Machine Learning Nano Degree

Capstone Project

31 March 2018

Ilyas Ahmed Mohammed

Udacity MLND

Predicting Car Acceptability Rate (CAR)

I. Definition

a. Project Overview: An important measure in Automotive industry is how reliable/acceptable a used car is. It is quite common to sell used cars to Car Junkyards after a minor crash and these junkyards simply disassemble the car and sell the individual parts without actually checking for the condition of the car and see if it can be repaired/reused. This is mainly due to lack of mechanical knowledge among general public and even the junkyard owners. If a car is determined to be in acceptable condition when it is sold to a junkyard, then with a minor repair this car can be resold /reused thus preventing the disassembly cost and the logistics cost of moving different components of the car assembly in order to sell it as scrap. An attempt has been in this project to predict the acceptability rate of a car given some simple features and information about the car's price and comfort. The Data collected from UCI Machine Learning repository has six input features describing a car's condition – Buying price, Maintenance price, number of doors, capacity of the car in terms of number of persons it can carry, size of the luggage boot and estimated safety of the car. Based on these features, a car is predicted to be either in unacceptable condition **or** acceptable condition **or** good condition **or** very good condition.

Although this project is aimed at helping the junkyard car owners increase their revenue by identifying cars in acceptable condition and repair them for reuse/resell, this model will be highly helpful to automotive manufacturers in determining the warranty period of the cars and providing excellent customer service by repairing the used cars which were considered as 'junk' otherwise and educating the general public about the reliability of their cars. A similar kind of problem was analyzed and reported using the same dataset in the past.

b. Problem Statement: Briefly, the problem can be summed up as follows: "Given some input features of a used car's price and comfort, predict the acceptability rate of the car"

The Dataset used in this project has a total six input features which are already available. All these features are categorical in nature. The Target variable is called Car Acceptability Rate (CAR) which describes the condition of the car and has four attributes. For the purpose of this model, these four attributes can be defined as follows:

- **unacc:** The car is in unacceptable condition and it's only use is to scrap and use the spare parts
- **acc:** The condition of the car is acceptable and it can potentially be reused by replacing some parts

- **good:** The car is in good condition and can be reused with minor repairs/modifications which can be done inhouse
- **vgood:** The car is in very good condition and requires some regular maintenance after which it can reused or sold to a pre-owned dealer or to other prospective buyers.

Given the input variables of a car, the model must predicts it's acceptability and assign it as one of the above four labels. This is a supervised learning multi-class classification problem. At first, the input data will be pre-processed wherein one-hot encoding will be applied. Since the complete data set is categorical in nature, no further normalization or logarithmic scaling is required. The Dataset has a total 1728 instances and does not have any missing values so imputing is also not required which makes the pre-processing very simple and straightforward. Two or Three classifiers will be first trained on the data set and evaluated and the highest performance classifier will be further optimized to predict the Target variable – CAR as one of the four labels – 'unacc', 'acc', 'good' or 'vgood'.

c. **Metrics:** The classifiers selected will be evaluated based on certain Evaluation Metrics and then the highest performance classifier will be further optimized and evaluated using the same Metrics to determine the improvement made. The target variable which needs to be predicted can be one of the four classes -- 'unacc', 'acc', 'good' or 'vgood' and this will be predicted based on the data which was provided to the classifier to train. A careful study of the data reveals that the four target labels are not equally distributed in the data and majority of the target label present in the data is 'unacc' (~70%) and the next highest label is 'acc'(~22%). The remaining two labels – 'good' and 'vgood' have same distribution (~3 -4%). This shows that the data is quite skewed towards 'unacc' label. Due to this uneven distribution of the target label in the data, accuracy will not be a good evaluation metric and it will predict an accuracy of over 70% even if everything is predicted as 'unacc' which can be quite misleading.

Predicting the right label for a car is extremely important because if an unacceptable car is predicted as acceptable or good, then the junkyard or the automotive manufacturers can incur some loss in trying to repair a car which is not reliable. For this reason, both Precision and Recall of the prediction must be evaluated and a combination metric of these two variables – F_1-score will be used in this project to evaluate the performance of the classifier and optimizing it to determine the amount of improvement made on the prediction. Since this is a multi-class classification problem, a confusion matrix will be developed with the individual precision and recall for each of the four labels – 'unacc'($Precision_{unacc}$ and $Recall_{unacc}$), 'acc'($Precision_{acc}$ and $Recall_{acc}$), 'good'($Precision_{good}$ and $Recall_{good}$), 'vgood'($Precision_{vgood}$ and $Recall_{vgood}$). The average Precision and average Recall is calculated.

$Precision_{avg} = (Precision_{unacc} + Precision_{acc} + Precision_{good} + Precision_{vgood}) / 4$

$Recall_{avg} = (Recall_{unacc} + Recall_{acc} + Recall_{good} + Recall_{vgood}) / 4$

Now the F_1-score will be calculated with the $Precision_{avg}$ and $Recall_{avg}$

F_1-score $= 2 * (Precision_{avg} * Recall_{avg}) / (Precision_{avg} + Recall_{avg})$

II. Analysis

a. Data Exploration: Data Set collected from the UCI Machine Learning Repository is loaded in a pandas Data Frame and explored using python packages and functions. Overall, the dataset has a total of 1728 examples of car's acceptability rate. There are six input features – 'Buying_price', 'Maint_price', 'doors', 'persons', 'lug_boot' and 'safety' and the Target label is 'CAR' which has four labels as explained in previous section.

```
In [13]: data.describe()
```

Out[13]:

	Buying_price	Maint_price	doors	persons	lug_boot	safety	CAR
count	1728	1728	1728	1728	1728	1728	1728
unique	4	4	4	3	3	3	4
top	vhigh	vhigh	2	2	small	high	unacc
freq	432	432	432	576	576	576	1210

Figure 1: Description of Data

The above code snippet provides a description of the data. The count for all the columns is same (1728) which means that there are no missing values in the dataset. All the features and the target variable (CAR) is categorical in nature since they have 3-4 unique values.

The Data is unevenly distributed with the majority of output labels as **'unacc'** which occupies about 70% of the dataset. **'acc'** label is about 22% and the remaining two labels – **'good'** and **'vgood'** is in between 3-4% each.

```
In [39]: #Data Exploration

number_of_instances = len(data)

def number(x):
    return len(data[data['CAR'] == x])


number_of_unacceptable_cars = number('unacc')
number_of_acceptable_cars = number('acc')
number_of_good_cars = number('good')
number_of_very_good_cars = number('vgood')

print("Total number of Instances = {}".format(number_of_instances))
print("Total number of Unacceptable condition cars in Data Set = {}".format(number_of_unacceptable_cars))
print("Total number of Acceptable condition cars in Data Set = {}".format(number_of_acceptable_cars))
print("Total number of Good condition cars in Data Set = {}".format(number_of_good_cars))
print("Total number of Very Good condition cars in Data Set = {}".format(number_of_very_good_cars))
```

```
Total number of Instances = 1728
Total number of Unacceptable condition cars in Data Set = 1210
Total number of Acceptable condition cars in Data Set = 384
Total number of Good condition cars in Data Set = 69
Total number of Very Good condition cars in Data Set = 65
```

Figure 2: Number of Instances of each CAR Label

```
In [41]: #Percentages

         def percentage(x):
             return ((x/number_of_instances) * 100.00)

         print("Percentage of Unacceptable condition cars = {:.2f}%".format(percentage(number_of_unacceptable_cars)))
         print("Percentage of Acceptable condition cars = {:.2f}%".format(percentage(number_of_acceptable_cars)))
         print("Percentage of Good condition cars = {:.2f}%".format(percentage(number_of_good_cars)))
         print("Percentage of Very Good condition cars = {:.2f}%".format(percentage(number_of_very_good_cars)))

         Percentage of Unacceptable condition cars = 70.02%
         Percentage of Acceptable condition cars = 22.22%
         Percentage of Good condition cars = 3.99%
         Percentage of Very Good condition cars = 3.76%
```

Figure 3: Percentages of each CAR Label

There are six input features the details of which are given below:

1. Buying_price: The price of buying the car. Attributes: v-high, high, med, low
2. Maint_price: The price of the maintenance of the car. Attributes: v-high, high, med, low
3. doors: Number of doors in the car. Attributes: 2, 3, 4, 5-more
4. persons: Capacity of the car in terms of persons it can carry. Attributes: 2, 4, more
5. lug_boot: Size of the luggage boot. Attributes: small, med, big
6. safety: estimated safety of the car. Attributes: low, med, high

```
In [38]: #Feature Set Exploration

         for column in features.columns:
             print("Attributes of '{}' feature = {}".format(column, features[column].unique()))

         Attributes of 'Buying_price' feature = ['vhigh' 'high' 'med' 'low']
         Attributes of 'Maint_price' feature = ['vhigh' 'high' 'med' 'low']
         Attributes of 'doors' feature = ['2' '3' '4' '5more']
         Attributes of 'persons' feature = ['2' '4' 'more']
         Attributes of 'lug_boot' feature = ['small' 'med' 'big']
         Attributes of 'safety' feature = ['low' 'med' 'high']
```

Figure 4: Feature Set Exploration

b. **Exploratory Visualization:** At the first intuition, it seems like for the cars which are in unacceptable condition, the price will be higher and the estimated safety of the car will be low.
This hypotheses needs to be tested with a visualization which gives the number the number of cars belonging to each label – 'unacc', 'acc', 'good', 'vgood' for each feature. At first, the 'Buying_price' feature is analyzed.

From the below graph, it looks like the majority of cars with unacceptable condition cars have very high Buying price however, a higher number of unacceptable condition cars also have low and medium Buying price so there is no good correlation there. Another important point to note here is that cars with very high and high Buying price are not in good or very good condition. Its only the medium and low Buying price cars which are either good or very good.

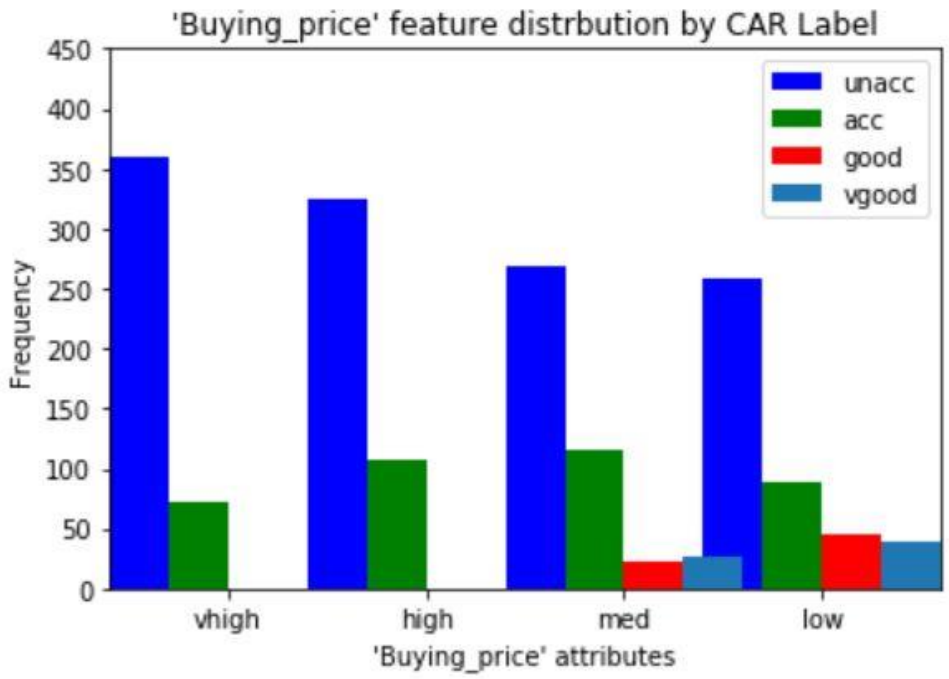

Figure 5: Buying_price feature analysis with respect to the target labels

Similar analysis with ‘safety’ feature is given below. All the cars with low safety rating are considered as unacceptable cars however not all unacceptable condition cars have low safety rating. A significant number of high safety cars are also considered unacceptable. All the cars which are considered as very good have high safety rating which seems to be in-line with the intuition.

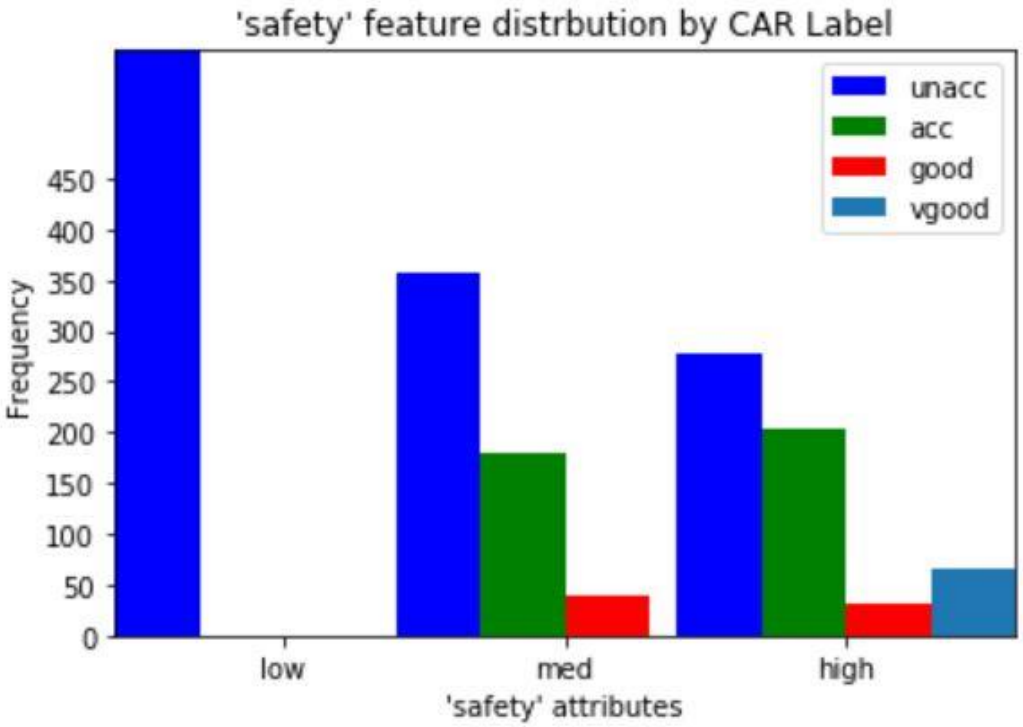

Figure 6: safety feature analysis with respect to the target label

Another interesting finding with the exploratory visualization was that the frequency graph of the Maintenance price feature and Buying price feature are very similar. They both have almost equal number of same condition cars (unacceptable, acceptable, good and very good) for each of their attributes. This begs the question if both these features are performing similarly with respect to the target variable – meaning does a car with unacceptable condition have some Buying_price and Maint_price?. IF that is the case then one feature can be eliminated – Buying_price in the pre-processing step which can significantly improve the performance of the classifier and this will reduce the number of features (after one-hot encoding) and will be faster. Further analysis showed that there are only 432 rows which have some value for Buying_price and Maint_price out of the total 1728 examples. So even though the frequency of occurrence is same for these features, they do not classify the label majority of the time which means that a car with very high Buying price and unacceptable condition does not necessarily have very high Maintenance price. So both these features are important in the final classification.

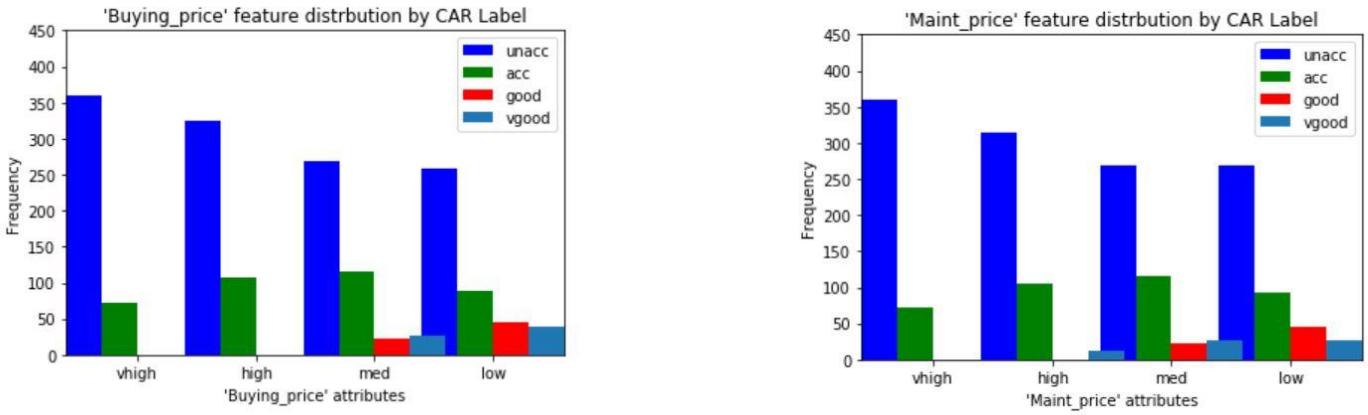

Figure 7: Comparison of Buying_price and Maint_price frequency

```
In [227]: len(data[data['Buying_price'] == data['Maint_price']]['CAR'])
Out[227]: 432
```

Figure 8: Code showing only 432 examples have common Buying_price and Maint_price

c. **Algorithms and Techniques:** The first technique which will be used in the preprocessing step to prepare the data for classification is one-hot encoding. The complete dataset is categorical in nature and in order to fit the data to the classifier, this categorical data needs to be converted to numerical data and one-hot encoding is an effective way of performing this task.

Pre-processing also include splitting the data in training and test dataset. Standard splitting will be implemented. 80% of the data will be used as training dataset and the remaining 20% of the data will be used as testing data set to evaluate the performance of the model. In order to avoid over-fitting, K-Fold Cross validation technique will be used when fitting the data to a certain classifier where the score on cross validation set will be used to evaluate the performance.

After the data is prepared and is ready, it will be fitted to three classifiers to do a comparison study of these three classifiers and select the classifier with the highest performance as measures on the evaluation metric – F_1 score and the time to complete the classification. The three classifiers which will be used in this project are given below:

i. **Decision Tree Classifier:** The Decision Tree Classifier predicts the target by asking a series of if-else questions and dividing the dataset. It's easy to implement and the tree which was used by the classifier to predict the outcome can be visualized which gives a clear picture of how the classifier arrived at the results and this can be used to further fine-tune the model. Since this dataset has all the categorical features, Decision Tree Classifier can fit the data well and should be able to produce good predictions by optimizing the parameters and Overfitting is avoided.

ii. **K-Nearest Neighbor:** K-Nearest Neighbor works on the principle of finding the commonality between the features and predicting the results. For example, if the Buying price and Maintenance price of a car is high then the car is more likely unacceptable. Intuitively, these kinds of patterns exist in the data given the it's categorical nature and K-Nearest Neighbor can find these patterns and use them to make good predictions.

iii. **Gaussian Naïve Bayes:** Gaussian Naïve Bayes classifier are known to perform well on classification problems with categorical datasets and limited dependencies. All the six features of this dataset are categorical and do not have then same attributes which makes them independent of each other. For this reason, Gaussian Naïve Bayes can be a good fit for this classification problem

All these three models will be fit to the dataset and evaluated based on the performance metrics as discussed above. A comparative study of the performance of these models will be done to select one classifier. The selected classifier is now refined to further improve the performance and to make sure it performs better than the Benchmark Model. In order to perform the refinement process, Grid Search CV technique will be implemented wherein the parameters of the selected model are given a range and presented in the form of a Grid. Each and every combination of the parameters is tested and evaluated and the optimum values of the parameters are given at the end which performs the best classification and has the highest metric. The prediction result of this classifier with the optimum parameters is compared with the Benchmark model using the same evaluation metric to ensure that it is performing better. The percentage improvement in performance by optimizing the parameters is also noted and reported.

d. **Bench Mark Model:** In the Introduction of this Report, it was noted that this problem is aimed at increasing the revenue of the junkyard owners and help them identify the cars which are in acceptable or good or very good condition. Usually, all the cars which are sold to the junkyard is considered to be in unacceptable condition and of no use and is dismantled straightway. This is a naïve practice and can be taken as a Benchmark model for this project i.e. predicting 'unacc' for all the cars no matter what the price or the safety of the car is. This is quite common in today's market and the aim of this model is to make predictions better than the Benchmark model i.e. identify the cars which are acceptable or good or very good given certain features of the car.

Precision is given by the ratio of number of examples predicted as positive which were actually positive (True Positives) to all the number of examples predicted as positive irrespective of whether they were positive or not (True Positives + False Positives)
Recall is given by the ratio of number of examples predicted as positive which were actually positive (True Positives) to the total number of examples which were positive whether they were predicted or not (True Positives + False Negatives)

First a series is created in which all the values are 'unacc' and this is taken as prediction:

In [14]:
```
# Creating Benchmark Model - predicting 'unacc' (0) for all the examples

def benchmark(value):
    return 0

Benchmark_CAR = Target_CAR.apply(benchmark)
```

Figure 9: Creating Benchmark Model

Then this is compared with the original Target Label – CAR to get the Benchmark Model F-score

In [15]:
```
#Evaluating Bench Mark Model performance

from sklearn.metrics import f1_score

benchmark_score = f1_score(CAR, Benchmark_CAR, average = 'micro' )
print("Benchmark Model F1 score = {}".format(benchmark_score))
```

```
Benchmark Model F1 score = 0.7002314814814815
```

Figure 10: Evaluating Benchmark Model

Benchmark F-score = 0.700

III. Methodology

a. Data Preprocessing: As mentioned previously, due to the fully categorical nature of the current dataset, the Pre-processing is not very extensive. No logarithmic scaling or transformation is required because there is no numerical data points and there are no missing values so no imputing techniques are needed as well.

The mainly thing which needs to be implemented in this step is to one-hot encode all the labels in order for the data to be fit properly in the model. This is done using the pandas built-in function 'get_dummies'. The number of features in the input data increases after this step since it creates one column for each attribute in a particular feature. Computing the results, we get 21 features after pre-processing.
For the Target Label – the labels are encoded such that if the value is 'unacc'- its replaced by 0, if it 'acc'- its replaced by 1, if it is 'good' – its replaced by 2 and if its 'vgood' – its replaced by 3

```
In [13]: #Prepocessing - 1: One Hot Encoding

         #Splitting features and Target variable
         Target_CAR = data['CAR']
         features = data.drop('CAR', axis = 1)

         #One Hot Encoding the Target Variable
         def one_hot_encode(example):
             if example =='unacc':
                 return 0
             elif example == 'acc':
                 return 1
             elif example == 'good':
                 return 2
             elif example == 'vgood':
                 return 3
             else:
                 print("Invalid entry")
         CAR = Target_CAR.apply(one_hot_encode)

         #Creating Dummies for features
         features_final = pd.get_dummies(features)
         print("Number of features after pre-processing: {}".format(len(features_final.columns)))

         Number of features after pre-processing: 21
```

Figure 11: One hot encoding features and Target Labels

After performing this task, the data is ready to be split into Training and Testing datasets and fit to a classifier for predictions.

```
In [17]: #Preprocessing - 2: Splitting into training and testing datasets

         from sklearn.model_selection import train_test_split

         X_train, X_test, y_train, y_test = train_test_split(features_final, CAR, test_size = 0.2, random_state = 0)

         print("Number of Training examples: {}".format(X_train.shape[0]))
         print("Number of Testing examples: {}".format(X_test.shape[0]))

         Number of Training examples: 1382
         Number of Testing examples: 346
```

Figure 12: Splitting data in training and testing sets

b. **Implementation:** The data is not fitted to all the three classifiers, the predictions are made and the respective f-scores are calculated to compare the performance of all these three classifiers and select the one with the highest performance

```
In [18]: from sklearn.tree import DecisionTreeClassifier
         from sklearn.naive_bayes import GaussianNB
         from sklearn.neighbors import KNeighborsClassifier
         from sklearn.metrics import f1_score


         score_dict = {}
         classifiers = [DecisionTreeClassifier(random_state = 0), GaussianNB(), KNeighborsClassifier()]
         f_scores = []
         def score(classifier):
             clf = classifier.fit(X_train, y_train)
             pred = clf.predict(X_test)
             score = f1_score(y_test, pred, average = 'micro' )
             return score

         for classifier in classifiers:
             name = classifier.__class__.__name__
             score_dict[name] = score(classifier)
             f_scores.append(score(classifier))
             print("{} score = {}".format(name, score_dict[name]))

         DecisionTreeClassifier score = 0.9739884393063584
         GaussianNB score = 0.7687861271676302
         KNeighborsClassifier score = 0.861271676300578
```

Figure 13: Performance of all three classifiers

At first, all the three classifiers are imported from their respective classes and the metric f1_score is imported from sklearn.metrics class.

A function score is defined which takes two inputs – testing set and predictions and returns the f1_score. Since this is a multi-class classification, the micro averaging technique is implemented to calculate the f1_score which takes into account all the true positives, true negatives, false positives and false negatives of all the labels.

A dictionary – score_dict is created, the keys of which are the names of the Classifiers and the respective values are the f-scores.

A list of all the classifiers is iterated, taking once classifier at a time, fitting it to the dataset, making the predictions and making the predictions which is then appended to the dictionary.

The results of f-scores are as follows:

- Decision Tree Classifier – 0.97
- Gaussian Naïve Bayes – 0.77
- K- Neighbors Classifiers – 0.86

The Decision Tre Classifiers gives the highest performance so this model is considered for further refinement by implementing the Grid Search CV technique and optimizing the parameters.

c. **Refinement:** Grid Search CV technique is where all the parameters of a given classifier is given a range of values and a performance metric is created to evaluate the performance – f-score in this case. A grid object is created with the parameter values and the performance metric and is fitted to the training data set. This object takes all the combinations of the given parameter values – one at a time, fits it to the dataset and makes the predictions. The performance of the classifier is measures with each combination of the parameters and the combination which performs the best is taken and used as the final parameter values.

The parameters selected for Decision Tree Classifier are – criterion – 'gini' or 'entropy' and max_depth – range of 1 to 15.
The grid object takes one combination of parameters[criterion = 'gini' and max_depth =1; criterion = 'gini' and max_depth = 2 and so on…..) at a time and uses those parameters in the classifier to fit to the dataset and make predictions.

The model with the optimum parameters is finally used to make predictions and the performance is evaluated. The goal is to improve the performance of the optimum model when compared to the unoptimized initial model.

```
In [21]: from sklearn.grid_search import GridSearchCV
         from sklearn.metrics import make_scorer
         from sklearn.metrics import f1_score

         def scorer(y_true, pred):
             return (f1_score(y_true, pred, average = 'micro'))

         classifier = DecisionTreeClassifier(random_state = 0)
         unoptimized_score = score_dict['DecisionTreeClassifier']
         param_grid = {"criterion": ["gini", "entropy"], "max_depth": list(range(1,15)),}
         grid = GridSearchCV(classifier, param_grid, make_scorer(scorer), cv = 5)
         grid = grid.fit(X_train, y_train)
         optimum_parameters = grid.best_estimator_
         optimum_predictions = optimum_parameters.predict(X_test)
         optimized_score = scorer(y_test, optimum_predictions)
         print("Unoptimized Initial score = {}".format(unoptimized_score))
         print("Optimum score with best parameters = {}".format(optimized_score))
         print("Improved Performance by optimizing the model by {:.2f}%".format(((optimized_score - unoptimized_score) / unoptimized_score
         print("Optimum Parameters: \n {}".format(grid.best_params_))

         Unoptimized Initial score = 0.9739884393063584
         Optimum score with best parameters = 0.9884393063583815
         Improved Performance by optimizing the model by 1.48%
         Optimum Parameters:
          {'criterion': 'entropy', 'max_depth': 13}
```

Figure 14: Grid Search CV technique to improve the performance of Decision Tree Classifier

The f-score of the optimized model after the Grid Search CV technique is implemented is 0.988 which is an improvement over the previous f-score of 0.974 of the unoptimized model.

IV. Methodology

a. Model Evaluation and Validation:

The optimum parameter values of the final value obtained from Grid Search CV are:

- criterion = 'entropy'
- max_depth = 13

Model with these parameter values is fitted to the training set and the prediction is evaluated.
As mentioned previously, the f-score obtained with this model is 0.98 which is over 1.48% improvement over the initial unoptimized model.

In order to check for the robustness of the model, different sample sizes of the training set is trained and the corresponding performance is evaluated and plotted. The graph shows that as the training set size increases, the f-score also increase.
The graph shows an increasing trend showing no sign of smoothing and a threshold value is not obtained. This shows that the model is not overfitting and as the number of training examples increases, the performance of the model further increases. With the current small dataset size of 1782 examples and the hyper-parameter tuning undergone using the Gris Search CV technique this prediction model is optimum. Any further improvement in this model will need more training examples.

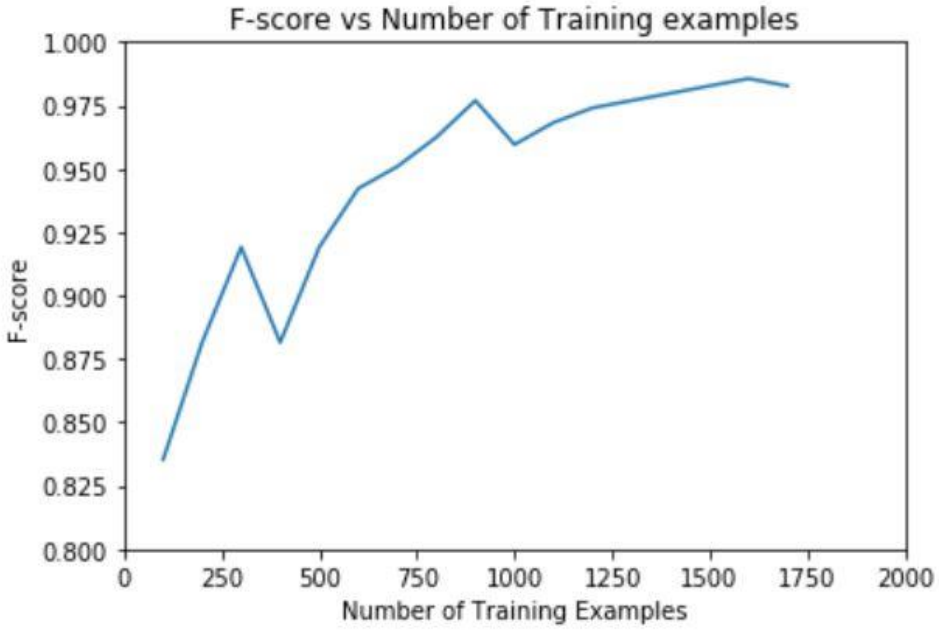

Figure 15: F-score vs Number of Training examples

b. **<u>Justification</u>**: The below graph shows the comparison of the optimized model with the Benchmark model.

As seen, the optimized model performs significantly better than the Benchmark model which is just to predict 'unacc' for all the cars.

This project is aimed at Car Junkyard owners and helping them improve their revenue. Instead of predicting all the cars sold to junkyard as unacceptable and simply dismantle and sell the spare parts as scrap, this model helps in identifying some cars which are either in acceptable condition and some are also in good and very good condition which requires only minor repairs.
With the help of this model, the car junkyard owners can asses the condition and the acceptability of a car and determine what's the best method to handle each car – either disassemble or repair and reuse/resell. Helping the junkyard car owners, identify potentially repairable cars with an accuracy of over 98% helps them increase their revenue and hence their profits by a huge margin.

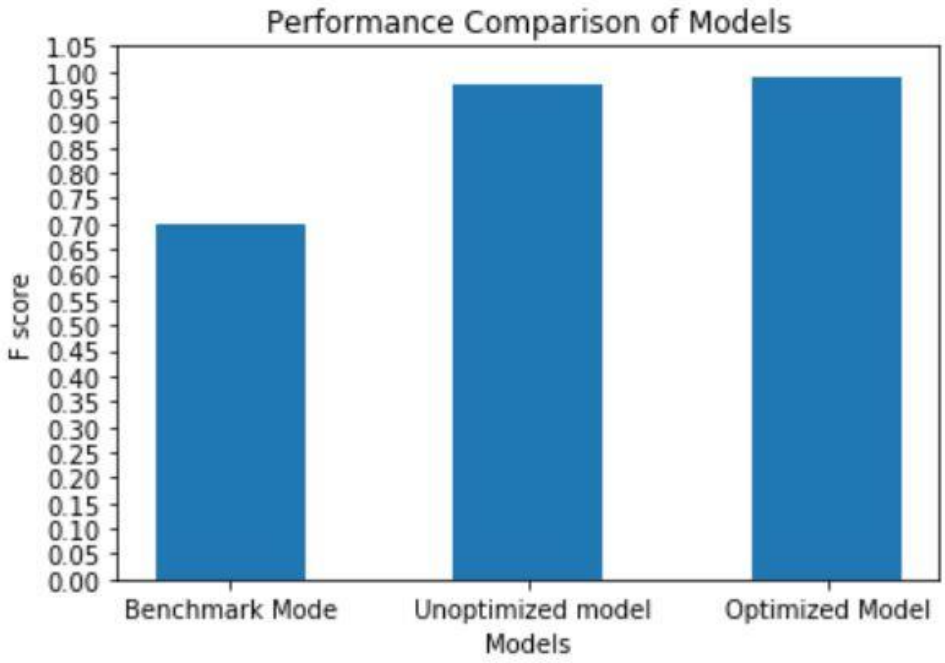

Figure 16: Performance comparison of all the models

V. Conclusion:

a. Free-Form Visualization: One interesting visualization which can help us gain insight into how the model is making predictions is by knowing the feature importance of each feature. This will help in analyzing which features are important for future predictions.

Feature importance is given by the attribute – feature_importances_ of the Decision Tree Classifier object.
The first five important features and their respective weights are given in the below chart.
As given in the chart – the capacity of a car in terms of the number of persons it can carry (persons) is the most important feature followed by safety, Maint_price, size of the luggage boot (lug_boot) and Buying_price. The data used in this project could be from a specified area where there are a lot of families and hence the persons capacity of a car in determining its condition/acceptability rate (whether to scrap or repair) is of utmost importance. If a car can be repaired but cannot be reused/ resell because of its persons capacity then there is no point in repairing the car.

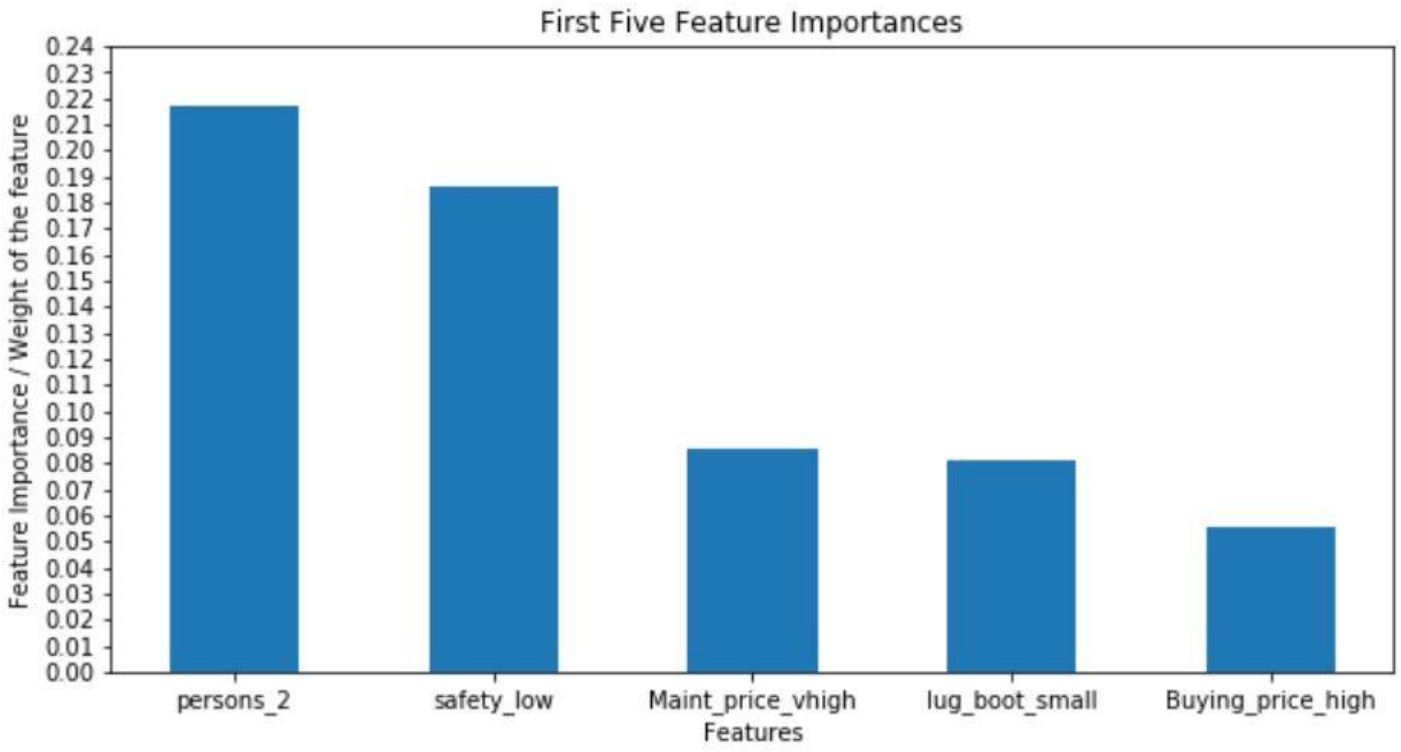

Figure 17: First Five important features and their respective weights

b. **Reflection:** Overall, the project has been a rewarding and a great learning experience. The project seemed quite ambiguous at first with no clarity as to what needed to predicted and how this can benefit but creating a narrative around the project of the Car Junkyard owner really helped in understanding the problem of how the majority of the car are not subjected to any kind of scrutiny before they are disassembled and sold. Looking the problem from the junkyard owner's perspective really helped in understanding the dataset. This kind of thinking brought about other uses and other areas of application where this solution can be useful – such as in warranty of automobiles of automotive manufacturers and also in educating and providing simple tools to the general public about the reliability of a car.

The skewed nature of the data – over 70% of the data belonging to one column greatly helped in deciding the evaluation metric. Selecting accuracy for such skewed data can be quite misleading and was hence avoided. One important learning in terms of deciding the evaluation metric is the averaging technique which needs to be used. Since this is multi class classification problem, either the 'macro' averaging technique needs to be used which calculates the f-scores of each label and then averages the f-score or the 'micro' averaging technique needs to be used which takes the precisions and recalls of all the labels and then calculates the final f-score by averaging all the precisions and recalls. Since the 'micro' technique considers all the labels equally, this averaging technique was used to avoid any bias towards one of the four labels.

The complete categorical nature of the dataset presented some challenges in the begging as this was the first time I was dealing with multi-class classification. One interesting learning was that not all classifiers accept the multiple columns in the output. At first, I was just applying the pd.dummies to the output label – CAR similar to what is done to the input features and fitting the data to the classifiers but this is giving the error that the output has multiple columns. So later on, I had to create a function which would encode different labels as number and apply that to the output column while maintaining the same size.

The Decision Tree Classifier produced very good results in the beginning without any optimization. So improving this further was quite challenging and although the final refinement by optimizing the parameters of the model is only by 1.48%, it is quite significant in the overall performance of the model. I tried different combinations of all the features of the Decision Tree Classifier other than max_depth but the performance did not improve and it was getting computationally expensive and taking longer for the grid object to iterate over all the combinations of the parameters. Since there was no improvement in the performance (f-score), I just removed all the other features and only kept max_depth which was significant in improving the performance. I feel that I reached a hard-stop as far hyper-parameter tuning is concerned for this project. Any further improvement in the performance of the model is possible only by collecting more data/examples which can improve the predictions as is clear from the graph of Number of training samples vs F-score where the F-score is continuously increasing and is on the upward trend.

c. **Improvement:** As mentioned previously, the dataset used in this project is very limited with only 1728 examples. For this reason, this model might not be ready for its application in real life. The model needs to be more robust which is achievable by training the model on more data points. Collecting more data will help in the improvement of this model's predictions.

   Though all the three classifiers used in this project works very well with the prediction of categorical data, it would be worthwhile to test any other classifier and see if it does better than the Decision Tree Classifier.