1. DT - Simple Example

November 29, 2022

1 Problem Statement

- Predict whether to play tennis or not using Decision Tree

2 Details of DataSet

14 samples

4 Attributes (all are categorical) - Outlook - Temperature - Humidty - Wind

Target Variable - play (yes or no)

3 Import necessary modules

In order to prepare data, train, evaluate, and visualize a decision tree, we will make use of several modules in the scikit-learn package. Run the cell below to import everything we'll need:

[ ]:
```
# For loading data and visualization
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Preprocessing
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import train_test_split

# For model building
from sklearn.tree import DecisionTreeClassifier

# For evaluating the model
from sklearn.metrics import accuracy_score, confusion_matrix,␣
 ↪classification_report

# For visualizing tree
from sklearn import tree
```

4 Load and Analyze the data

The play tennis dataset is available in the repo as `'tennis.csv'`. For this step, we'll start by importing the csv file as a pandas DataFrame.

[ ]:
```
# Load the dataset explore
df = pd.read_csv('../data/tennis.csv')
```

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

[ ]:
```
print(df.shape)
```

[ ]:
```
df.dtypes
```

[ ]:
```
df.outlook.value_counts()
```

[ ]:
```
df.describe()
```

[ ]:
```
# Check for missing values
df.isnull().sum()
```

[ ]:
```
# change data type of windy column to str.
# By default pandas interprets columns with true/false values as boolean
df.windy = df.windy.astype(str)
```

There are no missing values.

5 Create training and test sets

Before we do anything we'll want to split our data into ***training*** and ***test*** sets. We'll accomplish this by first splitting the DataFrame into features (`X`) and target (`y`), then passing `X` and `y` to the `train_test_split()` function to split the data so that 70% of it is in the training set, and 30% of it is in the testing set.

[ ]:
```
X = df.drop('play',axis=1)
```

[ ]:
```
# Separate attributes and target
X = df.iloc[:,0:4] # First 4 columns - Attributes df.drop('play',axis=1)
y = df['play'] # Last column - Target variable
```

[ ]:
```
# Divide the data into train and test
X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size = 0.3,
                                                    random_state = 42)
```

[ ]:
```
# Print the shape of train and test dataframes
X_train.shape, X_test.shape
```

[ ]:
```
X_train.outlook.value_counts()
```

6 Encode categorical attributes as numbers

For more info: https://towardsdatascience.com/categorical-encoding-using-label-encoding-and-one-hot-encoder-911ef77fb5bd

Here, categorical attributes are given as strings, we need to convert them into numbers. We can use two types of encoding.

a) Label Encoding - Each category of an attributes is mapped to a number

b) One-hot encoding - Refer: https://www.geeksforgeeks.org/ml-one-hot-encoding-of-datasets-in-python/

Since all of our data is currently categorical (recall that each column is in string format), we need to encode them as numbers. For this, we'll use a handy helper object from sklearn's `preprocessing` module called `OneHotEncoder`.

[ ]:
```
# One-hot encode the training data and show the resulting
# DataFrame with proper column names

ohe = OneHotEncoder() # Create onehot encoder object
```

[ ]:
```
ohe.fit(X_train) # Fit it with train data - the model learns here
```

[ ]:
```
X_train_ohe = ohe.transform(X_train).toarray() # Perform one-hot encoding
```

[ ]:
```
X_train_ohe
```

[ ]:
```
# Creating this DataFrame is not necessary its only to show the result of the␣
↪ohe
ohe_df = pd.DataFrame(X_train_ohe,
                      columns=ohe.get_feature_names_out(X_train.columns))
```

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

[ ]:
```
ohe_df.shape
```

7 Train the decision tree model

One awesome feature of scikit-learn is the uniformity of its interfaces for every classifier – no matter what classifier we're using, we can expect it to have the same important methods such as `.fit()` and `.predict()`. This means that this next part should feel familiar.

We'll first create an instance of the classifier with any parameter values we have, and then we'll fit our data to the model using `.fit()`.

[ ]:
```
# Create the classifier or an empty model object
model = DecisionTreeClassifier(criterion='entropy')
```

[ ]:
```
# Fit or train the classifier or build the model
# Her we should give both attributes as well class labels
model.fit(X_train_ohe, y_train)
```

Plot the decision tree You can see what rules the tree learned by plotting this decision tree, using matplotlib and sklearn's `plot_tree` function.

[ ]:
```
fig, axes = plt.subplots(nrows = 1,
                         ncols = 1,
                         figsize = (3,3),
                         dpi=300)
tree.plot_tree(model,
               feature_names = ohe.get_feature_names(),
               class_names=np.unique(y).astype('str'),
               filled = True)
plt.show()
```

8 Evaluate the model performance using test data

Now that we have a trained model, we can generate some predictions, and go on to see how accurate our predictions are. We can use a simple accuracy measure, AUC, a confusion matrix, or all of them. This step is performed in the exactly the same manner, so it doesn't matter which classifier you are dealing with.

[ ]:
```
# For test data also we need to perfrom on-hot encoding
X_test_ohe = ohe.transform(X_test)
```

[ ]:
```
# Make the predictions using model
y_preds = model.predict(X_test_ohe)
```

[ ]:
```
y_preds
```

[ ]:
```
# Accuracy
print('Accuracy: ', accuracy_score(y_test, y_preds))
```

[ ]:
```
# Print confusion matrix
confusion_matrix(y_test, y_preds)
```

[ ]:
```
from sklearn.metrics import ConfusionMatrixDisplay
```

[ ]:
```
cm = confusion_matrix(y_test, y_preds)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                              display_labels=model.classes_)
```

[ ]:
```
disp.plot()
```

[ ]:
```
# Print the entire classification report
print(classification_report(y_test, y_preds))
```

9 Save Models

[ ]:
```
import joblib
```

[ ]:
```
# save One hot encoder model
joblib.dump(ohe,'../models/DT_ohe.joblib')
```

[ ]:
```
# save Decision Tree model
joblib.dump(model,'../models/DT_model.joblib')
```