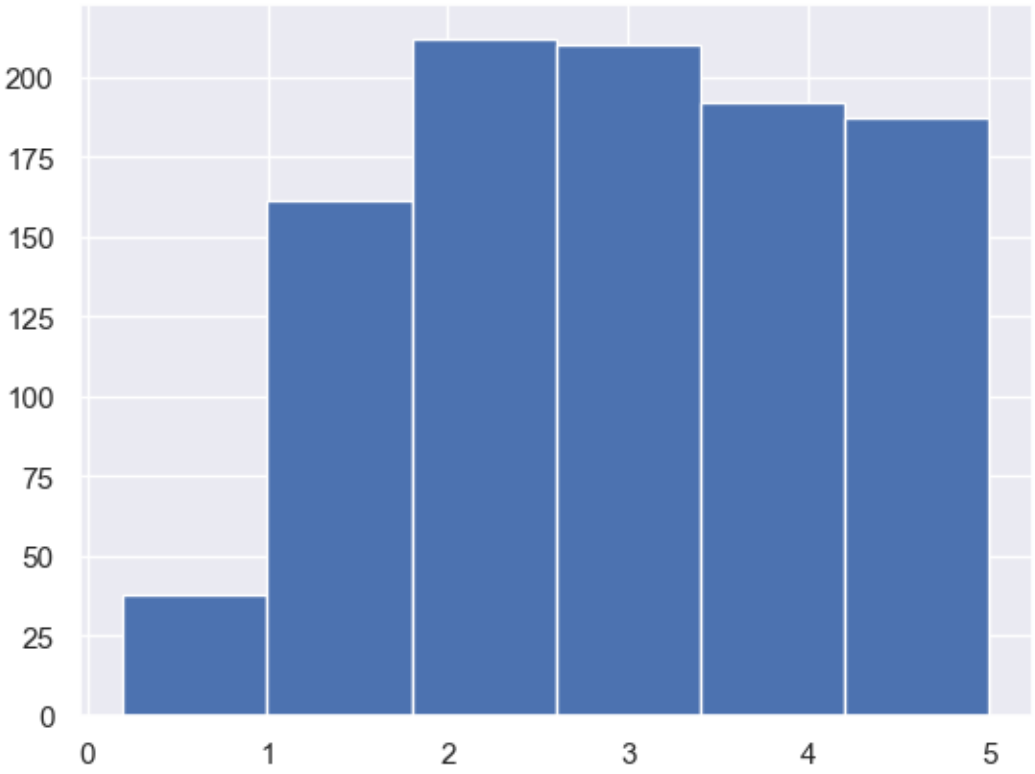
Visualizing Trees HW1 Q3

January 21, 2024

[1]:
```
%matplotlib inline

import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import norm

from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
from sklearn import tree
from matplotlib.patches import Rectangle
from matplotlib.collections import PatchCollection
from matplotlib import cm
from collections import Counter

sns.set()
```

[2]:
```
n = 1000
x = np.random.uniform(0, 1, n)
y = np.random.uniform(0, 1, n)
target = np.random.uniform(x+y,5)
# norm.pdf((x - 0.75) / 0.1) + norm.pdf((y - 0.75) / 0.1) \
#         + norm.pdf((x - 0.25) / 0.1) + norm.pdf((y - 0.25) / 0.1) \
#         + np.array(np.round(np.random.normal(-0.1,0.1, n), 2))
```

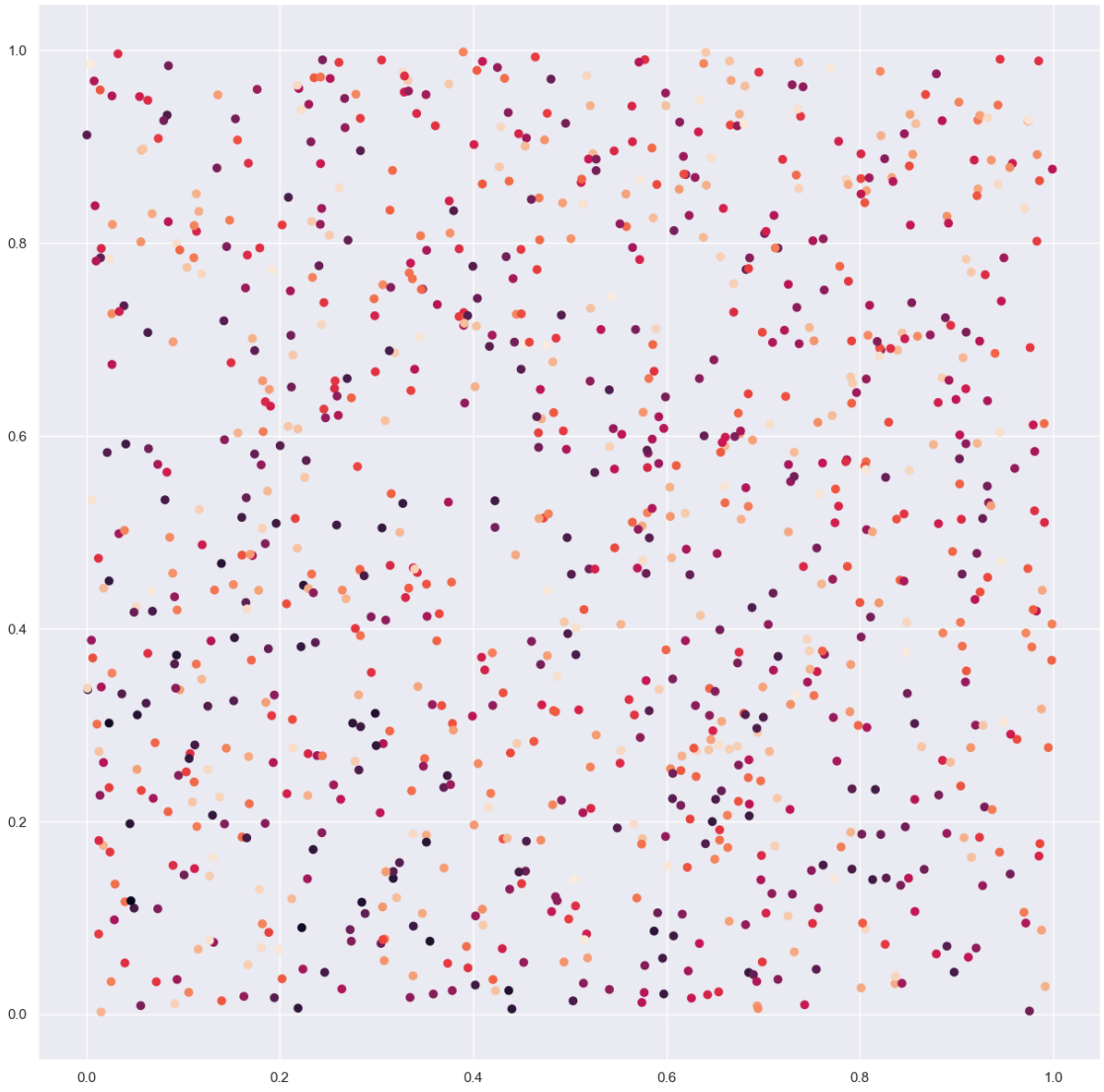
[3]:
```
a = plt.hist(target,bins=6)[1]
```



[4]:
```
a
```

[4]:
```
array([0.19834388, 0.99850434, 1.7986648 , 2.59882526, 3.39898572,
       4.19914618, 4.99930664])
```

[5]:
```
# Plot all points
fig, ax = plt.subplots(figsize = (15,15))
ax.scatter(x, y, c = target);
```



Note: Here I use a DecisionTreeRegressor instead of a DecisionTreeClassifier.
And I no longer transform the target into labels as we want to create a regression tree.

[6]:
```
data1 = pd.DataFrame({'x' : x, 'y' : y})
tree_1 = DecisionTreeRegressor(max_depth=5,min_samples_leaf = 50,max_features=0.
↪5)
tree_1.fit(data1,target)
```

[6]:
```
DecisionTreeRegressor(max_depth=5, max_features=0.5, min_samples_leaf=50)
```

[7]:
```
data1
```

[7]:
```
            x         y
0    0.623607  0.828632
1    0.032622  0.996253
2    0.929928  0.767134
3    0.761656  0.571742
4    0.665185  0.274939
..        …       …
995  0.379618  0.294699
996  0.737405  0.695517
997  0.434263  0.785849
998  0.034064  0.728959
999  0.762236  0.154673

[1000 rows x 2 columns]
```

In the following function, I adapted line 83 onwards so that now:
- The function works with target (continuous data) instead of labels (categorical data)
- It takes the average of the target in each rectangle instead of the max
- A colormap is added and a normalizing function to map the continuous values to a color

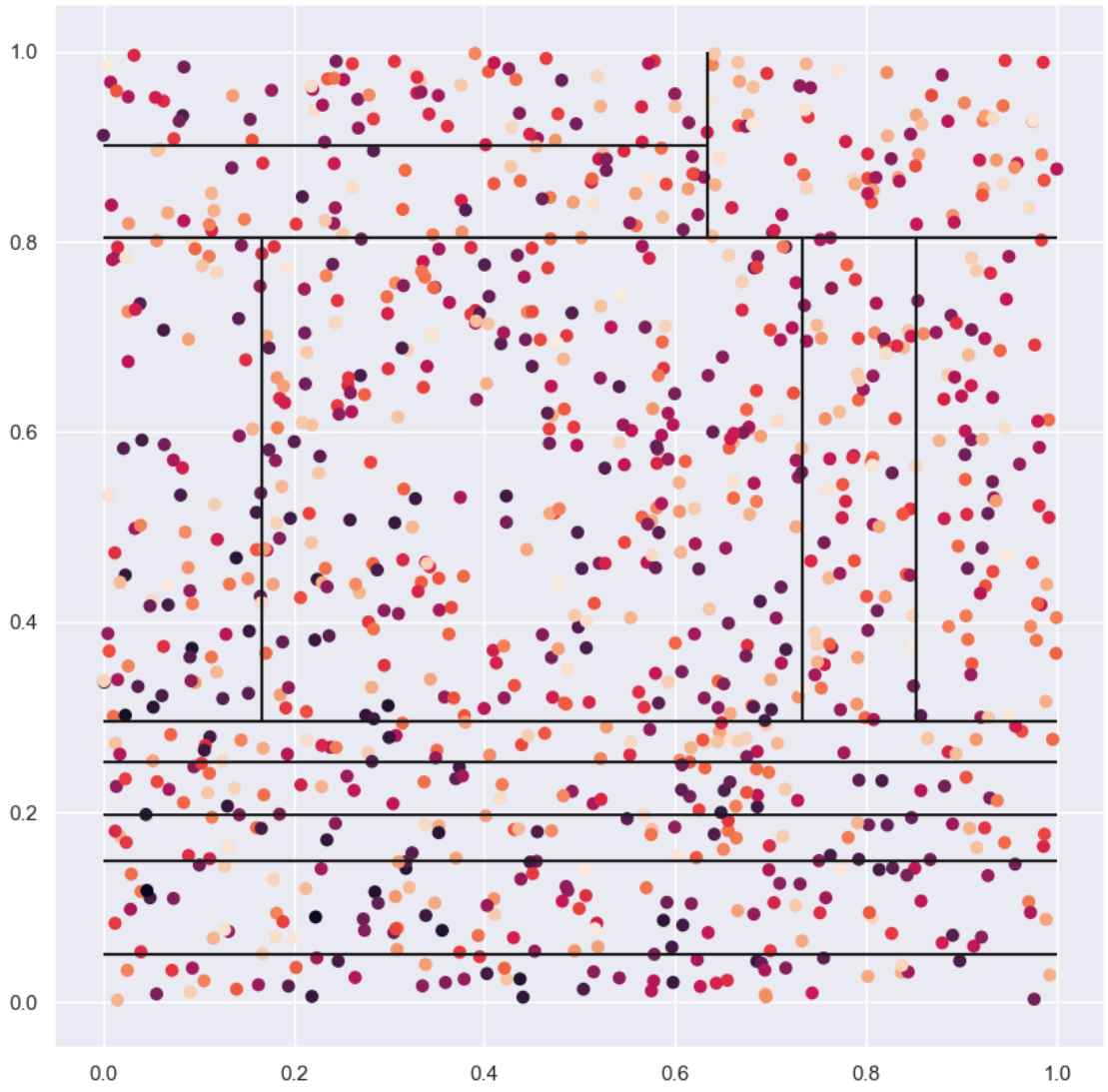
After experimenting with different colormaps, I decided to use the matplotlib "turbo" map as it is also a rainbow colormap and has some advantages for visualization purposes: https://blog.research.google/2019/08/turbo-improved-rainbow-colormap-for.html

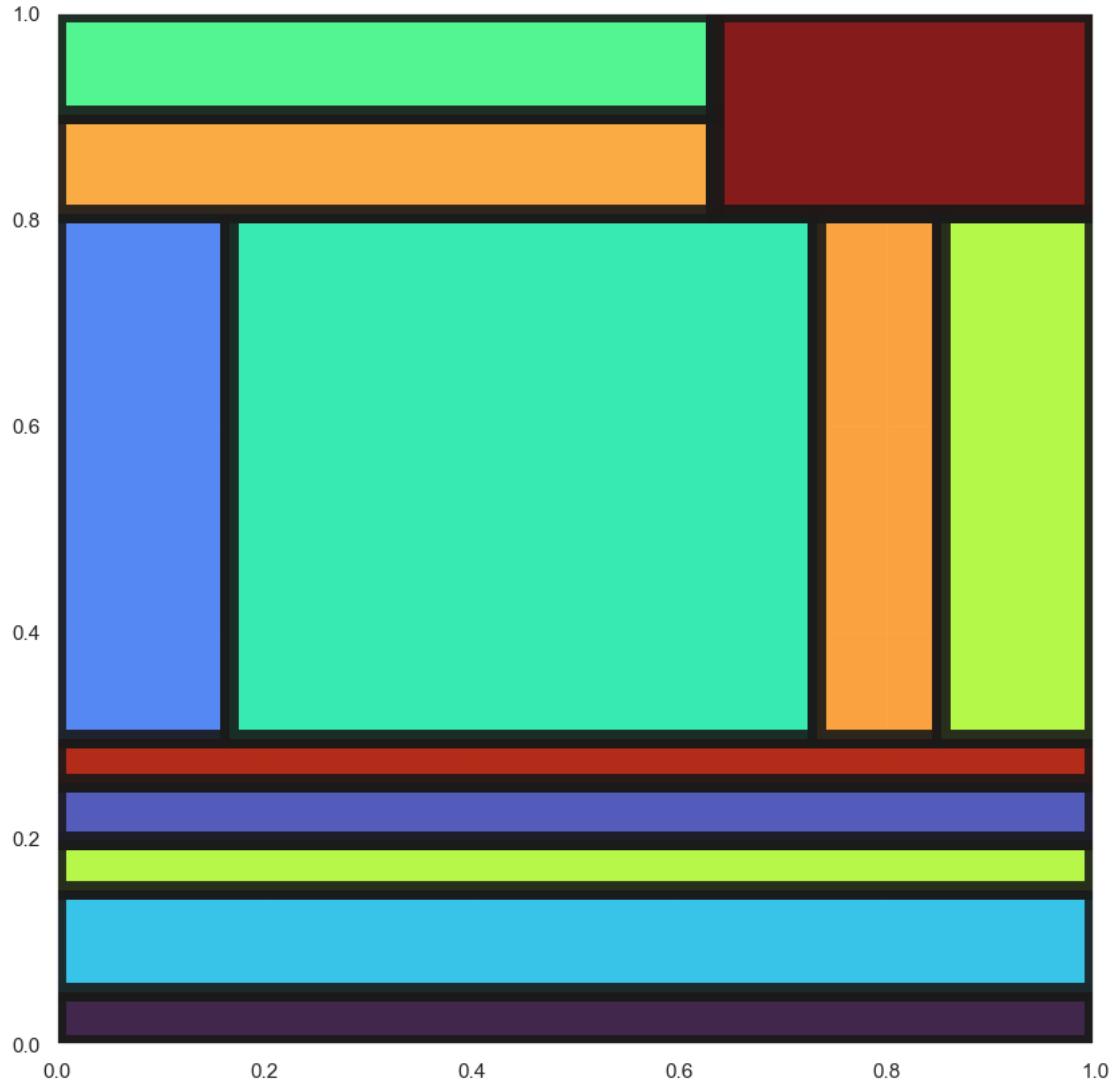
[8]:
```
def boxes(tree,data,target):

    n_nodes = tree.tree_.node_count
    children_left = tree.tree_.children_left
    children_right = tree.tree_.children_right
    feature = tree.tree_.feature
    threshold = tree.tree_.threshold

    def split(i):

        left = children_left[i]
        right = children_right[i]

        return (left,right)


    def parent(i):
        splits = enumerate([split(i) for i in range(n_nodes)])
        for a,b in splits:
            if (b[0] == i) or (b[1] == i):
                return a
            else: continue
```

```
def box(i):

    (a,b),(c,d) = (0,0),(0,0)

    if i == 0:
        (a,b) = (0,0)
        (c,d) = (1,1)
    else:
        j = parent(i)
        t = threshold[j]
        (a,b),(c,d) = box(j)

        if feature[j] == 0:
            if i == split(j)[0]:
                (a,b) = (a,b)
                (c,d) = (t,d)
            else:
                (a,b) = (t,b)
                (c,d) = (c,d)

        if feature[j] == 1:
            if i == split(j)[0]:
                (a,b) = (a,b)
                (c,d) = (c,t)
            else:
                (a,b) = (a,t)
                (c,d) = (c,d)

    return (a,b),(c,d)

boxes = []
for i in range(n_nodes):
    boxes.append(box(i))

fig, ax = plt.subplots(figsize = (10,10))
ax.scatter(x, y, c = target);



for i in range(1,n_nodes):

    j = parent(i)
    t = threshold[j]
    ((a,b),(c,d)) = boxes[j]
    if feature[j] == 0:
        ax.vlines(t, b, d, colors='k')
```

```
        else:
            ax.hlines(t,a,c,colors='k')

    leaves = [x for x in range(n_nodes) if split(x) == (-1,-1)]

    leaf_rects = []
    for leaf in leaves:
        ((a,b),(c,d)) = box(leaf)
        rect = Rectangle((a,b), c - a,d - b )
        leaf_rects.append(rect)

    rect_averages = []
    for leaf in leaves:
        data_points_in_rect = []
        for i in range(len(data1)):
            p = data1.iloc[i]
            ((a,b),(c,d)) = boxes[leaf]
            if (p['x'] > a) and (p['x'] <= c) and (p['y'] > b) and (p['y'] <=␣
↪d):
                data_points_in_rect.append(i)

        rect_averages.append(np.average(target[data_points_in_rect]))

    # Normalize range of values to colormap
    cmap = plt.get_cmap('turbo')
    norm = plt.Normalize(np.min(rect_averages), np.max(rect_averages))

    facecolor = []
    for i in range(len(leaves)):
        color = cmap(norm(rect_averages[i]))
        facecolor.append(color)

    pc = PatchCollection(leaf_rects, facecolor=facecolor, alpha=0.9,
                         edgecolor='k',linewidths = (10,))

    fig,ax = plt.subplots(figsize = (10,10))
    ax.add_collection(pc);
```

[9]:
```
boxes(tree_1,data1,target)
```


1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0



Note here we use a BaggingRegressor instead of a BaggingClassifier, and use a DecisionTreeRegressor as the base model.

[10]:
```
from sklearn.ensemble import BaggingRegressor
```

[11]:
```
bg_rgr = BaggingRegressor(DecisionTreeRegressor(min_samples_leaf=32),n_estimators=10,max_samples=0.4)
```

[12]:
```
bg_rgr.fit(data1,target)
```

[12]:
```
BaggingRegressor(estimator=DecisionTreeRegressor(min_samples_leaf=32),
                 max_samples=0.4)
```

[13]:
```
trees = bg_rgr.estimators_
```

Again, in following function, I adapted line 68 onwards so that now:
- The function works with target (continuous data) instead of labels (categorical data)
- It takes the average of the target in each rectangle instead of the max
- A colormap is added and a normalizing function to map the continuous values to a color

[14]:
```
def bagging_boxes(tree,data,labels):

    n_nodes = tree.tree_.node_count
    children_left = tree.tree_.children_left
    children_right = tree.tree_.children_right
    feature = tree.tree_.feature
    threshold = tree.tree_.threshold

    def split(i):

        left = children_left[i]
        right = children_right[i]

        return (left,right)


    def parent(i):
        splits = enumerate([split(i) for i in range(n_nodes)])
        for a,b in splits:
            if (b[0] == i) or (b[1] == i):
                return a
            else: continue



    def box(i):

        (a,b),(c,d) = (0,0),(0,0)

        if i == 0:
            (a,b) = (0,0)
            (c,d) = (1,1)
        else:
            j = parent(i)
            t = threshold[j]
            (a,b),(c,d) = box(j)

            if feature[j] == 0:
                if i == split(j)[0]:
                    (a,b) = (a,b)
                    (c,d) = (t,d)
```

```
                else:
                    (a,b) = (t,b)
                    (c,d) = (c,d)

            if feature[j] == 1:
                if i == split(j)[0]:
                    (a,b) = (a,b)
                    (c,d) = (c,t)
                else:
                    (a,b) = (a,t)
                    (c,d) = (c,d)

        return (a,b),(c,d)

    boxes = []
    for i in range(n_nodes):
        boxes.append(box(i))

    leaves = [x for x in range(n_nodes) if split(x) == (-1,-1)]

    leaf_rects = []
    for leaf in leaves:
        ((a,b),(c,d)) = box(leaf)
        rect = Rectangle((a,b), c - a,d - b )
        leaf_rects.append(rect)

    rect_averages = []
    for leaf in leaves:
        points_in_rect = []
        for i in range(len(data1)):
            p = data1.iloc[i]
            ((a,b),(c,d)) = boxes[leaf]
            if (p['x'] > a) and (p['x'] <= c) and (p['y'] > b) and (p['y'] <=␣
↪d):
                points_in_rect.append(i)
        rect_averages.append(np.average(target[points_in_rect]))

     # Normalize range of values to colormap
    cmap = plt.get_cmap('turbo')
    norm = plt.Normalize(np.min(rect_averages), np.max(rect_averages))

    facecolor = []
    for i in range(len(leaves)):
        color = cmap(norm(rect_averages[i]))
        facecolor.append(color)
```

```
    pc = PatchCollection(leaf_rects, facecolor=facecolor, alpha=0.1,
                         edgecolor='k',linewidths = (2,))

    ax.add_collection(pc);
```

[15]:
```
fig, ax = plt.subplots(figsize = (10,10))

for tree in trees:
    bagging_boxes(tree,data1,target)
```