Decision Tree Classifier Building in Scikit-learn.

Practical 1 – W5

This week, we will learn how to train a decision tree classifier using Scikit-learn library.

Download Week5_work.ipynb from Brightspace. The jupyter notebook already loads the data and pre-processed it using the techniques and concepts we covered in last week's lab sessions.

1. Importing Required Libraries

Load the required libraries.

```
# Load libraries
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report
from sklearn import metrics
from sklearn import preprocessing
import pandas as pd
```

1.1 Feature Selection

The Week5_work notebook already has a cleaned dataset stored in a local variable "df". Here we will split out data into predictors and target variables. Target variable is the feature/column we want to predict, and all the other features are predictors. Here we are training a decision tree to predict the maritial_status of our online customers.

```
# data Split
X = df.drop(columns="Marital_status")

y = df["Marital_status"]
```

1.2 Data Scaling

In the previous labs, we have learnt how to scale a data using two techniques, min-max scaling and standardise scaling. You only have to scale the predictors features and not the target feature. We will use the preprocessing method from scikit-learn library.

```
#Normalised Data
normalized_X = preprocessing.normalize(X)
# Standardised Data
standardized_X = preprocessing.scale(X)
```

For classification task, we have to transform our target feature as well.

```
#encode categorical data into digits
y = pd.get_dummies(y)
print(y.head())
```

1.3 Splitting Data

To understand model performance, dividing the dataset into a training set and a test set is a good strategy.

Let's split the dataset by using the function train_test_split(). You need to pass three parameters' features: target, and test_set size. Whereas stratify=y is the random sampling parameter that ensure class distribution in train and test dataset is same.

```
# train test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
stratify=y, random_state=1)
```

1.4 Building Decision Tree Model

Let's create a decision tree model using Scikit-learn.

```
# Create Decision Tree classifier object
dt = DecisionTreeClassifier()

# Train Decision Tree Classifier
Result_DT = dt.fit(X_train,y_train)

#Predict the response for test dataset
y_pred = dt.predict(X_test)
```

Accuracy can be computed by comparing actual test set values and predicted values.

```
#accuracy

data_accuracy = metrics.accuracy_score(y_test, y_pred)
```

```
########################################## Using Normal Data
# decision tree construction
dt = DecisionTreeClassifier(criterion = 'entropy', random_state=1)
dt= dt.fit(X_train, y_train)
#pridiction
y_pred = dt.predict(X_test)
#accuracy
data_accuracy = metrics.accuracy_score(y_test, y_pred)
print("Data Accuracy:",data_accuracy)
########################################## Using Normalised Data
# train test split
X_train, X_test, y_train, y_test = train_test_split(normalized_X,
y,test_size=0.3, stratify=y, random_state=1)
# decision tree construction
dt = DecisionTreeClassifier(criterion = 'entropy', random_state=1)
dt= dt.fit(X_train, y_train)
#prediction
y_pred = dt.predict(X_test)
#accuracy
data_accuracy = metrics.accuracy_score(y_test, y_pred)
print("Normalised data Accuracy:",data_accuracy)
########################################## Using Standardised Data
# train test split
X_train, X_test, y_train, y_test = train_test_split(standardized_X,
y,test_size=0.3,stratify=y, random_state=1)
# decision tree construction
dt = DecisionTreeClassifier(criterion = 'entropy', random_state=1)
dt= dt.fit(X_train, y_train)
#pridiction
y_pred = dt.predict(X_test)
#accuracy
data_accuracy = metrics.accuracy_score(y_test, y_pred)
print("Standardised Accuracy:",data accuracy)
```

1.5 Evaluating the Model

The classification report could be generated by calling classification_report method.

```
print(metrics.classification_report(y_test, y_pred,
digits=2,output_dict=False))
```

1.6 Feature Importance

The following code will give the feature importance in the decision tress classifier.

```
# Variable importance in classifier
print("Variable importacne in the classifier.")
pd.concat((pd.DataFrame(df.iloc[:, 1:].columns, columns =
['variable']),
            pd.DataFrame(dt.feature_importances_, columns =
['importance'])),
           axis = 1).sort_values(by='importance', ascending =
False)[:20]
```

1.7 Optimizing Decision Tree Performance

Here is a little more information on decision tree classifier and some of the important hyper-parameters that you can tweak to improve the model's performance.

Criterion: optional (default="gini") or Choose attribute selection measure. This parameter allows us to use the different-different attribute selection measure. Supported criteria are "gini" for the Gini index and "entropy" for the information gain.

Splitter: string, optional (default="best") or Split Strategy. This parameter allows us to choose the split strategy. Supported strategies are "best" to choose the best split and "random" to choose the best random split.

max_depth: int or None, optional (default=None) or Maximum Depth of a Tree. The maximum depth of the tree. If None, then nodes are expanded until all the leaves contain less than min_samples_split samples. The higher value of maximum depth causes overfitting, and a lower value causes underfitting (Source).

In Scikit-learn, optimization of decision tree classifier performed by only pre-pruning. Maximum depth of the tree can be used as a control variable for pre-pruning. In the following the example, you can plot a decision tree on the same data with max_depth=3. Other than pre-pruning parameters, you can also try other attribute selection measure such as entropy.

```
# Create Decision Tree classifer object

clf = DecisionTreeClassifier(criterion="entropy", max_depth=3)


# Train Decision Tree Classifer

clf = clf.fit(X_train,y_train)


#Predict the response for test dataset

y_pred = clf.predict(X_test)


# Model Accuracy, how often is the classifier correct?

print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```