index

March 20, 2022

1 Building Trees using scikit-learn - Lab

1.1 Introduction

Following the simple example you saw in the previous lesson, you'll now build a decision tree for a more complex dataset. This lab covers all major areas of standard machine learning practice, from data acquisition to evaluation of results. We'll continue to use the Scikit-learn and Pandas libraries to conduct this analysis, following the same structure we saw in the previous lesson.

1.2 Objectives

In this lab you will:

- Use scikit-learn to fit a decision tree classification model
- Use entropy and information gain to identify the best attribute to split on at each node
- Plot a decision tree using Python

1.3 UCI Banknote authentication dataset

In this lab, you'll work with a popular dataset for classification called the "UCI Bank note authentication dataset". This data was extracted from images that were taken from genuine and forged banknotes! The notes were first digitized, followed by a numerical transformation using DSP techniques. The final set of engineered features are all continuous in nature, meaning that our dataset consists entirely of floats, with no strings to worry about. If you're curious about how the dataset was created, you can visit the UCI link here!

We have the following attributes in the dataset:

1. **Variance** of wavelet transformed image (continuous)
2. **Skewness** of wavelet transformed image (continuous)
3. **Curtosis** of wavelet transformed image (continuous)
4. **Entropy** of image (continuous)
5. **Class** (integer) - Target/Label

1.4 Step 1: Import the necessary libraries

We've imported all the necessary modules you will require for this lab, go ahead and run the following cell:

[1]:
```
# Import necessary libraries
import numpy as np
```

```python
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, roc_curve, auc
from sklearn.preprocessing import OneHotEncoder
from sklearn import tree
```

1.5 Step 2: Import data

Now, you'll load our dataset in a DataFrame, perform some basic EDA, and get a general feel for the data you'll be working with.

- Import the file `'data_banknote_authentication.csv'` as a pandas DataFrame. Note that there is no header information in this dataset
- Assign column names `'Variance'`, `'Skewness'`, `'Kurtosis'`, `'Entropy'`, and `'Class'` to the dataset in the given order
- View the basic statistics and shape of the dataset
- Check for the frequency of positive and negative examples in the target variable

[12]:
```python
# Create DataFrame

df = pd.read_csv('data_banknote_authentication.csv', header = None)
df.columns = ['Variance', 'Skewness', 'Kurtosis', 'Entropy', 'Class']
df.head()
```

[12]:

	Variance	Skewness	Kurtosis	Entropy	Class
0	3.62160	8.6661	-2.8073	-0.44699	0
1	4.54590	8.1674	-2.4586	-1.46210	0
2	3.86600	-2.6383	1.9242	0.10645	0
3	3.45660	9.5228	-4.0112	-3.59440	0
4	0.32924	-4.4552	4.5718	-0.98880	0

[15]:
```python
# Describe the dataset
df.describe()
```

[15]:

	Variance	Skewness	Kurtosis	Entropy	Class
count	1372.000000	1372.000000	1372.000000	1372.000000	1372.000000
mean	0.433735	1.922353	1.397627	-1.191657	0.444606
std	2.842763	5.869047	4.310030	2.101013	0.497103
min	-7.042100	-13.773100	-5.286100	-8.548200	0.000000
25%	-1.773000	-1.708200	-1.574975	-2.413450	0.000000
50%	0.496180	2.319650	0.616630	-0.586650	0.000000
75%	2.821475	6.814625	3.179250	0.394810	1.000000
max	6.824800	12.951600	17.927400	2.449500	1.000000

[16]:
```python
# Shape of dataset
df.shape
```

[16]: (1372, 5)

[20]:
```
# Class frequency of target variable
target_frequency = dict(df["Class"].value_counts())
target_frequency
```

[20]: {0: 762, 1: 610}

1.6 Step 3: Create features, labels, training, and test data

Now we need to create our feature set `X` and labels `y`:
- Create `X` and `y` by selecting the appropriate columns from the dataset - Create a 80/20 split on the dataset for training/test. Use `random_state=10` for reproducibility

[86]:
```
# Create features and labels
y = df["Class"]
X = df.drop("Class", axis = 1)
```

[87]:
```
# Perform an 80/20 split
X_train, X_test, y_train, y_test = train_test_split(X,y ,
                                                    test_size = 0.2,
                                                    random_state=10)
```

1.7 Step 4: Train the classifier and make predictions

- Create an instance of a decision tree classifier with `random_state=10` for reproducibility
- Fit the training data to the model
- Use the trained model to make predictions with test data

[88]:
```
# Train a DT classifier

cls = DecisionTreeClassifier(random_state = 10)
cls.fit(X_train, y_train)
```

[88]: DecisionTreeClassifier(random_state=10)

[89]:
```
# Make predictions for test data
y_test_pred = cls.predict(X_test)
```

1.8 Step 5: Check predictive performance

Use different evaluation measures to check the predictive performance of the classifier: - Check the accuracy, AUC, and create a confusion matrix - Interpret the results

[90]:
```
# Calculate accuracy
acc = accuracy_score(y_test,y_test_pred)
print('Accuracy is :{0}'.format(acc))
```

```python
# Check the AUC for predictions
false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test,␣
 ↪y_test_pred)
roc_auc = auc(false_positive_rate, true_positive_rate)
print('\nAUC is :{0}'.format(round(roc_auc, 2)))

# Create and print a confusion matrix
print('\nConfusion Matrix')
print('----------------')

from sklearn.metrics import confusion_matrix
print(confusion_matrix(y_test, y_test_pred))
```

```
Accuracy is :0.9781818181818182

AUC is :0.98

Confusion Matrix
----------------
[[149   3]
 [  3 120]]
```

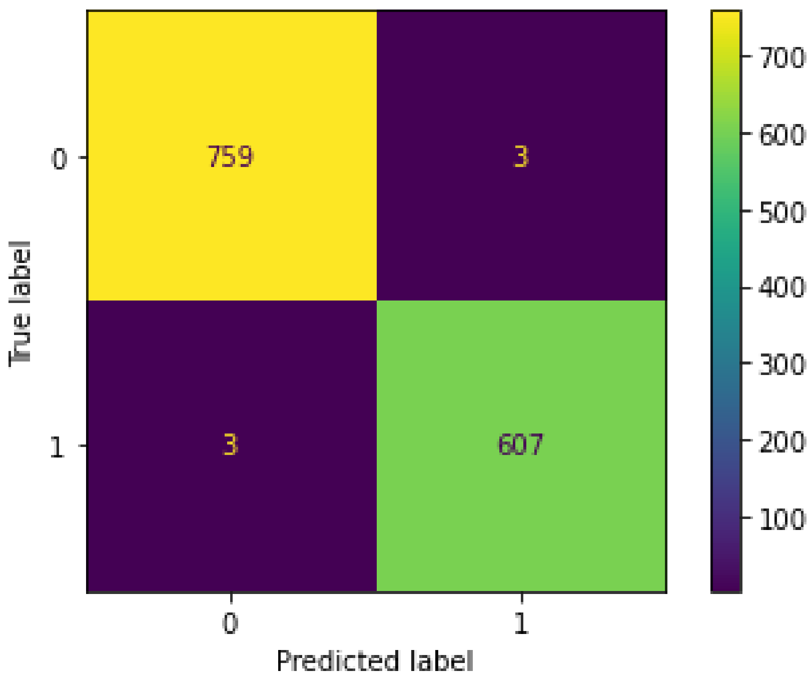
[91]:
```python
aa = confusion_matrix(y_test, y_test_pred)
pd.DataFrame(aa, columns = ["0", "1"], index = ["0", "1"])
```

[91]:

	0	1
0	149	3
1	3	120

[92]:
```python
pt_df = pd.concat([y_test,
                   pd.DataFrame(
                                y_test_pred,
                                columns = ["pred"],
                                index = y_test.index
                                        )
                   ],axis = 1)

condision_t0p0 = (pt_df["Class"]==pt_df["pred"]) & (pt_df["Class"]==0)
condision_t1p1 = (pt_df["Class"]==pt_df["pred"]) & (pt_df["Class"]==1)
condision_t0p1 = (pt_df["Class"]!=pt_df["pred"]) & (pt_df["Class"]==1)
condision_t1p0 = (pt_df["Class"]!=pt_df["pred"]) & (pt_df["Class"]==0)


t1p0 = len(pt_df.loc[condision_t1p0])
t0p1 = len(pt_df.loc[condision_t0p1])
t1p1 = len(pt_df.loc[condision_t1p1])
t0p0 = len(pt_df.loc[condision_t0p0])
```

[98]:
```
### From GitHub

pd.crosstab(y_test, y_test_pred,
            rownames=['True'],
            colnames=['Predicted'],
            margins=True)

from sklearn.metrics import plot_confusion_matrix

plot_confusion_matrix(cls, X, y, values_format='.3g')
plt.show()
```



1.9 Level up (Optional)

1.9.1 Re-grow the tree using entropy

The default impurity criterion in scikit-learn is the Gini impurity. We can change it to entropy by passing in the argument `criterion='entropy'` to the classifier in the training phase.

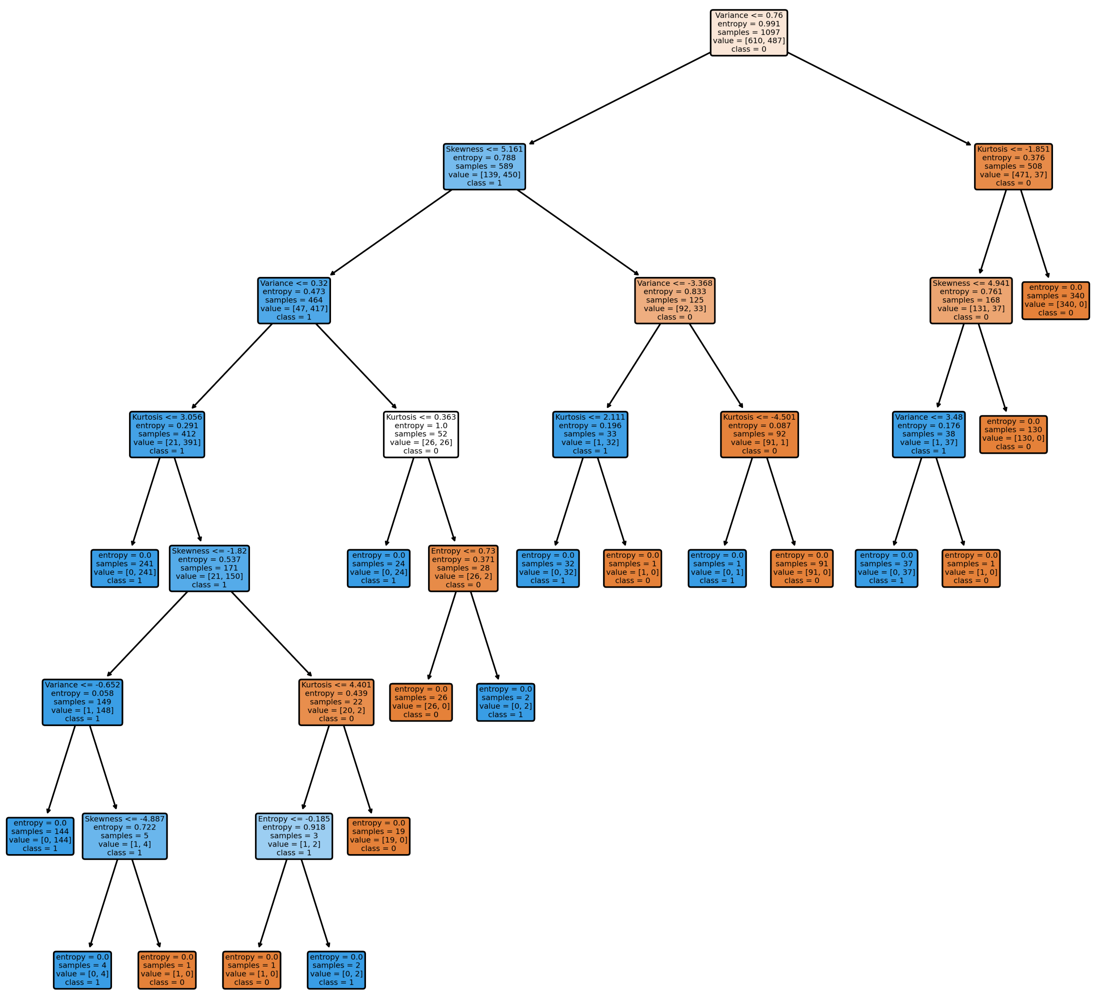
- Create an instance of a decision tree classifier with `random_state=10` for reproducibility. Make sure you use entropy to calculate impurity
- Fit this classifier to the training data
- Run the given code to plot the decision tree

[99]:
```
# Instantiate and fit a DecisionTreeClassifier
classifier_2 = DecisionTreeClassifier(criterion='entropy', random_state=10)
```

```
classifier_2.fit(X_train, y_train)
```

[99]: DecisionTreeClassifier(criterion='entropy', random_state=10)

[100]:
```
# Plot and show decision tree
plt.figure(figsize=(12,12), dpi=500)
tree.plot_tree(classifier_2,
               feature_names=X.columns,
               class_names=np.unique(y).astype('str'),
               filled=True, rounded=True)
plt.show()
```



- We discussed earlier that decision trees are very sensitive to outliers. Try to identify and remove/fix any possible outliers in the dataset.

[ ]:
```

```

- Check the distributions of the data. Is there any room for normalization/scaling of the data? Apply these techniques and see if it improves the accuracy score.

[ ]:
```

```

1.10 Summary

In this lesson, we looked at growing a decision tree for the banknote authentication dataset, which is composed of extracted continuous features from photographic data. We looked at data acquisition, training, prediction, and evaluation. We also looked at growing trees using entropy vs. gini impurity criteria. In following lessons, we shall look at more pre-training tuning techniques for ensuring an optimal classifier for learning and prediction.