**Yash Shah**

**J060**

In [3]:

**import**

**os**

print

(

os

.

getcwd

())

**import**

**pandas**

**as**

**pd**

**import**

**numpy**

**as**

**np**

**import**

**matplotlib.pyplot**

**as**

**plt**

%

**matplotlib**

inline

C:\Users\User\Downloads In [4]:

df

=

pd

.

read\_csv

(

'car\_evaluation.csv'

,

header

=

**None**

)

In [5]:

df

.

head

()

Out[5]:

**0 1 2 3 4 5 6**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **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 [6]:

col\_names

=

[

'buying'

,

'maint'

,

'doors'

,

'persons'

,

'lug\_boot'

,

'safety'

,

'class'

]

df

.

columns

=

col\_names

col\_names

Out[6]:

['buying', 'maint', 'doors', 'persons', 'lug\_boot', 'safety', 'class'] In [7]:

df

.

head

()

Out[7]: **buying maint doors persons lug\_boot safety class**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **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 [8]:

df

.

info

()

<class 'pandas.core.frame.DataFrame'> RangeIndex: 1728 entries, 0 to 1727

Data columns (total 7 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

1. buying 1728 non-null object
2. maint 1728 non-null object
3. doors 1728 non-null object
4. persons 1728 non-null object
5. lug\_boot 1728 non-null object
6. safety 1728 non-null object
7. class 1728 non-null object dtypes: object(7) memory usage: 47.3+ KB

In [9]:

**for**

i

**in**

col\_names

:

print

(

df

[

i

]

.

value\_counts

())

high 432 low 432 med 432 vhigh 432

Name: buying, dtype: int64 high 432 low 432 med 432 vhigh 432

Name: maint, dtype: int64

1. 432
2. 432

5more 432

2 432

Name: doors, dtype: int64 more 576 4 576

2 576

Name: persons, dtype: int64 small 576 big 576 med 576

Name: lug\_boot, dtype: int64 high 576 low 576 med 576

Name: safety, dtype: int64 unacc 1210 acc 384 good 69 vgood 65

Name: class, dtype: int64 In [10]:

df

.

shape

Out[10]:

(1728, 7) In [11]:

X

=

df

.

drop

([

'class'

]

,

axis

=

1

)

y

=

df

[

'class'

]

In [12]:

**from** **sklearn.model\_selection** **import** train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y,test\_size=0.3,random\_state=42)

In [13]:

**from**

**sklearn.preprocessing**

**import**

OrdinalEncoder

enc

=

OrdinalEncoder

()

X\_train

=

enc

.

fit\_transform

(

X\_train

)

X\_test

=

enc

.

transform

((

X\_test

))

**Gini index as criterion**

In [14]:

**from**

**sklearn.tree**

**import**

DecisionTreeClassifier

In [15]:

clf\_gini = DecisionTreeClassifier(criterion='gini', max\_depth=3, random\_state=42) clf\_gini.fit(X\_train, y\_train)

Out[15]:

DecisionTreeClassifier(class\_weight=None, criterion='gini', max\_depth=3, max\_features=None, max\_leaf\_nodes=None, min\_impurity\_decrease=0.0, min\_impurity\_split=None, min\_samples\_leaf=1, min\_samples\_split=2, min\_weight\_fraction\_leaf=0.0, presort=False, random\_state=42, splitter='best')

In [16]:

y\_pred

=

clf\_gini

.

predict

(

X\_test

)

# Grid Search Cv

In [17]:

**from** **sklearn.model\_selection** **import** GridSearchCV option=['gini','entropy'] weight\_option=['auto','sqrt','log2']

param\_grid = {'criterion': option , 'max\_features':[2,3,4,5,6] , 'max\_depth':[4,5,6,7]

, 'min\_samples\_split':[2,3,4,5]}

grid=GridSearchCV(clf\_gini,param\_grid,cv=3,scoring='accuracy') grid.fit(X\_train,y\_train) print(grid.best\_score\_) print(grid.best\_params\_)

0.9247311827956989

{'criterion': 'gini', 'max\_depth': 7, 'max\_features': 6, 'min\_samples\_spli t': 2}

In [18]:

**from**

**sklearn**

**import**

tree

plt

.

figure

(

figsize

=

(

15

,

8

))

tree

.

plot\_tree

(

clf\_gini

,

feature\_names

=

[

'buying'

,

'maint'

,

'doors'

,

'persons'

,

'lug\_boot'

,

'safet

y'

]

,

class\_names

=

list

(

set

(

y\_train

))

,

filled

=

**True**

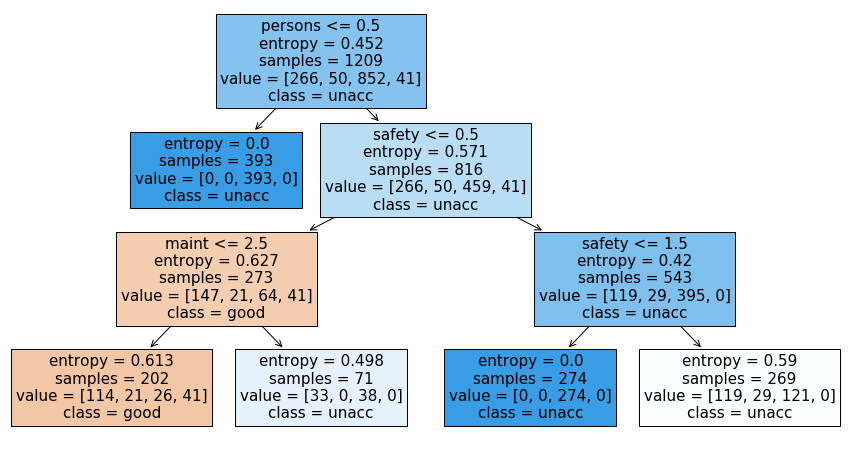
)

plt

.

show

()



In [19]:

*# Check for underfitting*

print

(

f

'Training set score: {clf\_gini.score(X\_train,y\_train)}'

)

print

(

f

'Test set score: {clf\_gini.score(X\_test,y\_test)}'

)

Training set score: 0.7775020678246485 Test set score: 0.7572254335260116

# Model after grid search

In [20]:

dtc = DecisionTreeClassifier(criterion='gini', max\_depth=7,max\_features = 6) dtc.fit(X\_train, y\_train)

Out[20]:

DecisionTreeClassifier(class\_weight=None, criterion='gini', max\_depth=7, max\_features=6, max\_leaf\_nodes=None,

min\_impurity\_decrease=0.0, min\_impurity\_split=None, min\_samples\_leaf=1, min\_samples\_split=2, min\_weight\_fraction\_leaf=0.0, presort=False, random\_state=None, splitter='best')

In [21]:

y\_pred

=

dtc

.

predict

(

X\_test

)

In [22]:

print(f'Training set score: {dtc.score(X\_train,y\_train)}') print(f'Test set score: {dtc.score(X\_test,y\_test)}')

Training set score: 0.9330024813895782

Test set score: 0.9344894026974951 In [23]:

**from**

**sklearn**

**import**

tree

plt

.

figure

(

figsize

=

(

15

,

8

))

tree

.

plot\_tree

(

dtc

,

feature\_names

=

[

'buying'

,

'maint'

,

'doors'

,

'persons'

,

'lug\_boot'

,

'safet

y'

]

,

class\_names

=

list

(

set

(

y\_train

))

,

filled

=

**True**

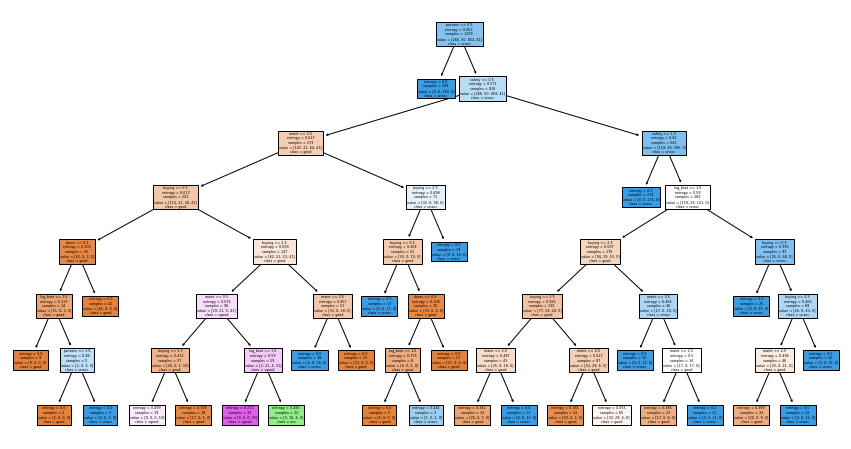
)

plt

.

show

()



**Cross validation**

In [24]:

**from** **sklearn.model\_selection** **import** cross\_val\_score score=cross\_val\_score(dtc,X\_train,y\_train,cv=10,scoring='accuracy') score.mean()

Out[24]:

0.9204997229758234

In [25]:

**from** **sklearn.model\_selection** **import** cross\_val\_score score=cross\_val\_score(dtc,X\_test,y\_test,cv=10,scoring='accuracy') score.mean() Out[25]:

0.891906223156897

In [26]:

**from** **sklearn.metrics** **import** confusion\_matrix, classification\_report cm = confusion\_matrix(y\_test, y\_pred) In [27]:

print

(

cm

)

[[109 4 1 4]

[ 10 6 0 3]

[ 11 0 346 1]

[ 0 0 0 24]] In [28]:

print

(

classification\_report

(

y\_test

,

y\_pred

))

precision recall f1-score support

acc 0.84 0.92 0.88 118 good 0.60 0.32 0.41 19 unacc 1.00 0.97 0.98 358 vgood 0.75 1.00 0.86 24

accuracy 0.93 519 macro avg 0.80 0.80 0.78 519 weighted avg 0.94 0.93 0.93 519