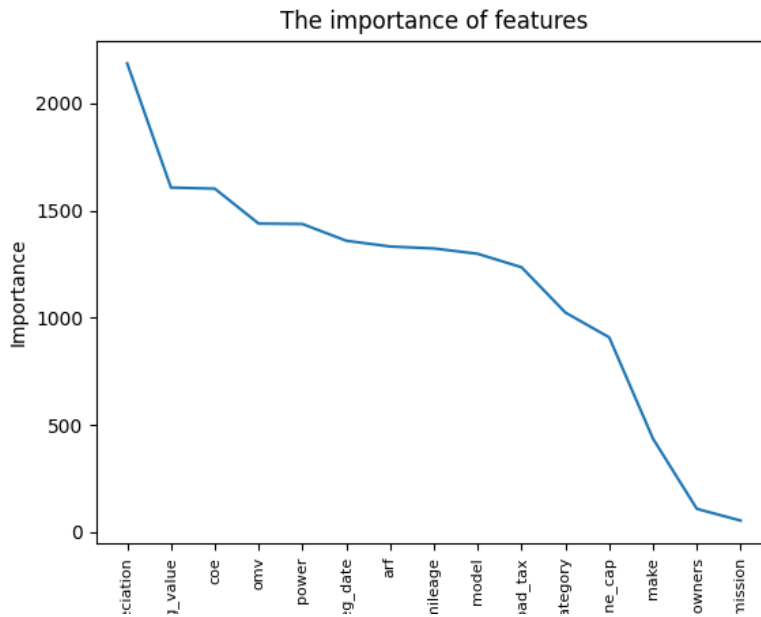


Figure 9: LightGBM feature importance

2.3.3 result

LightGBM did the best job in all our attempts, with 24000 rmse on the training set cross validation and 29000 on the kaggle scoring test set. We can tell from this part that the strategy LightGBM takes about the missing values is much better than the strategy we took to fill the values. Compared with other models, LightGBM trains faster and can do better on small data-sets with GOSS algorithm.

3 Task 2: Car Recommendation

3.1 Methodology

To utilize the features of the cars, we design the recommendation process as a content-based system. The idea is to vectorize the car items, calculate item similarities among all the cars. The top K cars with the most similar vectors are returned. We tried three ways to calculate vector similarity, including Cosine Distance, Euclidean Distance, and TS-SS [4].

3.1.1 Cosine Distance

Cosine Distance (CD) is one of the most common ways to calculate vector similarity. The range of CD measure is from -1 to 1.

$$\text{cosine}(A, B) = \frac{\sum_{n=1}^k A(n) \cdot B(n)}{|A| \cdot |B|} \quad (5)$$

However, the method does not take the magnitude of vectors into consideration, which indicates the numerical value of a particular feature, which might cause inaccurate results consequently if the vectors have the same angle but tremendously different magnitudes.

3.1.2 Euclidean Distance

Euclidean Distance (ED) is another popular geometrical method used to evaluate similarity. The maximum possible similarity value of ED is ∞ , while the minimum value is 0. It represents the distance between two points where the vectors point in the n dimensional space based on their coordinate.

$$\text{ED}(A, B) = \sqrt{\sum_{n=1}^k (A(n) - B(n))^2} \quad (6)$$

3.1.3 TS-SS

TS-SS stands for the combination of Triangle's Area Similarity (TS) and Sector's Area Similarity (SS) [4]. Different from other methods mentioned above, TS takes three characteristics into consideration for computing the similarity, including angle, magnitude and Euclidean Distance, making the value more precise.

$$\text{TS}(A, B) = \frac{|A| \cdot |B| \cdot \sin(\theta')}{2} \quad (7)$$

In the above equation, the value of θ' is $\cos^{-1}(V) + 10$, where V is the cosine value of θ . The reason to modify the angle between two target vectors is that the idea of forming the triangle fails when vectors are overlapped. To overcome the problem, θ is designed to increase by 10 degrees.

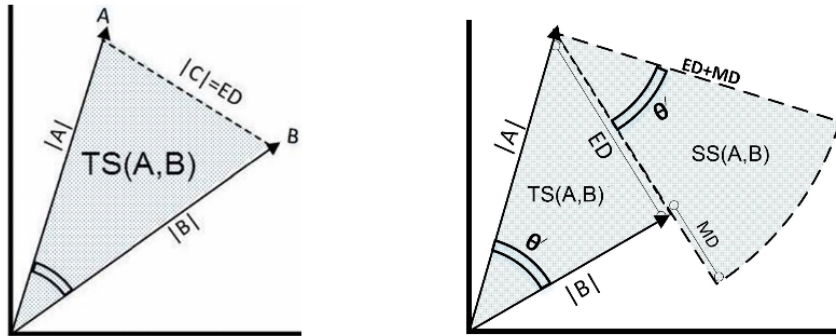


Figure 10: Triangle's Area Similarity (TS) and Sector's Area Similarity (SS)

However, TS alone is not robust enough to calculate accurate similarity results due to some critical missing components such as the difference between the vectors' magnitudes and Euclidean Distance (ED) between two vectors. Therefore, what SS does is to consider these components as well as a complementary of TS.

$$MD(A, B) = \left| \sqrt{\sum_{n=1}^k A_n^2} - \sqrt{\sum_{n=1}^k B_n^2} \right| \quad (8)$$

In the paper [4], the magnitude difference between two vectors is called MD. It increases proportionally with the increase of ED. At the same time, to leverage the combination of MD and ED on similarity values, SS adds Angular Difference (AD) to the calculation process, which is $AD(A, B) = \theta'$. Note that θ' is the same value used in Equation (7).

$$SS(A, B) = \pi \cdot (ED(A, B) + MD(A, B))^2 \cdot \left(\frac{\theta'}{360} \right) \quad (9)$$

The final similarity result is calculated by $TS \times SS$ since TS and SS form as a whole and complete each other. Moreover, multiplication deals well the situations when one value is quite larger or smaller than the other, making summation mainly represent the larger value. TS-SS measures from 0 to ∞ . Therefore, the final equation for TS-SS is as follows.

$$TS-SS(A, B) = \frac{|A| \cdot |B| \cdot \sin(\theta') \cdot \theta' \cdot \pi \cdot (ED(A, B) + MD(A, B))^2}{720} \quad (10)$$

3.2 Implementation

The whole process of the system is to take in the input feature vector, compare it with all the other available item vectors in the dataset and return top K recommendation results. The main function which takes in the vector and output the selected items is `get_top_recommendations`. It calls the functions from `similarity.py`, where all three of the methods are implemented. Among them, the implementation of TS-SS is referred to [5], while all the other are completed ourselves.

The note book attached records all the process of the system. Note that since Cosine Distance anyway normalizes the vector and then takes dot product of two, the vector normalization is only performed on Euclidean Distance and TS-SS.

3.3 Evaluation

3.3.1 Recommendation Precision

In the evaluation process, we first assess the recommendation results of different methods. After testing with different K values, we get an initial idea of these three methods. The results are shown in Root Mean Square Error (RMSE) and Mean Absolute Error (MAE). The error values are represented the mean of all the pair-wise errors (between input target and results) in the top K recommendation outputs.

As depicted in Figure 11, Cosine Distance has the highest error score among all of the three, which is nearly twice the value of Euclidean Distance and TS-SS. At the same time, Euclidean Distance and TS-SS performs similarly. From a detailed look of their MAE values, the results are quite close.

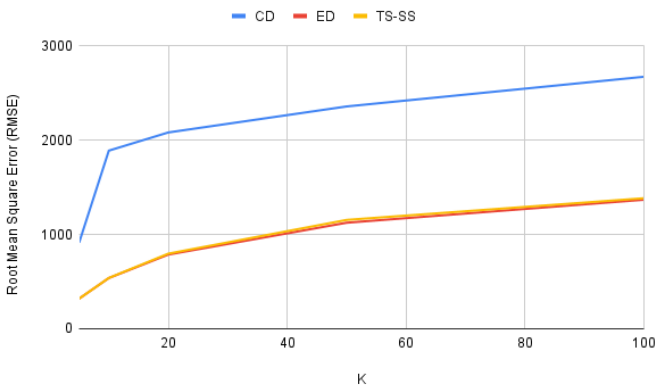


Figure 11: RMSE of Three Methods

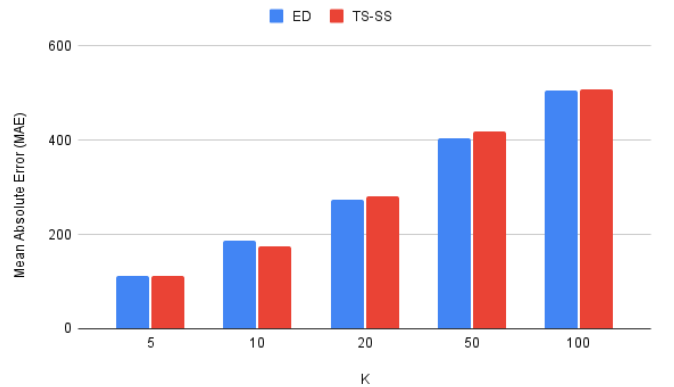


Figure 12: MAE of ED and TS-SS

Therefore, it is obvious that for the recommendation task on car resale dataset, both Euclidean Distance and TS-SS are suitable. However, since Cosine Distance ignores the item vector magnitude, it returns the worst results.

3.3.2 Running Time

The running time is another essential feature for evaluation. For both Cosine Distance and Euclidean Distance, we use the implementation from scikit-learn directly, while TS-SS is prepared in an separate python script. As listed in the table below, TS-SS takes the least time to run, ED follows and CD takes the most of the time.

| Cosine Distance | Euclidean Distance | TS-SS |
|-----------------|--------------------|-------|
| 4.3s | 2.4s | 1.9s |

Table 1: Running Time of Three Methods

3.4 Conclusion

From the evaluation part, we can conclude that both Euclidean Distance and TS-SS are suitable for the recommendation task on car resale dataset. They lead relatively better recommendation results while saving much running time compared to Cosine Distance, which also indicates that the magnitudes of item vectors in the dataset do greatly affect the vector similarity calculation in the recommendation task.

4 Task 3: Open Task: Car Retention Rate

4.1 Motivation

For car manufacturers and car buyers, a quantitative approach to evaluating the value retention of a vehicle allows for a logical analysis of the vehicle’s intrinsic value. Instinctively, the value retention of a car can be roughly assessed by the difference between the price of a brand new car and the price of a used car, divided by the mileage, as a criterion. In actual data analysis, this ratio also needs to be homogenized against the brand new car value to acquire comparability.

In this task, we will calculate the average car retention rate for different makes of models, according to the following formula.

$$\text{Car Retention Rate} = \frac{\text{New Car Price} - \text{Used Car Price}}{\text{Mileage} \times \text{New Car Price}} \times 100\%$$

4.2 Implementation

4.2.1 Additional Data Collection

In order to calculate the car retention rate, we need brand new car prices. This information is available on the Internet. We implemented a web crawler based on Selenium. Since the implementation of the web crawler is not very relevant to this course, the details of the implementation are omitted here. Since some of the models involved in the data set have been discontinued, their prices are not directly available. This issue of missing values will be dealt with in the next section.

4.2.2 Data Pre-processing

- For feature "model_price" (new car price), missing value is filled with the average model_price of the same make.
- For feature "mileage", missing value is filled with the global average mileage.

4.2.3 Data Screen

After data pre-processing, we found some issues in the data.

1. Some deals are sold second-hand for more than the original price.
2. Some models have too small a sample size, resulting in extreme values for the average results.

For the first problem, data points with negative price differences are eliminated. For the second problem, intuitively data from makes which have a smaller number of data points are not considered in the calculation. To achieve this goal, we use the Z-score as a metric to eliminate low-performing data points. In fact, the makes that are incorporated into the calculation essentially have more than 50 data points to ensure the general significance of the findings.

4.3 Result & Discussion

The result is shown as below (for display purposes, the x-axis unit displayed as 1e6):

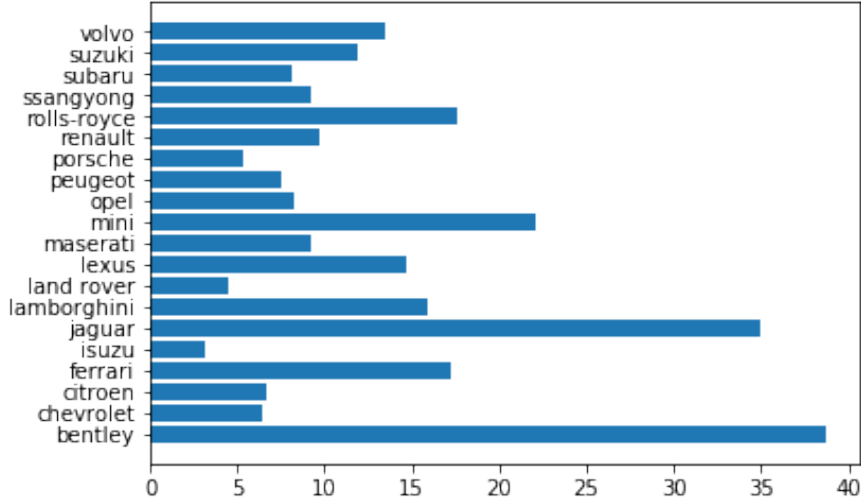


Figure 13: retention rate result

From the results, we can find that Bentley, Lamborghini, Rolls Royce and Ferrari have high retention rates, which is not surprising since they are luxury cars. A counter-intuitive finding is that MINI also has a high retention rate, which would be an interesting entry point for a more in-depth analysis and study. In the results we can also find more interesting findings, for example Isuzu has a lower retention rate compared to Suzuki, and both brands are considered to be in the same value range.

4.4 Practical Design Improvement

During the implementation of the algorithm it has been found that the distribution of data points owned by different vehicle makes varies greatly, which can affect the generality of the final results of the make. In order to solve this problem, more data points need to be collected. In addition, using the Z-score threshold to filter out outliers is an empirical behavior that can be replaced by a more potentially logical analysis method to figure out appropriate threshold.

References

- [1] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “BERT: Pre-training of deep bidirectional transformers for language understanding,” in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, Jun. 2019, pp. 4171–4186. [Online]. Available: <https://aclanthology.org/N19-1423>
- [2] G. C. Chen Tianqi, “Xgboost: A scalable tree boosting system,” *KDD '16: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, no. 785-794, 2016.
- [3] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, “Lightgbm: A highly efficient gradient boosting decision tree,” in *NIPS*, 2017.
- [4] A. Heidarian and M. J. Dinneen, “A hybrid geometric approach for measuring similarity level among documents and document clustering,” in *2016 IEEE Second International Conference on Big Data Computing Service and Applications (BigDataService)*. Los Alamitos, CA, USA: IEEE Computer Society, apr 2016, pp. 142–151. [Online]. Available: <https://doi.ieeecomputersociety.org/10.1109/BigDataService.2016.14>
- [5] J. Kim, “Vector_similarity,” https://github.com/taki0112/Vector_Similarity, 2019.