Assignment: Tuning Classifier Model with XGBoost

The comments/sections provided are your cues to perform the assignment. You don't need to limit yourself to the number of rows/cells provided. You can add additional rows in each section to add more lines of code.

If at any point in time you need help on solving this assignment, view our demo video to understand the different steps of the code.

Happy coding!

Tuning Classifier Model with XGBoost

DESCRIPTION

The used car market has significantly grown in recent times, with clients ranging from used car dealers and buyers. You are provided with a car evaluation dataset that has features like price, doors, safety, and so on.

Objective: You are required to create a robust model that allows stakeholders to predict the condition of a used vehicle.

Actions to Perform:

- Predict the condition of a vehicle based on its features.
- Plot the most important features.
- Train multiple classifiers and compare the accuracy.
- Evaluate the XGBoost model with K-fold cross-validation.

Predict the condition of a vehicle based on features

In [1]:

```
#import the libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

import warnings
warnings.filterwarnings('ignore')
```

In [2]:

```
#read the dataset
data=pd.read_csv('car_evaluation.csv')
```

In [3]:

```
data.head()
```

Out[3]:

	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 [4]:

```
data.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   class     1728 non-null   object
dtypes: object(7)
memory usage: 47.3+ KB
```

Columns are categorical, check for unique values of each column.

In [5]:

```
for i in data.columns:
    print(data[i].unique(),"\t",data[i].nunique())
```

```
['vhigh' 'high' 'med' 'low']      4
['vhigh' 'high' 'med' 'low']      4
['2' '3' '4' '5more']    4
['2' '4' 'more']         3
['small' 'med' 'big']    3
['low' 'med' 'high']     3
['unacc' 'acc' 'vgood' 'good']    4
```

Checking how these unique categories are distributed among the columns

In [6]:

```
for i in data.columns:
    print(data[i].value_counts())
    print()
```

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

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

4        432
2        432
3        432
5more    432
Name: doors, dtype: int64

4       576
2       576
more    576
Name: persons, dtype: int64

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

low     576
med     576
high    576
Name: safety, dtype: int64

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

From above output, it is clear that all the columns except 'class' are distributed equally among data.

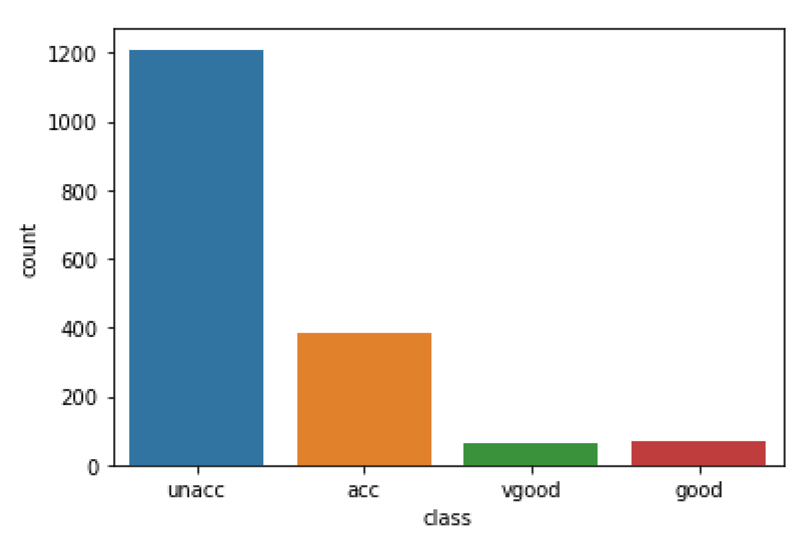
Plot the most important features

Class Distribution

In [7]:

```
sns.countplot(data['class'])
```

Out[7]: <AxesSubplot:xlabel='class', ylabel='count'>



It can be seen from the graph that the result 'class' is unbalanced with larger values of 'unacc'. So, this is an "unbalanced multiclass classification problem".

Dummy encoding

In [8]:

```
from sklearn.preprocessing import LabelEncoder
```

In [9]:

```
le=LabelEncoder()
```

In [10]:

```
for i in data.columns:
    data[i]=le.fit_transform(data[i])
```

In [11]:

```
data.head()
```

Out[11]:

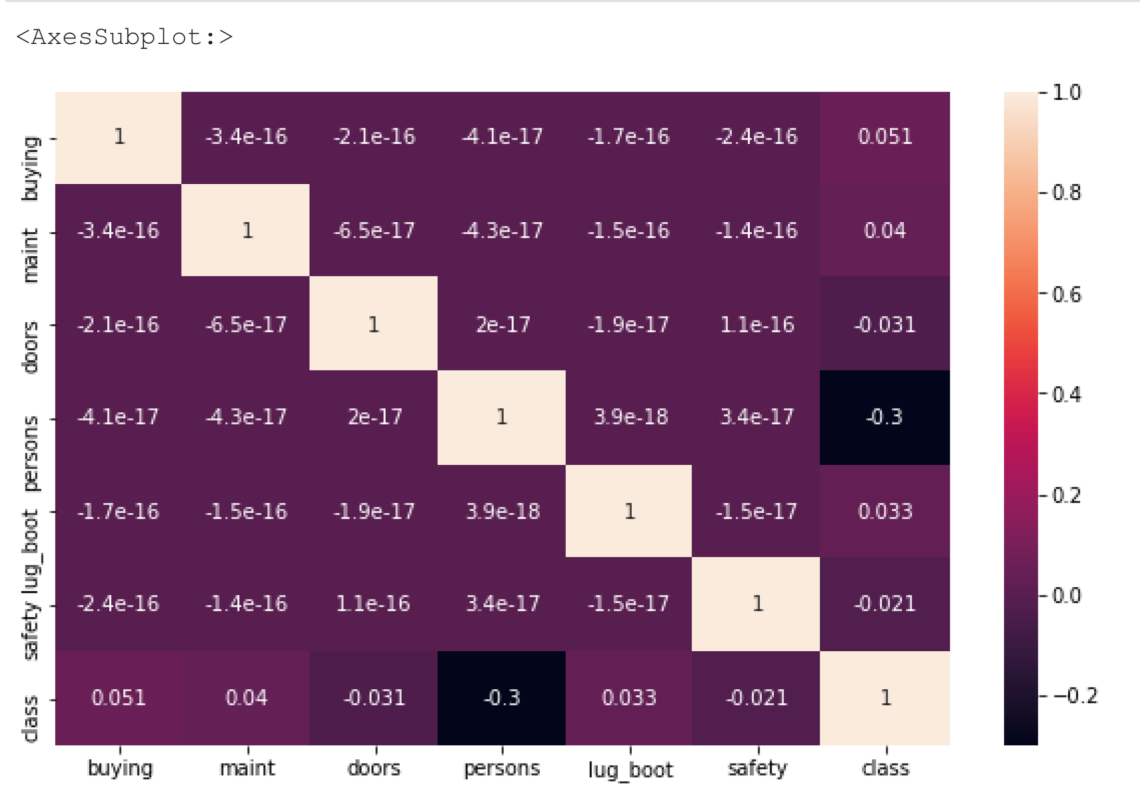
	buying	maint	doors	persons	lug_boot	safety	class
0	3	3	0	0	2	1	2
1	3	3	0	0	2	2	2
2	3	3	0	0	2	0	2
3	3	3	0	0	1	1	2
4	3	3	0	0	1	2	2

Correlation matrix

In [12]:

```
fig=plt.figure(figsize=(10,6))
sns.heatmap(data.corr(),annot=True)
```

Out[12]: <AxesSubplot:>



Most of the columns show very weak correlation with 'class' So, doing any analysis on them may not give any productive output.

Train multiple classifiers and compare the accuracy

In [13]:

```
from sklearn.model_selection import learning_curve
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_val_score
from sklearn.metrics import classification_report, confusion_matrix
```

In [14]:

```
X=data[data.columns[:-1]]
y=data['class']
```

In [15]:

```
X.head(2)
```

Out[15]:

	buying	maint	doors	persons	lug_boot	safety
0	3	3	0	0	2	1
1	3	3	0	0	2	2

In [16]:

```
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=
```

Logistic Regression

In [17]:

```
logreg=LogisticRegression(solver='newton-cg',multi_class='multinomial')
```

In [18]:

```
logreg.fit(X_train,y_train)
```

Out[18]: LogisticRegression(multi_class='multinomial', solver='newton-cg')

In [19]:

```
pred=logreg.predict(X_test)
```

In [20]:

```
logreg.score(X_test,y_test)
```

Out[20]: 0.6647398843930635

Logistic regression model is giving very less accuracy. Let's check with other algorithms.

KNN Classifier

In [21]:

```
knn=KNeighborsClassifier(n_jobs=-1)
```

In [22]:

```
knn.fit(X_train,y_train)
pred=knn.predict(X_test)
knn.score(X_test,y_test)
```

Out[22]: 0.9017341040462428

In [23]:

```
print(classification_report(y_test,pred))
```

```
              precision    recall  f1-score   support

           0       0.82      0.79      0.80       118
           1       0.77      0.53      0.62        19
           2       0.93      0.99      0.96       358
           3       1.00      0.50      0.67        24

    accuracy                           0.90       519
   macro avg       0.88      0.70      0.76       519
weighted avg       0.90      0.90      0.90       519
```

Accuracy can't be a fair criterion to evaluate unbalanced classification, so check 'f1-score' f1-score is 0.9 which is better than previous model

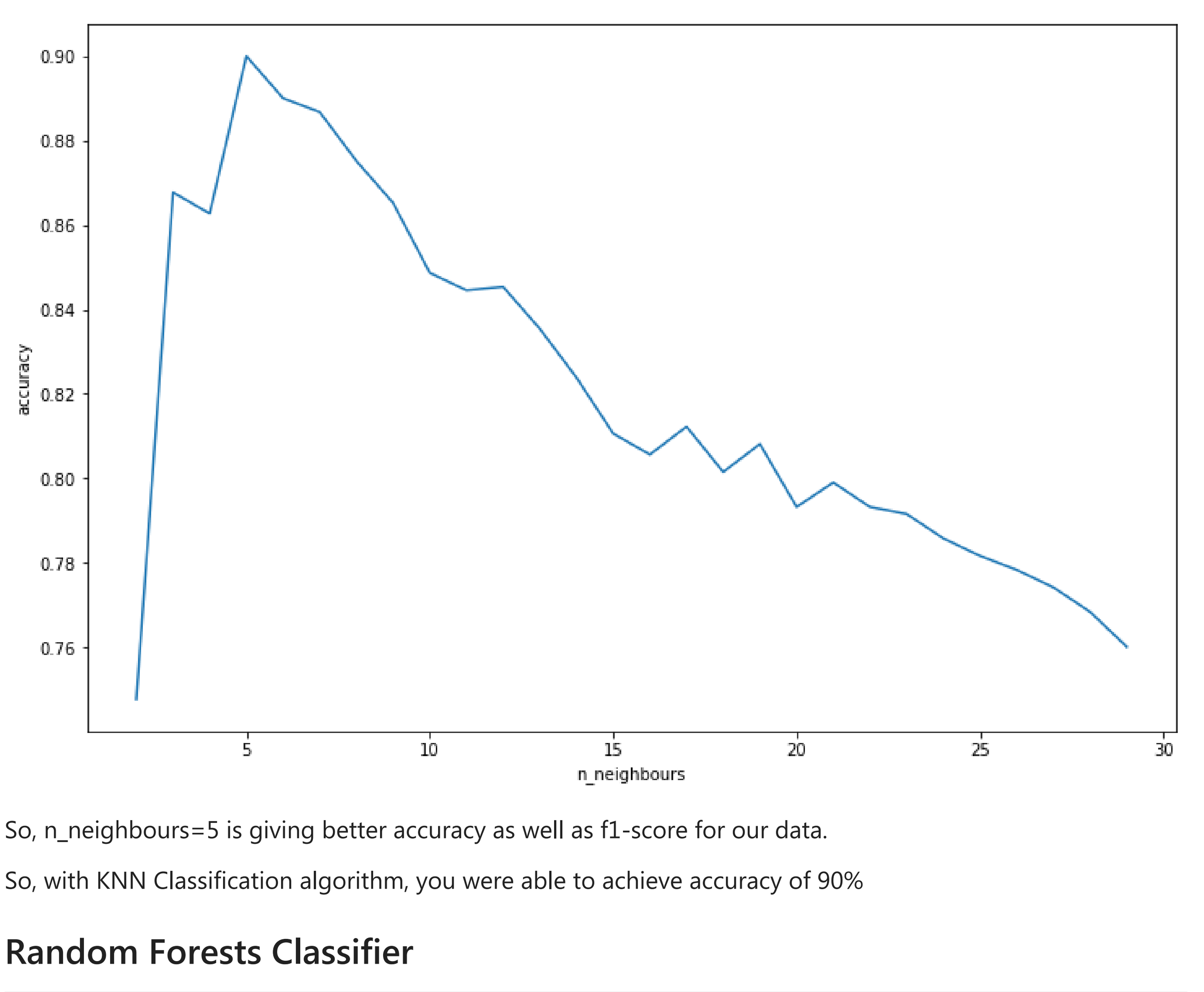
In [24]:

```
avg_score=[]
for k in range(2,30):
    knn=KNeighborsClassifier(n_jobs=-1,n_neighbors=k)
    score=cross_val_score(knn,X_train,y_train,cv=5,n_jobs=-1,scoring='accuracy')
    avg_score.append(score.mean())
```

In [25]:

```
plt.figure(figsize=(12,8))
plt.plot(range(2,30),avg_score)
plt.xlabel("n_neighbours")
plt.ylabel("accuracy")
#plt.xticks(range(2,30,2))
```

Out[25]: Text(0, 0.5, 'accuracy')



So, n_neighbours=5 is giving better accuracy as well as f1-score for our data.

So, with KNN Classification algorithm, you were able to achieve accuracy of 90%

Random Forests Classifier

In [26]:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import f1_score
```

In [27]:

```
rfc=RandomForestClassifier(n_jobs=-1,random_state=51)
```

In [28]:

```
rfc.fit(X_train,y_train)
print(rfc.score(X_test,y_test))
print(f1_score(y_test,rfc.predict(X_test),average='macro'))
```

```
0.9730250481695568
0.9245337130459484
```

RFC is providing 97% accuracy

Tuning Classifier Model with XGBoost

In [29]:

```
#Import the respective modules
from sklearn import model_selection
from xgboost import XGBClassifier

seed = 7
num_trees = 30

#Construct XGB Classifier model using kfolds technique
kfold = model_selection.KFold(n_splits=10)
model = XGBClassifier(n_estimators=num_trees, random_state=seed, eval_metric='rmse', u
#pass the model within the cross validation score function to evaluate the results
results = model_selection.cross_val_score(model,X_train,y_train,cv=kfold)
print(results.mean())
```

```
0.9809573002754822
```

XGBoost is providing 98% accuracy.

Among all the classifiers, XGBoost gives better accuracy.