CS 229 Final Report: Used Cars Price Prediction

Angel Raychev

December 6, 2024

1 Introduction

The goal of this project is to develop a machine learning model capable of predicting the auction price of used cars. We utilize a large dataset obtained from Kaggle [1], which contains detailed information on vehicle features and pricing. The motivation for this project lies in improving price predictions beyond the baseline provided by the Manheim Market Report (MMR), a leading source of wholesale vehicle pricing.

2 Dataset

The dataset contains 558,837 examples and the following features:

- **vin**: Unique identification number for each car.
- **production year**: Year of manufacture (1982–2015).
- **make**: Manufacturer (e.g., Toyota, Kia, Honda).
- **model**: Specific car model (e.g., Sorento, M5, Fusion).
- **trim**: Sub-model configuration (e.g., SE, LX, Limited).
- **body**: Body type (e.g., SUV, Sedan, Hatchback).
- **transmission**: Gearbox type (automatic or manual).
- **state**: State where the car was sold (e.g., CA, FL, TX).
- **condition**: Numerical rating (1 to 5) representing the car's condition (higher is better).
- **odometer**: Number of miles driven.
- **exterior color**: Color of the car's exterior.
- **interior color**: Color of the car's interior.
- **seller**: Auction seller (e.g., Kia Motors America, Volvo NA Rep/World Omni).
- **saledate**: Date of the auction sale (e.g., Thu Jan 29 2015 04:30:00 GMT-0800 (PST)).
- **mmr**: Manheim Market Report price, the baseline we aim to outperform.
- **price**: Actual selling price at the auction (target variable).

2.1 Dataset Preprocessing

To ensure data quality and relevance, we applied the following preprocessing steps:

1. **Handling missing and incorrect values:** Removed examples with missing feature values or incorrect formats.

2. **Standardization:** Standardized labels to lowercase and removed redundant spaces.

3. **Outlier removal:** Removed atypical examples, including:

    - Cars manufactured before 2000.
    - Cars with odometer readings below 1,000 miles or above 220,000 miles.
    - Cars sold for less than $1,000 or more than $50,000.
    - Cars with makes appearing in fewer than 1,000 examples.
    - Cars with models appearing in fewer than 100 examples.
    - Cars with body types appearing in fewer than 1,000 examples.

After preprocessing, the dataset was reduced to 435,651 examples.

2.2 Feature Engineering

We identified two missing yet important features: **engine volume** and **fuel type**. To extract these, we leveraged public APIs that provide detailed car information based on the VIN number. Due to request limitations, this process took approximately 4 hours. We further removed cars for which this information could not be retrieved. Lastly, we removed some outliers: car engines of volume greater than 7L or smaller than 1L, as well as any example whose fuel type had fewer than 120 occurrences in total. These preprocessing steps reduced the dataset to 431,328 examples.

2.3 Feature Selection

We excluded the following features due to limited predictive value:

- **Trim**
- **Seller**
- **Sale date**

Additionally, we chose not to include the **MMR (Manheim Market Report)** as a feature, even though it is highly predictive of auction prices. This decision was motivated by our goal to develop a model capable of outperforming MMR. Including MMR as a feature would likely cause the model to rely heavily on it, limiting the model's ability to learn richer relationships from the other features. By excluding MMR, we ensure that the model focuses on leveraging features like condition, mileage, make, and model to generate a more expressive and potentially superior pricing model.

2.4 Dataset Split

We split the dataset into training, development, and test sets as follows:

- **Training set:** 401,328 examples.
- **Development set:** 15,000 examples.
- **Test set:** 15,000 examples.

2.5 Feature Preprocessing

We applied three types of encodings to the categorical features, depending on the model and the specific feature:

- **Simple Labeling:** Each unique category was assigned a numeric label
- **One-Hot Encoding:** Each category was represented as a binary vector, with one dimension per category.
- **Target Encoding:** Categories were replaced with the average price of all training set examples corresponding to that label.

Rescaling was applied to all numerical features to normalize them between 0 and 1, given the non-negative nature of the data. Additionally, the target variable (price) was also rescaled to the [0,1] range to maintain consistency with the feature scaling.

3 Baseline Evaluation

We use **Root Mean Squared Deviation (RMSD)** as our evaluation metric. Since we have rescaled the prices to be in the range $[0, 1]$, the RMSD is normalized by the range of the target variable. Note that we applied the same rescaling to the *Manheim Market Report (MMR)* values to ensure consistent scaling when comparing the deviations between data points. The MMR, which represents the best in the industry for wholesale vehicle pricing, achieved an RMSD of **3.30%**. This establishes a baseline against which we compare our model's performance.

4 Methods and Results

4.1 Overview

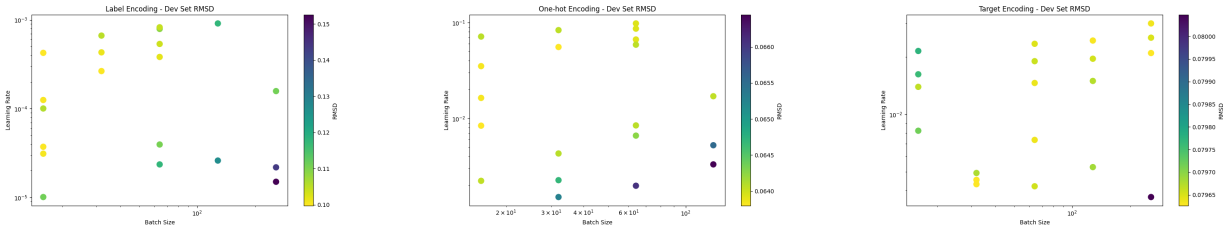
We trained and evaluated four models for the task of used car price prediction: **Linear Regression**, **Generalized Linear Models (GLM)**, **Neural Networks (NN)**, and **XGBoost**. The pipeline consists of preprocessing the dataset, selecting features, training the models, and evaluating their performance using normalized root mean squared deviation (NRMSD) as the primary metric.

4.2 Linear Regression

Setup: We experimented with three types of encodings for categorical features: simple labeling, one-hot encoding, and target encoding. Due to the dataset's size, we trained the model using mini-batch gradient descent.

Hyperparameter Search: We randomly searched for the optimal learning rate and batch size, running 20 experiments per encoding for 80 epochs each. Figure 1 shows the RMSE on the development set for the different encodings.

Results: The best-performing configuration used one-hot encoding with a learning rate of 0.01 and batch size of 16, achieving an RMSD of 6.38% on the development set. Running the model



(a) Labeling Encoding (b) One-Hot Encoding (c) Target Encoding

Figure 1: Random search RMSD on the dev set for the three types of encodings.

with this configuration on the test set yielded an RMSD of **6.41%**. While its performance was close to the MMR baseline, it highlighted the limitations of linear regression for this dataset.

4.3 GLM

Going one step further in complexity, we decided to use Generalized Linear Models (GLM). To determine the appropriate family of distributions, we first attempted to fit several well-known distributions to the target variable y. The distributions considered were Gamma, Gaussian, and Inverse Gaussian. Among these, the Gamma distribution provided the best fit. The plot below illustrates the quality of the fit for the Gamma distribution:

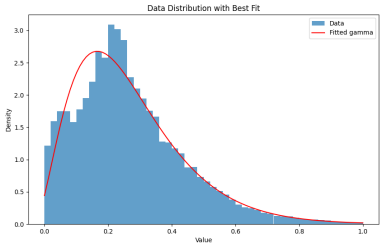


Figure 2: Fit of the Gamma distribution to the target variable.

We used the iteratively reweighted least squares (IRLS) algorithm to solve the optimization problem, as it is well-suited for GLMs with non-Gaussian distributions. This approach directly models the relationship between the features and the target variable while accounting for the distribution of y. The train and development set achieved RMSDs of 4.93% and 5.06%, respectively. Since GLMs do not involve hyperparameters, we directly evaluated the model's performance on the test set as well. There we achieved an RMSD of **4.96%**. These results demonstrate that GLMs provide a significant improvement over linear regression, particularly when the underlying distribution of the target variable is appropriately modeled.

4.4 XGBoost

For the XGBoost model, we implemented a regression approach with automated hyperparameter tuning using Optuna. The model was designed to process both categorical and numerical features. We set up a comprehensive hyperparameter search space, covering several key parameters for XGBoost optimization:

- **Number of trees**: 100 to 1000
- **Tree depth**: 3 to 10
- **Learning rate**: 0.01 to 0.3
- **Sampling parameters**:
  - **Subsample (row sampling)**: 0.6 to 1.0
  - **Colsample_bytree (column sampling)**: 0.6 to 1.0
- **Tree construction parameters**:
  - **Min child weight**: 1 to 7
  - **Gamma**: 0 to 0.5
- **Regularization terms**:
  - **Alpha**: 0 to 10
  - **Lambda**: 1 to 10

The optimization process ran 50 trials using Optuna, aiming to minimize the RMSE on the validation set. The best-performing model achieved an RMSE of 3.79% on the train set and 4.27% on the validation set. We then assessed the model on the test set, yielding an RMSD of **4.48%**. These results indicate that XGBoost performed reasonably well and outperformed both Linear Regression and GLM.

4.5 Neural Networks

We began by training a small neural network architecture, consisting of 4 layers with 16 units in each hidden layer. To optimize this architecture, we conducted a random hyperparameter search over the batch size and learning rate. A total of 10 experiments were run for 20 epochs each, with a learning rate decay set to 0.05. Based on the results, we performed a coarse-to-fine search, running 5 additional experiments for 30 epochs. The best configuration used a learning rate of 0.0004 and a batch size of 16.

Using this configuration, we trained the model for 100 epochs and achieved RMSD values of 3.70% on the training set and 3.86% on the development set. Below is a plot showing the convergence of training and validation losses over epochs:

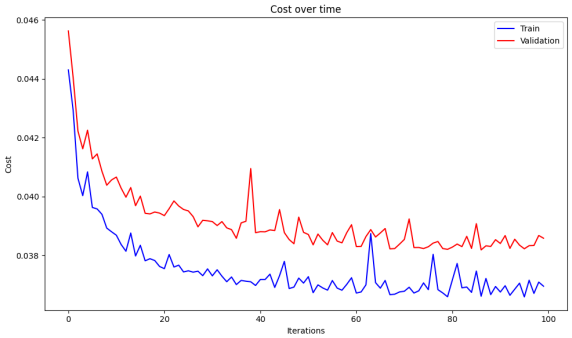


Figure 3: Training and development set RMSD convergence for the small neural network.

When evaluated on the test set, the model achieved an RMSD of **3.71%**, representing a significant improvement over all our previous approaches. This result also considerably narrowed the gap to the MMR baseline.

However, we observed that the training and validation losses plateaued at around the 50th epoch, despite not using any regularization techniques. This implies that the current architecture has reached its expressivity limit. Given that the model was unable to overfit the data even without regularization, scaling up the model's capacity is a logical next step to improve performance further.

Scaling Up To overcome the limitations of our initial implementation in NumPy, which restricted us from efficiently utilizing GPU acceleration, we rewrote the code in PyTorch. This allowed us to train a much larger neural network on an NVIDIA A100 GPU. The new architecture consisted of 3 hidden layers with 32 units each, along with skip connections to improve gradient flow. We incorporated the following techniques to enhance training efficiency and regularization:

- **Batch Normalization**: To accelerate convergence and stabilize training.
- **He Initialization**: For effective weight initialization, reducing vanishing or exploding gradients.
- **Adam Optimizer**: To ensure adaptive learning rates and efficient gradient updates.
- **Mini-batch size**: Increased to 128 for faster training on the GPU.

The model was trained for 30 minutes, achieving RMSD values of 3.62% on the training set and 3.73% on the development set. Below is a plot showing the convergence of the train and development losses over time:

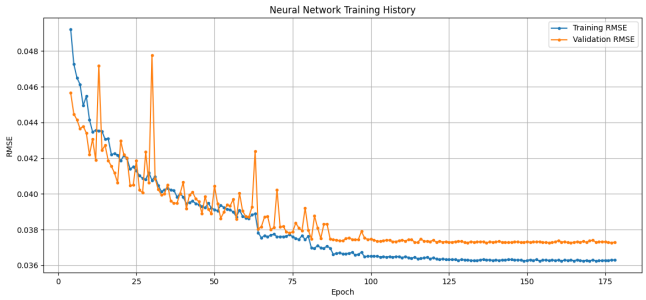


Figure 4: Training and development set RMSD convergence for the scaled neural network.

Since the RMSD on the development set was close to that achieved by the smaller model, we decided not to evaluate this larger model on the test set.

Scaling Further Building on our previous scaling efforts, we decided to train an even larger model to see if it could improve performance further. For this model, we chose hidden layer sizes of 128, 64, and 64 units. We retained key techniques from our prior model, including batch normalization, Adam optimization, He initialization, and skip connections between the two 64-unit layers.

During our previous scaling attempt, we observed a slight gap between the RMSDs of the training and development sets, suggesting a minor variance problem. To mitigate this, we introduced a very small L2 regularization with a regularization strength of $\lambda = 10^{-5}$. We trained this model for one hour on the A100 GPU and achieved RMSD values of 3.60% on the training set and 3.64% on the development set (see plot below). Upon evaluating the model

on the test set, we obtained an RMSD of **3.51%**. These results indicate that the larger model, along with the small L2 regularization, successfully reduced the variance and improved the test performance, suggesting that further scaling may lead to diminishing returns.

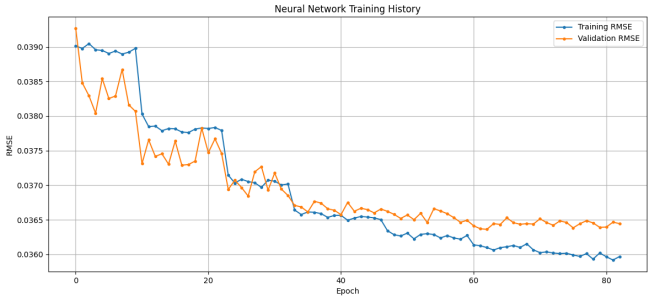


Figure 5: Training and development RMSD convergence for further scaled neural network.

Scaling Even Further To push the model further, we decided to scale it up even more. The architecture now consisted of 4 hidden layers with the following unit sizes: 192, 192, 128, and 128. Skip connections were added between the two 192-unit layers and between the two 128-unit layers to facilitate better gradient flow. We retained all previous optimizations, with the only modification being an increase in the L2 regularization strength to 2×10^{-5}. We trained this model for 2 hours on a single A100 GPU. The resulting RMSD values on the development and test sets were 3.54% and 3.61%, respectively, as shown in the plot below. Upon evaluating the model on the test set, we achieved an RMSD of **3.48%**.

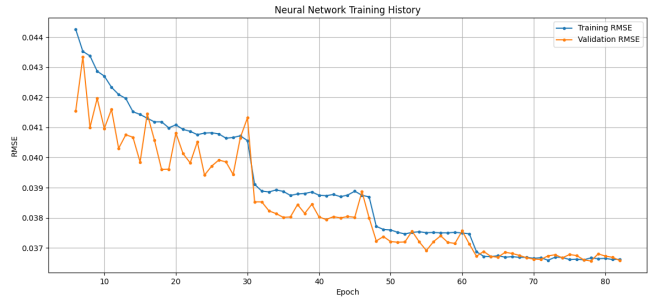


Figure 6: Training, development, and test set RMSD convergence for the further scaled neural network.

Although scaling the model improved performance and allowed it to outperform its predecessor, it still fell short of surpassing the baseline RMSD set by MMR.

4.6 Comparison of Models

A summary of the results is provided in Table 1. Among the models, XGBoost demonstrated strong performance, though it was ultimately surpassed by the neural network models. Linear regression and GLM served as effective first steps but were outclassed by the more sophisticated

approaches. The neural network models, particularly the larger architectures, showed the most promise, closing the gap to the industry-leading Manheim Market Report (MMR).

Table 1: Performance of models and baseline (MMR) on the test set.

Model	RMSD (%)
MMR (Baseline)	3.30
Linear Regression	6.41
GLM	4.96
XGBoost	4.48
Neural Network (small)	3.71
Neural Network (large)	3.51
Neural Network (huge)	3.48

5 Conclusion

In this study, we compared the performance of four distinct machine learning models—Linear Regression (LR), Generalized Linear Models (GLM), XGBoost, and Neural Networks (NN)—on a regression task using a dataset of car prices. Each model was evaluated using RMSD as the primary performance metric, with various preprocessing and encoding techniques applied to the features. Our results showed that linear regression and GLM were effective foundational models but were significantly outperformed by more advanced approaches. XGBoost demonstrated strong performance, while neural networks, particularly when scaled, achieved the best results among the models evaluated. However, even with the most advanced models, the MMR baseline remained unbeaten. Future work could focus on further model refinement, such as fine-tuning hyperparameters, exploring alternative neural network architectures, and leveraging additional feature engineering techniques. These efforts could help close the remaining gap to the MMR baseline and unlock further improvements in predictive accuracy.

6 Acknowledgements

We would like to express our gratitude to the teaching team of CS 229 for their guidance and support throughout the course of this project. We also extend our thanks to Bojan Tunguz, the creator of the Kaggle dataset, for making the data available, which served as the foundation of our work.

References

[1] Tunguz. Used car auction prices dataset. `https://www.kaggle.com/datasets/tunguz/used-car-auction-prices/data`, 2024. Accessed: 2024-12-04.