pr3-admission-gs

November 7, 2024

[ ]:
```
pip install pandas
```

```
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages
(2.2.2)
Requirement already satisfied: numpy>=1.22.4 in /usr/local/lib/python3.10/dist-
packages (from pandas) (1.26.4)
Requirement already satisfied: python-dateutil>=2.8.2 in
/usr/local/lib/python3.10/dist-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-
packages (from pandas) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/dist-
packages (from pandas) (2024.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-
packages (from python-dateutil>=2.8.2->pandas) (1.16.0)
```

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

[ ]:
```
df = pd.read_csv('Admission_Predict.csv')
```

[ ]:
```
df.head(10)
```

[ ]:

	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	\
0	1	337	118	4	4.5	4.5	9.65	
1	2	324	107	4	4.0	4.5	8.87	
2	3	316	104	3	3.0	3.5	8.00	
3	4	322	110	3	3.5	2.5	8.67	
4	5	314	103	2	2.0	3.0	8.21	
5	6	330	115	5	4.5	3.0	9.34	
6	7	321	109	3	3.0	4.0	8.20	
7	8	308	101	2	3.0	4.0	7.90	
8	9	302	102	1	2.0	1.5	8.00	
9	10	323	108	3	3.5	3.0	8.60	

	Research	Chance of Admit
0	1	0.92

```
1          1               0.76
2          1               0.72
3          1               0.80
4          0               0.65
5          1               0.90
6          1               0.75
7          0               0.68
8          0               0.50
9          0               0.45
```

[ ]:
```
df.shape
```

[ ]:
```
(400, 9)
```

[ ]:
```
df['Chance of Admit '] = [1 if each > 0.75 else 0 for each in df['Chance of␣
↪Admit ']]
df.head()
```

[ ]:
```
   GRE Score  TOEFL Score  University Rating  SOP  LOR   CGPA  Research  \
0        337          118                  4  4.5   4.5  9.65         1
1        324          107                  4  4.0   4.5  8.87         1
2        316          104                  3  3.0   3.5  8.00         1
3        322          110                  3  3.5   2.5  8.67         1
4        314          103                  2  2.0   3.0  8.21         0

   Chance of Admit
0                1
1                1
2                0
3                1
4                0
```

==========

Alternate Method

from sklearn.preprocessing import Binarizer

bi = Binarizer(threshold=0.75) # here we are changing values less than 0.75 to 0 and above 0.75 to 1

df[‘Chance of Admit’] = bi.fit_transform(df[[‘Chance of Admit’]])

df.head()

Essentially, the threshold determines the point at which the data should switch from one binary value (0) to another (1). This is commonly used for feature transformation in machine learning tasks, where you want to classify data points into two categories based on a numerical threshold. **********

transform the Chance of Admit column into binary values (0 or 1) based on a threshold of 0.75.

Here's how it works:

Binarizer(threshold=0.75): The Binarizer class is initialized with a threshold of 0.75. It will convert any value in the Chance of Admit column:

To 1 if the value is greater than or equal to 0.75. To 0 if the value is less than 0.75.

bi.fit_transform(df[['Chance of Admit']]): This method applies the transformation to the Chance of Admit column, and the result is stored back in the same column.

So, after running this code, the df['Chance of Admit'] column will be updated with binary values based on the 0.75 threshold.

[ ]:
```
#df = df.drop('Serial No.',axis=1)
```

axis=1: This specifies that you're dropping a column (as opposed to a row). In pandas, axis=0 refers to rows, while axis=1 refers to columns.

[ ]:
```
df.shape
```

[ ]:
```
(400, 8)
```

[ ]:
```
x = df.drop('Chance of Admit ',axis=1) # dropping the admitted column
y = df['Chance of Admit ']
```

x = df[['GRE Score', 'TOEFL Score', 'University Rating', 'SOP', 'LOR', 'CGPA', 'Research']]

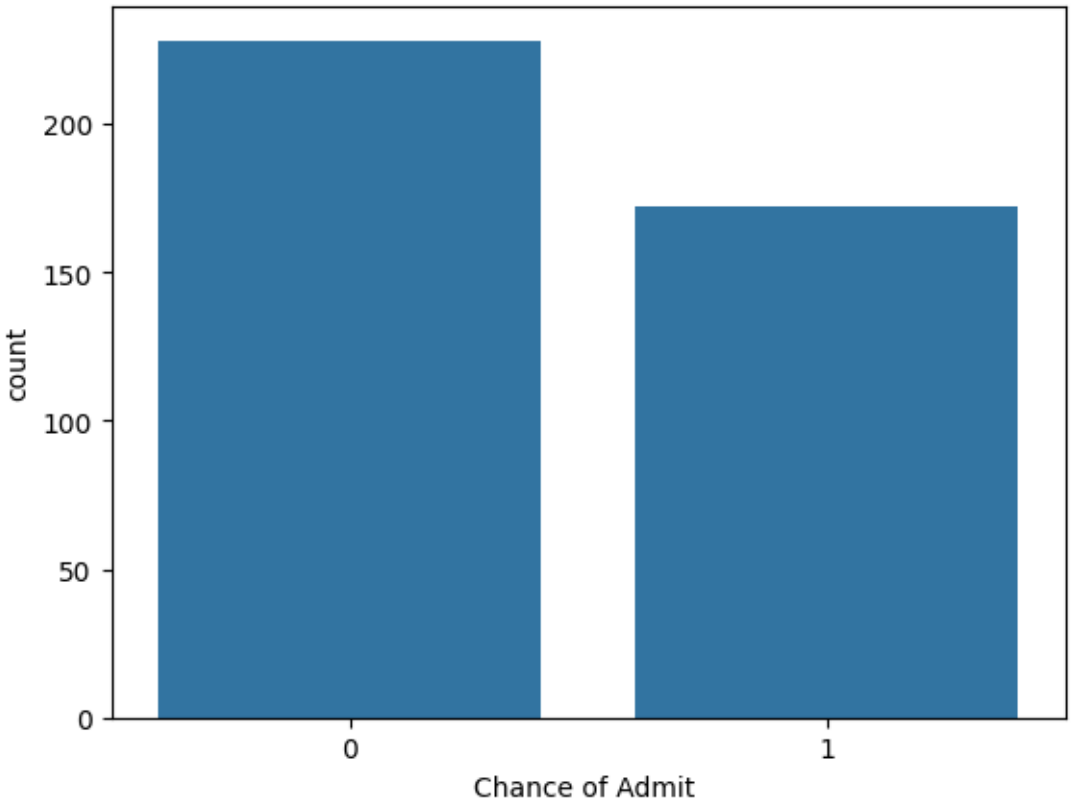
y = df['Chance of Admit']

[ ]:
```
#y

#if not int then
#y = y.astype('int')
```

[ ]:
```
sns.countplot(x=y)
```

[ ]:
```
<Axes: xlabel='Chance of Admit ', ylabel='count'>
```



sns.countplot(): This function is used to create a bar plot that shows the frequency of unique values in a categorical variable.

x=y: This means you are passing y as the data for the x-axis. y should be a categorical variable (like a column from a DataFrame) that you want to visualize the count of its unique values.

The result will be a bar plot where the x-axis shows the unique values in y, and the y-axis shows the frequency (or count) of those values.

[ ]:
```
y.value_counts()
```

[ ]:
```
Chance of Admit
0    228
1    172
Name: count, dtype: int64
```

[ ]:
```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.
  ↪25,random_state=0)
```

[ ]:
```
x_train.shape
```

[ ]: (300, 7)

[ ]:
```
x_test.shape
```

[ ]: (100, 7)

[ ]:
```
x_test.head()
```

[ ]:

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research
132	309	105	5	3.5	3.5	8.56	0
309	308	110	4	3.5	3.0	8.60	0
341	326	110	3	3.5	3.5	8.76	1
196	306	105	2	3.0	2.5	8.26	0
246	316	105	3	3.0	3.5	8.73	0

[ ]:
```
from sklearn.tree import DecisionTreeClassifier
```

[ ]:
```
classifier = DecisionTreeClassifier(random_state=0)
```

[ ]:
```
classifier.fit(x_train,y_train)
```

[ ]:
```
DecisionTreeClassifier(random_state=0)
```

[ ]:
```
y_pred = classifier.predict(x_test)
```

[ ]:
```
result = pd.DataFrame(
{
    'actual':y_test,
    'predicted':y_pred
})
```

creates a DataFrame that compares the actual target values (y_test) with the predicted values (y_pred) from your model. This is useful for evaluating the performance of your model and understanding how well it predicts on the test data.

[ ]:
```
result
```

[ ]:

	actual	predicted
132	0	0
309	0	0
341	1	1
196	0	0
246	0	1
..	…	…
146	0	0
135	1	1
390	0	0

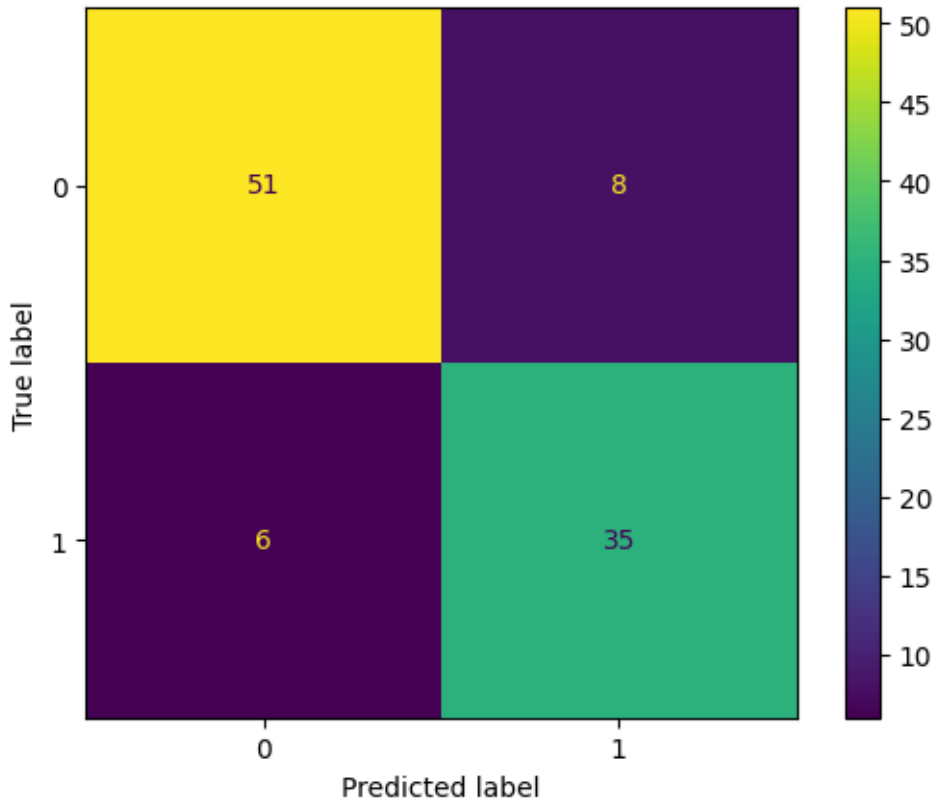
```
264       0          0
364       1          0

[100 rows x 2 columns]
```

[ ]:
```
from sklearn.metrics import␣
  ↪confusion_matrix,accuracy_score,ConfusionMatrixDisplay
from sklearn.metrics import classification_report
```

[ ]:
```
ConfusionMatrixDisplay.from_predictions(y_test,y_pred)
```

[ ]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x7c15862bee60>
```



Predicted 0 Predicted 1

actual 0 TN .|. FP

……

actual 1 FN .|. TP

[ ]:
```
accuracy_score(y_test,y_pred)
```

[ ]: 0.86

[ ]:
```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.89      0.86      0.88        59
           1       0.81      0.85      0.83        41

    accuracy                           0.86       100
   macro avg       0.85      0.86      0.86       100
weighted avg       0.86      0.86      0.86       100
```

[ ]:
```
new=[[322,110,3,3.5,2.5,8.67,1]]
classifier.predict(new)[0]
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:493: UserWarning: X does
not have valid feature names, but DecisionTreeClassifier was fitted with feature
names
  warnings.warn(
```

[ ]: 1

new:

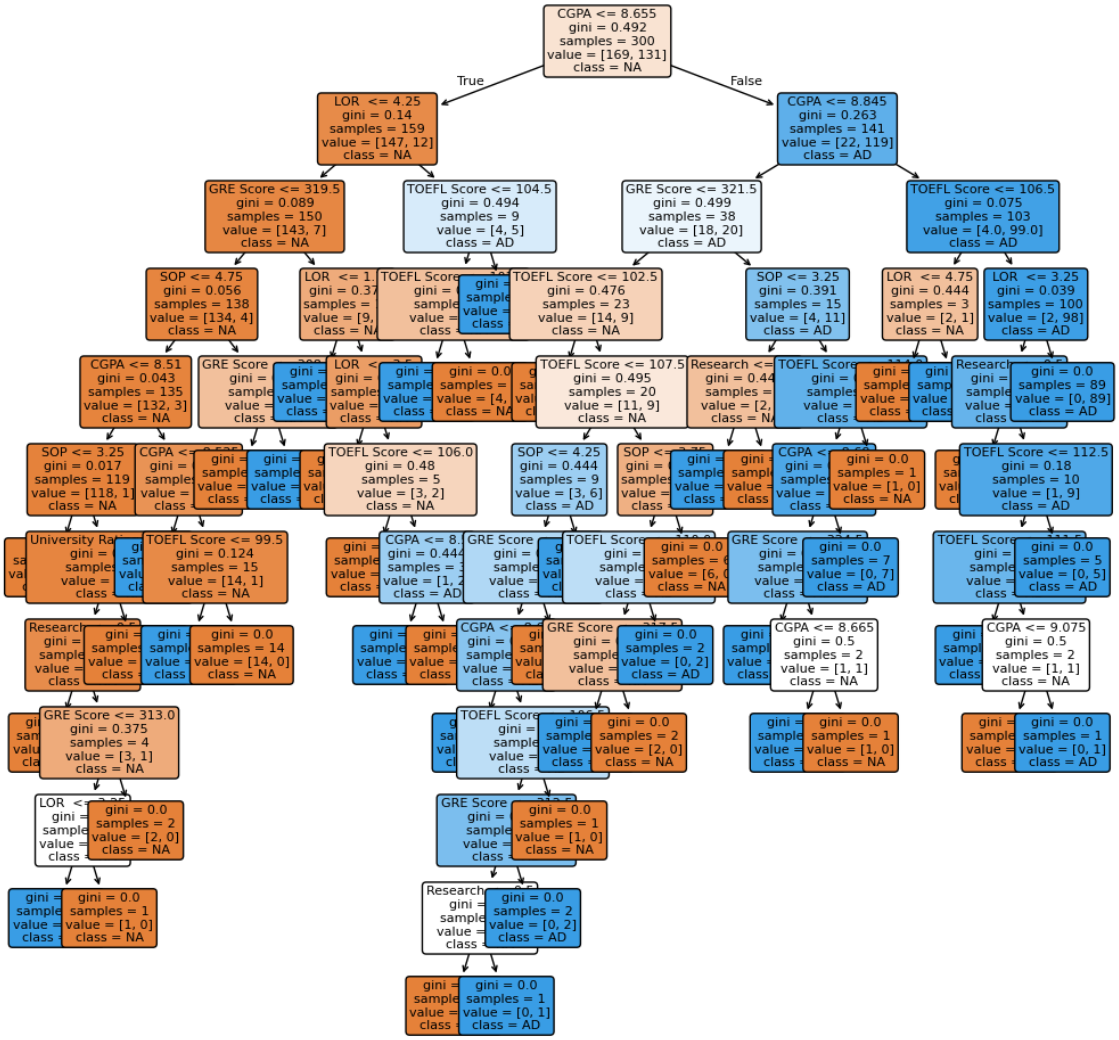
This is a list (or array) representing a new sample for which you want to make a prediction. In this case, new is a 2D list with one row of feature values. Each value in the list represents a feature (or independent variable) corresponding to the model's training data.

This method takes the new sample(s) (new) and makes a prediction using the trained decision tree model (classifier). It returns an array of predicted class labels for each input sample in new. [0]:

Since new contains just a single sample (a list of feature values), classifier.predict(new) will return an array with one predicted value. The [0] is used to extract the first (and only) element from that array, which represents the predicted label for that sample.

[ ]:
```
from sklearn.tree import plot_tree

plt.figure(figsize=(12,12))
plot_tree(classifier,fontsize=8,filled=True,rounded=True,feature_names=x.
  ↪columns,class_names=['NA','AD']);
```



from sklearn.tree import plot_tree:

This imports the plot_tree function from sklearn.tree, which is used to create a graphical representation of a decision tree.

plt.figure(figsize=(12,12)):

This line sets the figure size for the plot. The figsize argument defines the dimensions of the figure in inches, where (12, 12) means the figure will be 12 inches wide and 12 inches tall.

classifier: This is the trained decision tree model that you want to visualize. fontsize=8: This sets the font size of the text in the tree. In this case, the text will be sized at 8 points.

filled=True: This fills the nodes with color. The color represents the class that is most frequent in that node (based on the target variable's class). rounded=True: This rounds the corners of the decision tree nodes, giving it a more visually appealing look.

feature_names=x.columns: This argument defines the names of the features (columns) used in the

decision tree. It assumes that x is a DataFrame, and x.columns will give the feature names for the tree.

class_names=['NA', 'AD']: These are the names of the classes the model is predicting. In this case, ['NA', 'AD'] might represent two possible categories, like "Not Admitted" (NA) and "Admitted" (AD), based on the values of y.

Accuracy

Accuracy is the overall correctness of the model across all classes, measuring the proportion of true results (both true positives and true negatives) out of all predictions.

Calculated as the sum of true positives and true negatives divided by the total number of samples.

Precision

Precision measures how many of the positive predictions made by the model are actually correct. Calculated as the number of true positives divided by the sum of true positives and false positives.

Recall

Recall measures how many of the actual positives were correctly identified by the model.

Calculated as the number of true positives divided by the sum of true positives and false negatives.

F1 Score

The F1-score is the harmonic mean of precision and recall. It balances the two metrics, offering a single performance metric when you want to find a compromise between precision and recall.

Calculated as 2 * (Precision * Recall) / (Precision + Recall).

[ ]: