COMP 6721 Applied Artificial Intelligence (Summer 2023)

Assignment #5: Decision Trees & Evaluation Metrics

Solutions

Question 1 Given the training instances below, use information theory to find whether 'Outlook' or 'Windy' is the best feature to decide when to play a game of golf.

Outlook	Temperature	Humidity	Windy	Play / Don't Play
sunny	85	85	false	Don't Play
sunny	80	90	true	Don't Play
overcast	83	78	false	Play
rain	70	96	false	Play
rain	68	80	false	Play
rain	65	70	true	Don't Play
overcast	64	65	true	Play
sunny	72	95	false	Don't Play
sunny	69	70	false	Play
rain	75	80	false	Play
sunny	75	70	true	Play
overcast	72	90	true	Play
overcast	81	75	false	Play
rain	71	80	true	Don't Play

$$H(Output) = H\left(\frac{5}{14}, \frac{9}{14}\right) = -\left(\frac{5}{14}\log_2\frac{5}{14} + \frac{9}{14}\log_2\frac{9}{14}\right) = 0.94$$

$$H(Output|sunny) = H\left(\frac{3}{5}, \frac{2}{5}\right) = -\left(\frac{3}{5}\log_2\frac{3}{5} + \frac{2}{5}\log_2\frac{2}{5}\right) = 0.97$$

$$H(Output|overcast) = H\left(0, 1\right) = -\left(0\log_2 0 + 1\log_2 1\right) = 0$$

$$H(Output|rain) = H\left(\frac{2}{5}, \frac{3}{5}\right) = -\left(\frac{2}{5}\log_2\frac{2}{5} + \frac{3}{5}\log_2\frac{3}{5}\right) = 0.97$$

$$H(Output|Outlook) = \frac{5}{14}H(Output|sunny) + \frac{4}{14}H(Output|overcast) + \frac{5}{14}H(Output|rain)$$

$$H(Output|Outlook) = \frac{5}{14}0.97 + \frac{4}{14}0 + \frac{5}{14}0.97 = 0.69$$

$$gain(Outlook) = H(Output) - H(Output|Outlook) = 0.94 - 0.69 = 0.25$$

$$H(Output|Windy = true) = H\left(\frac{1}{2}, \frac{1}{2}\right) = 1$$

$$H(Output|Windy = false) = H\left(\frac{1}{4}, \frac{3}{4}\right) = 0.81$$

$$H(Output|Windy) = \frac{6}{14}1 + \frac{8}{14}0.81 = 0.89$$

$$gain(Windy) = H(Output) - H(Output|Windy) = 0.94 - 0.89 = 0.05$$

$\Rightarrow$ 'Outlook' is a better feature because it has a bigger information gain.

Question 2 It's time to leave the calculations to your computer: Write a Python program that uses scikit-learn's *Decision Tree Classifier:*[1]

```
import numpy as np
from sklearn import tree
from sklearn import preprocessing
```

Here is the training data from the first question:

```
dataset = np.array([
['sunny', 85, 85, 0, 'Don\'t Play'],
['sunny', 80, 90, 1, 'Don\'t Play'],
['overcast', 83, 78, 0, 'Play'],
['rain', 70, 96, 0, 'Play'],
['rain', 68, 80, 0, 'Play'],
['rain', 65, 70, 1, 'Don\'t Play'],
['overcast', 64, 65, 1, 'Play'],
['sunny', 72, 95, 0, 'Don\'t Play'],
['sunny', 69, 70, 0, 'Play'],
['rain', 75, 80, 0, 'Play'],
['sunny', 75, 70, 1, 'Play'],
['overcast', 72, 90,1, 'Play'],
['overcast', 81, 75, 0, 'Play'],
['rain', 71, 80, 1, 'Don\'t Play'],
])
```

Note that we changed `True` and `False` into `1` and `0`.

For our feature vectors, we need the first four columns:

```
X = dataset[:, 0:4]
```

and for the training labels, we use the last column from the dataset:

```
y = dataset[:, 4]
```

However, you will not be able to use the data as-is: All features must be numerical for training the classifier, so you have to transform the strings into numbers. Fortunately, scikit-learn has a preprocessing class for *label encoding* that we can use:[2]

```
le = preprocessing.LabelEncoder()
X[:, 0] = le.fit_transform(X[:, 0])
```

(Note: you will need to transform `y` as well.)

[1] See https://scikit-learn.org/stable/modules/tree.html#classification

[2] See https://scikit-learn.org/stable/modules/preprocessing_targets.html#preprocessing-targets

Now you can create a classifier object:

```
dtc = tree.DecisionTreeClassifier(criterion="entropy")
```

Note that we are using the *entropy* option for building the tree, which is the method we've studied in class. Train the classifier to build the tree:

```
dtc.fit(X, y)
```

Now you can predict a new value using `dtc.predict(test)`, just like you did for the Naïve Bayes classifier last week. Note: if you want the string output that you transformed above, you can use the `inverse_transform(predict)` function of a label encoder to change the `predict`ed result back into a string.

You can also print the tree:

```
tree.plot_tree(dtc)
```

but this can be a bit hard to read; to get a prettier version you can use the *Graphviz* visualization software,[3] which you can call from Python like this:

```
# print a nicer tree using graphviz
import graphviz
dot_data = tree.export_graphviz(dtc, out_file=None,
   feature_names=['Outlook', 'Temperature', 'Humidity', 'Windy'],
   class_names=['Don\'t Play', 'Play'],
   filled=True, rounded=True)
graph = graphviz.Source(dot_data)
graph.render("mytree")
```

The result will be stored in a file `mytree.pdf` and should look like Figure 1.

Most of the code is provided above. For transforming the labels, you can use the same label encoder:

```
y = le.fit_transform(dataset[:, 4])
```

To predict a label for a new feature vector:

```
y_pred = dtc.predict([[2, 81, 95, 1]])
print("Predicted output: ", le.inverse_transform(y_pred))
```

To print the tree using the encoded class names:

```
dot_data = tree.export_graphviz(dtc, out_file=None,
   feature_names=['Outlook', 'Temperature', 'Humidity', 'Windy'],
   class_names=le.classes_,
   filled=True, rounded=True)
```

[3] See https://www.graphviz.org/

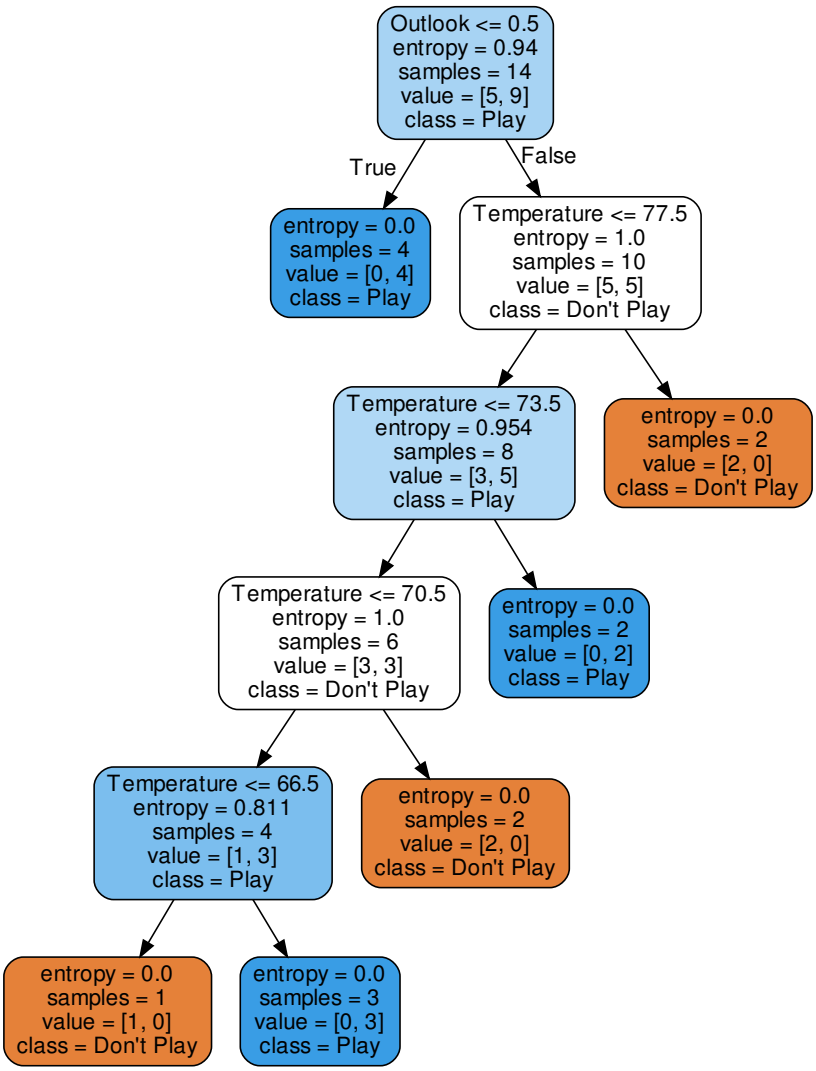

Figure 1: Generated Decision Tree using scikit-learn: Note that the string values for *Outlook* have been changed into numerical values ('overcast'= 0, 'rain'= 1, 'sunny' = 2)

Question 3 For the tree of the last question, report the following parameters or metrics:

(a) Number of leaves

(b) Height (depth) of the tree

(c) External Path Length

(d) Weighted External Path Length

(e) Accuracy, recall, and precision

(f) Now assume that we do pruning considering the hyper parameter of "Minimum Leaf Size" =2. What will be the accuracy, recall, and precision of this tree (training)?

(a) Number of leaves = 6 (3 dark blue, 3 orange)

(b) Height (depth) of the tree = 5

(c) External Path Length = 20 (1+2+3+4+5+5) (Avg:3.3)

(d) Weighted External Path Length = $\frac{4}{14}$ x1 + $\frac{2}{14}$ x2 + $\frac{2}{14}$ x3 + $\frac{2}{14}$ x4 +$\frac{1}{14}$ x5 + $\frac{3}{14}$ x5 = $\frac{42}{14}$ = 3.

(e) As can be seen, all leaves are pure (entropy=0), so the training accuracy, recall, and precision are all equal to 1 (100%)

(f) With the condition of "Minimum Leaf Size"=2, the last split (deepest) will be pruned, and the light blue node with [1,3] becomes the new leaf. The tree will predict class "Play" in this leaf (3 Play vs 1 Don't), so there will be 1 out 14 classification error, which gives an accuracy of $\frac{13}{14} = 0.93$. Since this missclassifed sample has a label of "Don't Play" (class 0), the recall of the tree will still be 1 (100%) while the precision will be $\frac{9}{10} = 0.9$.

Question 4 Let's start working with some "real" data: scikit-learn comes with a number of example datasets, including the *Iris* flower dataset. If you do not know this dataset, start by reading https://en.wikipedia.org/wiki/Iris_flower_data_set.[4]

(a) Change your program from the previous question to work with this new dataset:

```
from sklearn.datasets import load_iris
X, y = load_iris(return_X_y=True)
```

Train a decision tree and print it out like before.

Here is a minimal solution:

```
from sklearn import tree
from sklearn.datasets import load_iris

# load the Iris dataset
iris = load_iris()
X, y = iris.data, iris.target

# create and print the decision tree
dtc = tree.DecisionTreeClassifier(criterion="entropy")
dtc.fit(X, y)
tree.plot_tree(dtc)
```

For a nicer tree plot, again use graphviz:

```
# print a nicer tree using graphviz
import graphviz
dot_data = tree.export_graphviz(dtc, out_file=None,
                                feature_names=iris.feature_names,
                                class_names=iris.target_names,
                                filled=True, rounded=True,
                                special_characters=True)
graph = graphviz.Source(dot_data)
graph.render("iristree")
```

(b) Now you have to *evaluate* the performance of your classifier. Use scikit-learn's `train_test_split` helper function[5] to split the Iris dataset into a *training* and *testing* subset, as discussed in the lecture. Now run an *evaluation* to compute the *Precision, Recall, F_1-measure, and Accuracy* of your classifier using the evaluation tools in scikit-learn.[6]

[4] Also see https://scikit-learn.org/stable/auto_examples/datasets/plot_iris_dataset.html for the scikit-learn documentation of this dataset.

[5] See https://scikit-learn.org/stable/modules/cross_validation.html

[6] See https://scikit-learn.org/stable/modules/model_evaluation.html

To create an 80%/20% split of the training/testing data, use:

```
X_train, X_test, y_train, y_test = train_test_split(
   X, y, test_size=0.2, random_state=0)
```

scikit-learn has a helper function to produce a report for all the metrics, `classification_report`:[7]

```
y_pred = dtc.predict(X_test)
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

Which should result in a report like this:

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        11
           1       1.00      1.00      1.00        13
           2       1.00      1.00      1.00         6

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```

As you can see, the data was easy to learn and your classifier has a perfect score. Try decreasing the training data to just 50% of the dataset to see some errors.

Note: Since we have more than two classes, the above metrics are an *average* of the values for each individual class.

(c) Finally, compute and print out the *confusion matrix*.[8]

This is easy using the helper function `confusion_matrix`:

```
from sklearn.metrics import confusion_matrix
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

which results in

```
Confusion Matrix:
 [[11  0  0]
 [ 0 13  0]
 [ 0  0  6]]
```

[7] See https://scikit-learn.org/stable/modules/model_evaluation.html#classification-report

[8] See https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html