Car analyser

Faculty of Science and Engineering
Anglia Ruskin University

Car Evaluation using Machine Learning Models

Description

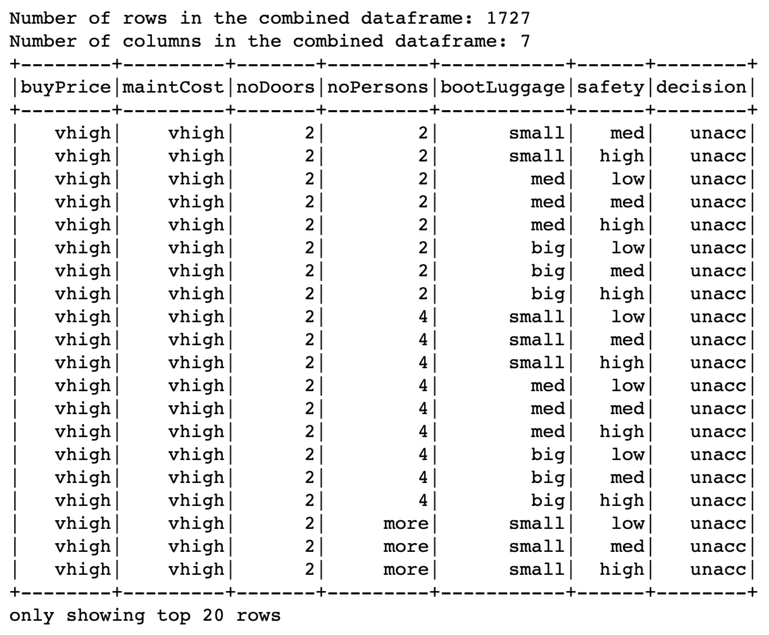
This task consists of building two Machine Learning models using **Apache Spark's Machine Learning(ML) library**[1], to classify type of car based on the car evaluation dataset. Decision tree and Random forest classifiers were used to classify the type of car based on attributes **buying price, price of maintenance, number of doors, capacity in terms of persons to carry, the relative size of luggage boot** and the estimated **safety** value of each car and it classifies cars as

1) **unacceptable (unacc)**, 2) **acceptable(acc)**, 3) **good** or 4) **very good(vgood)**

Pre-processing

The first step that was taken was to pre-process the data.The car evaluation data that was split into 4 files, was first concatenated and put into a single dataframe object by first loading the .csv files into an array.The dataFrames into a single dataFrame.Initialize a new dataframe object with the first dataFrame which is the "car_evaluation_0" content, and the rest of the files content starting from "car_evaluation_1.csv" to "car_evaluation_3.csv" were looped over and concatenated using union which makes sure the other dataframes have the same columns as first dataframe "car_evaluation_0".

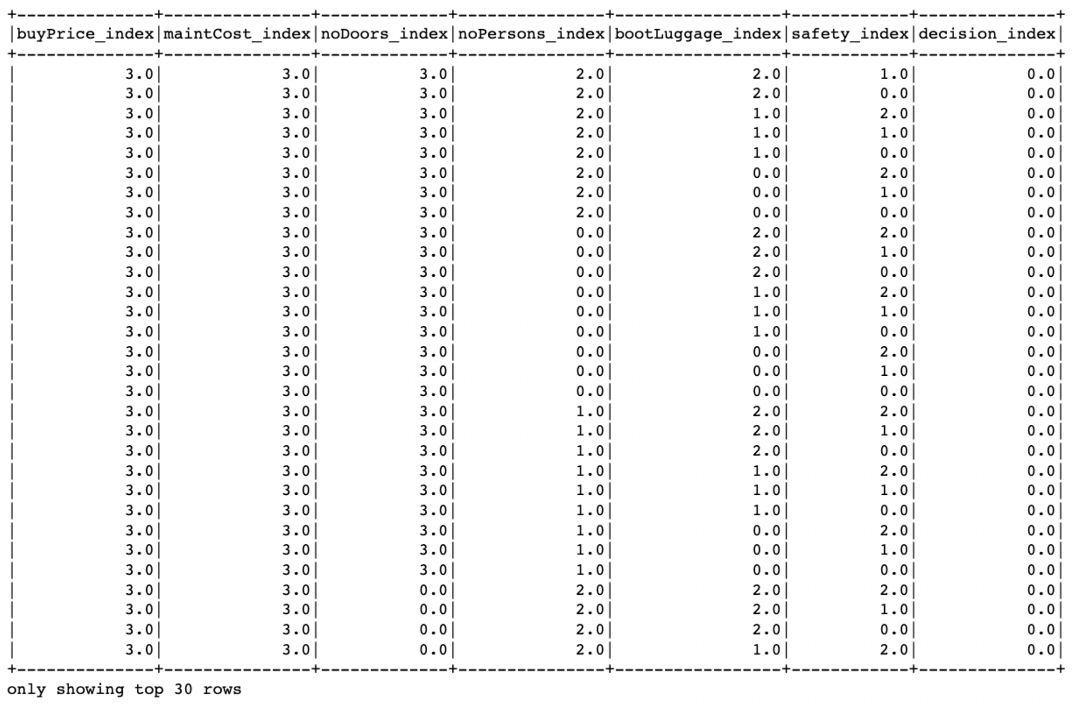
[1]`https://spark.apache.org/mllib/`



```
Number of rows in the combined dataframe: 1727
Number of columns in the combined dataframe: 7
+--------+---------+-------+---------+-----------+------+--------+
|buyPrice|maintCost|noDoors|noPersons|bootLuggage|safety|decision|
+--------+---------+-------+---------+-----------+------+--------+
|   vhigh|    vhigh|      2|        2|      small|   med|   unacc|
|   vhigh|    vhigh|      2|        2|      small|  high|   unacc|
|   vhigh|    vhigh|      2|        2|        med|   low|   unacc|
|   vhigh|    vhigh|      2|        2|        med|   med|   unacc|
|   vhigh|    vhigh|      2|        2|        med|  high|   unacc|
|   vhigh|    vhigh|      2|        2|        big|   low|   unacc|
|   vhigh|    vhigh|      2|        2|        big|   med|   unacc|
|   vhigh|    vhigh|      2|        2|        big|  high|   unacc|
|   vhigh|    vhigh|      2|        4|      small|   low|   unacc|
|   vhigh|    vhigh|      2|        4|      small|   med|   unacc|
|   vhigh|    vhigh|      2|        4|      small|  high|   unacc|
|   vhigh|    vhigh|      2|        4|        med|   low|   unacc|
|   vhigh|    vhigh|      2|        4|        med|   med|   unacc|
|   vhigh|    vhigh|      2|        4|        med|  high|   unacc|
|   vhigh|    vhigh|      2|        4|        big|   low|   unacc|
|   vhigh|    vhigh|      2|        4|        big|   med|   unacc|
|   vhigh|    vhigh|      2|        4|        big|  high|   unacc|
|   vhigh|    vhigh|      2|     more|      small|   low|   unacc|
|   vhigh|    vhigh|      2|     more|      small|   med|   unacc|
|   vhigh|    vhigh|      2|     more|      small|  high|   unacc|
+--------+---------+-------+---------+-----------+------+--------+
only showing top 20 rows
```

Figure 1: After concatenating the files

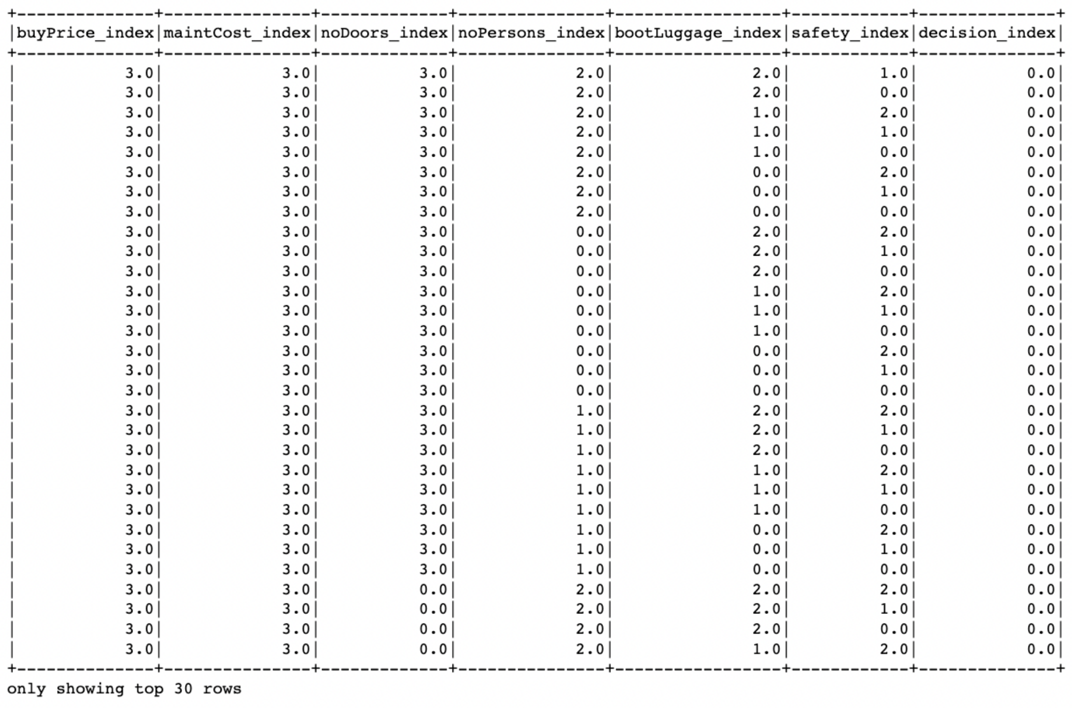
Fig 1 shows the dataframe after being concatenated and confirming the number of rows and columns as 1727 and 7 including the **decision** column. The data then had to be further processed since it consisted of categorical data like ”low”,”medium”, high” for ”maintCost” column, similarly other columns like noDoors, noPersons had a mix of numerical and string data like ”5more” and ”more”.The data has to be converted to numerical data either way to feed into a ML model either way. This was done using String Indexing and One hot encoding.String indexing converted the categorical data for all the attributes columns into numerical data.The decision column which is the column used to make the prediction had to be indexed as well. Estimator accepted a dataframe and implemented fit() method to produce a model Transformer.A Pipeline was used to chain multiple Transformers and Estimators together to specify our machine learning workflow.The goal of an Estimator in this case is to figure out how the attribute name maps to a it’s indexes. The transform() method after fit() takes one DataFrame and turns it into another by adding the indexed columns as seen in Fig 3



```
+--------------+---------------+-------------+---------------+-----------------+------------+--------------+
|buyPrice_index|maintCost_index|noDoors_index|noPersons_index|bootLuggage_index|safety_index|decision_index|
+--------------+---------------+-------------+---------------+-----------------+------------+--------------+
|           3.0|            3.0|          3.0|            2.0|              2.0|         1.0|           0.0|
|           3.0|            3.0|          3.0|            2.0|              2.0|         0.0|           0.0|
|           3.0|            3.0|          3.0|            2.0|              1.0|         2.0|           0.0|
|           3.0|            3.0|          3.0|            2.0|              1.0|         1.0|           0.0|
|           3.0|            3.0|          3.0|            2.0|              1.0|         0.0|           0.0|
|           3.0|            3.0|          3.0|            2.0|              0.0|         2.0|           0.0|
|           3.0|            3.0|          3.0|            2.0|              0.0|         1.0|           0.0|
|           3.0|            3.0|          3.0|            2.0|              0.0|         0.0|           0.0|
|           3.0|            3.0|          3.0|            0.0|              2.0|         2.0|           0.0|
|           3.0|            3.0|          3.0|            0.0|              2.0|         1.0|           0.0|
|           3.0|            3.0|          3.0|            0.0|              2.0|         0.0|           0.0|
|           3.0|            3.0|          3.0|            0.0|              1.0|         2.0|           0.0|
|           3.0|            3.0|          3.0|            0.0|              1.0|         1.0|           0.0|
|           3.0|            3.0|          3.0|            0.0|              1.0|         0.0|           0.0|
|           3.0|            3.0|          3.0|            0.0|              0.0|         2.0|           0.0|
|           3.0|            3.0|          3.0|            0.0|              0.0|         1.0|           0.0|
|           3.0|            3.0|          3.0|            0.0|              0.0|         0.0|           0.0|
|           3.0|            3.0|          3.0|            1.0|              2.0|         2.0|           0.0|
|           3.0|            3.0|          3.0|            1.0|              2.0|         1.0|           0.0|
|           3.0|            3.0|          3.0|            1.0|              2.0|         0.0|           0.0|
|           3.0|            3.0|          3.0|            1.0|              1.0|         2.0|           0.0|
|           3.0|            3.0|          3.0|            1.0|              1.0|         1.0|           0.0|
|           3.0|            3.0|          3.0|            1.0|              1.0|         0.0|           0.0|
|           3.0|            3.0|          3.0|            1.0|              0.0|         2.0|           0.0|
|           3.0|            3.0|          3.0|            1.0|              0.0|         1.0|           0.0|
|           3.0|            3.0|          3.0|            1.0|              0.0|         0.0|           0.0|
|           3.0|            3.0|          0.0|            2.0|              2.0|         2.0|           0.0|
|           3.0|            3.0|          0.0|            2.0|              2.0|         1.0|           0.0|
|           3.0|            3.0|          0.0|            2.0|              2.0|         0.0|           0.0|
|           3.0|            3.0|          0.0|            2.0|              1.0|         2.0|           0.0|
+--------------+---------------+-------------+---------------+-----------------+------------+--------------+
only showing top 30 rows
```

Figure 2: String Indexing attribute columns

Then One Hot Encoder(OHE) was used on the indexed columns to break the classified column into number of unique categories in the column. Giving each column a binary value (0 or 1) that shows whether or not the data point has the has that attribute or not.

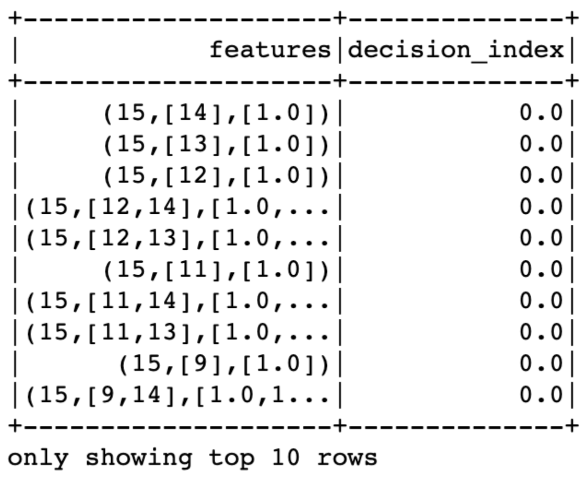


```
+--------------+---------------+-------------+---------------+-----------------+------------+--------------+
|buyPrice_index|maintCost_index|noDoors_index|noPersons_index|bootLuggage_index|safety_index|decision_index|
+--------------+---------------+-------------+---------------+-----------------+------------+--------------+
|           3.0|            3.0|          3.0|            2.0|              2.0|         1.0|           0.0|
|           3.0|            3.0|          3.0|            2.0|              2.0|         0.0|           0.0|
|           3.0|            3.0|          3.0|            2.0|              1.0|         2.0|           0.0|
|           3.0|            3.0|          3.0|            2.0|              1.0|         1.0|           0.0|
|           3.0|            3.0|          3.0|            2.0|              1.0|         0.0|           0.0|
|           3.0|            3.0|          3.0|            2.0|              0.0|         2.0|           0.0|
|           3.0|            3.0|          3.0|            2.0|              0.0|         1.0|           0.0|
|           3.0|            3.0|          3.0|            2.0|              0.0|         0.0|           0.0|
|           3.0|            3.0|          3.0|            0.0|              2.0|         2.0|           0.0|
|           3.0|            3.0|          3.0|            0.0|              2.0|         1.0|           0.0|
|           3.0|            3.0|          3.0|            0.0|              2.0|         0.0|           0.0|
|           3.0|            3.0|          3.0|            0.0|              1.0|         2.0|           0.0|
|           3.0|            3.0|          3.0|            0.0|              1.0|         1.0|           0.0|
|           3.0|            3.0|          3.0|            0.0|              1.0|         0.0|           0.0|
|           3.0|            3.0|          3.0|            0.0|              0.0|         2.0|           0.0|
|           3.0|            3.0|          3.0|            0.0|              0.0|         1.0|           0.0|
|           3.0|            3.0|          3.0|            0.0|              0.0|         0.0|           0.0|
|           3.0|            3.0|          3.0|            1.0|              2.0|         2.0|           0.0|
|           3.0|            3.0|          3.0|            1.0|              2.0|         1.0|           0.0|
|           3.0|            3.0|          3.0|            1.0|              2.0|         0.0|           0.0|
|           3.0|            3.0|          3.0|            1.0|              1.0|         2.0|           0.0|
|           3.0|            3.0|          3.0|            1.0|              1.0|         1.0|           0.0|
|           3.0|            3.0|          3.0|            1.0|              1.0|         0.0|           0.0|
|           3.0|            3.0|          3.0|            1.0|              0.0|         2.0|           0.0|
|           3.0|            3.0|          3.0|            1.0|              0.0|         1.0|           0.0|
|           3.0|            3.0|          3.0|            1.0|              0.0|         0.0|           0.0|
|           3.0|            3.0|          0.0|            2.0|              2.0|         2.0|           0.0|
|           3.0|            3.0|          0.0|            2.0|              2.0|         1.0|           0.0|
|           3.0|            3.0|          0.0|            2.0|              2.0|         0.0|           0.0|
|           3.0|            3.0|          0.0|            2.0|              1.0|         2.0|           0.0|
+--------------+---------------+-------------+---------------+-----------------+------------+--------------+
only showing top 30 rows
```

Figure 3: One Hot Encoding the attribute columns

The encoded columns were then transformed into "features" using **VectorAssembler** and then added to the dataframe using trasform as shown in Fig 4.The 'features' and 'decision_index" will then be used to feed into the machine learning models as input.

The data was then split into training and test data, a 80% and 20% split respectively. There were then 1399 training samples and 328 samples.



```
+--------------------+--------------+
|            features|decision_index|
+--------------------+--------------+
|     (15,[14],[1.0])|           0.0|
|     (15,[13],[1.0])|           0.0|
|     (15,[12],[1.0])|           0.0|
|(15,[12,14],[1.0,...|           0.0|
|(15,[12,13],[1.0,...|           0.0|
|     (15,[11],[1.0])|           0.0|
|(15,[11,14],[1.0,...|           0.0|
|(15,[11,13],[1.0,...|           0.0|
|      (15,[9],[1.0])|           0.0|
|(15,[9,14],[1.0,1...|           0.0|
+--------------------+--------------+
only showing top 10 rows
```

Figure 4: Transforming attributes columns into features

Decision Trees

Decision Tree is a Supervised Machine Learning Algorithm that uses a set of rules to make decisions in the same way that humans do[1].Data points are continuously divided based on specific parameters and/or the problem that the algorithm is attempting to answer.Decision tree consists of a root node, some branches, and leaf nodes.The algorithm will continue to split the tree until the data is sufficiently uniform.

The time required to build a decision tree grows as the number of splits increases. Trees with a high number of splits, on the other hand, are prone to overfitting, resulting in low accuracy."max-depth" or the maximum depth allowed for the tree to grow throughout the learning process is used to avoid **overfitting**. Overfitting happens when a decision tree becomes too complicated and captures noise in the training data, causing it to be less successful at generalising to new, previously unknown data.To prevent overfitting further,during the learning process, you can influence the tree's growth. If a node's number of instances falls below this level, the node will not be split further.

The model hyper-parameters tuned were **max-depth** which was set to 10, **minimum instances per node** was set to 5 and impurity was set to **gini** formula of which is given below.

The minimum information gain required to perform a split at a node measures the improvement in impurity (e.g., Gini impurity or entropy) by splitting a node. It can help avoid splits that do not provide significant improvements in impurity reduction. All of which prevent overfitting and increase accuracy.

$$Gini = 1 - \sum_j p_j s^2$$

Random Forests

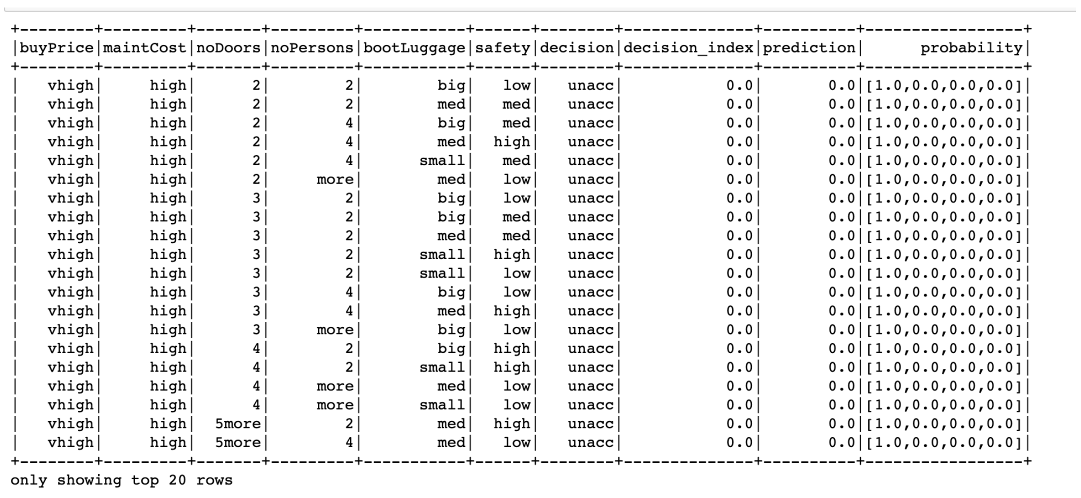
Random Forests are a powerful ensemble learning algorithm that combine multiple decision trees to make more accurate predictions.The number of decision trees to build in the forest. More trees generally lead to better performance, but it also increases computational complexity. The maximum depth of each decision tree in the forest. It controls the complexity of the individual trees. A deeper tree can capture more complex relationships in the data, but it can also lead to overfitting. Overfitting can cause the model to perform well on training data but poorly unseen data, which is not desired.

The number of trees was set to 1, maximum depth was set to 30, minimum instances per node to 3,feature subset strategy to auto.Minimum instances per node, nodes with less than 3 instances will become leaf nodes, which helps to prevent **overfitting**. The "**featureSubsetStrategy**" argument specifies the approach for selecting the subset of features to consider when determining the optimum split at each tree node. Setting it to "auto" implies that the algorithm will select a strategy based on the number of features and trees in the forest.

Evaluating model

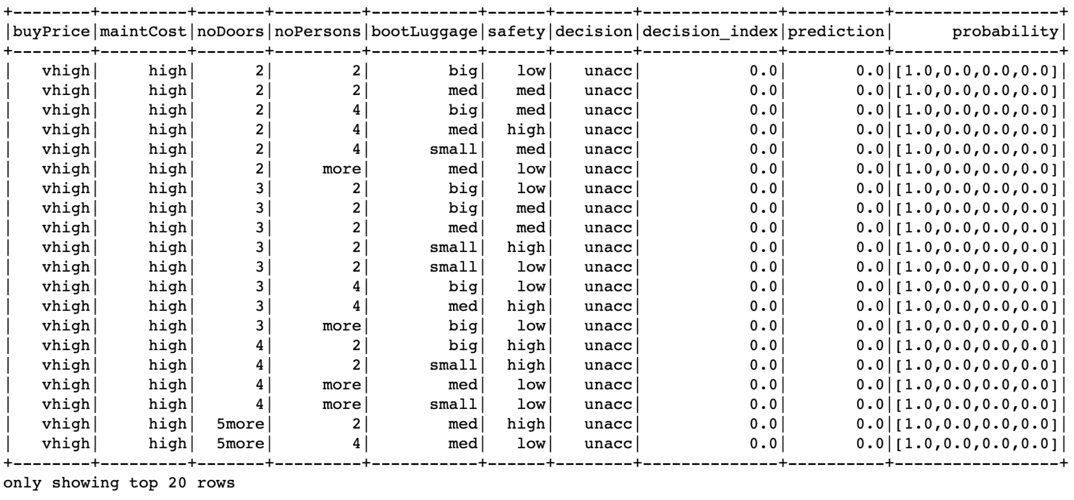
Since the problem of classification was a multiclass problem meaning it had to classify 4 types of car,**MulticlassClassificationEvaluator** was used to evaluate the performance and provide an overall evaluation of the model's performance, in this case the overall accuracy was evaluated.

CrossValidator was used to obtain a more reliable assessment of your model's performance and to provide a better notion of how well the model generalises to previously unseen data.**numFolds** or number of folds was set to 5 meaning data was be divided into 5 folds, and the cross-validation process was done 5 times. These folds were iteratively used to train and evaluate the model. Each time, one fold is used as the validation set, and the rest of the folds are used for training the models.The model was then fit() then trained using the training data for both models.Using the **transform()** method next on the model,with test data the predictions column was appended to the dataframe and shown for both models in Fig 5 and 6.The original attributes columns, decision,decision_index','prediction' and 'probabilty' were selected to show specifically.



```
+--------+---------+-------+---------+-----------+------+--------+--------------+----------+-----------------+
|buyPrice|maintCost|noDoors|noPersons|bootLuggage|safety|decision|decision_index|prediction|      probability|
+--------+---------+-------+---------+-----------+------+--------+--------------+----------+-----------------+
|   vhigh|     high|      2|        2|        big|   low|   unacc|           0.0|       0.0|[1.0,0.0,0.0,0.0]|
|   vhigh|     high|      2|        2|        med|   med|   unacc|           0.0|       0.0|[1.0,0.0,0.0,0.0]|
|   vhigh|     high|      2|        4|        big|   med|   unacc|           0.0|       0.0|[1.0,0.0,0.0,0.0]|
|   vhigh|     high|      2|        4|        med|  high|   unacc|           0.0|       0.0|[1.0,0.0,0.0,0.0]|
|   vhigh|     high|      2|        4|      small|   med|   unacc|           0.0|       0.0|[1.0,0.0,0.0,0.0]|
|   vhigh|     high|      2|     more|        med|   low|   unacc|           0.0|       0.0|[1.0,0.0,0.0,0.0]|
|   vhigh|     high|      3|        2|        big|   low|   unacc|           0.0|       0.0|[1.0,0.0,0.0,0.0]|
|   vhigh|     high|      3|        2|        big|   med|   unacc|           0.0|       0.0|[1.0,0.0,0.0,0.0]|
|   vhigh|     high|      3|        2|        med|   med|   unacc|           0.0|       0.0|[1.0,0.0,0.0,0.0]|
|   vhigh|     high|      3|        2|      small|  high|   unacc|           0.0|       0.0|[1.0,0.0,0.0,0.0]|
|   vhigh|     high|      3|        2|      small|   low|   unacc|           0.0|       0.0|[1.0,0.0,0.0,0.0]|
|   vhigh|     high|      3|        4|        big|   low|   unacc|           0.0|       0.0|[1.0,0.0,0.0,0.0]|
|   vhigh|     high|      3|        4|        med|  high|   unacc|           0.0|       0.0|[1.0,0.0,0.0,0.0]|
|   vhigh|     high|      3|     more|        big|   low|   unacc|           0.0|       0.0|[1.0,0.0,0.0,0.0]|
|   vhigh|     high|      4|        2|        big|  high|   unacc|           0.0|       0.0|[1.0,0.0,0.0,0.0]|
|   vhigh|     high|      4|        2|      small|  high|   unacc|           0.0|       0.0|[1.0,0.0,0.0,0.0]|
|   vhigh|     high|      4|     more|        med|   low|   unacc|           0.0|       0.0|[1.0,0.0,0.0,0.0]|
|   vhigh|     high|      4|     more|      small|   low|   unacc|           0.0|       0.0|[1.0,0.0,0.0,0.0]|
|   vhigh|     high|  5more|        2|        med|  high|   unacc|           0.0|       0.0|[1.0,0.0,0.0,0.0]|
|   vhigh|     high|  5more|        4|        med|   low|   unacc|           0.0|       0.0|[1.0,0.0,0.0,0.0]|
+--------+---------+-------+---------+-----------+------+--------+--------------+----------+-----------------+
only showing top 20 rows
```

Figure 5: Decision Tree after adding predictions column using transform



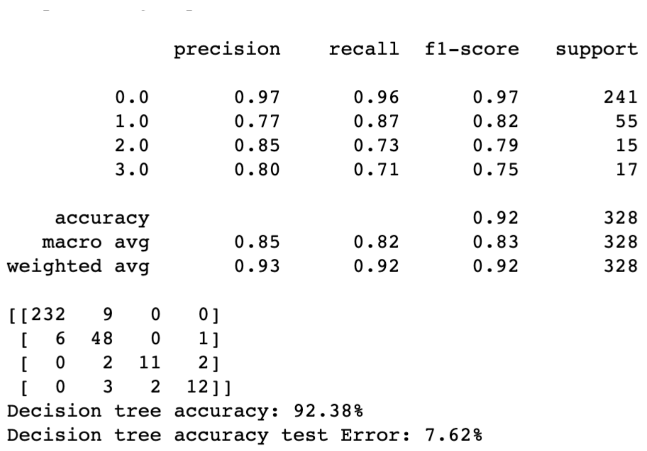
```
+--------+---------+-------+---------+-----------+------+--------+--------------+----------+-----------------+
|buyPrice|maintCost|noDoors|noPersons|bootLuggage|safety|decision|decision_index|prediction|      probability|
+--------+---------+-------+---------+-----------+------+--------+--------------+----------+-----------------+
|   vhigh|     high|      2|        2|        big|   low|   unacc|           0.0|       0.0|[1.0,0.0,0.0,0.0]|
|   vhigh|     high|      2|        2|        med|   med|   unacc|           0.0|       0.0|[1.0,0.0,0.0,0.0]|
|   vhigh|     high|      2|        4|        big|   med|   unacc|           0.0|       0.0|[1.0,0.0,0.0,0.0]|
|   vhigh|     high|      2|        4|        med|  high|   unacc|           0.0|       0.0|[1.0,0.0,0.0,0.0]|
|   vhigh|     high|      2|        4|      small|   med|   unacc|           0.0|       0.0|[1.0,0.0,0.0,0.0]|
|   vhigh|     high|      2|     more|        med|   low|   unacc|           0.0|       0.0|[1.0,0.0,0.0,0.0]|
|   vhigh|     high|      3|        2|        big|   low|   unacc|           0.0|       0.0|[1.0,0.0,0.0,0.0]|
|   vhigh|     high|      3|        2|        big|   med|   unacc|           0.0|       0.0|[1.0,0.0,0.0,0.0]|
|   vhigh|     high|      3|        2|        med|   med|   unacc|           0.0|       0.0|[1.0,0.0,0.0,0.0]|
|   vhigh|     high|      3|        2|      small|  high|   unacc|           0.0|       0.0|[1.0,0.0,0.0,0.0]|
|   vhigh|     high|      3|        2|      small|   low|   unacc|           0.0|       0.0|[1.0,0.0,0.0,0.0]|
|   vhigh|     high|      3|        4|        big|   low|   unacc|           0.0|       0.0|[1.0,0.0,0.0,0.0]|
|   vhigh|     high|      3|        4|        med|  high|   unacc|           0.0|       0.0|[1.0,0.0,0.0,0.0]|
|   vhigh|     high|      3|     more|        big|   low|   unacc|           0.0|       0.0|[1.0,0.0,0.0,0.0]|
|   vhigh|     high|      4|        2|        big|  high|   unacc|           0.0|       0.0|[1.0,0.0,0.0,0.0]|
|   vhigh|     high|      4|        2|      small|  high|   unacc|           0.0|       0.0|[1.0,0.0,0.0,0.0]|
|   vhigh|     high|      4|     more|        med|   low|   unacc|           0.0|       0.0|[1.0,0.0,0.0,0.0]|
|   vhigh|     high|      4|     more|      small|   low|   unacc|           0.0|       0.0|[1.0,0.0,0.0,0.0]|
|   vhigh|     high|  5more|        2|        med|  high|   unacc|           0.0|       0.0|[1.0,0.0,0.0,0.0]|
|   vhigh|     high|  5more|        4|        med|   low|   unacc|           0.0|       0.0|[1.0,0.0,0.0,0.0]|
+--------+---------+-------+---------+-----------+------+--------+--------------+----------+-----------------+
only showing top 20 rows
```

Figure 6: Random forests after adding predictions column using transform

The classifier performances were then evaluated using the **evaluate()** method. The minimum instances per node when tweaked to a higher number seemed to produce less accuracy for the Random forests model.The decision tree model increased accuracy up to 92.38% after lowering the minimum instances per node to 2.Increasing the max_depth to 30 and reducing the number of trees from 5 to 3, to 1 increased the accuracy to 89.94% but was indeed computationally expensive.It took longer to train and evaluate.

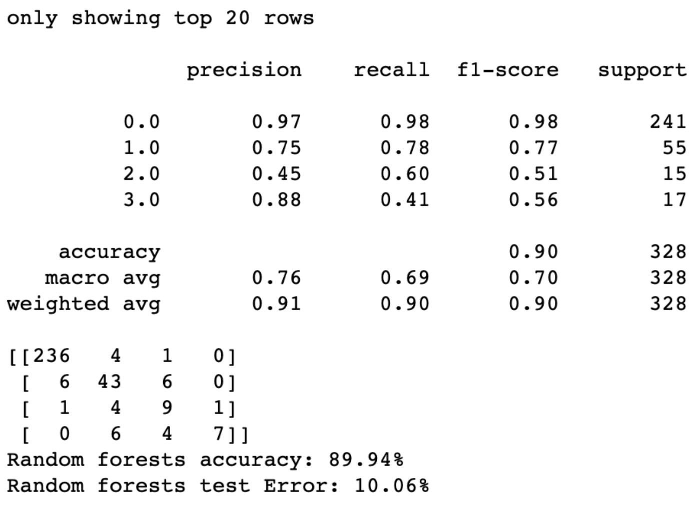
The accuracy and test errors for each model was then recorded.The Decision tree classifier performed better than the Random Forests classifier with the first getting an accuracy of 92.38%,test error 7.62% and 89.94% accuracy, 10.06% test error respectively.The models performance on unseen data was further evaluated using precision, recall,f1-score and confusion matrix. The decision tree model.

Since cars are expensive, it is important to make high accurate results,so F1-score is selected to compare the models[2].F-1 score, which is a harmonic mean of precision and recall for Decision tree was higher than Random forests. Decision tree f1 score was 0.83 was significantly higher than Random Forests score which was 0.70.Thus, it can be concluded that Decision tree would be a better classifier to predict car type. The model hyperparameters were tuned several times to achieve the best accuracy result of 92.38% and error 7.62% which is good.The confusion matrix for the Decision tree classified 303(232+48+11+12) out of 328 test samples correctly and misclassified 12(6+2+3+2+9) samples.Random forests classifier classified 295(236+43+9+7) out of 328 samples correctly and misclassified 37(6+1+4+6+4+1+6+4+4+1) samples.It was observed that 236 samples out of 241 of unacceptable car types were predicted better by the Random forests model than the decision tree which also signifies the samples for the car type unacceptable is higher than the other types.The support values from both results in Fig 7 and 8 make it clear unacceptable car type has 241 samples and there aren't enough samples for other car types which are 55,15,17 for acceptable,good and very good car types respectively.Adding more data for the other types could increase the accuracy further for both models.



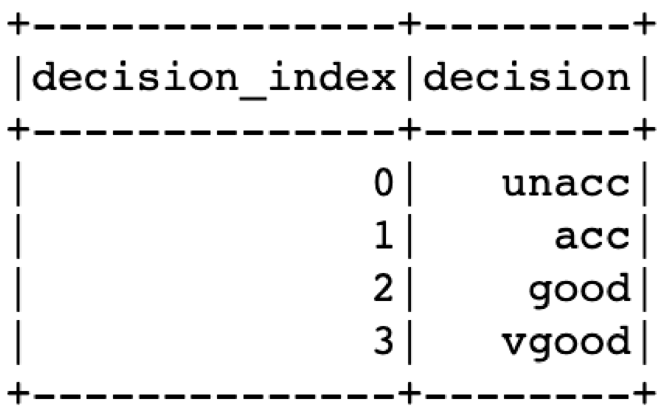
```
              precision    recall  f1-score   support

         0.0       0.97      0.96      0.97       241
         1.0       0.77      0.87      0.82        55
         2.0       0.85      0.73      0.79        15
         3.0       0.80      0.71      0.75        17

    accuracy                           0.92       328
   macro avg       0.85      0.82      0.83       328
weighted avg       0.93      0.92      0.92       328

[[232   9   0   0]
 [  6  48   0   1]
 [  0   2  11   2]
 [  0   3   2  12]]
Decision tree accuracy: 92.38%
Decision tree accuracy test Error: 7.62%
```

Figure 7: Decision tree results



```
only showing top 20 rows

              precision    recall  f1-score   support

         0.0       0.97      0.98      0.98       241
         1.0       0.75      0.78      0.77        55
         2.0       0.45      0.60      0.51        15
         3.0       0.88      0.41      0.56        17

    accuracy                           0.90       328
   macro avg       0.76      0.69      0.70       328
weighted avg       0.91      0.90      0.90       328

[[236   4   1   0]
 [  6  43   6   0]
 [  1   4   9   1]
 [  0   6   4   7]]
Random forests accuracy: 89.94%
Random forests test Error: 10.06%
```

Figure 8: Random forests results



```
+--------------+--------+
|decision_index|decision|
+--------------+--------+
|             0|   unacc|
|             1|     acc|
|             2|    good|
|             3|   vgood|
+--------------+--------+
```

Figure 9: Decision labels mapped to its indexes

References

[1]Bento, C., 2022. Decision Tree Classifier explained in real-life: picking a vacation destination. [online] Medium. Available at: https://towardsdatascience.com/decision-tree-classifier-explained-in-real-life-picking-a-vacation-destination-6226b2b60575: :text=Decision%20tr
(Accessed 12th July 2023)

[2]Saxena, S., 2023. Precision vs Recall - shruti saxena - Medium. [online] Medium. Available at: https://medium.com/@shrutisaxena0617/precision-vs-recall-386cf9f89488. (Accessed 12th July 2023)

[3]Raj, A., 2022. An Exhaustive Guide to Decision Tree Classification in Python 3.x. [online] Medium. Available at: https://towardsdatascience.com/an-exhaustive-guide-to-classification-using-decision-trees-8d472e77223f. (Accessed 12th July 2023)