614 Project Report

Members: *Ayman Shahriar, Harmandeep Singh Teja, Jae Kang, Jinyu Wang, Xin Wang*

Instructor: Sarah Shah

Introduction And Motivation

The demand for pre-owned items, particularly vehicles, has been steadily increasing as consumers seek cost-effective alternatives to purchasing new products. Many individuals prefer not to pay the full price for brand-new items when better deals are available for used goods. For others, financial constraints make buying new items unaffordable. In the case of cars, depreciation begins the moment a new vehicle is driven off the lot, further driving demand for used vehicles.

In 2023, sales of used light vehicles in the United States reached approximately 38.2 million units, significantly surpassing the 15.5 million new vehicle sales in the same year [1]. This trend underscores the growing consumer preference for used cars. Additionally, the U.S. used car market was valued at USD 195.84 billion in 2021 and is projected to reach USD 302.47 billion by 2027, registering a compound annual growth rate (CAGR) of 7.51% during the forecast period [2].

For our project, we will utilize data related to used cars in order to develop and evaluate models that accurately predict the price a used car should be sold for when given the properties of that car as input. The motivation behind this project stems from the growing demand for used cars and the value this project could bring to both buyers and sellers. Buyers can gain insights into fair market pricing, helping them negotiate better deals with dealers. Similarly, sellers can use this analysis to determine competitive prices for their vehicles when listing them on online platforms. In the following sections, we will describe our data collection, data processing, model building, model evaluation and analysis, and potential next steps.

Data Collection

We have obtained a dataset from Kaggle that contains more than 3 million rows of used car details that was scraped from Cargurus in September 2020 [6]. Each of the 3 million rows contains the details of a single used car, including the model, year, and price. The dataset consists of 66 columns, with the price column serving as the target and the other 65 columns being potential features. The data types of the columns include numerical (integer or float), string and boolean. The kaggle page from which we downloaded the data also included a description of each column, which helped us better understand the data.

The size of this dataset is close to 10 gigabytes. However, due to a limitation of Google Colab (the environment which we used to host our python notebook), we used the first 100,000 rows of the data (which has a size of 362 megabytes) for our processing and model development. The limitation is that the notebook will stop running if the user has not manually interacted with it for a certain time period (less than an hour). Since it is not practical to constantly monitor the running of our notebook for hours on end to prevent timeout, we only used the first 100,000

rows of the data for our project. It took more than 6 hours to process and build models using this reduced dataset.

Data Preprocessing, Wrangling and Exploration

Given the large number of columns in our dataset, there were many opportunities for us to process and explore the data. Here is a sequential summary of all the preprocessing that was carried out:

- Drop columns that definitely won't be used, such as ids and urls.
- Missing string values are indicated by '--', replace with None.
- Drop string columns that have None for every value.
- Some columns should be numerical, but are string because each value includes the unit of the measurement. For example, the length column's values include '210 in', '224 in', etc. where 'in' indicates inches. We converted these string columns into numerical by removing the units. We also made sure that each column had a consistent unit before removing the units.
- The power column consists of two values, horsepower and rpm (eg. '177 hp @ 5,750 RPM'). These two values were extracted into their own separate columns, and were converted to numerical type.
- The torque column consists of two values, torque and rpm (eg. '200 lb-ft @ 1,750 RPM'). These two values were extracted into their own separate columns, and were converted to numerical type.
- The dealer_zip was converted from string to numeric. We looked into the structure of US zip codes, and discovered that each digit divides a region into more granular regions. We decided that keeping all 5 digits of the dealer_zip results in regions that are too granular. So we decided to keep the first 3 digits only.
- The values in wheel_system are just abbreviations of the values in wheel_system_display, so we dropped wheel_system_display.
- The description column is of type text data, not categorical data. Since we will not do any text analysis / natural language processing with this dataset, we dropped the description column.
- Remove engine_cylinders because its values are identical to the values of engine_type.
- The models of PySpark's MLlib library throw errors when they encounter None values, so we replaced all None values with 0 for numerical columns, False for boolean columns and "NULL" string for string columns.
- String columns cannot be used by the models of PySpark's MLlib library, we need to encode them. In each of our string columns, their values have no inherent hierarchy, so

they are all nominal. Nominal features can be encoded using one-hot encoding, so we used PySpark's OneHotEncoder and CountVectorizer to encode them.

- The listed_date column is of type date, but PySpark's MLlib library does not support this data type. So we converted listed_date into UNIX timestamps (integers representing seconds since epoch), which is compatible with MLlib.

The data exploration stage also included some exploratory data analysis to help us better understand the data:

- For numeric columns, print out the data type, null count, unique value count, min and max of each column.
- For string columns, print out the data type, null count, unique value count, min length and max length of each column.
- We noticed that the minimum value of the back_legroom column is 0, which does not make sense. After investigating the rows whose back_legroom value is 0, we discovered that they are pickup_trucks (by looking at the body_type column).
- Visualize the first few rows to inspect the values of each column.

Model Building and Results

We have trained three models using the cleaned data: linear regression, lasso regression and decision tree. Both linear regression and lasso regression are linear models, while a decision tree is a non-linear model. Having both linear and non-linear models will enable us to capture both linear and non-linear correlations within the data. We used PySpark's MLlib library to build and evaluate our models. The MLlib models require all the input features to be merged into a single array, so we had to use PySpark's VectorAssembler to transform the input before they are used to train the models. 75 percent of the data was used for training the model and hyperparameter tuning, and 25 percent was used to evaluate the performance of the model when given new, unseen data.

Linear regression:

A linear regression model captures the linear correlations in a dataset. Linear regression models do not have any hyperparameters, so we created the linear regression model without performing any hyperparameter tuning that is done for the other two models.

Lasso Regression:

A lasso regression model is similar to linear regression, except that it uses regularization. One way to think of regularization is that it restricts the model to prevent overfitting. Lasso regression uses L1 regularization, which sets some of the coefficients of the model to be zero. This results

in some features being entirely ignored by the model, resulting in a type of automatic feature selection.
Given the large number of features in this dataset, there is a high likelihood that some features are not very important for predicting the target. So using lasso regression is well suited for our use-case, as it will ignore features that are not relevant for predicting the target.

The lasso regression model uses a parameter that controls the L1 regularization (the higher the parameter, the more regularization will be applied). In order to find the optimal value for the L1 regularization parameter, we performed hyperparameter tuning using a combination of grid search and 5-fold cross validation. We found that the optimal L1 regularization parameter value for our data was 10.

Decision Tree:

We created a decision tree model to capture any non-linear correlations that our data may have. In order to prevent overfitting, we can limit the maximum depth of a decision tree. To find the optimal depth of our decision tree, we performed hyperparameter tuning using a combination of grid search and 5-fold cross validation. We found that the optimal depth for a decision tree built using our data was 11.

Model Results:

R2 Scores:

	Linear Regression	Lasso Regression	Decision Tree
Train Score	0.984	0.978	0.940
Test Score	0.691	0.711	0.678

RMSE Scores:

	Linear Regression	Lasso Regression	Decision Tree
Train Score	2,512.376	2,988.879	4,944.755
Test Score	12,790.756	12,361.408	13,050.859

Figure 1: Validation Metrics of the Three Models

Two validation metrics were calculated for each model: the coefficient of determination (R2), and the root mean square error (RMSE). The R2 score is bounded between 0 and 1, and indicates the amount of variance in the target that can be explained by the features. A higher R2 score

indicates better performance [3]. The RMSE sums the squared difference between the predicted value and the actual value of each datapoint. A lower RMSE score indicates better performance [3]. For all three models, and for both the training and test data, the R2 score and RMSE were inversely correlated. In other words, if a model had a higher R2 score than another model, then it also had a lower RMSE score than that other model (and vice-versa).

When evaluating the performance of a model, we are mainly concerned with the test score, which indicates a model's generalization performance (i.e. how good the model is when predicting on new data). All three models have a large difference between their train R2 score and test R2 score, which indicates overfitting.

To prevent the overfitting that is happening in the linear regression model, we created the lasso regression model. We benefit from the regularization that is being done in the lasso regression model as its performance is better than the linear regression model, with a test R2 score of 0.711. However, the model is still overfitting as it still has a very high training R2 score.

The decision tree has the lowest training R2 score (0.678), which means that out of the three models, the decision tree has the worst performance. The decision tree also has a large difference between its training and test R2 scores, which means that just like the two linear models, it is also overfitting on the data. Decision trees have a tendency to overfit on the data, which is why ensemble methods like random forest and gradient-boosted decision trees are used more often than single decision trees [4].

Conclusion and Next Steps

Given that our best model has a low test R2 score of 0.711, we can conclude that we were not successful in achieving our goal, which was to develop a model that can accurately predict the price of a used car. A potential next step is to use other models such as random forest, gradient-boosted decision trees and neural networks and see if we get better results. However, some of these new models would be computationally intensive, so we could also try switching our environment from Colab to a real distributed computing environment (such as Databricks) to scale up our computing power. If we utilized distributed computing, we could potentially increase our computing power enough to handle the entire 10 gigabyte dataset.

Another approach we could consider for improving model performance would be to reduce the dimensionality of our dataset using unsupervised dimensionality reduction techniques, such as Principal Component Analysis. Dimensionality reduction would reduce the number of features while still retaining the meaningful properties of the data, which would reduce the effects of the “curse of dimensionality” and potentially improve model performance [5].

References:

[1]: New and used light vehicle sales in the United States from 2010 to 2023, Statista, https://www.statista.com/statistics/183713/value-of-us-passenger-cas-sales-and-leases-since-1990/?utm_source=chatgpt.com

[2]: US Used Car Market Size & Share Analysis - Growth Trends & Forecasts (2024 - 2029), Mordor Intelligence, https://www.mordorintelligence.com/industry-reports/united-states-used-car-market

[3]: RMSE vs. R-Squared: Which Metric Should You Use?, Statology, https://www.statology.org/rmse-vs-r-squared/

[4]: Leanne Dawson, Week 4/5 – Decision Trees, Slide 32, ENSF 611: Machine Learning for Software Engineers, University of Calgary, 2024

[5]: What is dimensionality reduction?, IBM, https://www.ibm.com/think/topics/dimensionality-reduction

[6]: US Used Cars Dataset, Kaggle, https://www.kaggle.com/datasets/ananaymital/us-used-cars-dataset