COMP9417 - Machine Learning
Tutorial: Tree Learning

Weekly Problem Set: Please submit questions 1a, 1d, 4a, 4b on Moodle by 11:55am Tuesday 15th March, 2022. Please only submit these requested questions and no others.

Question 1. Expressiveness of Trees

Give decision trees to represent the following Boolean functions, where the variables `A`, `B`, `C` and `D` have values `t` or `f`, and the class value is either `True` or `False`. Can you observe any effect of the increasing complexity of the functions on the form of their expression as decision trees ?

(a) $A \wedge \neg B$

Solution:

```
A = t:
|    B = f:  True
|    B = t:  False
A = f:  False
```

(b) $A \vee [B \wedge C]$

Solution:

$A \vee [B \wedge C]$

```
A = t:  True
A = f:
|    B = f:  False
|    B = t:
|    |    C = t:  True
|    |    C = f:  False
```

(c) A XOR B

Solution:

```
A = t:
|    B = t:  False
|    B = f:  True
A = f:
```

```
|    B = t:  True
|    B = f:  False
```

(d) $[A \land B] \lor [C \land D]$

Solution:

```
A = t:
|    B = t:  True
|    B = f:
|    |    C = t:
|    |    |    D = t:  True
|    |    |    D = f:  False
|    |    C = f:  False
A = f:
|    C = t:
|    |    D = t:  True
|    |    D = f:  False
|    C = f:  False
```

Notice the *replication* effect of repeated subtrees as the target expression becomes more complex, for example, in the tree for *d*. This is a situation where the hypothesis class of models (here, decision trees) can fit any Boolean function, but in order to represent the function the tree may need to be very complex. This makes it hard to learn, and will require a lot data !

Question 2. Decision Tree Learning

(a) Assume we learn a decision tree to predict class Y given attributes A, B and C from the following training set, with no pruning.

A	B	C	Y
0	0	0	0
0	0	1	0
0	0	1	0
0	1	0	0
0	1	1	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	0	1
1	1	1	0
1	1	1	1

What would be the training set error for this dataset ? Express your answer as the number of examples out of twelve that would be misclassified.

Solution:

2. There are two pairs of examples with the same values for attributes A, B and C but a different (contradictory) value for class Y. One example from each of these pairs will always be misclassified (noise).

(b) One nice feature of decision tree learners is that they can learn trees to do *multi-class* classification, i.e., where the problem is to learn to classify each instance into exactly one of $k > 2$ classes.

Suppose a decision tree is to be learned on an arbitrary set of data where each instance has a discrete class value in one of $k > 2$ classes. What is the maximum training set error, expressed as a fraction, that any dataset could have ?

Solution:

First consider the case where all k class values are evenly distributed, so there are $\frac{1}{k}$ examples of each class in the training set. In the worst case a decision tree can be learned that predicts one of the k classes for all examples. This will get $\frac{1}{k}$ of the training set correct, and make $1 - \frac{1}{k} = \frac{k-1}{k}$ mistakes as a fraction of the training set.

If any one class has more than $\frac{1}{k}$ examples then the worst case decision tree is guaranteed to predict that class, which will reduce the error since that class now represents more than $\frac{1}{k}$ of the training set.

Question 3. ID3 Algorithm

Here is small dataset for a two-class prediction task. There are 4 attributes, and the class is in the rightmost column (homeworld). Look at the examples. Can you guess which attribute(s) will be most predictive of the class ?

species	rebel	age	ability	homeworld
pearl	yes	6000	regeneration	no
bismuth	yes	8000	regeneration	no
pearl	no	6000	weapon-summoning	no
garnet	yes	5000	regeneration	no
amethyst	no	6000	shapeshifting	no
amethyst	yes	5000	shapeshifting	no
garnet	yes	6000	weapon-summoning	no
diamond	no	6000	regeneration	yes
diamond	no	8000	regeneration	yes
amethyst	no	5000	shapeshifting	yes
pearl	no	8000	shapeshifting	yes
jasper	no	6000	weapon-summoning	yes

You probably guessed that attributes 3 and 4 were not very predictive of the class, which is true. However, you might be surprised to learn that attribute "species" has higher information gain than attribute "rebel". Why is this ?

Suppose you are told the following: for attribute "species" the Information Gain is 0.52 and *Split Information* is 2.46, whereas for attribute "rebel" the Information Gain is 0.48 and *Split Information* is 0.98.

Which attribute would the decision-tree learning algorithm select as the split when using the *Gain Ratio* criterion instead of Information Gain ? Is Gain Ratio a better criterion than Information Gain in this case ?

Solution:

Gain Ratio corrects for a bias in Information Gain that favours attributes with many values by dividing it by the Split Information, which is the information of the partition of the data according to the values of the attribute in question. This is shown in the formula on slide 72 for Split Information of some attribute A on a sample S, where c is the number of values for A.

For attribute "species" Gain Ratio is $\frac{0.52}{2.46} \simeq 0.21$, whereas for attribute "rebel" Gain Ratio is $\frac{0.48}{0.98} \simeq 0.49$. So attribute "rebel" would be selected under the Gain Ratio criterion.

In this case, it is a better choice, since the "species" attribute has higher information gain simply by having many more values in this dataset. Note however that other corrections to information gain have been proposed that have other advantages in different settings, and in general there is no "best" splitting criterion for all settings.

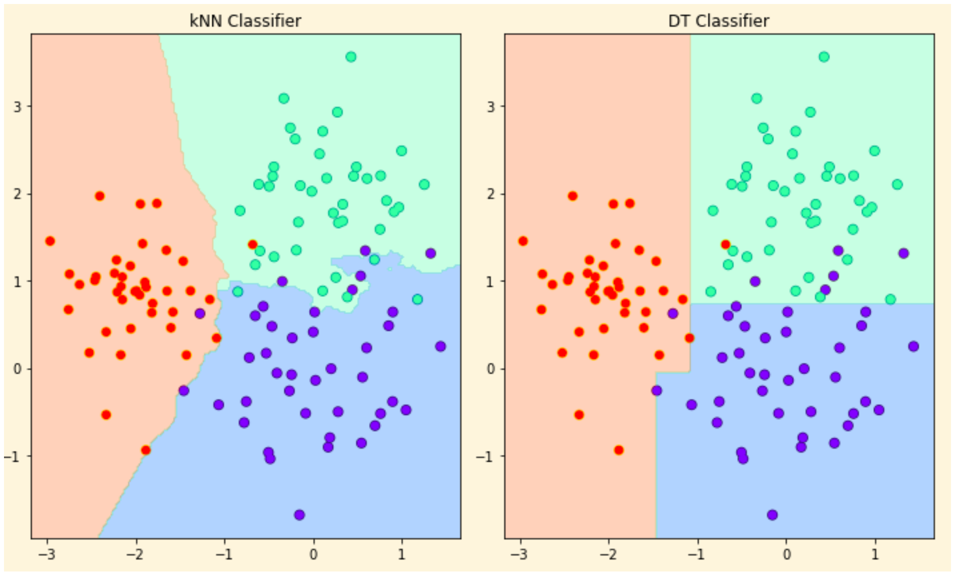
Question 4. Working with Decision Trees

In `utils.py` you will find the implementation of the function `visualize_classifier` which allows us to visualise any classifier that has a `predict` method. You can use this function as a black box throughout. The `sklearn.datasets.make_blobs` function gives us a quick way to create toy data for classification. In the following we'll create a 3 class classification problem:

```python
from sklearn.datasets import make_blobs
X, y = make_blobs(n_samples=120,                        # total number of samples
                  centers=[[0,0], [0,2], [-2,1]],       # cluster centers of the 3 classes
                  random_state=123,                     # reproducibility
                  cluster_std=0.6)                      # how spread out are the samples
    from their center

plt.scatter(X[:, 0], X[:, 1], c=y, s=50)                # scatter with color=label
plt.show()
```

(a) Use `sklearn.neighbors.KNeighborsClassifier` and `sklearn.tree.DecisionTreeClassifier` objects to demonstrate the `visualize_classifier` function. Explain the differences between the decision boundaries of the two classifiers.

Solution:



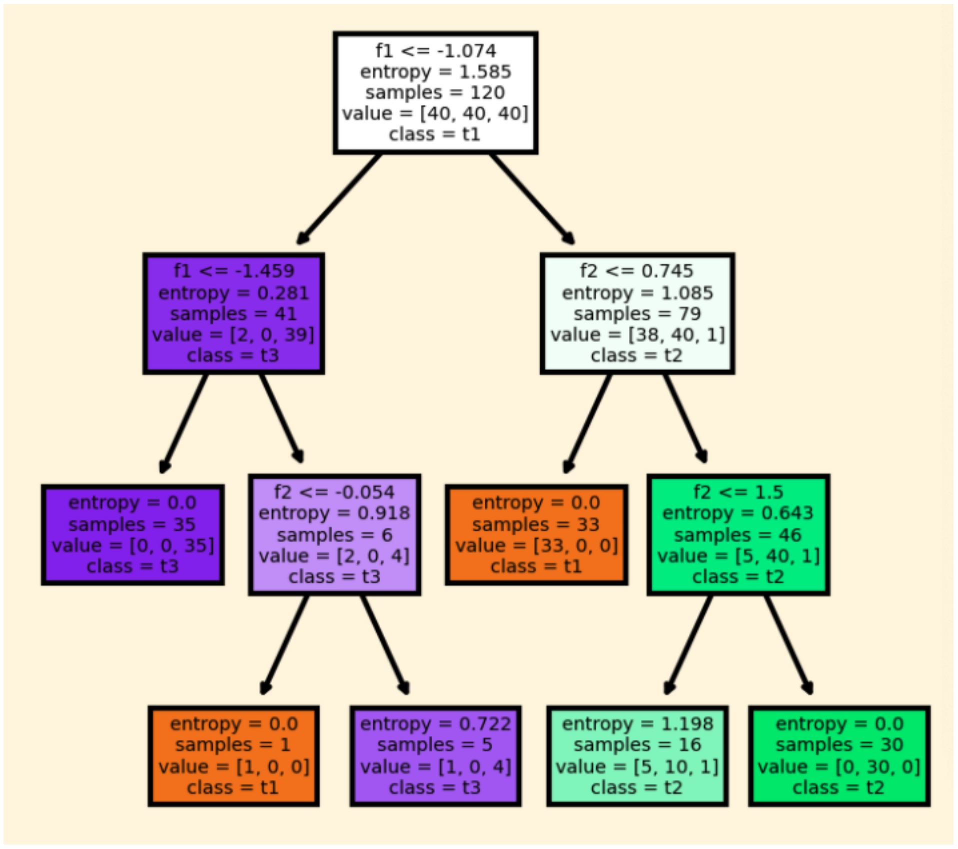
The shaded regions correspond to how the classifier would classify a point that falls in that region. Note that for the kNN classifier, the regions are quite jagged, whereas the regions for the DT are always constructed by splitting using straight lines on a particular axis.

(b) Another way to visualise a tree can be done by running:

```
    from sklearn import tree

    fig, axes = plt.subplots(1, 1,figsize = (3,3), dpi=300)
    tree.plot_tree(model1,          # fitted decision tree
    feature_names=['f1', 'f2'],     # names for features
    class_names=['t1', 't2', 't3'], # names for class labels
    filled=True)
    plt.show()

```

Explain what is going on in the resulting plot. What do the colors represent? What does the `value` argument tells us? What about `entropy`?

Solution:



Note that the color of the labels correspond to the majority class at that particular point. The plot also gives us the distribution of the three classes at each node (value), and tells us the number of samples that falls into a particular node. We also can see the information gain (entropy) for each node.

(c) Generate data using the following code:

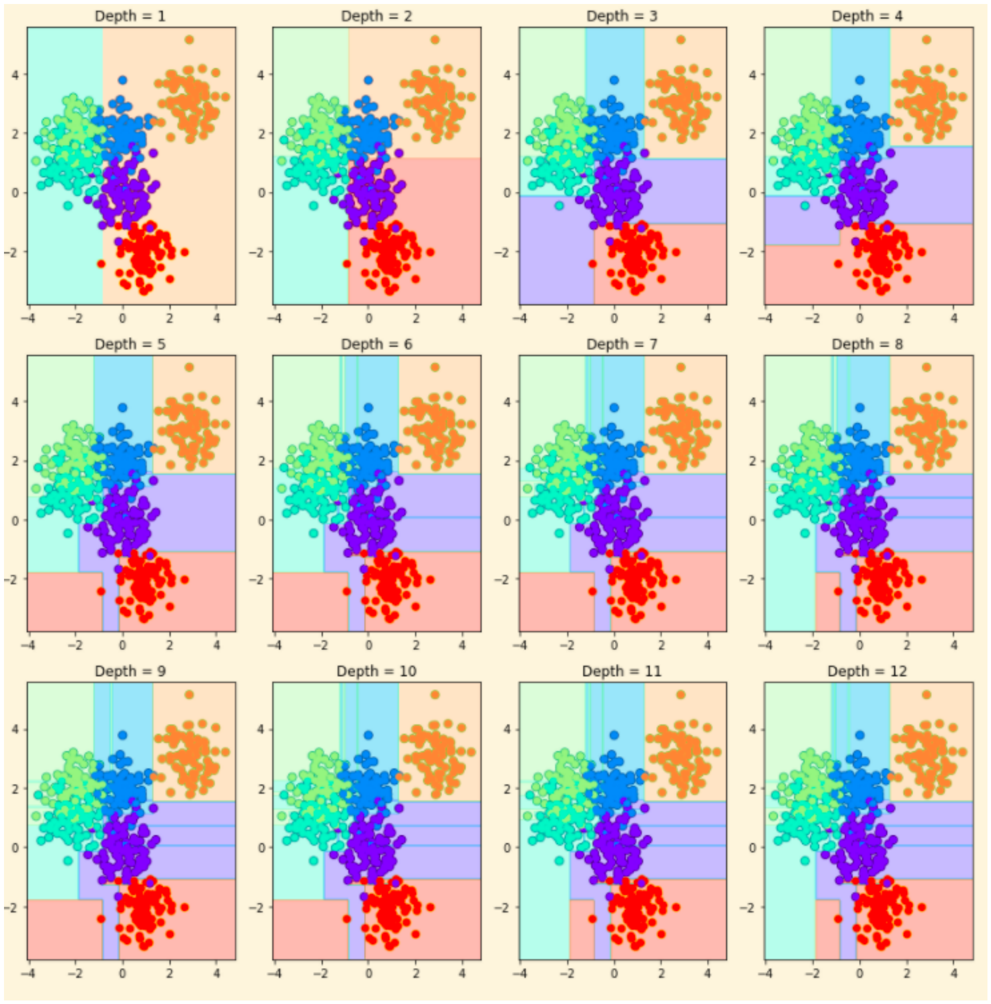
```
X, y = make_blobs(n_samples=500,
                  centers=[[0,0], [0,2], [-2,1], [-2,2], [3,3], [1,-2]],
                  random_state=123,
                  cluster_std=0.6)

```

Then fit a decision tree (using information gain for splits) with `max_depth` set to $1, 2, \ldots, 12$ and visualize the classifier (use a 3×4 grid). What do you observe? Why do you think decision trees are described as performing 'recursive partitioning'?

Solution:
When depth=1, the tree is just a decision stump (a single if/else statement) and so it can only choose a single feature to split on (either feature 1 which we can call f1 (x axis) or f2 (y axis)).

Here we see that it splits on f1, and classifies everything to the left of the split as aqua blue, and everything to the right of the split as purple, since these are the majority classes on each side. This is a partition of the 2d space into two rectangles.

When we increase the depth by 2, the tree has more flexibility (chained if/else), and can now partition the original partition. It does so in this case by splitting on f2 to isolate the red cluster. In depth 3 it partitions the previous partition. This is why we say that trees perform recursive partitioning, at each depth they simply look at the partition from the previous depth and partition it in a way to isolate new clusters. As we keep increasing the depth, the partitions become finer and finer (of course in practice, we need to trade-off the fine-ness of the partition with the risk of over fitting).



```
X, y = make_blobs(n_samples=500,
    centers=[[0,0], [0,2], [-2,1], [-2,2], [3,3], [1,-2]],
    random_state=123,
```

```python
    cluster_std=0.6)

fig, axes = plt.subplots(3, 4, figsize=(12,12))

for i, ax in enumerate(axes.flat):

    ax.scatter(X[:, 0], X[:, 1], c=y, s=50)
    model = DecisionTreeClassifier(max_depth=i+1, criterion='entropy').fit(X,y)
    visualize_classifier(model, X, y, ax=ax)
    ax.set_title(f"Depth = {i+1}")

plt.tight_layout()
plt.show()

```