Data Science 3: Big Data Analytics

**Car Feature Predictions Using Spark and Databricks**

By Mike Coppola, Raghav Kumar, Christopher McQuarrie, Ruchika Ruchika, Jagdish Saggu

horizontal line

# Objective

In this project, we have analyzed the Car Features dataset from Kaggle [[1](#refer_kaggle)]. The data includes car features like Make, Model, Year, Engine, etc which are common to cars on the road today. We have tackled the dataset using both Classification and Regression techniques. The classification models would predict whether the car belonged to the high- or low-end categories in terms of price. The regression models followed the same strategy but modeled the price of the car directly from the features. A sample data is included below.

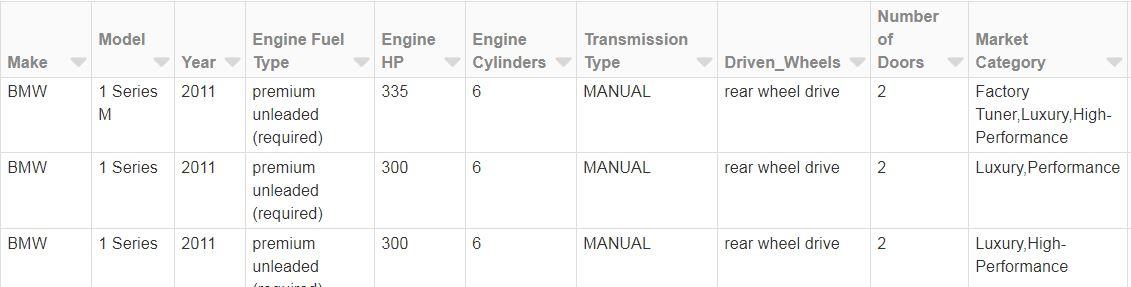


Figure 1: Sample from the Car Features Kaggle Dataset

# 

# Analysis

## Support Vector Machine Classification

Support Vector Machines (SVM) can be used for a variety of machine learning tasks. In this case, a linear classifier (LinearSCV from the pyspark machine learning classification library) was chosen to predict the price class of a vehicle given the input features in the dataset. This linear classifier attempts to draw a hyperplane through the dimensions of features in the set, which will maximize the division in the data (value vs. expensive). The mean of the MSRP was chosen as the classification point. It should be noted that the distribution of the MSRP was not normal; therefore, the mode of the MSRP was also used as the classification point for comparison, but it is not included in this report as the difference was negligible.

Initial tuning of the classifier with all features (1000+) enabled resulted in an ROC AUC of 0.99 and an F1 score of 0.89. The fitting of this model took approximately 4.5 minutes. Unfortunately, automated feature selection was not possible with the LinearSVC library, as it is an experimental library in pyspark and is not as rich in features as other, more mature, libraries. In addition, the evaluator for the LinearSVC model only outputs the area under the ROC; therefore, a confusion matrix and F1 score has to be calculated manually in order to properly asses the fit, especially given that the output variable, MSRP, was not normal.

In a linear SVM classifier, the weight of each coefficient from the resulting model can be used to determine the importance of the feature relative to the other features. Higher negative coefficients have more weight to classify the vehicle as value, and higher positive coefficients have more weight to classify the vehicle as expensive. In order to determine the most important features, the largest 500 coefficients (absolute value) were extracted and plotted.

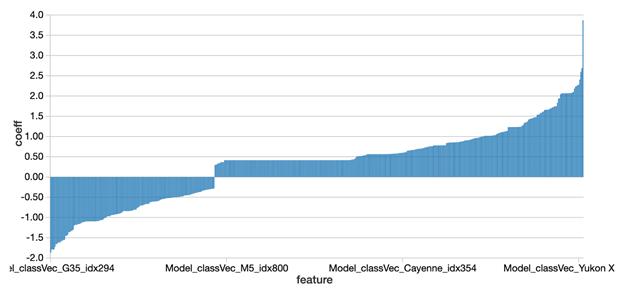


Figure 2: Top 500 Features from Initial SVM Fit

The above showed that the Vehicle Model feature for each vehicle was the most influential in determining the class. The pyspark feature library function, VectorSlicer, was used to select a subset of these most important features from the original feature list and run the fit again. The reduced feature set fit resulted in an ROC AUC of 0.98 and an F1 score of 0.89; however, the time to run the model, approximately 5 minutes, took longer than the original fit.

Since Vehicle Model was so prominent in the top features, a fit was attempted with this feature excluded from the inputs. This fit resulted in an ROC AUC of 0.98 and an F1 score of 0.89, and took approximately 2.5 minutes to run. The same methodology was then used to extract the top 150 features from this fit.

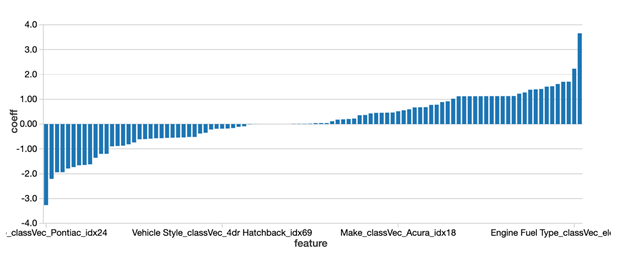


Figure 3: Top 150 Features from SVM Fit without Vehicle Model

With Vehicle Model removed, Vehicle Make was the most common type of feature that was influencing the classifier (as seen above). Some of the other notable features that were important in classifying the vehicle as expensive were the following: Engine Fuel Type and Market Category of Exotic, High-Performance, or Luxury. Other important features to classify the vehicle as value were the following: Vehicle Style such as Midsize or Compact and Market Category of Hatchback.

A final fit was performed using the top 150 features (which already excluded the Vehicle Model features) above. The result was an ROC AUC of 0.98 and an F1 score of 0.88 and took approximately 2.6 minutes.

As can be seen, the Support Vector Machine classifier is a robust and versatile model that performs well with large or small features sets. Its only downside when used in pyspark is that it is not as feature rich as most other modelling tools and requires manual intervention to evaluate models and perform feature selection.

## Logistic Regression (Classification)

Several criteria were used to get the best model for accurate prediction. The first main criterion was the Elastic Net parameter. The second criterion was the largest number of iterations to run. Each criterion had three values each, for a total of nine different models.

The Elastic Net Parameter was used so that we can regularize the model depending on which of L1 or L2 loss is more predominant. This will ensure that we don’t incorrectly focus on either L1 or L2 losses. This criterion was swept from 0.25 to 0.75 in increments of 0.25 (0.25, 0.5, 0.75).

The second criterion was the maximum number of iterations. We wanted to see how the model predicts based on the greatest number of times it’s allowed to “learn” with the training data. The maximum number of iterations was set from 10 to 30 in increments of 10 (10, 20, 30).

Table 1: List of Logistic Regression models (Green is best model)

|  |  |  |
| --- | --- | --- |
|  | Elastic Net Parameter | Max # of Iterations |
| Model 1 | 0.25 | 10 |
| Model 2 | 0.25 | 20 |
| Model 3 | 0.25 | 30 |
| Model 4 | 0.5 | 10 |
| Model 5 | 0.5 | 20 |
| Model 6 | 0.5 | 30 |
| Model 7 | 0.75 | 10 |
| Model 8 | 0.75 | 20 |
| Model 9 | 0.75 | 30 |

Once we decided what the two criteria values were going to be, we built a parameter grid with them. Logistic Regression was then applied with each of the nine models (as seen above in [Table 1](#tab_logr)) using a 3-fold Cross Validation technique. We used this method to prevent overfitting the model. We then selected the best model.

The model that gave us the best results had an Elastic Net Parameter of 0.25 with maximum number of iterations of 30. This model was then analyzed to check the ROC curve and accuracy.

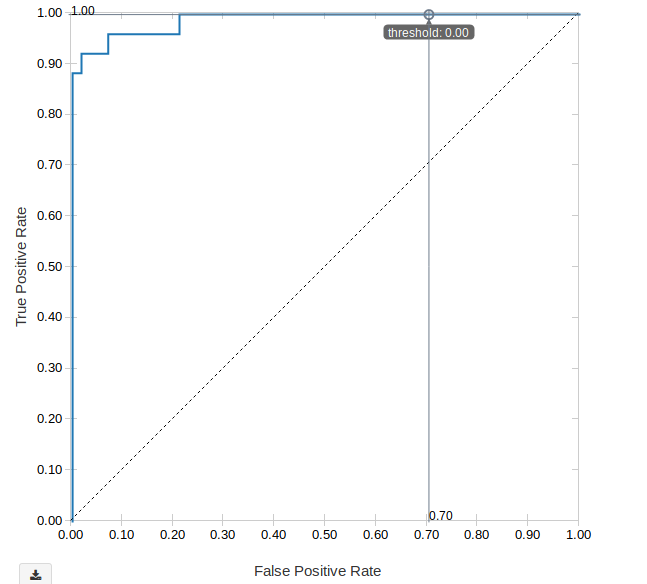


Figure 4: ROC Curve on test dataset using Logistic model with ENP = 0.25 and MaxIter = 30

We calculated the area under the ROC curve (ROC AUC), area under the PR curve (PR AUC) and accuracy of the model. The model learned from the training set. It then predicted the binary results with the test dataset. We wanted to compare whether the model was prone to overfitting the data; we expected cross validation to mitigate this in training.

Table 2: Area under ROC/PR Curves and Accuracy (Initial Fit)

|  |  |  |  |
| --- | --- | --- | --- |
|  | Training Data | Test Data | % Difference |
| Area under ROC Curve | 0.9921 | 0.9296 | 6.30% |
| Area under PR Curve | N/A | 0.8678 | N/A |
| Accuracy | 0.9543 | 0.9470 | 0.77% |

There’s a more dramatic decrease in ROC AUC than Accuracy. Yet, neither difference suggests the logistic regression model is overfitting the data. This is because we see that for the test data our model still pulls off predictions in the low 90s for both metrics.

The PR AUC was also calculated to contrast against the ROC AUC. The PR AUC calculation was unavailable in Spark for the training dataset. But, we can see a 6.65% difference between the ROC AUC and PR AUC when we used the logistic model with the test dataset. This tells us that ROC alone is insufficient in gauging the strength of our model (a fact corroborated in [[2](#refer_link1)].

In the next part of the analysis, we removed unwanted coefficients. We did this to check the impact of such removals on the performance of the logistic model. The original logistic regression model had about 1000 coefficients, making it impractical.

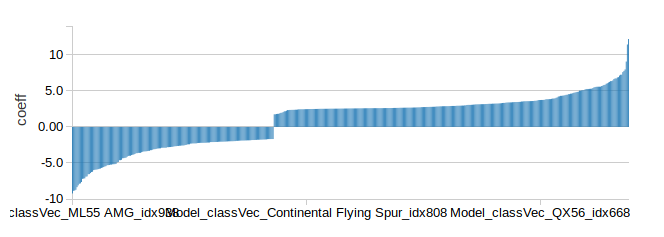


Figure 5: Top 500 Features from Initial Logistic Fit

After removing nearly half the coefficients (from 1000 to 500), re-training the model and testing it, we found the performance to be slightly better than the initial fit with all coefficients.

So, removing nearly half the coefficients resulted in a slight boost in performance with the test dataset. We wanted to glean the consequence of further reducing the number of coefficients to 150.

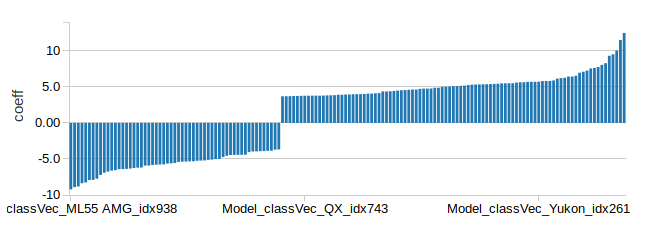


Figure 6: Top 150 Features from Initial Logistic Fit

After using 150 coefficients, we saw no difference in the ROC AUC; accuracy and PR AUC increased only slightly. So, we could still get away with using only 150 coefficients and getting a better result with this model.

Notice that the top 150 features contain the car’s Model (from x-axis of Figure 6). On probing deeper, we found that Model makes up a large proportion of the top 150 features. This suggests the Model feature plays a crucial role in prediction. So, if we removed the Model feature, this would undermine the logistic regression model’s predictive ability.

On the other hand, in the top 500 features, Vehicle Style rarely appeared as a predominant predictor. So, as a final step we need to check the effect of removing the Vehicle Style feature from the list and calculating the impact on the logistic model’s ROC AUC, PR AUC and accuracy.

Removing the Vehicle style from the feature list only reduced the ROC AUC by 0.73% (0.9353 to 0.9285) and reduced the PR AUC by 0.69% (0.8850 to 0.8789). The accuracy decreased by 0.46% (0.9511 to 0.9467), a fraction in terms of actual performance on test data.[[1]](#footnote-0)

So, removing Vehicle Style reduced the overall performance of the logistic regression model by tiny increments, but benefited the model in terms of the resources it needs to make accurate predictions on par with 500+ coefficients. Other features like Vehicle Style should also be checked and removed, but this will be part of future analyses of this dataset.

## Linear Regression

Going through analysis of the data set, we can see that we have both numerical values and categorical values. We have created two different models one with using only the numerical values other with converting the categorical into numerical using StringIndexer, and VectorAssembler.

Starting with linear regression, we set the feature columns to be just the available numerical columns. Which were: 'Year', 'Engine HP', 'Engine Cylinders', 'Number of Doors', 'highway MPG', 'city mpg', 'Popularity'. Since we know that the Make of the vehicle has a huge impact on MSRP, we are expecting a bad model, and since they are categorical pieces of data we are not including them at this point. Case in point we know that Bugatti has an average MSRP of about 1.7M, Maybach with average of 546k, and in the low point we have Plymouth at 3k. So, it is not hard to see that the linear regression without usage of the categorical values will give us a bad model. See below for graph.

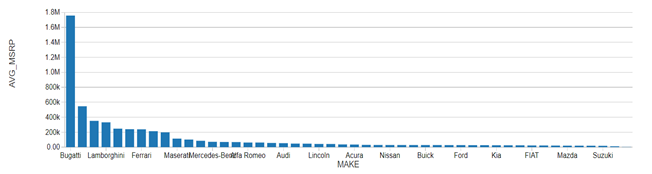


Figure 7: Average MSRP (price) of a vehicle versus the make of the vehicle

After running through the model, we set ranges against the predicted value and the actual MSRP, since getting a MSRP to be exact will be very difficult. We tried to use a 20% range, If the predicted value fell in between MSRP – MSRP\*0.2 and MSRP + MSRP\*0.2 then the model was successful in prediction. See below for code.Even with a large range such a 20% of the MSRP we still get a model which is only accurate for about 23%.

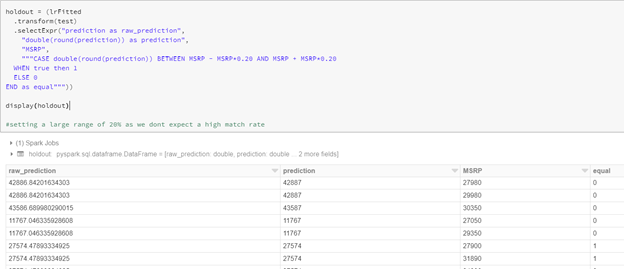


Figure 8: Holdout Data used to calculate MSRP prediction accuracy

Since this model was not very useful we decided to try using the stringIndexer to convert categorical data to usable points of data in a linear model. The full set of columns we are using now are: Make, Model, Year, Engine Fuel Type, Engine HP, Engine Cylinders, Transmission Type, Driven\_Wheels, Number of Doors, Market Category, Vehicle Size, Vehicle Style, highway MPG, city mpg, Popularity. We expect a much better model this time as we are including all columns provided. We know the biggest contributors to the model should be Model, Make, Engine HP, which are all now included.

Again, running through the same steps, we fit the data and tested against the 20% range of MSRP. We saw a big improvement. We see our model is now about 82% accurate in the predictions of MSRP using all available columns. Since we know 20% range is not useful for us in use in real life. Setting the range to be 10% is a little more feasible, i.e. MSRP between MSRP - MSRP\*0.10 AND MSRP + MSRP\*0.10. After running the below we can see the model accuracy has decreased to about 58% which is still better than what we had before.

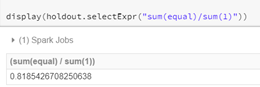


Figure 9: Accuracy of model for +/- 20% range

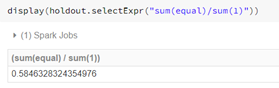


Figure 10: Accuracy of model for +/- 10% range

Dropping the range to only 5% which is more feasible (MSRP between MSRP - MSRP\*0.05 AND MSRP + MSRP\*0.05), we see a huge drop in accuracy, we get about 34%. If we want to improve the model we can try removing columns which may not be as useful for us, like Number of Cylinders. If we already have HP, it may not be needed. City MPG probably does not impact price as much either. We will try other modeling methods to improve model accuracy.

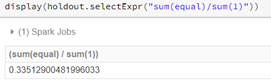


Figure 11: Accuracy of model for +/- 5% range

Summary of linear regression models below, as we can see the accuracy drops drastically as the range is decreased. Which is expected. Even at 5% range we still have higher accuracy when compared to Model 1 which doesn't use Categorical Data.

Table 3: Accuracy results as tolerance window changes

|  |  |  |
| --- | --- | --- |
| Model | Range +/- MSRP | Accuracy |
| Model 1 without Categorical | 20% | 23% |
| Model 2 with Categorical | 20% | 82% |
| Model 3 with Categorical | 10% | 58% |
| Model 4 with Categorical | 5% | 34% |

## Random Forest Regression

The Random Forest Regression model was used to predict the price of a vehicle based on the given features. All 15 features were used to predict MSRP of different cars, as was the case with the previous section on Linear Regression.

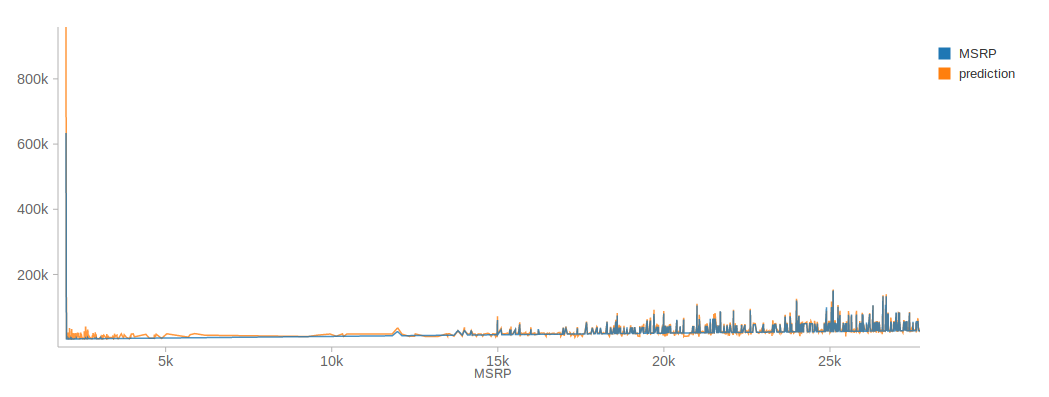


Figure 12: MSRP predictions from Random Forest Model

For this model, the most important criteria are maximum depth, number of trees and maximum number of bins. We set each criterion to two different values so that we could build a parameter grid and pass off to cross validation. Please see the table in Appendix A for more information on the different criteria.

After choosing the best model through cross validation, we made predictions on MSRP (price) using this model. The model only predicted the exact price 0.028% of the time. The rest of the time it was more practical to check within a given tolerance range whether the random forest regressor was close enough to the actual price.

Based on Figure 12 above, it is important to note that the random forest regressor did not perform particularly well on cheaper cars (we see much more variance on the left side of the graph). On the other hand, when we take a look at the right side of the graph, we see that the prediction picks up as there’s more variation in price on the more expensive side.

Table 4: Accuracy results as tolerance window changes

|  |  |
| --- | --- |
| Tolerance | Accuracy |
| 0% | 0.028% |
| +/- 5% | 24.44% |
| +/- 10% | 48.51% |

The accuracy in predicting the MSRP of a car is low because of the large variation in price across this dataset. We can experiment with larger values for tree depths and bins as well as with a larger number of trees. Another way to increase accuracy is to experiment with removing the least beneficial coefficients (those that have values less than 0.01). This will be undertaken in future and is beyond the scope of this project.

## Gradient Boosted Tree Classifier

We have used gradient-boosted trees (GBTs) classifier to predict if the price of the car is above the mean value or below the mean. The data is split into 70:30 and model was tested on held-out test data. The results from different models are included below:

Table 5: Results of the Gradient Boosted Tree Classifier

|  |  |  |  |
| --- | --- | --- | --- |
| Model ID | Number of Trees | Error in Prediction | Area Under Curve |
| 1 | 10 | 5.69% | 98.17% |
| 2 | 20 | 5.05% | 98.43% |
| 3 | 30 | 5.01% | 98.57% |
| 4 | 40 | 4.91% | 98.68% |
| 5 | 50 | 4.83% | 98.73% |
| 6 | 60 | 4.83% | 98.78% |
| 7 | 70 | 4.83% | 98.80% |
| 8 | 80 | 4.75% | 98.83% |
| 9 | 90 | 4.77% | 98.85% |

Studies have shown that GBTs are more prone to overfitting than RF.

# Conclusions

## Classification

In this project, we used three different models to classify whether a car, based on its MSRP (price), falls into the higher, more expensive end (1) or the lower, inexpensive end (0). The mean MSRP of all data points was calculated and then used as the decision boundary to differentiate between these two classes.

The three classification models used were Support Vector Machines (SVM), Logistic Regression (LR) and Gradient Boost Tree Classifier (GBT). With all 16 features taken into consideration, SVM showed an ROC AUC of 99%, LR showed 92% and GBT showed roughly 98%. From these results we may summarise that SVM and GBT are the best models to choose when classifying expensive vs inexpensive cars. Logistic Regression can also be used, but it falls a few points short for predictions.

In the case of SVM and LR, we also experimented with fewer coefficients to see if this would affect the performance of the two models. For SVM, when the total coefficients were reduced by half (1000+ to 500), this only reduced the predictive ability by one percentage point. A further reduction from 500 to 150 did not show any major decreases in predictive ability. Even after removing the Model feature, one that was quite crucial to the prediction, SVM showed no drastic reduction in predictive ability, staying a constant 98%. In the case of the LR model, the same procedure produced similar results. The ROC AUC remained stable at roughly 93%. The Vehicle Style feature was removed to check whether it would impact the LR model’s predictive ability; it showed only fractional changes in ROC AUC. So, in future, we may subtract such features that don’t have a major impact on the overall performance of the LR model. This will help make the model more practical to work with.

## Regression

We tried two different regression methods, Linear Regression, and Random Forest to try and predict the MSRP of the vehicle. From our model results in Table 3 (Linear Regression) and table 4 (Random Forest) we can see the comparisons between the two. We will ignore the model with does not use the categorical values, and the models which use +/-20% of MSRP, as they are not practical. Linear Regression provided an accuracy of 58% and 34% for 10% and 5% +/- MSRP respectively. Whereas, Random Forest, provided approx 46% and 24% for 10% and 5% +/- MSRP respectively. As seen above we have an 8% increase in accuracy and a 10% increase in accuracy for 10% and 5% +/- MSRP respectively. For a better prediction we can consider removing some columns which may not help in prediction of MSRP much. And for Random forest increase depth and iterations.

# References

[[1](#ref_kaggle)] "Car Features and MSRP", *Kaggle.com*, 2019. [Online]. Available: https://www.kaggle.com/CooperUnion/cardataset. [Accessed: 19- Aug- 2019].

[[2](#ref_link1)] J. Davis and M. Goadrich, "The Relationship Between Precision-Recall and ROC Curves", in *23rd International Conference on Machine Learning (ICML)*, Pittsburgh, 2006.

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# Appendix A

## Logistic Regression

Table 6: Area under ROC/PR Curves and Accuracy (Top 500)

|  |  |  |  |
| --- | --- | --- | --- |
|  | Training Data | Test Data | % Difference |
| Area under ROC Curve | 0.9921 | 0.9353 | 5.73% |
| Area under PR Curve | N/A | 0.8836 | N/A |
| Accuracy | 0.9543 | 0.9489 | 0.57% |

Table 7: Area under ROC/PR Curves and Accuracy (Top 150)

|  |  |  |  |
| --- | --- | --- | --- |
|  | Training Data | Test Data | % Difference |
| Area under ROC Curve | 0.9924 | 0.9353 | 5.75% |
| Area under PR Curve | N/A | 0.8850 | N/A |
| Accuracy | 0.9574 | 0.9511 | 0.66% |

Table 8: Area under ROC/PR Curves and Accuracy (Vehicle Style removed + Top 150)

|  |  |  |  |
| --- | --- | --- | --- |
|  | Training Data | Test Data | % Difference |
| Area under ROC Curve | 0.9925 | 0.9285 | 6.45% |
| Area under PR Curve | N/A | 0.8789 | N/A |
| Accuracy | 0.9577 | 0.9467 | 1.15% |

Table 9: All differences between four different models for Logistic Regression[[2]](#footnote-1)

|  |  |  |  |
| --- | --- | --- | --- |
| Differences (Δ) | Area under ROC Curve | Area under PR Curve | Accuracy |
| O - T500 | -0.61% | -1.82% | -0.20% |
| T500 - T150 | 0.00% | -0.16% | -0.23% |
| T150 - T150VSR | 0.73% | 0.69% | 0.46% |
| O - T150 | -0.61% | -1.98% | -0.43% |
| O - T150VSR | 0.12% | -1.28% | 0.03% |
| T500 - T150VSR | 0.73% | 0.53% | 0.23% |

## Random Forest Regression

Table 10: List of Random Forest Regression models

|  |  |  |  |
| --- | --- | --- | --- |
|  | Max Depth | Number of Trees | Max Number of Bins |
| Model 1 | 5 | 20 | 5 |
| Model 2 | 5 | 20 | 10 |
| Model 3 | 5 | 60 | 5 |
| Model 4 | 5 | 60 | 10 |
| Model 5 | 10 | 20 | 5 |
| Model 6 | 10 | 20 | 10 |
| Model 7 | 10 | 60 | 5 |
| Model 8 | 10 | 60 | 10 |

## 

1. Please take a look at Appendix A for all other tables including a complete table of differences between all the logistic regression model test results. [↑](#footnote-ref-0)
2. O - Original/All features; T500 - Top 500 features; T150 - Top 150 features; T150VSR - Top 150 with Vehicle Style removed [↑](#footnote-ref-1)