Data Analytics Lab: Assignment - 4
A mathematical essay on Decision Tree

Nagappan N
Department of Metallurgical and Materials Engineering
IIT Madras
Chennai, India
mm19b040@smail.iitm.ac.in

***Abstract*—The aim of this project is to predict the safety of a car based on its features. We use the decision tree model to attain our goal. Our model makes this prediction with an accuracy of 95%.**

I. Introduction

The main aim of our project is to predict the safety of a car depending on different features related to a car. Knowing the safety of a car is very crucial as thousands of people die everyday due to road accidents. Hence, the safety aspects of a car are important to save lives of people.

Here, we employ decision trees to achieve our task. It is a non-parametric supervised learning method that can be used for both classification and regression problems. Decision trees classify instances by sorting them down the tree from the root to some leaf node, which provides the classification of the instance.

In this project, we classify the cars based on its safety using the decision tree model. Our model is able to make this prediction at an accuracy of 95%. We use input features such as the price of the car, maintenance cost, features of the car and estimated safety for making our predictions.

This paper starts with the description of decision trees. It is followed by the description of the datasets. This includes data visualization. The following section includes details about data processing and how the model has been implemented. Finally, we conclude with the key inferences from our project.

II. Decision Tree

Decision Trees (DTs) are a non-parametric supervised learning method used for classification and regression. A decision tree is a type of tree structure that resembles a flowchart, where each internal node represents a test on an attribute, each branch a test result, and each leaf node (terminal node) a class label.

By dividing the source set into subgroups based on an attribute value test, a tree can be ”trained”. It is known as recursive partitioning to repeat this operation on each derived subset. When the split no longer improves the predictions or when the subset at a node has the same value for the target variable, the recursion is finished.

Decision trees classify instances by arranging them in a tree from the root to a leaf node, which gives the instance's categorization. To classify an instance, one tests the attribute given by the root node of the tree before continuing down the branch of the tree that corresponds to the attribute's value. The subtree rooted at the new node is then subjected to the same procedure once more.

The main benefits of employing the decision tree technique include its simplicity in understanding, minimal data preparation requirements, and cost that scales linearly with the number of data points required to train the tree.

The most important hyperparameters for decision tree are:

- Criterion: The function to measure the quality of a split. Supported criteria are "gini" for the Gini impurity and "log_loss" and "entropy.
- Max_depth: The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.
- Min_samples_split: The minimum number of samples required to split an internal node.
- Min_samples_leaf: The minimum number of samples required to be at a leaf node. A split point at any depth will only be considered if it leaves at least min_samples_leaf training samples in each of the left and right branches.
- Min_weight_fraction_leaf: The minimum weighted fraction of the sum total of weights (of all the input samples) required to be at a leaf node. Samples have equal weight when sample_weight is not provided.
- Max_features: The number of features to consider when looking for the best split.
- Class_weight: Weights associated with classes in the form class_label: weight. If None, all classes are supposed to have weight one.

III. Datasets

First we look at the features in our initial dataset. It has the following features:

1) buying: It is the buying price of the car. This is a categorical variable that takes one of the 4 values: vhigh, high, med, low.
2) maint: Price of maintenance which agains takes either vhigh, high, med or low as its value.
3) doors: Number of doors present in the car which can be 2, 3, 4, 5 or more.
4) persons: It is the number of people who can travel in the car which is also a categorical variable. It takes one of the follwing: 2, 4 or more.
5) lug_boot: This is a categorical variable that describes the size of the luggage boot. It can be small, med or big.
6) safety: This tells us about the estimated safety of the car. This feature takes the values low, med or high.
7) target: This is our target variable which again is a categorical variable that can take one of these values: unacc, acc, good, vgood.

A. Data Visualization

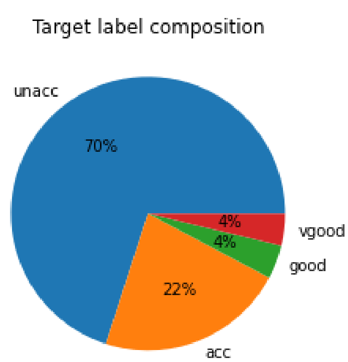


Fig. 1. Pie chart showing the proportion of each class in the target column.

We see that the proportion of classes is very unbalanced in the target column. A majority of them are unacc and only about 8% of them are good or vgood cumulatively.

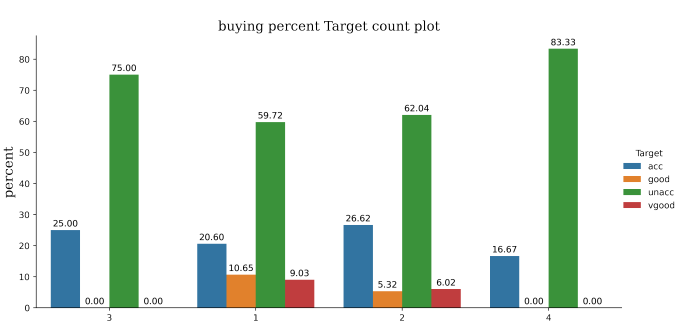


Fig. 2. Percentage of each target class based on the buying price of the car.

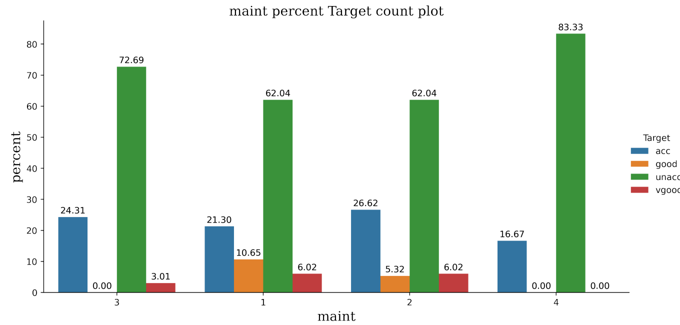


Fig. 3. Percentage of each target class based on the price of maintenance.

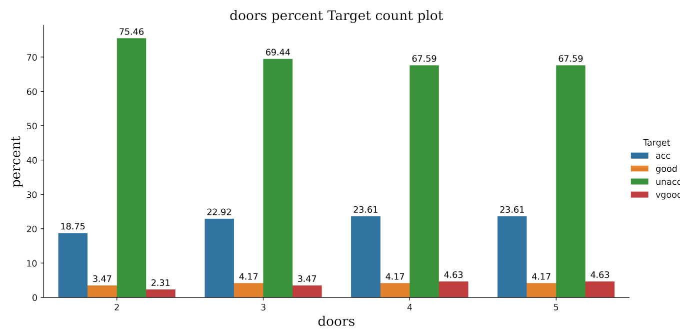


Fig. 4. Percentage of each target class based on the number of doors present in the car.

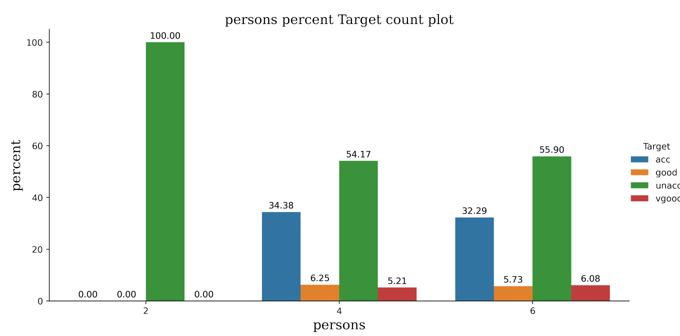


Fig. 5. Percentage of each target class based on the people capacity of the car

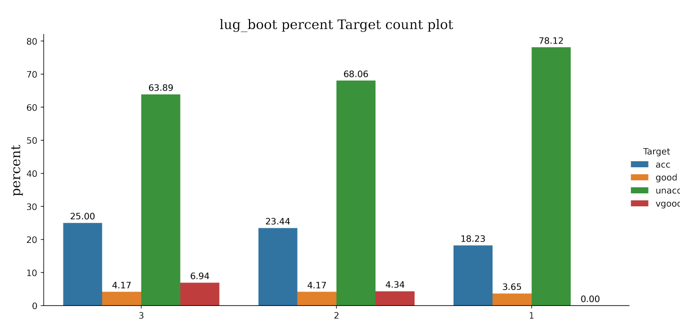


Fig. 6. Percentage of each target class based on the size of the luggage boot.

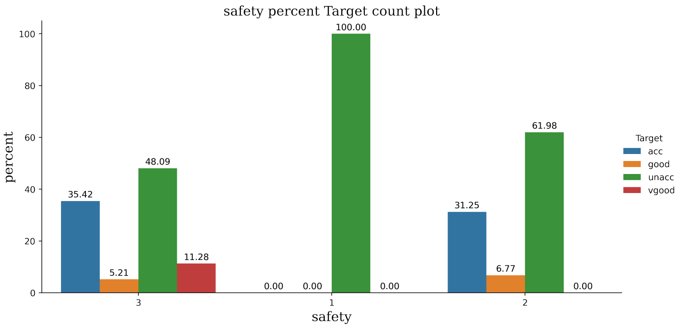
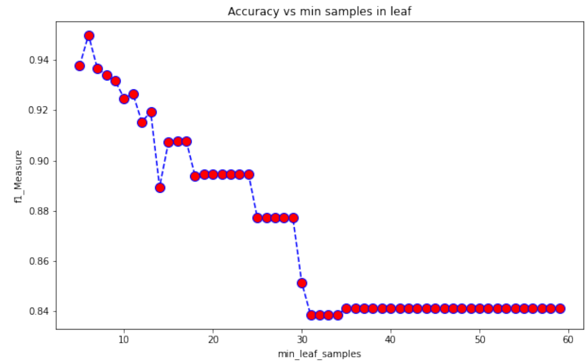


Fig. 7. Percentage of each target class based on the estimated safety of the car.



IV. Modelling

All of the input features may be label encoded with ease because they are all ordinal in nature. Each one starts with one. Because the decision tree module in Scikit Learn does not support nominal categorical variables, this is crucial to our model. Next, the train test split is carried out.

Prior to inputting data into our decision tree model, we compare the composition of the target variable between the train and original compositions. We find that the target variables' composition hasn't altered notably due to the quantity of the dataset.

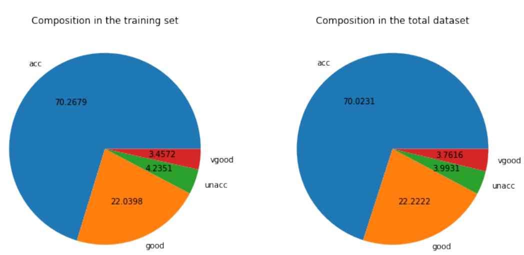


Fig. 8. Pie charts representing the composition of classes in the training dataset and the full dataset.

The decision tree is trained for different min_samples_leaf parameter and their corresponding weighted f1 scores taken into account, to plot and find the best minimum samples in leaf node. The following f1 scores are a weighted average of the normal f1 outcomes for each class. Since F1 scores are the harmonic mean of precision and recall and show how well the model performs when the target variable contains imbalanced classes, they are a more accurate evaluation for classification problems than accuracy scores.

	precision	recall	f1-score	support
acc	0.91	0.88	0.90	129
good	0.59	0.85	0.69	20
unacc	1.00	0.97	0.98	397
vgood	0.80	0.96	0.87	25
accuracy			0.95	571
macro avg	0.82	0.92	0.86	571
weighted avg	0.96	0.95	0.95	571

Fig. 10. Classification report

We have also provided a representation of the tree which has been attached in the appendix.

V. Conclusions

We find that a simple model like a decision tree is good to make predictions for real world problems like this. We have implemented a decision tree on the dataset to predict the safety of a car with an accuracy of 95%. We also see that the F1 scores for most classes is good enough and thus our model is a good fit even for the imbalanced dataset.


safety <= 1.5 gini = 0.455 samples = 1157 value = [255, 49, 813, 40] class = unacc
True
False
gini = 0.0 samples = 386 value = [0, 0, 386, 0] class = unacc
persons <= 3.0 gini = 0.577 samples = 771 value = [255, 49, 427, 40] class = unacc
gini = 0.0 samples = 246 value = [0, 0, 246, 0] class = unacc
buying <= 2.5 gini = 0.631 samples = 525 value = [255, 49, 181, 40] class = acc
maint <= 1.5 gini = 0.654 samples = 254 value = [131, 49, 34, 40] class = acc
maint <= 2.5 gini = 0.496 samples = 271 value = [124, 0, 147, 0] class = unacc
safety <= 2.5 gini = 0.617 samples = 64 value = [9, 34, 3, 18] class = good
lug_boot <= 1.5 gini = 0.541 samples = 190 value = [122, 15, 31, 22] class = acc
safety <= 2.5 gini = 0.382 samples = 140 value = [104, 0, 36, 0] class = acc
buying <= 3.5 gini = 0.259 samples = 131 value = [20, 0, 111, 0] class = unacc
lug_boot <= 1.5 gini = 0.464 samples = 30 value = [9, 20, 1, 0] class = good
lug_boot <= 1.5 gini = 0.547 samples = 34 value = [0, 14, 2, 18] class = vgood
doors <= 2.5 gini = 0.551 samples = 71 value = [40, 6, 25, 0] class = acc
maint <= 2.5 gini = 0.483 samples = 119 value = [82, 9, 6, 22] class = acc
lug_boot <= 1.5 gini = 0.5 samples = 70 value = [36, 0, 34, 0] class = acc
doors <= 2.5 gini = 0.056 samples = 70 value = [68, 0, 2, 0] class = acc
maint <= 3.5 gini = 0.441 samples = 61 value = [20, 0, 41, 0] class = unacc
gini = 0.0 samples = 70 value = [0, 0, 70, 0] class = unacc
gini = 0.198 samples = 9 value = [8, 0, 1, 0] class = acc
lug_boot <= 2.5 gini = 0.091 samples = 21 value = [1, 20, 0, 0] class = good
gini = 0.298 samples = 11 value = [0, 9, 2, 0] class = good
doors <= 2.5 gini = 0.34 samples = 23 value = [0, 5, 0, 18] class = vgood
persons <= 5.0 gini = 0.443 samples = 17 value = [4, 1, 12, 0] class = unacc
safety <= 2.5 gini = 0.489 samples = 54 value = [36, 5, 13, 0] class = acc
safety <= 2.5 gini = 0.656 samples = 35 value = [14, 9, 0, 12] class = acc
buying <= 1.5 gini = 0.325 samples = 84 value = [68, 0, 6, 10] class = acc
gini = 0.0 samples = 24 value = [0, 0, 24, 0] class = unacc
doors <= 3.5 gini = 0.34 samples = 46 value = [36, 0, 10, 0] class = acc
persons <= 5.0 gini = 0.219 samples = 16 value = [14, 0, 2, 0] class = acc
gini = 0.0 samples = 54 value = [54, 0, 0, 0] class = acc
safety <= 2.5 gini = 0.428 samples = 29 value = [20, 0, 9, 0] class = acc
gini = 0.0 samples = 32 value = [0, 0, 32, 0] class = unacc
gini = 0.219 samples = 8 value = [1, 7, 0, 0] class = good
gini = 0.0 samples = 13 value = [0, 13, 0, 0] class = good
gini = 0.49 samples = 7 value = [0, 4, 0, 3] class = good
buying <= 1.5 gini = 0.117 samples = 16 value = [0, 1, 0, 15] class = vgood
gini = 0.5 samples = 6 value = [4, 1, 1, 0] class = acc
gini = 0.0 samples = 11 value = [0, 0, 11, 0] class = unacc
maint <= 2.5 gini = 0.497 samples = 28 value = [15, 0, 13, 0] class = acc
maint <= 2.5 gini = 0.311 samples = 26 value = [21, 5, 0, 0] class = acc
buying <= 1.5 gini = 0.463 samples = 22 value = [14, 8, 0, 0] class = acc
buying <= 1.5 gini = 0.142 samples = 13 value = [0, 1, 0, 12] class = vgood
maint <= 3.5 gini = 0.405 samples = 41 value = [30, 0, 1, 10] class = acc
doors <= 3.5 gini = 0.206 samples = 43 value = [38, 0, 5, 0] class = acc
lug_boot <= 2.5 gini = 0.491 samples = 23 value = [13, 0, 10, 0] class = acc
gini = 0.0 samples = 23 value = [23, 0, 0, 0] class = acc
gini = 0.0 samples = 10 value = [10, 0, 0, 0] class = acc
gini = 0.444 samples = 6 value = [4, 0, 2, 0] class = acc
lug_boot <= 1.5 gini = 0.48 samples = 15 value = [6, 0, 9, 0] class = unacc
gini = 0.0 samples = 14 value = [14, 0, 0, 0] class = acc
gini = 0.219 samples = 8 value = [0, 1, 0, 7] class = vgood
gini = 0.0 samples = 8 value = [0, 0, 0, 8] class = vgood
gini = 0.0 samples = 11 value = [11, 0, 0, 0] class = acc
maint <= 3.5 gini = 0.36 samples = 17 value = [4, 0, 13, 0] class = unacc
gini = 0.469 samples = 8 value = [3, 5, 0, 0] class = good
gini = 0.0 samples = 18 value = [18, 0, 0, 0] class = acc
gini = 0.0 samples = 8 value = [0, 8, 0, 0] class = good
gini = 0.0 samples = 14 value = [14, 0, 0, 0] class = acc
gini = 0.278 samples = 6 value = [0, 1, 0, 5] class = vgood
gini = 0.0 samples = 7 value = [0, 0, 0, 7] class = vgood
safety <= 2.5 gini = 0.496 samples = 22 value = [12, 0, 0, 10] class = acc
doors <= 3.5 gini = 0.1 samples = 19 value = [18, 0, 1, 0] class = acc
safety <= 2.5 gini = 0.363 samples = 21 value = [16, 0, 5, 0] class = acc
gini = 0.0 samples = 22 value = [22, 0, 0, 0] class = acc
doors <= 2.5 gini = 0.355 samples = 13 value = [3, 0, 10, 0] class = unacc
gini = 0.0 samples = 10 value = [10, 0, 0, 0] class = acc
gini = 0.0 samples = 6 value = [0, 0, 6, 0] class = unacc
gini = 0.444 samples = 9 value = [6, 0, 3, 0] class = acc
gini = 0.5 samples = 8 value = [4, 0, 4, 0] class = acc
gini = 0.0 samples = 9 value = [0, 0, 9, 0] class = unacc
gini = 0.0 samples = 9 value = [9, 0, 0, 0] class = acc
doors <= 3.5 gini = 0.355 samples = 13 value = [3, 0, 0, 10] class = vgood
gini = 0.245 samples = 7 value = [6, 0, 1, 0] class = acc
gini = 0.0 samples = 12 value = [12, 0, 0, 0] class = acc
gini = 0.496 samples = 11 value = [6, 0, 5, 0] class = acc
gini = 0.0 samples = 10 value = [10, 0, 0, 0] class = acc
gini = 0.0 samples = 6 value = [0, 0, 6, 0] class = unacc
gini = 0.49 samples = 7 value = [3, 0, 4, 0] class = unacc
gini = 0.49 samples = 7 value = [3, 0, 0, 4] class = vgood
gini = 0.0 samples = 6 value = [0, 0, 0, 6] class = vgood

EE4708: Data Analytics Lab

Assignment 4

Name: Nagappan N

Roll number: MM19B040

Importing Libraries

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
```

In [2]:
```
d = pd.read_excel('car_evaluation.xlsx')
```

In [3]:
```
d.head()
```

Out[3]:

	buying	maint	doors	persons	lug_boot	safety	Target
0	vhigh	vhigh	2	2	small	low	unacc
1	vhigh	vhigh	2	2	small	med	unacc
2	vhigh	vhigh	2	2	small	high	unacc
3	vhigh	vhigh	2	2	med	low	unacc
4	vhigh	vhigh	2	2	med	med	unacc

In [4]:
```
d.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1728 entries, 0 to 1727
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   buying    1728 non-null   object
 1   maint     1728 non-null   object
 2   doors     1728 non-null   object
 3   persons   1728 non-null   object
 4   lug_boot  1728 non-null   object
 5   safety    1728 non-null   object
 6   Target    1728 non-null   object
dtypes: object(7)
memory usage: 94.6+ KB
```

Exploratory Data analysis of features

In [5]:
```
d['buying'].value_counts()
```

Out[5]:
```
high     432
vhigh    432
low      432
med      432
Name: buying, dtype: int64
```

In [6]:
```
d['maint'].value_counts()
```

Out[6]:
```
high     432
vhigh    432
low      432
med      432
Name: maint, dtype: int64
```

In [7]:
```
d['doors'].value_counts()
```

Out[7]:
```
2        432
3        432
4        432
5more    432
Name: doors, dtype: int64
```

In [8]:
```
d['persons'].value_counts()
```

Out[8]:
```
2       576
4       576
more    576
Name: persons, dtype: int64
```

In [9]:
```
d['lug_boot'].value_counts()
```

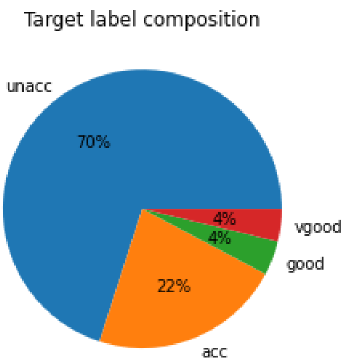
Out[9]:
```
big      576
small    576
med      576
Name: lug_boot, dtype: int64
```

In [10]:
```
d['safety'].value_counts()
```

Out[10]:
```
high    576
low     576
med     576
Name: safety, dtype: int64
```

We can see all the input features are well balanced

In [11]:
```
data=d['Target'].value_counts()
keys=d['Target'].value_counts().keys()
plt.title('Target label composition')
a=plt.pie(data, labels=keys, autopct='%.0f%%',)
plt.savefig('Targetpie.png')
```



The output target variable is an imbalanced multiclass variable as we can see from the pie chart above

In [12]:
```
d['buying'] = d['buying'].astype('category')
d['maint'] = d['maint'].astype('category')
d['doors'] = d['doors'].astype('category')
d['persons'] = d['persons'].astype('category')
d['lug_boot'] = d['lug_boot'].astype('category')
d['safety'] = d['safety'].astype('category')
d['Target'] = d['Target'].astype('category')
```

In [13]:

```
d.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1728 entries, 0 to 1727
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   buying    1728 non-null   category
 1   maint     1728 non-null   category
 2   doors     1728 non-null   category
 3   persons   1728 non-null   category
 4   lug_boot  1728 non-null   category
 5   safety    1728 non-null   category
 6   Target    1728 non-null   category
dtypes: category(7)
memory usage: 13.1 KB
```

Data Visualization

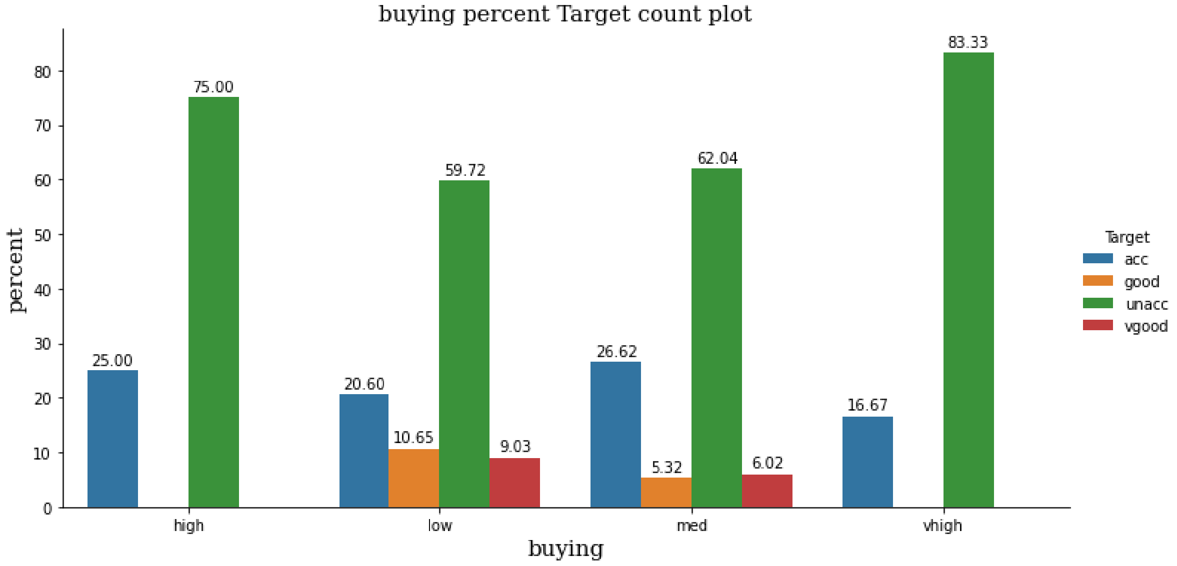
In [14]:

```
def funcplot(x1):
    y='Target'
    dfp=(d
    .groupby(x1)[y]
    .value_counts(normalize=True)
    .mul(100)
    .rename('percent')
    .reset_index())
    plt.figure(figsize=[15,7])
    g=sns.catplot(x=x1,y='percent',hue=y,data=dfp,kind='bar',height=5,aspect=2)
    for container in g.ax.containers:
        g.ax.bar_label(container, fmt='%.2f', padding=2)
    font1 = {'family':'serif','color':'black','size':15}
    font2 = {'family':'serif','color':'black','size':15}
    plt.title(x1+' percent Target count plot', fontdict=font1)
    plt.ylabel('percent',fontdict=font2)
    plt.xlabel(x1,fontdict=font2)
    plt.savefig(x1+'catplot.png',pad_inches=0,bbox_inches='tight',dpi=400)
```

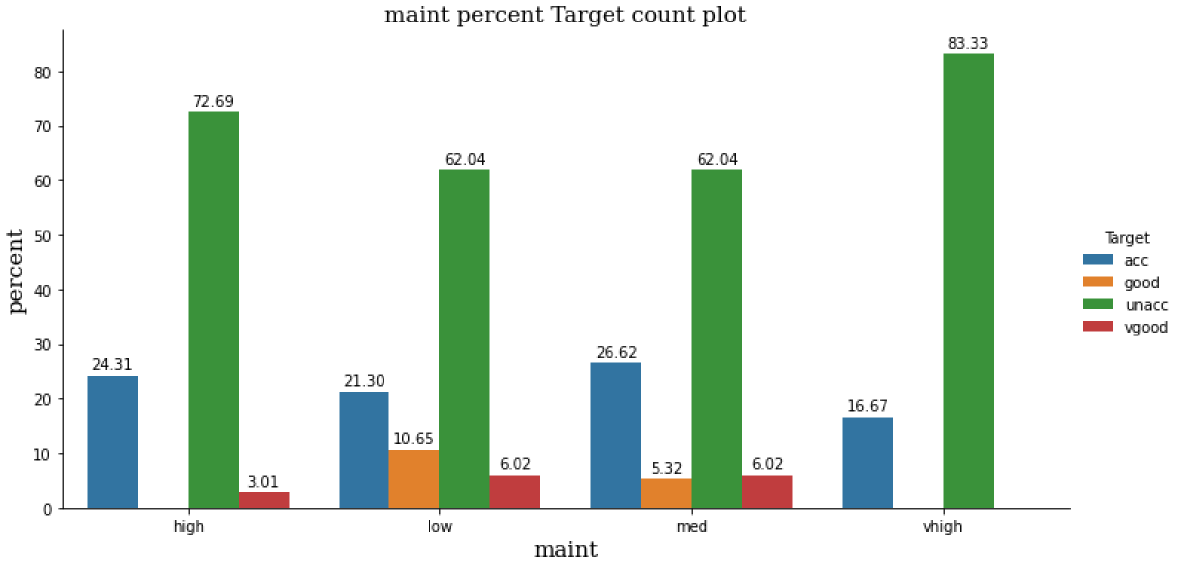
The following graphs visualize the composition of target class among each input variable category to analyse.

In [15]:
```python
for x in d.columns:
    if(x!='Target'):
        funcplot(x)
```

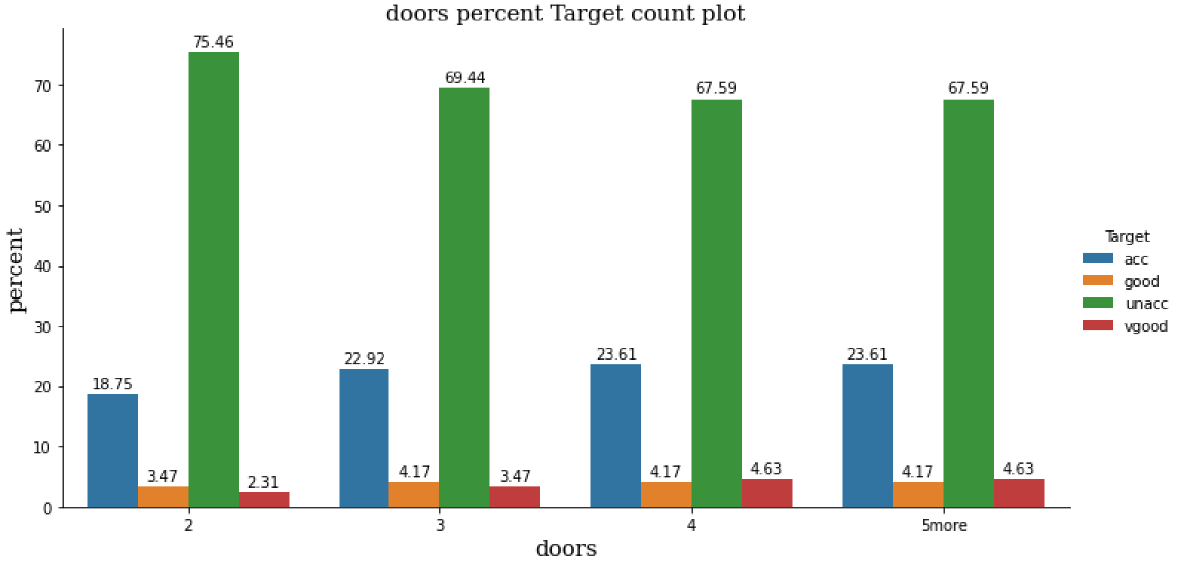
<Figure size 1080x504 with 0 Axes>



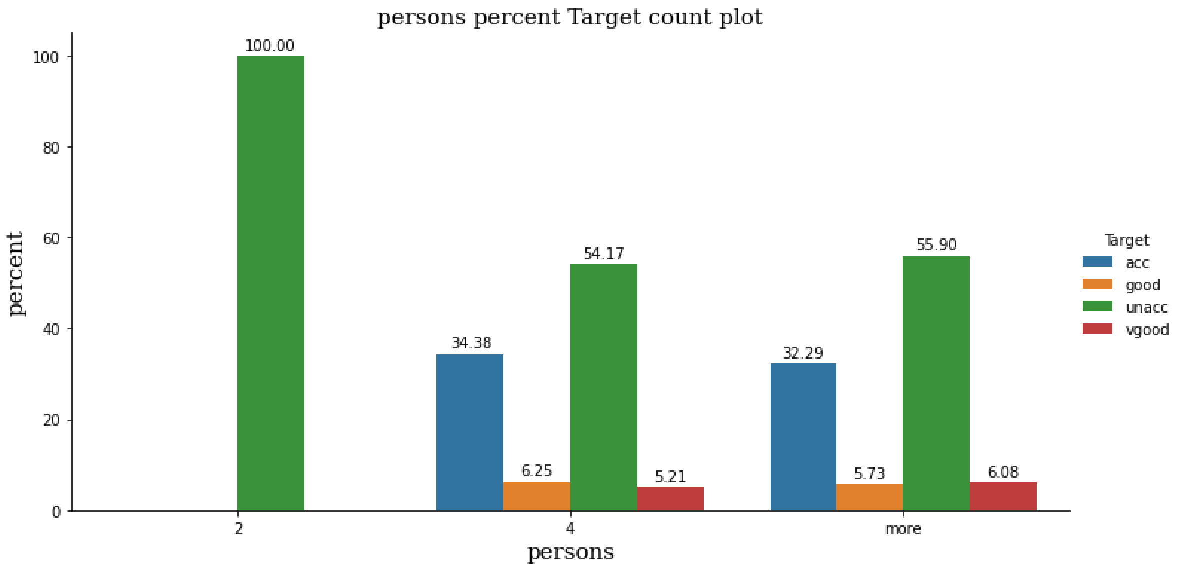
<Figure size 1080x504 with 0 Axes>



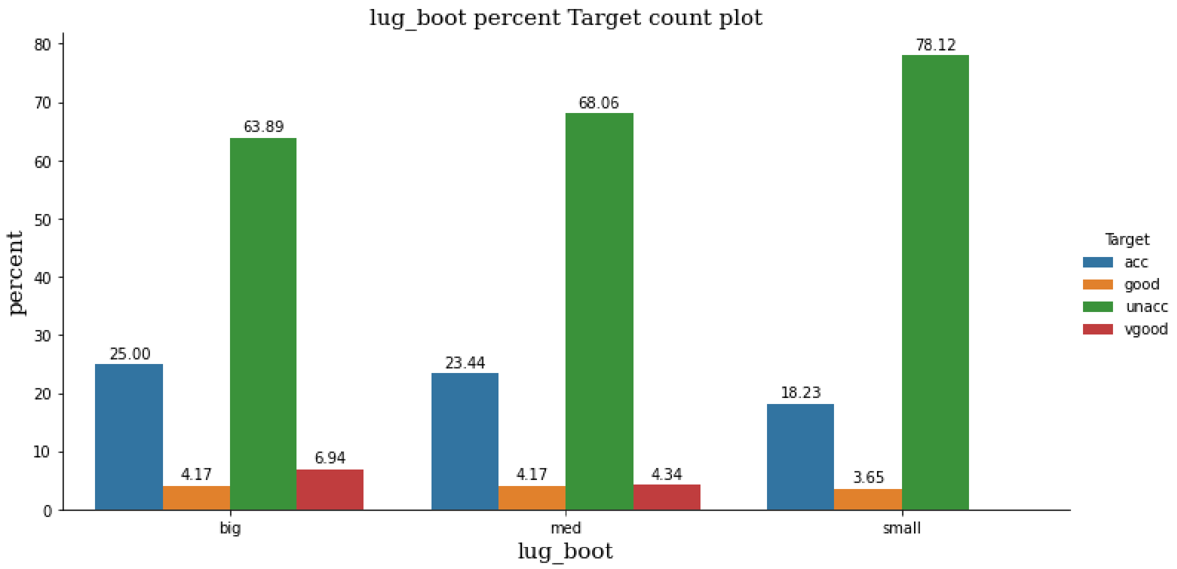
<Figure size 1080x504 with 0 Axes>



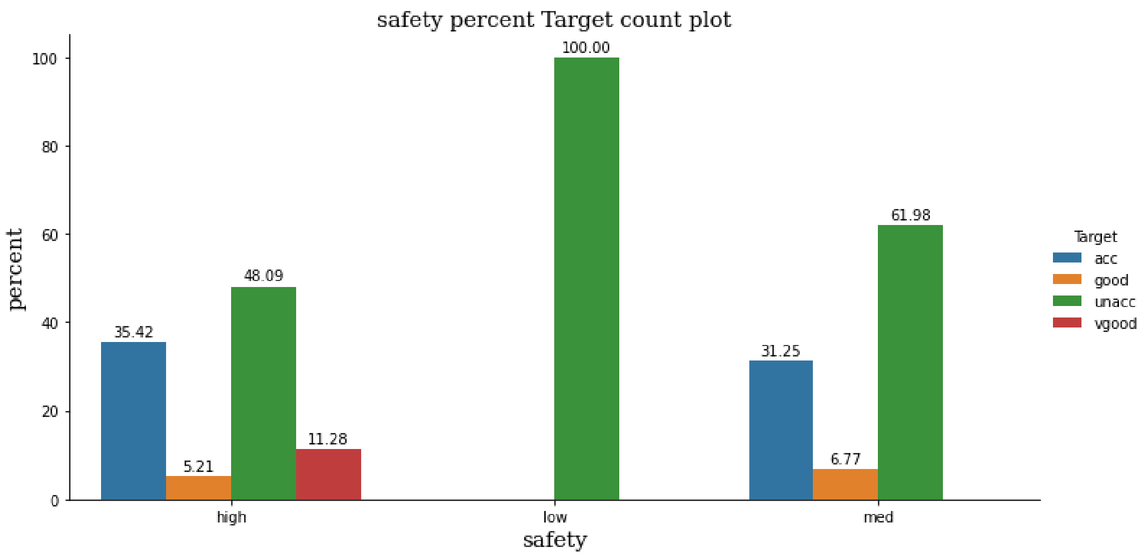
<Figure size 1080x504 with 0 Axes>



<Figure size 1080x504 with 0 Axes>



<Figure size 1080x504 with 0 Axes>



Label Encoding of ordinal input features

As we observe all the input features are ordinal we just map them into respective labels.

In [16]:
```
mappings={'low':1,'med':2,'high':3,'vhigh':4,'5more':5,'more':6,'small':1,'big':3}
d.replace(mappings,inplace=True)
```

Train Test Split

In [17]:
```
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report,f1_score
```

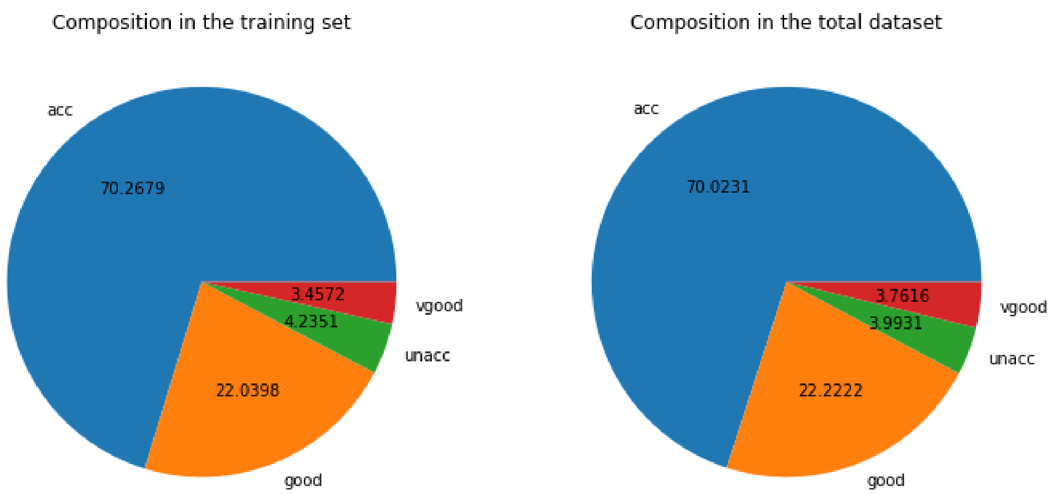
In [18]:
```
X = d.drop('Target',axis=1)
y = d['Target']
```

In [19]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)
```

In [20]:
```
y_train.value_counts(normalize=True)
```

Out[20]:
```
unacc    0.702679
acc      0.220398
good     0.042351
vgood    0.034572
Name: Target, dtype: float64
```

In [21]:
```
plt.figure(figsize=(12,8))
plt.subplot(1,2,1)
data=y_train.value_counts(normalize=True)
keys=np.unique(y_train)
data2=y.value_counts(normalize=True)
plt.title('Composition in the training set')
a=plt.pie(data, labels=keys, autopct='%.4f')
plt.subplot(1,2,2)
plt.title('Composition in the total dataset')
b=plt.pie(data2, labels=keys, autopct='%.4f')
plt.savefig('compo.png')
```



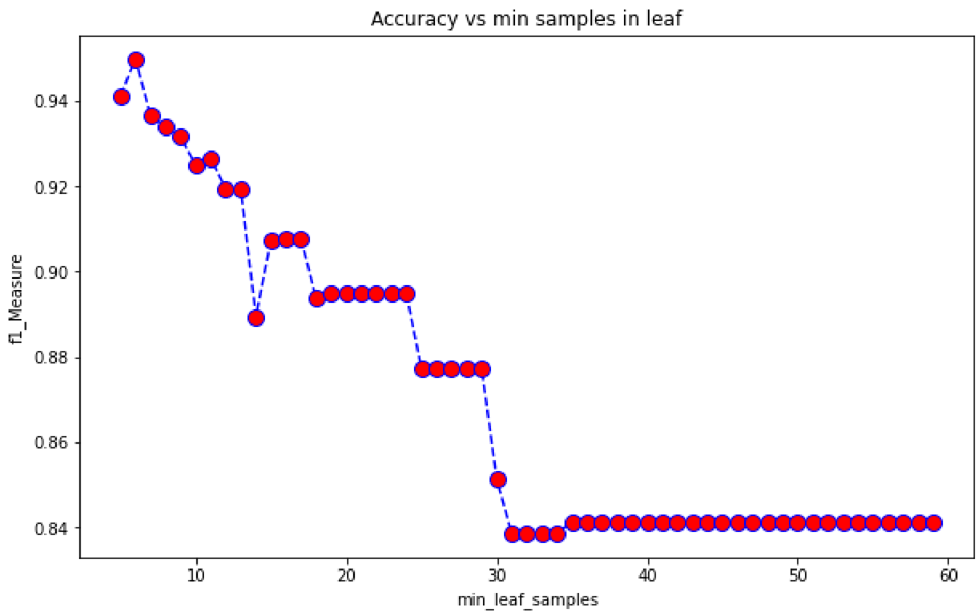
In [22]:
```
y.value_counts(normalize=True)
```

Out[22]:
```
unacc    0.700231
acc      0.222222
good     0.039931
vgood    0.037616
Name: Target, dtype: float64
```

Decision Tree

In [23]:
```
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier,plot_tree
```

In [24]:
```
a=[]
for i in range(5,60,1):
    dt = DecisionTreeClassifier(min_samples_leaf=i,criterion='gini')
    dt.fit(X_train,y_train)
    pred=dt.predict(X_test)
    a.append(f1_score(y_test,pred,average='weighted'))
plt.figure(figsize=(10,6))
plt.plot(range(5,60,1),a,color='blue', linestyle='dashed', marker='o',
         markerfacecolor='red', markersize=10)
plt.title('Accuracy vs min samples in leaf')
plt.xlabel('min_leaf_samples')
plt.ylabel('f1_Measure')
plt.savefig('f1.png')
```



In [25]:
```
dt=DecisionTreeClassifier(min_samples_leaf=6)
dt.fit(X_train,y_train)
```

Out[25]:
```
DecisionTreeClassifier(min_samples_leaf=6)
```

In [26]:
```
pred= dt.predict(X_test)
```

In [27]:
```
confusion_matrix(y_test, pred)
```

Out[27]:
```
array([[114,  11,   1,   3],
       [  0,  17,   0,   3],
       [ 11,   0, 386,   0],
       [  0,   1,   0,  24]])
```

In [28]:
```
print(classification_report(y_test, pred))
```

```
              precision    recall  f1-score   support

         acc       0.91      0.88      0.90       129
        good       0.59      0.85      0.69        20
       unacc       1.00      0.97      0.98       397
       vgood       0.80      0.96      0.87        25

    accuracy                           0.95       571
   macro avg       0.82      0.92      0.86       571
weighted avg       0.96      0.95      0.95       571
```

Visual interpretation of the tree

In [29]:
```
from sklearn import tree
import graphviz
dot_data = tree.export_graphviz(dt, out_file=None,
                                feature_names=X.columns,class_names=['acc','good','unacc','vgood'
                                filled=True)
G=graphviz.Source(dot_data)
G.render(filename='Decisiontree',format='pdf')
```

Out[29]: 'Decisiontree.pdf'