COMPSCI 371D Homework 7

Problem 0 (3 points) ¶

Part 1: Decision Trees

In [6]:

```
import geometry as geo
```

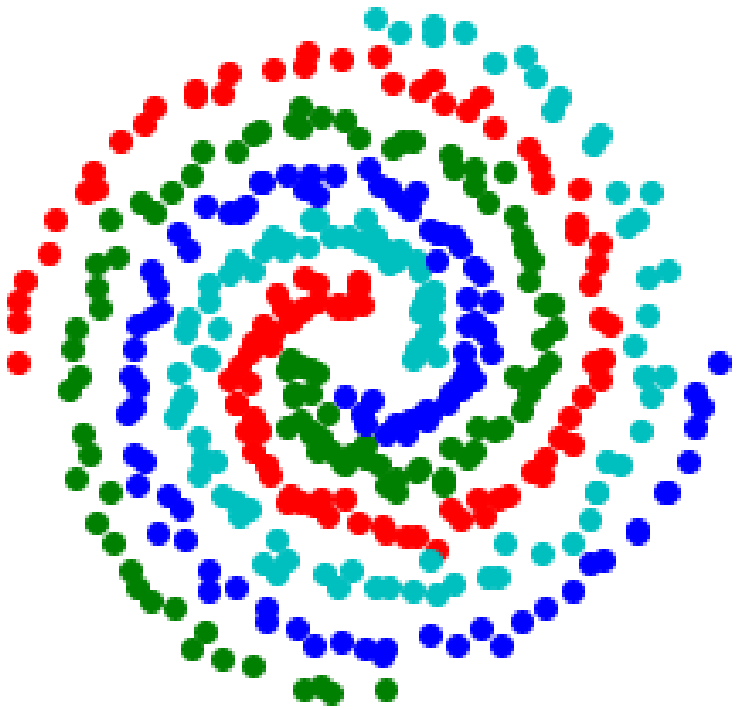
In [7]:

```
import pickle
import numpy as np

with open('data.pkl', 'rb') as file:
    spirals = pickle.load(file)

x_train, y_train = tuple(zip(*spirals['train']))
x_train, y_train = np.array(x_train), np.array(y_tra
in)

x_test, y_test = tuple(zip(*spirals['test']))
x_test, y_test = np.array(x_test), np.array(y_test)

label_colors = spirals['colors']
n_classes = len(spirals['colors'])
```

Problem 1.1

In [8]:

```
import matplotlib.pyplot as plt
x_train1=x_train[:,0]
x_train2=x_train[:,1]
print()
for i in range(len(x_train1)):
    ycolor=label_colors[y_train[i]]
    plt.scatter(x_train1[i],x_train2[i], color= ycol
or)

plt.axis('off')
plt.gca().set_aspect(1)
plt.show()
print("The data is not linearly separable.")
```



```
The data is not linearly separable.
```

Problem 1.2

In [9]:

```
def plot_decision_regions(t, node_id, region, colors
):


    if(t.feature[node_id]<0):
        color='r'
        colorarr=t.value[node_id]
        val=colorarr[0]

        for i in range(len(val)):
            check=val[i]
            if(check>0):
                color=colors[i]
                geo.plot_polygon(region, fill_color=
color)

    else:

        dim=t.feature[node_id]
        u=0
        v=0
        if(dim==0):
            u=1
            v=0
        if(dim==1):
            u=0
            v=1

        vector=geo.vector(u, v)
        line=geo.line(vector, -1*t.threshold[node_id
])
        cut=geo.cut_polygon(region, line)
        geo.plot_segment(cut.segment)
```

```
        left_id, right_id = t.children_left[node_id
], t.children_right[node_id]
        plot_decision_regions(t, left_id, cut.negati
ve, colors)
        plot_decision_regions(t, right_id, cut.posit
ive, colors)

        if(node_id==0):
            geo.plot_polygon(region, boundary_color=
'k')
            plt.gca().set_aspect(1)
            plt.axis('off')
```

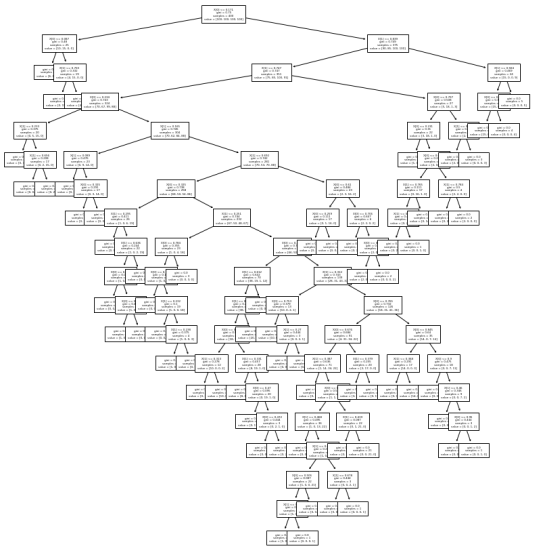
In [55]:

```
from sklearn import tree
from sklearn.metrics import accuracy_score
from sklearn.tree import DecisionTreeClassifier

region=geo.data_space()
treefig= plt.figure(figsize=(30, 15))
regionfig = plt.figure(figsize=(15, 15))
treetype=['best','random']

for i in range(2):

    decTree=tree.DecisionTreeClassifier(splitter=tre
etype[i])
    title=treetype[i]



    decTree.fit(x_train, y_train)

    axt=treefig.add_subplot(1,2,i+1)
    tree.plot_tree(decTree, ax=axt, fontsize=4)
    axt.set_title(title, fontsize=40)
    treefig.show()

    decTreepredict=decTree.predict(x_test)
    score=accuracy_score(y_test, decTreepredict)*100

    axr=regionfig.add_subplot(1,2,i+1)
    axr.set_title(title, fontsize=20)
    plot_decision_regions(decTree.tree_, 0, region,
label_colors)
    axr.set_xlabel(accurstr)
    regionfig.show()
```

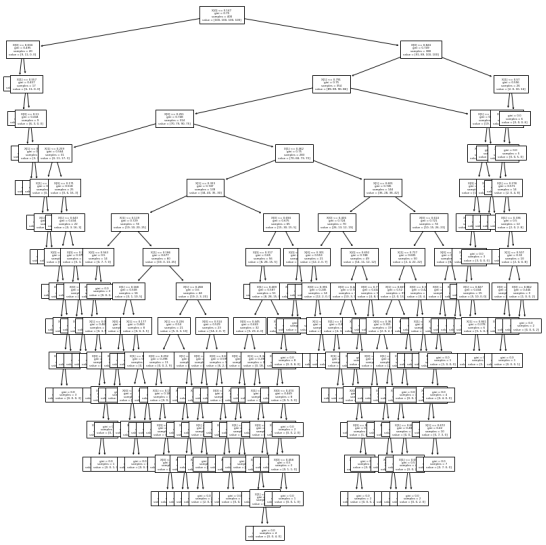
```
    print(title + " accuracy: {:0.2f}\n".format(scor
e))
```

```
<ipython-input-55-c10e1ad875f4>:21: User
Warning: Matplotlib is currently using m
odule://ipykernel.pylab.backend_inline,
which is a non-GUI backend, so cannot sh
ow the figure.
  treefig.show()
<ipython-input-55-c10e1ad875f4>:30: User
Warning: Matplotlib is currently using m
odule://ipykernel.pylab.backend_inline,
which is a non-GUI backend, so cannot sh
ow the figure.
  regionfig.show()

best accuracy: 92.45


<ipython-input-55-c10e1ad875f4>:21: User
Warning: Matplotlib is currently using m
odule://ipykernel.pylab.backend_inline,
which is a non-GUI backend, so cannot sh
ow the figure.
  treefig.show()
<ipython-input-55-c10e1ad875f4>:30: User
Warning: Matplotlib is currently using m
odule://ipykernel.pylab.backend_inline,
which is a non-GUI backend, so cannot sh
ow the figure.
  regionfig.show()

random accuracy: 90.42
```

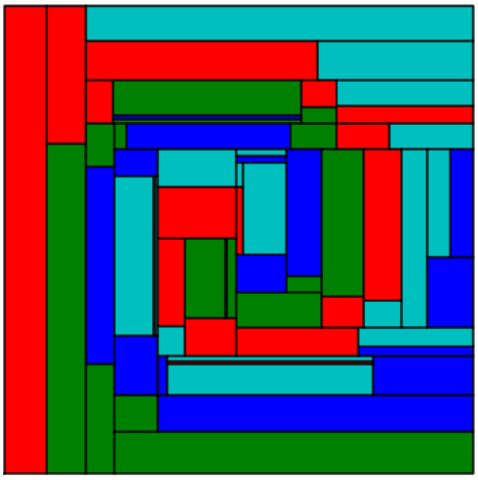
best



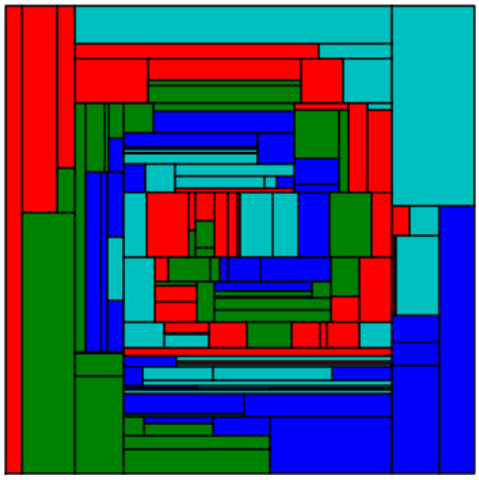
random



best



random



Problem 1.3 (Exam Style)

Looking at the results, the decision tree created with the 'best' split is less deep than the decision tree create with the 'random split', and the ‘best’ decision tree has greater accuracy by about 2-3%. It’s clear by inspection that the ‘random’ tree has many longer and denser paths than the ‘best’. ‘Best’ takes longer to train because you have to go through all possible thresholds of each dimension to calculate the impurity, so O(n^2). The random splitter doesn't have to go through all dimensions, it just picks one and goes through all thresholds, so training time is faster, O(n). Since the randomly split tree is longer and more dense, testing might take more time to go through the tree, but more time is saved by sampling randomly during training when compared to using 'best' during testing. Looking at the accuracy, both methods seem similar in their accuracy by inspection of the figures.

Part 2: Random Decision Forests

Problem 2.1

In [67]:

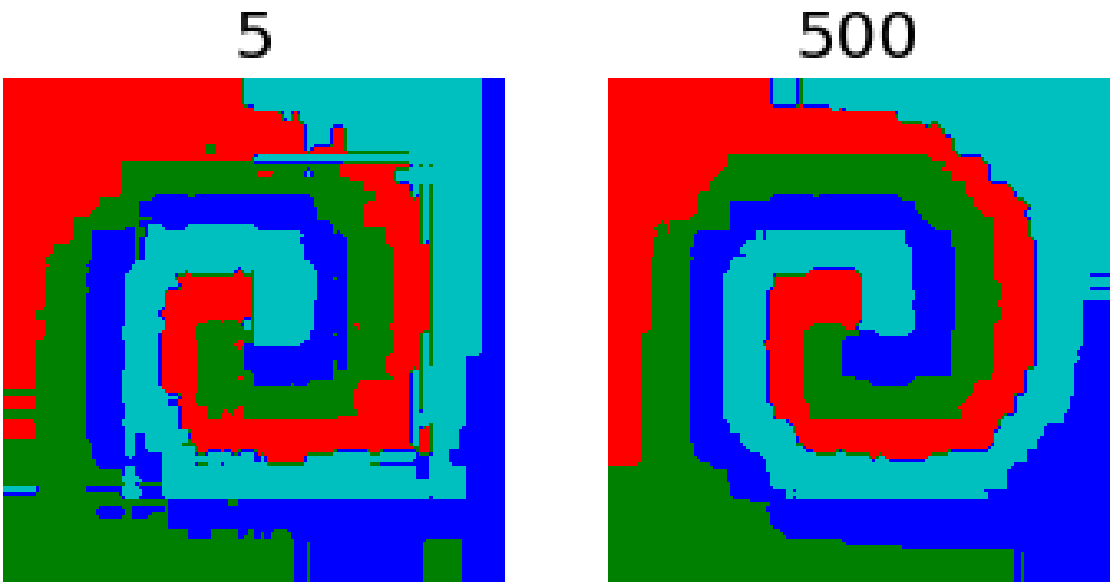
```
def coarse_regions(h, space, colors, step=0.01):

    color_map = ListedColormap(colors)
    x=space[:,0]
    y=space[:,1]
    xx, yy = np.meshgrid(np.arange(np.amin(x), np.amax(x), step), np.arange(np.amin(y), np.amax(y), step))
    predictions=h.predict(np.c_[xx.ravel(), yy.ravel()])
    predictions=predictions.reshape(xx.shape)
    cs = plt.contourf(xx, yy, predictions, cmap=color_map)
    plt.gca().set_aspect(1)
    plt.axis('off')
```

In [76]:

```
from sklearn.ensemble import RandomForestClassifier
from matplotlib.colors import ListedColormap
n_estimators=[5,500]
space=geo.data_space()
forestFig=plt.figure()
for i in range(2):
    decForest=RandomForestClassifier(n_estimators=n_
estimators[i], max_depth=None,
                                     min_samples_split=2
,random_state=0,oob_score=True)

    decForest.fit(x_train, y_train)
    title=str(n_estimators[i])

    decForestpredict=decForest.predict(x_test)
    testscore=accuracy_score(y_test, decForestpredic
t)*100
    print(title + " test accuracy: {:0.2f}\n".format
(testscore))

    oobscore=decForest.oob_score_*100

    print(title + " out-of-bag score: {:0.2f}\n".for
mat(oobscore))

    axF=forestFig.add_subplot(1,2,i+1)
    axF.set_title(n_estimators[i], fontsize=20)
    coarse_regions(decForest, space, label_colors, s
tep=0.01)
    forestFig.show()
```

```
/opt/anaconda3/lib/python3.8/site-packag
es/sklearn/ensemble/_forest.py:540: User
Warning: Some inputs do not have OOB sco
res. This probably means too few trees w
ere used to compute any reliable oob est
imates.
  warn("Some inputs do not have OOB scor
es. "
/opt/anaconda3/lib/python3.8/site-packag
es/sklearn/ensemble/_forest.py:544: Runt
imeWarning: invalid value encountered in
true_divide
  decision = (predictions[k] /
<ipython-input-76-7b7d72bef246>:24: User
Warning: Matplotlib is currently using m
odule://ipykernel.pylab.backend_inline,
which is a non-GUI backend, so cannot sh
ow the figure.
  forestFig.show()



5 test accuracy: 91.12

5 out-of-bag score: 65.25

500 test accuracy: 96.47

500 out-of-bag score: 90.25
```

```
<ipython-input-76-7b7d72bef246>:24: User
Warning: Matplotlib is currently using m
odule://ipykernel.pylab.backend_inline,
which is a non-GUI backend, so cannot sh
ow the figure.
  forestFig.show()
```



Problem 2.2 (Exam Style)

OOB score is calculated using training data from the training set that wasn't used in all of the trees in the forest. When there are fewer trees, or M<20 trees, the probability that a sample is in the OOB set is lower based on 1 – $(1 - 0.37)^M$, so the set isn't as representative of the actual data. This can be seen with the much lower OOB score of 65.25 and causes the accuracy to be off. The probability of a sample being in the OOB set is much higher with 500 trees, so the set is a more representative sample of the forest, and there are more trees voting on the results. This can be seen by the OOB score and the test risk being better with 500 trees. The decision region for 5 trees is less smooth and has more gaps where one class breaks into another whereas the decision region for 500 trees is more well defined. As dimension grows, unless training data also grew with it, the decision regions would become less accurate and more grainy like the decision region is for 5 trees.