Python Programming

Classification using Decision Tree

Rathindra Nath Dutta

Senior Research Fellow
Advanced Computing & Microelectronics Unit
Indian Statistical Institute, Kolkata

[Online]

Decision Tree

- Input: labelled categorical dataset
- Result: A learned decision tree to make the classification
- Metrics: Entropy, Gini index, ...

Using Decision Tree in Python

- We will use the `scikit-learn` library[1]

```
pip install -U scikit-learn
```

[1] https://scikit-learn.org/stable/

Using Decision Tree in Python

- We will use the `scikit-learn` library[1]

```
pip install -U scikit-learn
```

- Let `X_train` be the training input data
  and `y_train` be their corresponding labels
- Let `X_test` be the test input data
  and `y_test` be their corresponding labels

[1] https://scikit-learn.org/stable/

Using Decision Tree in Python

- We will use the `scikit-learn` library[1]

```
pip install -U scikit-learn
```

- Let `X_train` be the training input data and `y_train` be their corresponding labels
- Let `X_test` be the test input data and `y_test` be their corresponding labels

- Decision tree based predicted labels can be obtained as follows:

```
from sklearn.tree import DecisionTreeClassifier

clf = DecisionTreeClassifier(criterion='entropy')
clf = clf.fit(X_train, y_train) # build the tree

predictions = clf.predict(X_test)
```

[1] https://scikit-learn.org/stable/

Loading a Sample Dataset

- Iris Dataset consists of 150 samples of 3 different types of irises: Setosa, Versicolour, and Virginica
- Each sample specifies the Sepal Length, Sepal Width, Petal Length and Petal Width along with the iris type

Loading a Sample Dataset

- Iris Dataset consists of 150 samples of 3 different types of irises: Setosa, Versicolour, and Virginica
- Each sample specifies the Sepal Length, Sepal Width, Petal Length and Petal Width along with the iris type

- Can be easily loaded as follows:

```
from sklearn import datasets

iris = datasets.load_iris()
X, y = iris.data, iris.target # data and labels
```

- All data values are categorical, labels are also encoded as integers

Splitting the Data into Training and Test Sets

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y,
                                    test_size = 0.33, random_state = 42)
```

- Here a 2:1 training/test split is being done
- Fixing the `random_state` controls the shuffling applied to the data
- Thus the result is reproducible across multiple calls

Confusion Matrix

- A table to visualize and summarize the performance of a classification algorithm
- Also known as error matrix
- For a binary classification problem the matrix is as follows:

		Predicted	
		Negative **(N)** -	Positive **(P)** +
Actual	Negative -	True Negative **(TN)**	**False Positive (FP) Type I Error**
	Positive +	**False Negative (FN) Type II Error**	True Positive **(TP)**

Confusion Matrix

- A table to visualize and summarize the performance of a classification algorithm
- Also known as error matrix
- For a binary classification problem the matrix is as follows:

		Predicted	
		Negative **(N)** -	Positive **(P)** +
Actual	Negative -	True Negative **(TN)**	**False Positive (FP) Type I Error**
	Positive +	**False Negative (FN) Type II Error**	True Positive **(TP)**

- For a multilevel classification problem, the miss predictions are spread out over the other classes

Displaying Confusion Matrix in Python

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay
import matplotlib.pyplot as plt

cm = confusion_matrix(y_test, predictions,
                                        labels=clf.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                                    display_labels=clf.classes_)
disp.plot()
plt.show()
```

Displaying Confusion Matrix in Python

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay
import matplotlib.pyplot as plt

cm = confusion_matrix(y_test, predictions,
                                        labels=clf.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                                  display_labels=clf.classes_)
disp.plot()
plt.show()
```

- To get a pairwise confusion matrix use:

```
from sklearn.metrics import multilabel_confusion_matrix

mcm = multilabel_confusion_matrix(y_test, predictions)
print(mcm)
```

Classification Report

- Use the following to get the classification performance measures: *precision*, *recall*, *f1-score* and *support*[2]

```
from sklearn.metrics import classification_report

class_names = ['Setosa', 'Versicolour', 'Virginica']
report = classification_report(y_test, predictions,
                               target_names=class_names)
print(report)
```

[2] See https://en.wikipedia.org/wiki/Confusion_matrix for the definitions

Integer Encoding of Categorical Data

- We can use the Category Encoders library[3] for encoding categorical variables into numerical ones

```
pip install category_encoders
```

- Example usage:

```
import category_encoders as ce

encoder = ce.OrdinalEncoder(cols=['col1', 'col2', 'col3'])
X_train = encoder.fit_transform(X_train)
X_test = encoder.transform(X_test)
```

[3] https://contrib.scikit-learn.org/category_encoders

Integer Encoding of Categorical Data

- We can use the Category Encoders library[3] for encoding categorical variables into numerical ones

```
pip install category_encoders
```

- Example usage:

```
import category_encoders as ce

encoder = ce.OrdinalEncoder(cols=['col1', 'col2', 'col3'])
X_train = encoder.fit_transform(X_train)
X_test = encoder.transform(X_test)
```

- See the documentation page for more examples

[3] https://contrib.scikit-learn.org/category_encoders