mwptirpd9

March 26, 2023

0.1 Problem Statement

This dataset is created for the prediction of Graduate admissions from an Indian perspective. The dataset contains various features that are important during the application for the Master's Program. The predicted output obtained from the classification algorithm gives a fair idea about the chances of a student for admission.

0.2 About the dataset (Graduate admissions prediction data)

Serial No.: Serial number of student

GRE Scores: GRE score (out of 340)

TOEFL Scores: TOEFL score (out of 120)

University Rating: University rating (out of 5)

SOP: Strength of Statement of Purpose (out of 5)

LOR: Strength of Letter of Recommendation (out of 5)

CGPA: Undergraduate CGPA (out of 10)

Chance of Admit: Chance of admission (target/dependent variable)

0.3 Table of Content

1. **Import Libraries**
2. **Data Preparation**
    - 2.1 - **Read the Data**
    - 2.2 - **Check the Data Type**
    - 2.3 - **Remove Insignificant Variables**
    - 2.4 - **Missing Value Treatment**
    - 2.5 - **Train-Test Split**
3. **Decision Tree for Classification**

1. Import Libraries

Let us import the required libraries.

[1]:
```
!pip install pydotplus
```

```
Requirement already satisfied: pydotplus in d:\anaconda\lib\site-packages
(2.0.2)
```

```
Requirement already satisfied: pyparsing>=2.0.1 in d:\anaconda\lib\site-packages
(from pydotplus) (3.0.9)
```

[2]:

```python
# import 'Pandas'
import pandas as pd

# import 'Numpy'
import numpy as np

import urllib.request as urllib

# import subpackage of Matplotlib
import matplotlib.pyplot as plt

# import 'Seaborn'
import seaborn as sns

# to suppress warnings
from warnings import filterwarnings
filterwarnings('ignore')

# display all columns of the dataframe
pd.options.display.max_columns = None

# display all rows of the dataframe
pd.options.display.max_rows = None

# to display the float values upto 6 decimal places
pd.options.display.float_format = '{:.6f}'.format

# import train-test split
from sklearn.model_selection import train_test_split

# import various functions from sklearn
from sklearn.metrics import classification_report
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree

# import the functions for visualizing the decision tree
import pydotplus
from IPython.display import Image

from sklearn import metrics

from sklearn.metrics import␣
↪roc_auc_score,roc_curve,classification_report,confusion_matrix
```

[3]:
```
# set the plot size using 'rcParams'
# once the plot size is set using 'rcParams', it sets the size of all the␣
↪forthcoming plots in the file
# pass width and height in inches to 'figure.figsize'
plt.rcParams['figure.figsize'] = [15,8]
```

2. Data Preparation

2.1 Read the Data

Read the dataset and print the first five observations.

[4]:
```
# load the csv file
# store the data in 'df_bupa'
# df_bupa = pd.read_table('https://archive.ics.uci.edu/ml/
↪machine-learning-databases/liver-disorders/bupa.data', sep = ',',header =␣
↪None,encoding = 'utf-8')
df_bupa = pd.read_table('bupa.data', sep = ',',header = None,encoding = 'utf-8')

# display first five observations using head()
df_bupa.head()
```

[4]:

	0	1	2	3	4	5	6
0	85	92	45	27	31	0.000000	1
1	85	64	59	32	23	0.000000	2
2	86	54	33	16	54	0.000000	2
3	91	78	34	24	36	0.000000	2
4	87	70	12	28	10	0.000000	2

Let us now see the number of variables and observations in the data.

[5]:
```
df_bupa = df_bupa.rename({0: 'mcv', 1: 'alkphos', 2: 'sgpt', 3: 'sgot', 4:␣
↪'gammagt', 5: 'drinks', 6 : 'selector'}, axis=1)
```

[6]:
```
# use 'shape' to check the dimension of data
df_bupa.shape
```

[6]: (345, 7)

[7]:
```
df_bupa.head()
```

[7]:

	mcv	alkphos	sgpt	sgot	gammagt	drinks	selector
0	85	92	45	27	31	0.000000	1
1	85	64	59	32	23	0.000000	2
2	86	54	33	16	54	0.000000	2
3	91	78	34	24	36	0.000000	2
4	87	70	12	28	10	0.000000	2

Interpretation: The data has 345 observations and 7 variables.

2.2 Check the Data Type

Check the data type of each variable. If the data type is not as per the data definition, change the data type.

[8]:
```
# use 'dtypes' to check the data type of a variable
df_bupa.dtypes
```

[8]:
```
mcv           int64
alkphos       int64
sgpt          int64
sgot          int64
gammagt       int64
drinks      float64
selector      int64
dtype: object
```

Interpretation: The variables `GRE Score`, `TOEFL Score`, `University Rating`, `SOP`, `LOR`, and `CGPA` are numerical.

2.3 Remove Insignificant Variables

The column `Serial No.` contains the serial number of the student, which is redundant for further analysis. Thus, we drop the column.

[9]:
```
df_bupa.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 345 entries, 0 to 344
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   mcv       345 non-null    int64
 1   alkphos   345 non-null    int64
 2   sgpt      345 non-null    int64
 3   sgot      345 non-null    int64
 4   gammagt   345 non-null    int64
 5   drinks    345 non-null    float64
 6   selector  345 non-null    int64
dtypes: float64(1), int64(6)
memory usage: 19.0 KB
```

[10]:
```
df_bupa.nunique()
```

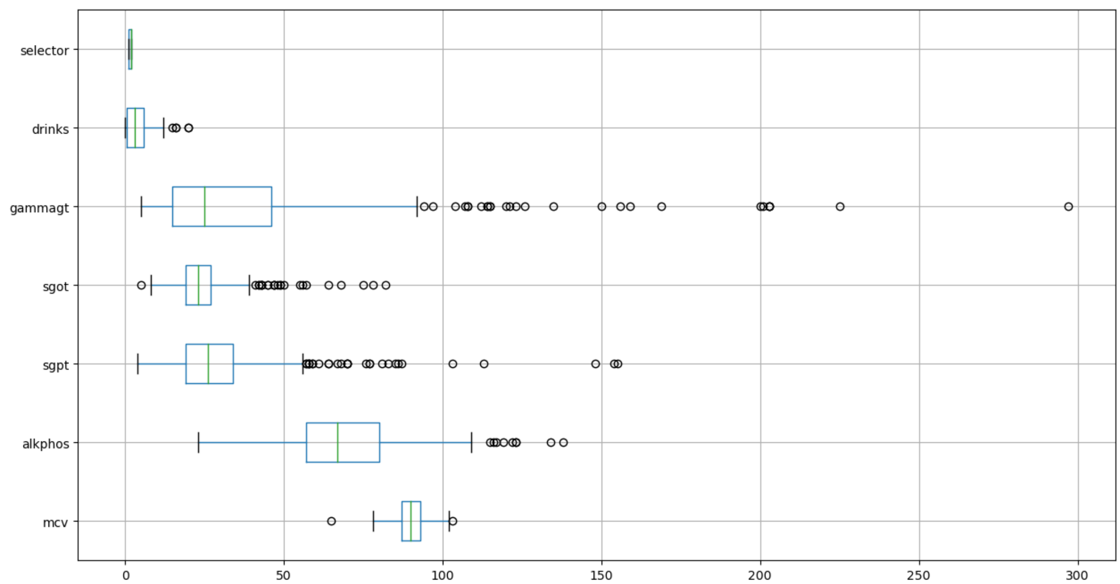
[10]:
```
mcv         26
alkphos     78
sgpt        67
sgot        47
gammagt     94
```

```
drinks      16
selector     2
dtype: int64
```

[11]:
```
df_bupa['selector'].value_counts()
```

[11]:
```
2    200
1    145
Name: selector, dtype: int64
```

[12]:
```
df_bupa.boxplot(vert=0)
plt.show()
```



2.4 Missing Value Treatment

First run a check for the presence of missing values and their percentage for each column. Then choose the right approach to treat them.

[13]:
```
Total = df_bupa.isnull().sum().sort_values(ascending=False)
Percent = (df_bupa.isnull().sum()*100/df_bupa.isnull().count()).
↪sort_values(ascending=False)
missing_data = pd.concat([Total, Percent], axis = 1, keys = ['Total',␣
↪'Percentage of Missing Values'])
missing_data
```

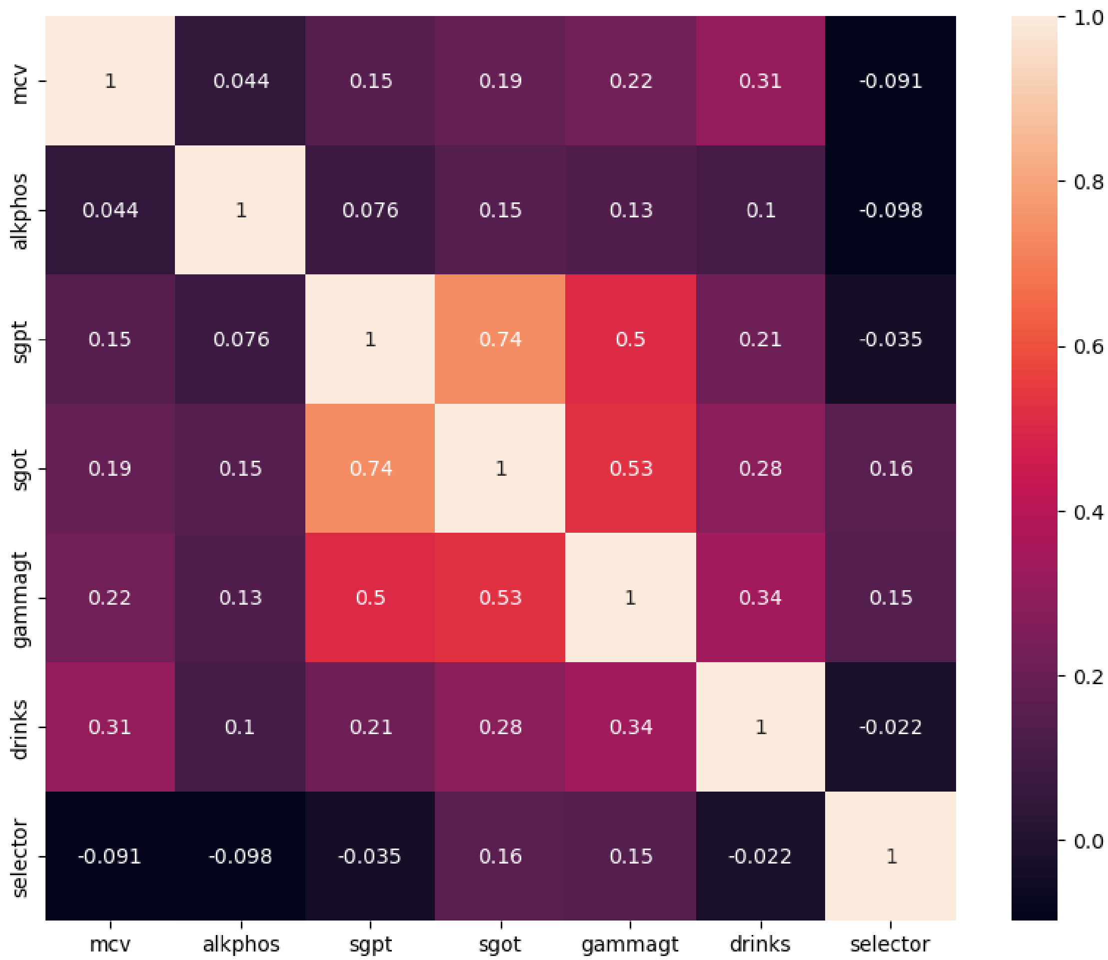
[13]:
```
         Total  Percentage of Missing Values
mcv          0                      0.000000
alkphos      0                      0.000000
```

```
sgpt         0                      0.000000
sgot         0                      0.000000
gammagt      0                      0.000000
drinks       0                      0.000000
selector     0                      0.000000
```

Interpretation: The above output shows that there are no missing values in the data.

[14]:
```
impute=df_bupa.columns
for i in impute:
    df_bupa[i].fillna(df_bupa[i].median(),inplace=True)
```

[15]:
```
plt.figure(figsize=(10,8))
sns.heatmap(df_bupa.corr(),annot=True)
plt.show()
```



2.5 Train-Test Split

Before applying various classification techniques to predict the category of selector,let us split the dataset in train and test set.

[16]:
```
# Segregate the dataset into predictor (X) and target (y) variables.
X = df_bupa.iloc[:, :-1]
df_target = df_bupa.iloc[:, -1]
```

[17]:
```
# split data into train subset and test subset
# set 'random_state' to generate the same dataset each time you run the code
# 'test_size' returns the proportion of data to be included in the test set
X_train, X_test, y_train, y_test = train_test_split(X, df_target, random_state␣
↪= 10,  test_size = 0.2)

# check the dimensions of the train & test subset using 'shape'
# print dimension of train set
print('X_train', X_train.shape)
print('y_train', y_train.shape)

# print dimension of test set
print('X_test', X_test.shape)
print('y_test', y_test.shape)
```

```
X_train (276, 6)
y_train (276,)
X_test (69, 6)
y_test (69,)
```

Create a generalized function to calculate the metrics for the train and the test set.

[18]:
```
print(X_train.head())
```

```
     mcv  alkphos  sgpt  sgot  gammagt   drinks
121   79      101    17    27       23 4.000000
211   78       69    24    18       31 0.500000
325   91       52    76    32       24 8.000000
78    91       55     9    25       16 2.000000
173   92       94    18    17        6 8.000000
```

[19]:
```
print(y_train.head())
```

```
121    2
211    1
325    1
78     2
173    1
Name: selector, dtype: int64
```

[20]:
```python
# create a generalized function to calculate the metrics values for train set
def get_train_report(model):

    # for training set:
    # train_pred: prediction made by the model on the train dataset 'X_train'
    # y_train: actual values of the target variable for the train dataset

    # predict the output of the target variable from the train data
    train_pred = model.predict(X_train)

    # return the performace measures on train set
    return(classification_report(y_train, train_pred))
```

[21]:
```python
# create a generalized function to calculate the metrics values for test set
def get_test_report(model):

    # for test set:
    # test_pred: prediction made by the model on the test dataset 'X_test'
    # y_test: actual values of the target variable for the test dataset

    # predict the output of the target variable from the test data
    test_pred = model.predict(X_test)

    # return the performace measures on test set
    return(classification_report(y_test, test_pred))
```

3. Decision Tree for Classification

Decision Tree is a non-parametric supervised learning method. It builds a model in the form of a tree structure. It breaks down a dataset into smaller and smaller subsets, which is called splitting. A decision node is a node on which a decision of split is to be made. A node that can not be split further is known as the terminal/leaf node. A leaf node represents the decision. A decision tree can work with both numerical and categorical variables.

A decision tree for classification is built using criteria like the Gini index and entropy.

0.4 Gini Index

Gini index measures the probability of the sample being wrongly classified. The value of the Gini index varies between 0 and 1. We choose the variable with a low Gini index. The Gini index of the variable is calculated as:

$ Gini = 1 - _{i = 1}^{n}p_{i}^{2} $

Where, p_i: Probability of occurrence of the class 'i'

0.5 Entropy

Entropy is one of the criteria used to build the decision tree. It calculates the heterogeneity of the sample. The entropy is zero if the sample is completely homogeneous, and it is equal to 1 if the sample is equally divided. Entropy of the variable 'X' is calculated as:

\$ E(X) = - *{i = 1}^{c}p*{i}log_{2}p_{i} \$

Where, p_i: Probability of occurrence of the class 'i'

And the conditional entropy of the variable is given as:

\$ E(T, X) = _{c X}P(c)E(c) \$

Where, $P(c)$: Probability of occurrence of the class 'c' $E(c)$: Entropy of the class 'c'

The information gain is the difference between the entropy of the target variable and the entropy of the target variable given an independent variable. We split the on the variable that corresponds to the highest information gain.

Build a full decision tree model on a train dataset using 'entropy'.

[22]:
```
# instantiate the 'DecisionTreeClassifier' object using 'entropy' criterion
# pass the 'random_state' to obtain the same samples for each time you run the␣
↪code
decision_tree_classification = DecisionTreeClassifier(criterion = 'entropy',␣
↪random_state = 10)

# fit the model using fit() on train data
decision_tree = decision_tree_classification.fit(X_train, y_train)
```

Plot a decision tree. To visualize our decision tree we will use `Graphviz.` If you are an anaconda user then install it by using `conda install graphviz` otherwise write the command `pip install graphviz.`

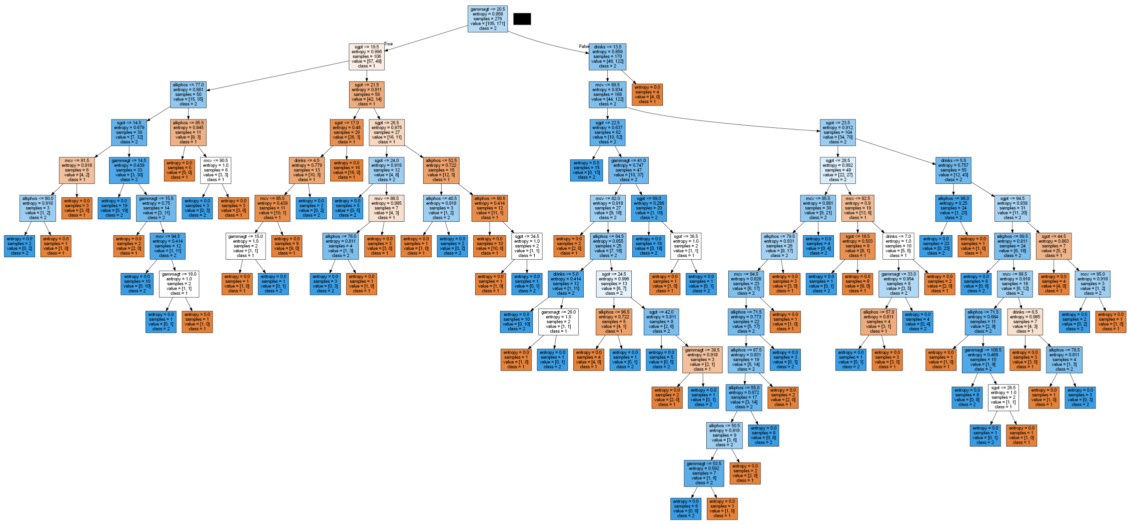
[23]:
```
pip install graphviz
```

```
Requirement already satisfied: graphviz in d:\anaconda\lib\site-packages
(0.20.1)
Note: you may need to restart the kernel to use updated packages.
```

[24]:
```
# save the column names in 'labels'
labels = X_train.columns

# export a decision tree in DOT format
# pass the 'decision_tree' to export it to Graphviz
# pass the column names to 'feature_names'
# pass the required class labels to 'class_names'
dot_data = tree.export_graphviz(decision_tree, feature_names = labels,␣
↪class_names = ["1","2"], filled = True)

# plot the decision tree using DOT format in 'dot_data'
graph = pydotplus.graph_from_dot_data(dot_data)

# display the decision tree
```

```
Image(graph.create_png())

# double-click on the image below to get an expanded view
```

[24]:



0.6 Over-fitting in Decision Tree

The decision tree is said to be over-fitted if it tries to perfectly fit all the observations in the training data. We can calculate the difference between the train and test accuracy to identify if there is over-fitting.

Calculate performance measures on the train set.

[25]:

```
# compute the performance measures on train data
# call the function 'get_train_report'
# pass the decision tree to the function
train_report = get_train_report(decision_tree)

# print the performance measures
print(train_report)
```

```
              precision    recall  f1-score   support

           1       1.00      1.00      1.00       105
           2       1.00      1.00      1.00       171

    accuracy                           1.00       276
   macro avg       1.00      1.00      1.00       276
weighted avg       1.00      1.00      1.00       276
```

Calculate performance measures on the test set.

[26]:
```
# compute the performance measures on test data
# call the function 'get_test_report'
# pass the decision tree to the function
test_report = get_test_report(decision_tree)

# print the performance measures
print(test_report)
```

```
              precision    recall  f1-score   support

           1       0.71      0.62      0.67        40
           2       0.56      0.66      0.60        29

    accuracy                           0.64        69
   macro avg       0.64      0.64      0.63        69
weighted avg       0.65      0.64      0.64        69
```

Interpretation: From the above output, we can see that there is a difference between the train and test accuracy; thus, we can conclude that the decision tree is over-fitted on the train data.

If we tune the hyperparameters in the decision tree, it helps to avoid the over-fitting of the tree.

Build a decision tree using 'criterion = gini', 'max_depth = 5', 'min_samples_split = 4', 'max_leaf_nodes = 6'.

[27]:
```
# pass the criteria 'gini' to the parameter, 'criterion'
# max_depth: that assigns maximum depth of the tree
# min_samples_split: assigns minimum number of samples to split an internal node
# max_leaf_nodes': assigns maximum number of leaf nodes in the tree
# pass the 'random_state' to obtain the same samples for each time you run the␣
 ↪code
dt_model = DecisionTreeClassifier(criterion = 'gini',
                                  max_depth = 5,
                                  min_samples_split = 4,
                                  max_leaf_nodes = 6,
                                  random_state = 10)

# fit the model using fit() on train data
decision_tree = dt_model.fit(X_train, y_train)

# compute the performance measures on train data
# call the function 'get_train_report'
# pass the decision tree to the function
train_report = get_train_report(decision_tree)

# print the performance measures
print('Train data:\n', train_report)
```

```
# compute the performance measures on test data
# call the function 'get_test_report'
# pass the decision tree to the function
test_report = get_test_report(decision_tree)

# print the performance measures
print('Test data:\n', test_report)
```

```
Train data:
               precision    recall  f1-score   support

           1       0.75      0.55      0.64       105
           2       0.76      0.89      0.82       171

    accuracy                           0.76       276
   macro avg       0.76      0.72      0.73       276
weighted avg       0.76      0.76      0.75       276

Test data:
               precision    recall  f1-score   support

           1       0.72      0.53      0.61        40
           2       0.53      0.72      0.61        29

    accuracy                           0.61        69
   macro avg       0.62      0.62      0.61        69
weighted avg       0.64      0.61      0.61        69
```

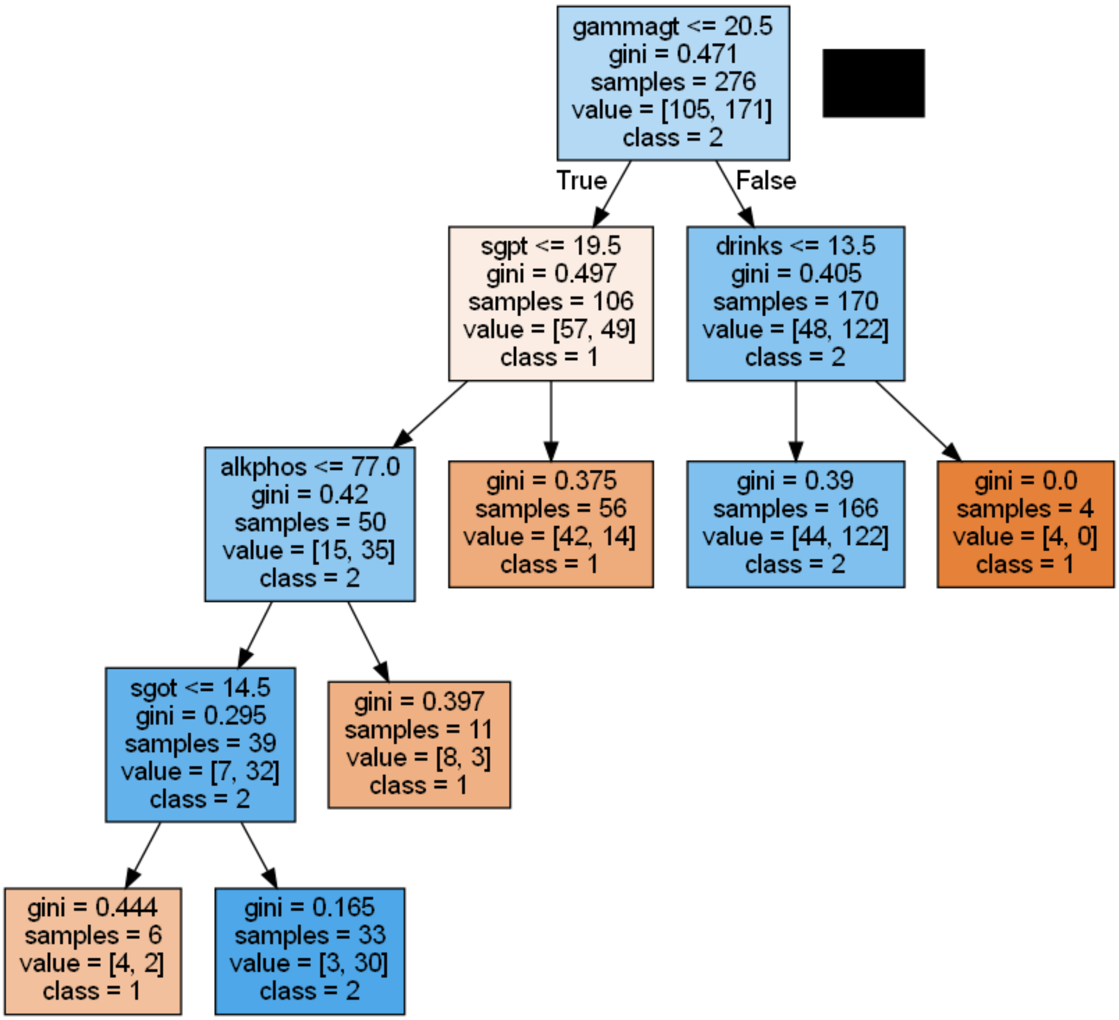
Interpretation: From the above output, we can see that there is slight significant difference between the train and test accuracy; thus, we can conclude that the decision tree is less over-fiited after specifying some of the hyperparameters.

Decision Tree Post-Pruning

[28]:
```
# save the column names in 'labels'
labels = X_train.columns

# export a decision tree in DOT format
# pass the 'decision_tree' to export it to Graphviz
# pass the column names to 'feature_names'
# pass the required class labels to 'class_names'
dot_data = tree.export_graphviz(decision_tree, feature_names = labels,␣
 ↪class_names = ["1","2"], filled = True)

# plot the decision tree using DOT format in 'dot_data'
graph = pydotplus.graph_from_dot_data(dot_data)
```

```
# display the decision tree
Image(graph.create_png())

# double-click on the image below to get an expanded view
```

[28]:



[29]:
```
print (pd.DataFrame(decision_tree.feature_importances_, columns = ["Imp"],
index = X_train.columns))
```

```
             Imp
mcv      0.000000
alkphos  0.161168
sgpt     0.334848
sgot     0.105354
gammagt  0.266563
drinks   0.132067
```

[30]:
```python
# implement GridsearchCV to find the best parameters for the␣
↪DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV, KFold
param_grid = {'max_depth': range(2,10),
              'min_samples_split': range(2,10),
              'max_leaf_nodes': range(2,10),
              'min_samples_leaf': range(3,10),
              'criterion':['gini','entropy']}
```

[31]:
```python
kf = KFold(n_splits=5, shuffle=True, random_state=10)
```

[32]:
```python
dt_grid = DecisionTreeClassifier()
grid_search = GridSearchCV(dt_grid,param_grid=param_grid,cv=kf)
grid_search.fit(X_train, y_train)
```

[32]:
```
GridSearchCV(cv=KFold(n_splits=5, random_state=10, shuffle=True),
             estimator=DecisionTreeClassifier(),
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': range(2, 10),
                         'max_leaf_nodes': range(2, 10),
                         'min_samples_leaf': range(3, 10),
                         'min_samples_split': range(2, 10)})
```

[33]:
```python
grid_search.best_params_
```

[33]:
```
{'criterion': 'gini',
 'max_depth': 3,
 'max_leaf_nodes': 4,
 'min_samples_leaf': 6,
 'min_samples_split': 2}
```

[34]:
```python
best_model = grid_search.best_estimator_
```

[35]:
```python
train_report = get_train_report(best_model)
print(train_report)
```

```
              precision    recall  f1-score   support

           1       0.75      0.48      0.58       105
           2       0.74      0.90      0.81       171

    accuracy                           0.74       276
   macro avg       0.74      0.69      0.70       276
weighted avg       0.74      0.74      0.72       276
```

[36]:

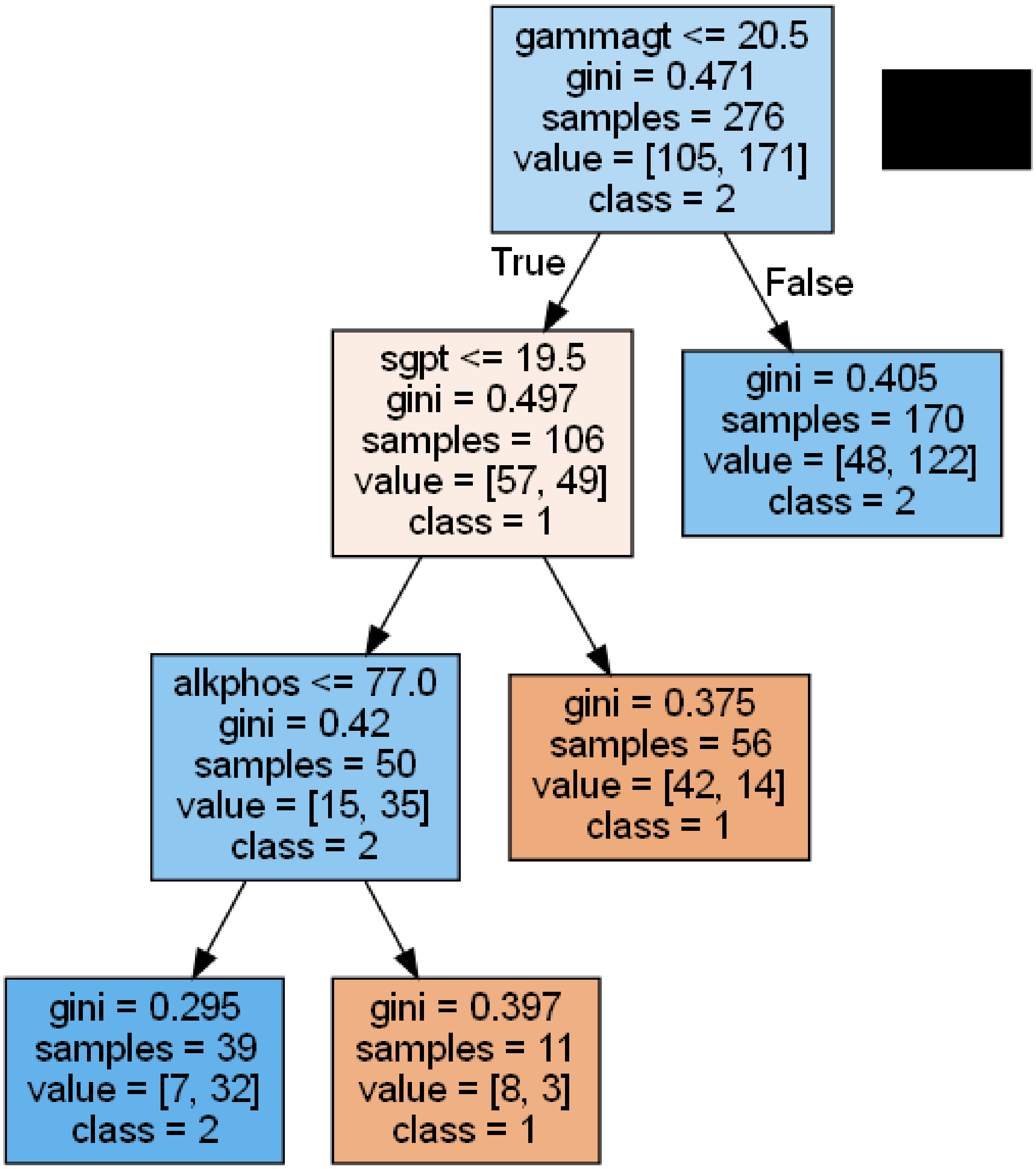
```
test_report = get_test_report(best_model)
print(test_report)
```

```
              precision    recall  f1-score   support

           1       0.72      0.53      0.61        40
           2       0.53      0.72      0.61        29

    accuracy                           0.61        69
   macro avg       0.62      0.62      0.61        69
weighted avg       0.64      0.61      0.61        69
```

[37]:

```
labels = X_train.columns

dot_data = tree.export_graphviz(best_model, feature_names = labels, class_names␣
 ↪= ["1","2"], filled = True)

graph = pydotplus.graph_from_dot_data(dot_data)

Image(graph.create_png())
```

[37]:



[38]:
```
print (pd.DataFrame(best_model.feature_importances_, columns = ["Imp"], index =␣
↪X_train.columns))
```

```
              Imp
mcv      0.000000
alkphos  0.211346
sgpt     0.439099
sgot     0.000000
gammagt  0.349555
```

```
drinks   0.000000
```

0.7 Random Forest

[39]:
```
from sklearn.ensemble import RandomForestClassifier

RF_model=RandomForestClassifier(n_estimators=100,random_state=1)
RF_model.fit(X_train, y_train)
```

[39]:
```
RandomForestClassifier(random_state=1)
```

[40]:
```
## Performance Matrix on train data set
y_train_predict = RF_model.predict(X_train)
model_score =RF_model.score(X_train, y_train)
print(model_score)
print(metrics.confusion_matrix(y_train, y_train_predict))
print(metrics.classification_report(y_train, y_train_predict))
```

```
1.0
[[105   0]
 [  0 171]]
              precision    recall  f1-score   support

           1       1.00      1.00      1.00       105
           2       1.00      1.00      1.00       171

    accuracy                           1.00       276
   macro avg       1.00      1.00      1.00       276
weighted avg       1.00      1.00      1.00       276
```

[59]:
```
rf_train_auc=roc_auc_score(y_train,RF_model.predict_proba(X_train)[:,1])
print('AUC TRAIN DATA', rf_train_auc)
```

```
AUC TRAIN DATA 1.0
```

[41]:
```
## Performance Matrix on test data set
y_test_predict = RF_model.predict(X_test)
model_score = RF_model.score(X_test, y_test)
print(model_score)
print(metrics.confusion_matrix(y_test, y_test_predict))
print(metrics.classification_report(y_test, y_test_predict))
```

```
0.7101449275362319
[[26 14]
 [ 6 23]]
              precision    recall  f1-score   support
```

```
           1       0.81      0.65      0.72        40
           2       0.62      0.79      0.70        29

    accuracy                           0.71        69
   macro avg       0.72      0.72      0.71        69
weighted avg       0.73      0.71      0.71        69
```

[60]:
```
rf_test_auc=roc_auc_score(y_test,RF_model.predict_proba(X_test)[:,1])
print('AUC TEST DATA', rf_test_auc)
```

```
AUC TEST DATA 0.794396551724138
```

[42]:
```
# Variable Importance
print (pd.DataFrame(RF_model.feature_importances_, columns = ["Imp"], index =␣
 ↪X_train.columns).sort_values('Imp',ascending=False))
```

```
             Imp
sgpt    0.203842
gammagt 0.202866
alkphos 0.185564
sgot    0.164354
mcv     0.137761
drinks  0.105612
```

[43]:
```
param_grid = {
'max_depth': [10],
'max_features': [5,6,7],
'min_samples_leaf': [5,10,15],
'min_samples_split': [30,40,50],
'n_estimators': [300]
}
rfcl = RandomForestClassifier(random_state=2)
grid_search = GridSearchCV(estimator = rfcl, param_grid = param_grid, cv = 5)
```

[44]:
```
grid_search.fit(X_train, y_train)
```

[44]:
```
GridSearchCV(cv=5, estimator=RandomForestClassifier(random_state=2),
             param_grid={'max_depth': [10], 'max_features': [5, 6, 7],
                         'min_samples_leaf': [5, 10, 15],
                         'min_samples_split': [30, 40, 50],
                         'n_estimators': [300]})
```

[45]:
```
grid_search.best_params_
```

[45]:
```
{'max_depth': 10,
 'max_features': 6,
 'min_samples_leaf': 5,
```

```
 'min_samples_split': 30,
 'n_estimators': 300}
```

[46]:
```
best_grid = grid_search.best_estimator_
```

[47]:
```
best_grid
```

[47]:
```
RandomForestClassifier(max_depth=10, max_features=6, min_samples_leaf=5,
                       min_samples_split=30, n_estimators=300, random_state=2)
```

[48]:
```
ytrain_predict = best_grid.predict(X_train)
ytest_predict = best_grid.predict(X_test)
```

[49]:
```
confusion_matrix(y_train,ytrain_predict)
```

[49]:
```
array([[ 60,  45],
       [ 10, 161]], dtype=int64)
```

[50]:
```
rf_train_acc=best_grid.score(X_train,y_train)
rf_train_acc
print(classification_report(y_train,ytrain_predict))
```

```
              precision    recall  f1-score   support

           1       0.86      0.57      0.69       105
           2       0.78      0.94      0.85       171

    accuracy                           0.80       276
   macro avg       0.82      0.76      0.77       276
weighted avg       0.81      0.80      0.79       276
```

[51]:
```
rf_metrics=classification_report(y_train, ytrain_predict,output_dict=True)
df=pd.DataFrame(rf_metrics).transpose()
rf_train_precision=round(df.loc["1"][0],2)
rf_train_recall=round(df.loc["1"][1],2)
rf_train_f1=round(df.loc["1"][2],2)
print ('rf_train_precision ',rf_train_precision)
print ('rf_train_recall ',rf_train_recall)
print ('rf_train_f1 ',rf_train_f1)
```

```
rf_train_precision  0.86
rf_train_recall  0.57
rf_train_f1  0.69
```

[52]:
```
rf_train_auc=roc_auc_score(y_train,best_grid.predict_proba(X_train)[:,1])
print('AUC TRAIN DATA', rf_train_auc)
```

```
AUC TRAIN DATA 0.9154553049289891
```

0.7.1 RF Model Performance Evaluation on Test data

[53]:
```python
confusion_matrix(y_test,ytest_predict)
```

[53]:
```
array([[16, 24],
       [ 6, 23]], dtype=int64)
```

[54]:
```python
rf_test_acc=best_grid.score(X_test,y_test)
rf_test_acc
```

[54]:
```
0.5652173913043478
```

[55]:
```python
print(classification_report(y_test,ytest_predict))
```

```
              precision    recall  f1-score   support

           1       0.73      0.40      0.52        40
           2       0.49      0.79      0.61        29

    accuracy                           0.57        69
   macro avg       0.61      0.60      0.56        69
weighted avg       0.63      0.57      0.55        69
```

[56]:
```python
rf_metrics=classification_report(y_test, ytest_predict,output_dict=True)
df=pd.DataFrame(rf_metrics).transpose()
rf_test_precision=round(df.loc["1"][0],2)
rf_test_recall=round(df.loc["1"][1],2)
rf_test_f1=round(df.loc["1"][2],2)
print ('rf_test_precision ',rf_test_precision)
print ('rf_test_recall ',rf_test_recall)
print ('rf_test_f1 ',rf_test_f1)
```

```
rf_test_precision  0.73
rf_test_recall  0.4
rf_test_f1  0.52
```

[57]:
```python
rf_test_auc=roc_auc_score(y_test,best_grid.predict_proba(X_test)[:,1])
print('AUC TEST DATA', rf_test_auc)
```

```
AUC TEST DATA 0.7663793103448275
```

[58]:
```python
# Variable Importance
print (pd.DataFrame(best_grid.feature_importances_, columns = ["Imp"], index =␣
 ↪X_train.columns).sort_values('Imp',ascending=False))
```

```
            Imp
gammagt 0.274891
sgpt    0.268564
sgot    0.158683
alkphos 0.132764
mcv     0.090086
drinks  0.075011
```