Blue Book for Bulldozers

Amna Shahnawaz
23030024

Waleed Arshad
24100303

Muhammad Usman Zafar
24100213

Usama Ali
23060018

Usama Tariq Khan
23060016

Amina Waheed
24020136

***Abstract*—This project investigates the applicability of various machine learning techniques for predicting prices in the Bluebook for Bulldozers dataset. The primary goal is to identify the model that yields the lowest Root Mean Squared Log Error (RMSLE), indicating superior predictive performance. The models under consideration include k-Nearest Neighbors (KNN), Linear Regression, Random Forest, Support Vector Regression (SVR), and Neural Network. The study involves thorough preprocessing of the dataset, including handling missing values, feature scaling, and one-hot encoding of categorical variables. Each model is trained, tuned, and evaluated using a consistent experimental setup. Results indicate that the Neural Network model outperforms other models, demonstrating the lowest RMSLE on the test dataset. This finding underscores the importance of considering model selection carefully in predictive modeling tasks, as different algorithms may yield varying performance. The research contributes valuable insights into the suitability of machine learning models for price prediction in the context of bulldozer sales. It provides practical recommendations for practitioners seeking accurate and reliable price predictions in the heavy equipment industry.**

Keywords: Machine Learning, Price Prediction, Random Forest, RMSLE, Bulldozer Sales, Feature Engineering.

I. Introduction

This report analyzes and compares the performance of 5 different models and their efficiency in using a publicly available dataset on Kaggle, a Blue Book for Bulldozers, to predict the sale prices of bulldozers sold at an auction. After applying a uniform set of preprocessing steps to the data itself, the following five models are used in turn: KNN, Random Forest, Neural Networks, SVM, and Linear Regression, with the Root Mean Squared Logarithmic Error (RMSLE) evaluation metric used to assess and determine the most effective model.

II. Evaluation Metric

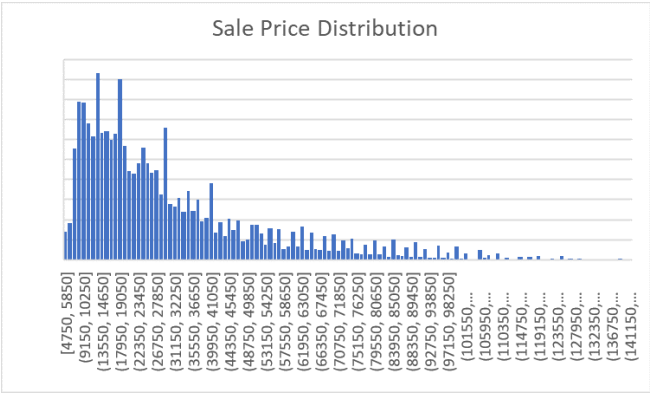
We have chosen Root Mean Squared Logarithmic Error(RMSLE) for our project since it was specified by Kaggle in the original competition and provides us with a way to compare our performance with the competition-winning models. RMSLE is calculated using following:

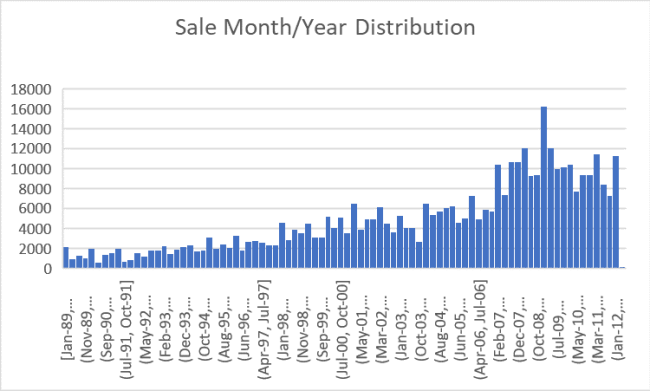
$$\text{RMSLE} = \sqrt{(log(y_i + 1) - log(\hat{y}_i + 1))^2}$$

The model is penalized by RMSLE if the predicted value is smaller than the actual value and vice versa. It does not penalize high errors due to the log. Hence the model has a larger penalty for underestimation than overestimation. This can be helpful in situations where we are not bothered by overestimation but underestimation is not acceptable, which is the case here. Customers should not be given an underestimation of the worth of their heavy equipment fleet at auctions, thus saving them from potential loss.

III. Analyzing the Dataset

The dataset was initially presented at a Kaggle contest, hosted by Fast Iron to create a "Blue Book for Bulldozers", whose goal was to predict the sale price of a bulldozer based on 53 provided features like model, age and usage, amongst others. The data contains three sets: a training set with data till 2011, a validation set containing data from January 1 - April 30, 2012, and a test set with data from May 1 - November 2012. Many values are missing or dubious, like some trucks being sold before they were manufactured, some being sold more than once, others being manufactured in the year 1000 and different years of manufacturing associated with the same Machine Id.





IV. Preprocessing

Accounting for inconsistent or strange behavior as outlined above, each of the following preprocessing steps is applied in turn:

1. ***Removing zeros in machine hours:*** Remove rows with zero values in machine hours current meter, since it incorrectly indicates no usage of the machine
2. ***Remove missing values:*** Replace all missing values in each column with the most frequently occurring value in that column: mode, for minimal distortion and to maintain data integrity
3. ***Dropping pre-1900 data***: Remove rows for which bulldozers were manufactured before the year 1923 (when they were invented), to filter out the year 1000s we encounter in the data
4. ***Remove insignificant variables:*** Remove columns evaluated to not have a significant effect on predictions, e.g if 80% or more of each of the variable's values are null
5. ***Enrich DataFrame:*** Enrich the dataframe with 'saleYear', 'saleMonth', 'saleDay', 'saleDayOfWeek' and 'saleDayOfYear'
6. ***Add new variable for Age:*** Account for age of bulldozer by creating a new variable showing the number of years from the year of manufacture to the Sale Year

V. Models: implementation, results and limitations

Splitting dataset:

When handling time-series data, it's advisable to partition the training, validation, and test sets based on temporal order. This ensures that the model learns from historical data and is tested on future data. For this purpose, we will utilize data up to the year 2011 for training, data specifically from the year 2011 for validation, and data from the year 2012 onwards for testing.

Feature set:

Except for Neural Networks, three different selections of feature sets were used for each of the models.

1. Single Feature*(FS1)*: To use as a baseline for a particular model and to see how a singular feature affects the prediction.
2. Multiple Numerical Features(FS2): To use how adding complexity can make the predictions better
3. Multiple Numerical + Encoded Features(FS3): To put in all the crafted features and see if one-hot encoding of categorical features was indeed useful.

Five different machine learning models are employed for prediction:

K nearest neighbors (KNN):

A non parametric method, KNN makes no assumptions about the underlying data distribution, convenient for when our data does not follow a known distribution. Due to local interpolation, in a dataset where local patterns or similarities strongly influence sale prices, KNN can more effectively make predictions based on local patterns.

Implementation:

We configured a RandomizedSearchCV instance to optimize the hyperparameters of a K-Nearest Neighbors (KNN) model. It involved defining a parameter grid as follows:

```
param_grid = {
    'n_neighbors': range(1, 10),
    'weights': ['uniform', 'distance'],
    'metric': ['euclidean', 'manhattan', 'minkowski']
}
```

The RandomizedSearchCV method sampled 54 different combinations from this grid, testing each combination to determine which yields the best performance. For each feature set, 54 combinations were tested and validation loss for the best model was calculated. In total we tested 54*3 combinations.

Results of Model:

KNN	FS1	FS2	FS3
Validation loss(RMSLE)	0.7613	0.5806	0.4739

The best model used FS3, and it gave an RMSLE of 0.5247 on the test set.

Limitations:

KNN is a fairly simple model and suffers from the "Curse of Dimensionality". It can also be computationally intensive, especially with large datasets, as it involves calculating the distance between data points for each prediction.
However, owing to the fact that KNN offers simplicity and can effectively model non-linear relationships without making assumptions about the data distribution, we used it as a baseline model and to use models that do not have inherent limitations as that of KNN.

Linear Regression

The Linear Regression Model was deemed appropriate as an initial, baseline model to evaluate the simplicity and linearity of our dataset. It is simple, easily interpretable, quick to train, and can provide a strong predictive performance if the relationship between the features and target variables are approximately linear.

Implementation:

We instantiate the linear regression model from the Sklearn library. In the first step, only a single feature of age was taken as input. The model gave the poor results with that single feature. Then in the second step, multiple numerical features were used and the results of the model were improved as compared to the single feature implementation. Then the model was implemented with multiple numerical encoded features and gave some better results. The main idea here is that categorical data (like make, model, or location) can have significant predictive power, as demonstrated in the last approach as well. Using one-hot encoding allows these categorical variables to be effectively incorporated into the model and the model can make good predictions as compared to single or multi numerical features.

Results of Model:

Linear Regression	FS1	FS2	FS3
Validation loss(RMSLE)	0.73	0.69	0.55

The best model used FS3, and it gave an RMSLE of 0.57 on the test set.

Limitations:

The target variable and the input characteristics are assumed to have a linear relationship in linear regression. A linear model might not be able to adequately represent the intricacy of the underlying patterns if the genuine connection is nonlinear. The scale of the input characteristics affects linear regression. Variations in the magnitude of features can impact the performance of the model.

Support vector Regression (svr):

SVMs are effective for datasets with numerous features (as is the case for our bulldozers dataset) and can effectively model non linear and complex relationships using the kernel trick, allowing greater adaptability to real world data like our bulldozer set.

Implementation:

To understand the effect of the model's performance, three different input features are implemented. In the first approach, SVR is implemented using a single variable. In the second approach, SVR using multiple variables is implemented. And in the third approach we have tried encoding categorical variables using one-hot encoding and examined the impact on the model. For feature scaling we used "StandardScaler()" to scale the numeric features. Radial basis function is used for the kernel, which is good for capturing complex, non-linear relationships. For the evaluation metric RMSLE (Root Mean Squared Logarithmic Error) is used. The reason is that the target variable has a wide range. To check if the model is overfitting or generalizing well, the model's performance is evaluated on both the training and validation sets.

Results of Model:

SVR	FS1	FS2	FS3
Validation loss(RMSLE)	0.712	0.7100	0.6995

The best model used FS3, and it gave an RMSLE of 0.7323 on the test set.

Limitations:

In terms of input features, SVR is really sensitive. So, the data needs to be scaled properly before applying SVR on it. The choice of kernel has a great impact on the model performance. So, it should be chosen very carefully keeping in mind the model requirement and condition. Finding the right combination can require extensive experimentation and fine-tuning. SVR has parameters like regularization parameters and kernel parameters that need to be tuned for optimal performance. SVR takes large computational time to train for a large dataset like in our case.

Random Forest:

Our dataset includes many types of variables: time based, numerical and categorical. Random Forest can handle this variety without requiring extensive preprocessing for categorical variables, as it inherently manages different feature types. It is robust to overfitting, is flexible and interpretable.

Implementation:

We configured a RandomizedSearchCV instance to optimize the hyperparameters of a Random Forest. It involved defining a parameter grid as follows:

```
param_grid = {
    'n_estimators': [100, 200],
    'max_features': ['auto', 'sqrt', 'log2'],
    'max_depth' : [4,5,6,7,8],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
```

The RandomizedSearchCV method then utilized this grid to randomly sample different hyperparameter combinations, evaluating each combination's performance through cross-validation with 5 folds. This approach allowed for an efficient exploration of the hyperparameter space, aiming to optimize the RandomForest model by identifying the best set of parameters from a comprehensive yet manageable subset, thereby balancing exploration and computational efficiency. The setup also ensured reproducibility and parallel processing through the random_state and n_jobs parameters.

Results of Model:

Random Forest	FS1	FS2	FS3

Validation loss(RMSLE)	0.7047	0.5248	0.4074

The best model used FS3, and it gave an RMSLE of 0.4686 on the test set.

Limitations:

Random Forest algorithms can struggle with very high-dimensional data, particularly when many features are not relevant. The algorithm is computationally intensive with a large number of trees and deep trees and can also consume a significant amount of memory.

Neural Network:

Neural Networks are sufficient for high dimensional and complex, non linear data. Unlike other algorithms, Neural Networks can automatically detect and learn interactions between features without explicit manual engineering. This is useful when interaction between features (like brand and model year) might influence sale price.

Implementation:

We only used FS3 for NN as it consistently outperformed other feature selections in all the models, and also because we did not have computational resources to try all the combinations. Adam optimizer was used in both architectures and we let the training run for 1000 epochs.

Architecture 1:

We structured the model in layers, with each layer performing a specific function. It begins with a densely connected layer featuring 256 neurons and a sigmoid activation function, integrating both L1 and L2 regularization to mitigate overfitting. This is followed by a dropout layer that randomly sets a proportion of input units to 0 at each update during training time to prevent over-reliance on any one node. This pattern of a dense layer followed by a dropout layer is repeated, gradually decreasing the number of neurons in subsequent dense layers (256, 128, and then 64). Each dense layer maintains the same activation function and regularization strategy. The final layer of the network is a densely connected layer with a single neuron, indicating that this model is intended for a regression task. The use of multiple layers with dropout and regularization focuses on creating a model that generalizes well to new, unseen data, reducing the risk of overfitting.

Architecture 2:

We constructed a series of layers, starting with a dense layer of 512 neurons, using the 'relu' activation function, and employing both L1 and L2 regularization to reduce overfitting by penalizing complex models. A dropout layer follows, set to drop out 30% of the nodes, thereby promoting model robustness and preventing over-reliance on specific neurons. This design pattern of alternating dense and dropout layers continues, with each subsequent dense layer halving the number of neurons (512, 256, and 128) while maintaining the same 'relu' activation and regularization. The architecture culminates in a final dense layer with a single neuron, indicating the model's focus on a regression task. The overall structure, marked by layers with 'relu' activations, dropout for regularization, and a gradual reduction in neuron count, indicates our deliberate approach to balance learning capacity with generalization, aiming to create a model that performs effectively on both training and unseen data.

Results of Model:

Neural Network	Arch 1	Arch 2
Test loss(RMSLE)	0.7857	0.3456

Overall Results

Limitations

Neural networks generally require a substantial amount of data to perform well and avoid overfitting. Training neural networks, especially deep learning models, can be computationally intensive and may require specialized hardware like GPUs. Without proper regularization and tuning, neural networks can easily overfit to the training data, especially if it's not sufficiently large or diverse. Neural networks involve many hyperparameters (like the number of layers, number of neurons, learning rate, etc.) that need careful tuning to achieve optimal performance.

VI. Discussion

There are additional avenues for preprocessing and cleaning the dataset that may influence prediction efficiency, and are worth mentioning here. For example, the 2008 recession, lasting from December 2007 to June 2009, may have influenced sale prices to deviate significantly in a systematic way for years impacted by the economic crisis, leading to a possible bias in our prediction. Techniques and models could be modified to learn our models on different periods, for example by removing the year 2009 which was impacted by the economic crisis.

For this project, we used KNN and linear Regression among the simpler algorithms, and with each approach we introduced more complexity, moving from SVM to Random Forests and ultimately to Neural Networks. The best performing model was Neural Network using ReLu. This can be attributed to the ability of NN to learn the patterns and useful relations between features from data and use it to make good predictions.

As chatGPT would narrate this, “Embarking on the "Bluebook for Bulldozers" project, I assembled a veritable 'Avengers' team of machine learning algorithms, each bringing its unique superpowers to the fore. There was the neighborly KNN, always keen to learn from those closest to it; the linear regression, a straight shooter adept at finding the simplest path; the SVM, cutting through data complexities with the finesse of a master swordsman; and the decision tree, branching out its knowledge with sage wisdom. But in this ensemble of algorithmic might, it was the neural network that emerged as the Tony Stark of the group. With its intricate layers and deep learning prowess, it outshone its peers, triumphing not just in performance but in showcasing the sheer power of modern AI. In this computational saga, neural networks proved, yet again, that when it comes to tackling complex, real-world problems, they often have the upper 'node'.”

References

[1] https://github.com/alzmcr/kaggle-Fast-Iron

[2] https://www.thoughtco.com/history-of-the-bulldozer-1991353

[3] http://webmining.olariu.org/trees-ridges-and-bulldozers-made-in-1000-ad/

[4] https://www.kaggle.com/c/bluebook-for-bulldozers/discussion (All the discussions)