Car Scam Prevention

ECS 171 Machine Learning

Group 21 Project Report

Group Members:

Clayton Chan, Juan Diaz, Saili Karkare, Jiawei Zheng

Github Repository:

https://github.com/jiaweizheng1/ECS-171-Term-Project-21

Introduction and Background

In this day and age, it is very common for people to buy secondhand cars. However, it is also very common that people are naive when it comes to the car market and can easily get scammed/ripped off when buying a secondhand car.

This gave us the inspiration to build some algorithms that will prevent these kinds of scams from happening in the first place. In order to prevent these scam, we needed something to help us accurately predict the car's real value in terms of price and also a classifier that determines whether a car is in relatively new condition or if it was old.

That way, we can have some sort of means to detect when someone is trying to scam you. For example, if the seller's price is a lot higher than what the algorithm predicted, that should set off an alarm that the seller is trying to scam you. Alternatively, if someone is trying to pass off a car as new but the algorithm predicts it to be on the older side, that could be another sign of a potential scam to help you not fall for it.

Literature Review

In the field of auto insurance fraud detection, previous research has primarily focused on using structural data for modeling. However using natural language processing and computer vision techniques is largely unexplored. To address this gap, Yang et al. developed the Auto Insurance Multi-modal Learning (AIML) framework. This framework incorporates text and visual data to improve fraud detection efficiency and performance. The authors conducted experiments using a real-world auto insurance dataset consisting of 4,946 cases, including both non-fraud and confirmed fraud cases. They compared the performance of AIML with a baseline model that only utilized structural data. The baseline model achieved an overall accuracy of 0.8364, with precision of 0.7095, recall of 0.4441, and F1-score of 0.5462. The model with the combined text and image processing increased accuracy to 0.8713, precision of 0.7143, recall of 0.6107 and F1-score of 0.6584. This shows how good their baseline is compared to our models using structural data and also shows how much our data could be improved by further extending the modes of data that we focus on.

Gupta et al. conducted research on fraud detection in car insurance. They focused on developing classification models to determine fraud in car insurance. The data was split into a 70:30 ratio for training and testing purposes. The primary model used in their study was a Multi-Layer Perceptron (MLP). They experimented with various parameters, such as batch size and the number of layers in the discriminator, to improve the performance scores. The researchers also addressed the slight data imbalance issue in their report.

Dataset Description and Exploratory Data Analysis of the Dataset

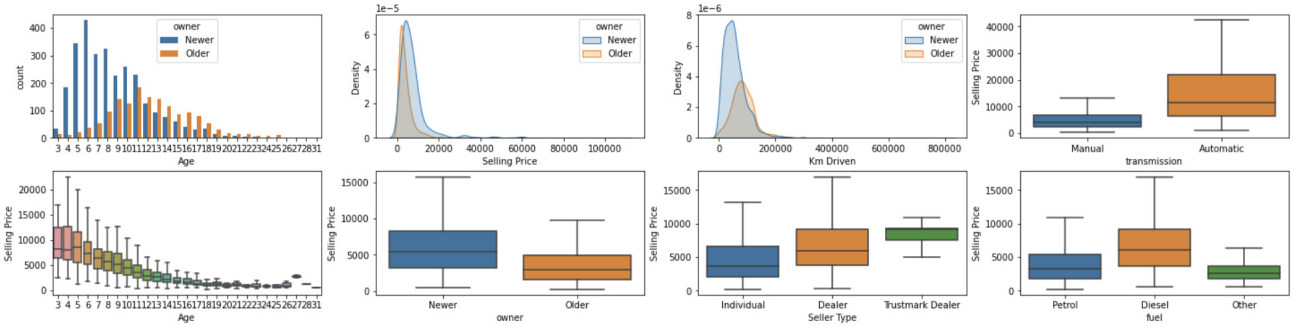
The dataset is taken from Kaggle and sourced from CarDekho. CarDekho is a popular company and website based in India that specializes in helping users buy cars that are right for them. Their website is rich with automatic content such as expert reviews, car specs and prices, and comparisons as well as pictures of all car models available in India. CarDekho also has ties with many car manufacturers and about 4000 car dealers to facilitate the purchase of vehicles.

The attributes of the dataset are

Name - The name of the car, a nominal variable. We ended up dropping this column because there are simply too many unique names. We feel, because of the high variance, it could probably cause our models to perform worst.
Year - The year the car was made, a numeric variable. We transformed the year column into a column representing the car's age in years.
Km Driven - The mileage on the car, a numeric variable.
Fuel - The type of fuel the car uses, a nominal categorical variable. It has 3 levels which are Diesel, Petrol, and Other.
Seller Type - The seller type describes what kind of person/organization is selling the car. This is a nominal categorical variable with 3 levels being an Individual, Car Dealership, and Trustmark Dealership.
Transmission - The transmission type of the car. This is a nominal categorical variable with 2 levels, Automatic and Manual.

We have 2 response variables of interest being

Selling Price - How much the car was sold for, a numeric variable.
Owner - The number of previous owners the car had. This was originally an ordinal variable with 4 levels, One Owner, Two Owners, Three Owners, and Four+ Owners, however, due to the strong imbalance of cars having only one previous owner we decided to combine any car with two or more previous owners into one class. Owner is now a binary variable with labels 0 representing that the car is in newer condition with one previous owner and 1 representing that the car is in older condition with two or more previous owners.



Visualizations:

We can see a noticeable right skewness regarding the age for newer cars. This means that the newer cars tend to be more young. Meanwhile we can see that the older cars tend to have less young cars and noticeably more older cars compared to the newer cars and seems to be more symmetrically distributed.

Based on the density plot, we can see that newer cars tend to be more expensive on average compared to older cars. Notice how there's more density for higher prices. We can also see that the newer cars on average have less mileage on them compared to the older cars.

Automatic cars tend to be more expensive on average. We also see that manual cars have less spread compared to automatic cars.

We can see that there's a general trend where selling price tends to go up the more recent the car was made. Also notice that the more recent the car was made the more variation and wiggle room you have for the price.

The selling price is more pricey for cars with less owners compared to cars with more owners on average. Cars with less owners also have more spread in price.

Individuals tend to sell their cars at a cheaper rate on average as opposed to Trustmark dealers being the most expensive and with dealers somewhat in the middle. Trustmark Dealers also have less spread compared to the other two types of sellers.

Diesel cars tend to be the most expensive on average but also the most spread out. Then with petrol cars and the middle in terms of price and spread and finally other fuel types being the least expensive and spread out.

Pre-processing

The data set was tidy and seemed to have no spelling errors and missing values. However, there was work to be done regarding feature engineering.

The pre-processing may vary slightly depending on the model but here are the methods we used that applies to every model.

When it comes to dealing with the categorical variables Transmission, Seller Type, and Fuel, since

they are nominal and have no particular order, we applied one-hot encoding.

We also converted the seller price from INR to USD by dividing by roughly 82.

Also, we transformed the year column into a column representing age by subtracting from 2023.

Any other methods applied unique to one model will be briefly discussed in the respective section.

Proposed Methodology

Regression for Selling Price

Since the selling price of the car is continuous, we are dealing with a regression problem where we're predicting a continuous variable. We've selected some models fit for regression which included linear regression, a neural network, and random forest. The overall goal is to minimize the bias and predict a car's price close to its actual worth. We'll also compare these models based on their R^2.

Classification for No. of Owners

On the other hand, the condition of the car (new vs old determined by the number of previous owners) is categorical hence making this a classification problem. Also, the response is binary which opens up a lot more options. This allows us to try models such as Logistic Regression, k-NN, and a Neural Network. To compare these models, we'll focus on the accuracy and the recall rate for old cars (label 1). We want to minimize the cases of false negatives where old cars are predicted as new because we're interested in scam prevention where we don't want consumers paying more money for an old car labeled as new. We don't care too much about the recall rate for new cars (label 0) that much as a false positive is not as bad as a false negative. A false positive in this context means that a new car is classified as old. If that's the case, the consumer is more likely to turn away cars that are perceived as old and won't really lose money or get scammed.

Experimental Results

Table 1: Metric Comparison for All Regression Models

Model	R^2	MSE
Linear Regression	0.6889	0.0429
Random Forest	0.7403	0.0358
Neural Network	0.6945	0.3167

Linear Regression

Linear Regression was done using the built in function from sklearn. Prior to running the model, we decided to log both the selling price and KM driven attributes to make them easier to work with and visualize. As depicted in the code, linear regression showed a negative correlation between the

age of a car and its selling price and the KM driven and the selling price. This model had an R^2 score of 0.6889 and an MSE of 0.0429.

Random Forest

In order to run Random Forest to the best of its ability and make the most accurate predictions, we use Grid Search to tune our hyperparameters. The hyperparameters considered were: the maximum depth, the maximum number of features, the maximum number of estimators. For these hyperparameters, these were the best results found: 10, 2, and 94. The n jobs parameter was incorporated to make running the model faster. This tuning showed that an average depth of the tree and a minimal number of features proved to have the best predictions. This model proved to be the best with the highest R^2 score of 0.7403 and the lowest MSE of 0.0358.

Neural Network

For the Neural Network, the hyperparameters were: activation function, hidden layer sizes, alpha, and the maximum number for iterations. The parameters found were: relu, 0.01, 8 or 6, and 1000, which shows that approximately 8 or 6 hidden layers and the relu function was the best function for this model. This model had an R^2 score of 0.6945 and an MSE of 0.3167.

Table 2: Metric Comparison for All Classification Models

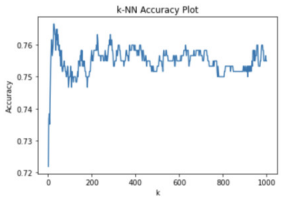
Model	Accuracy	Recall-0	Recall-1
k-NN	77%	73%	81%
Logistic Regression	73%	69%	78%
MLP	78%	77%	79%

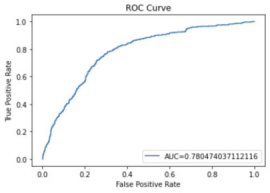
k-NN

For k-NN, the pre-processing involved was scaling all the numeric variables with standardization. Then we decided to balance the data out with undersampling. We didn't want to oversample because the duplicated observations will introduce bias where they'll have 0 distance from the same point and cause overfitting. Despite losing information, we can improve upon this area by collecting more data in the future.

The main hyperparameter of interest is k, how many nearest neighbors are we looking at relative to our observation to help determine a classification? In order to choose the best k, we did a grid search where k can be from 1 to 999 excluding even values as we don't want any ties for our model given that the response is binary. We trained the model for each k and then computed the testing accuracy for each k. We then found that k=25 had the highest testing accuracy.

Out of all the models, k-NN performs the best in the Recall-1 metric. It has a high accuracy of 77% classifying 77% of the observations in the testing dataset correctly. Also, it's very good at detecting old cars with a recall rate of 81% for class 1 where it classifies 81% of the cars with the old label in the testing dataset properly.





Logistic Regression

For logistic regression, we also applied standardization but also applied oversampling to balance the classes as there were less older cars compared to the newer ones.

First, we created the ROC curve to determine the performance of logistic regression and found the AUC to be roughly .78 and therefore conclude that the model performs fairly well but not outstanding at separating the two classes. And from this ROC curve, we found the optimal threshold by maximizing the true positive rate - false positive rate which is around .4704.

Logistic Regression also had fairly similar results to k-NN with only a slightly lower accuracy at 73% and recall rate for class 1 at 78%.

MLP

For MLP, multilayer perceptron, the preprocessing involved is exactly the same as the preprocessing done for k-NN via standardization. Then we decided to balance the data out by oversampling the samples where a car had 2 or more owners. Finally, we call ".fit" on the MLP neural network with the training data set, which trains the neural network for us by performing the necessary feed-forward pass, stochastic gradient descent(via Adam), and backpropagation per training sample.

Further, to squeeze a little bit more accuracy out of our MLP model, we took advantage of hyperparameter grid-search. For example, we grid searched different numbers of hidden layers and found that any hidden layer sizes of more than 1 do not really yield more accuracy. Further, we also grid search different activation functions, max_iterations, and learning rates(alpha) to find the best hyperparameters for our model that will lead to the highest accuracy.

Out of all the models, MLP performs the best in the accuracy metric with an accuracy of 78%. However, for Recall-1, it falls a little bit behind k-NN with 79%.

Flask Front-End

```
...
├── models
│   ├── Classification_MLP.pkl
│   ├── Classification_knn.pkl
│   ├── Regression_LinReg.pkl
│   └── Regression_RF.pkl
├── templates
│   └── car_predict.html
└── app.py
```

Flask is a web framework for building web applications in Python. It provides a simple API to handle various HTTP methods like GET and POST, and we can easily collect or access data from incoming requests.

After we trained all the models, we save them with pickle to a file directory called "models/" with their respective names, say "Classification_MLP.pkl". To simplify the demoing process, we decided to only showcase the top two best performing models for each type, classification and regression.

Then, "app.py" contains most of the code for the implementation of the font end. On the initial startup of "app.py", we use pickle to load up all the saved models and assign them to variable names so we can call on them later. Then, we POST the initial template of "car_predict.html" to the local webserver on port "localhost:5000", which just prompts the user to select any of our models. After the user selects a model, we rerender the template to prompt the appropriate inputs for that model. In the case of "Classification_MLP.pkl", for example, it would be age, selling price, kms driven, fuel type, seller type, and transmission type. Once the user type those inputs and click submit, we would GET the input data from the html, preprocess the input data(because the models were trained on preprocessed input data), and then message-pass them into the corresponding model they selected to make the prediction. Finally, we POST and rerender the "car_predict.html" template to include the prediction. When necessary, when doing regression or predicting car prices for example, because we preprocessed the selling price with log_10 before they are fed into the models, we have to perform the inverse (10^p) on the output predicted selling price of the models before we send them back to the user through the web framework.

For the drop-down list in "car_predict.html", we use html "form" (for collecting user input) with "select" and "option" to build it. Additionally, the switch between different display inputs for classification and regression models is done by using two "forms" and we hide one or the other with "style = 'display:none'" depending on the value of a string variable called "model" which is set when the user clicks on models from the models drop-down list.

Conclusion and Discussion

Through using the dataset "CAR DETAILS FROM CAR DEKHO", which contains many different attributes of cars, their number of previous owners, and their selling price, and 6 different models(Linear Regression, Random Forest, Neural Network, k-NN, Logistic Regression, and MLP), we were able to build a model with high R^2 of 0.7403 and low MSE of 0.0358 for regression using Random Forest and a model with an high accuracy of 77% and high Recall-1 of 81% for classification using k-NN.

For front end, we used Pickle and Flask to implement a simple-to-use and interactive web app. The users can select, from a drop-down list, the top two models we trained for each type, regression and classification. Then, they can type in the respective input attributes of the particular car they are interested in knowing more about and click "submit". We will process their request, message-pass it to the model they selected to obtain a prediction and then we send the prediction back to the user through the web app so they can see it.

Some limitations of our project begins with the fact that these are cars only from India therefore, it doesn't generalize to the entire car market very well. This brings us to the next point where if we were to improve upon this project, we would collect more data. Not only would we collect more observations, we would also think about which additional features to collect data on because our regression models had a lot of unexplained variance in the R^2 values. Also, there was one feature that we had trouble utilizing which was the car name. There were too many car models to make this attribute useable so maybe we could've done more feature engineering to include this variable.

References

Dataset:

Unfortunately, the dataset we used was taken down from Kaggle but the dataset is still available in our repository.

Literature Review:

Yang, J., Chen, K., Ding, K., Na, C., & Wang, M. (October 2022). Auto Insurance Fraud Detection with Multimodal Learning. https://direct.mit.edu/dint/article/5/2/388/114793

Gupta, X., Mudidgonda, N., & Baruah, R. (February 2021). TGANs with Machine Learning Models in Automobile Insurance Fraud Detection and Comparative Study with Other Data Imbalance Techniques. https://www.researchgate.net/publication/349095432