COMPSCI 371D Homework 8

Problem 0 (3 points)

Part 1: Impurity

Problem 1.1 (Exam Style)

$$p(class1|S) = 21/140$$
$$p(class2|S) = 0$$
$$p(class3|S) = 105/140$$
$$p(class4|S) = 14/140$$

In [39]:
```
import numpy as np
sum = 0
for i in [21/140, 0, 105/140, 14/140]:
    sum+=i**2
print("Gini index: " + str(1-sum))
```

```
Gini index: 0.405
```

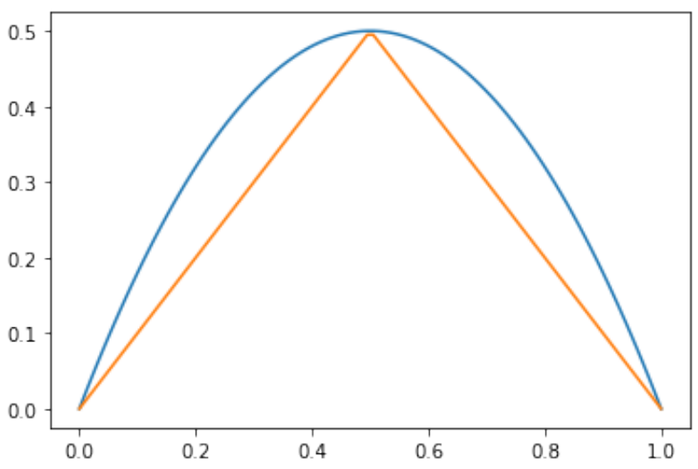
In [36]:
```
print("Error-rate impurity: " + str(1-105/140))
```

```
Error-rate impurity: 0.25
```

Problem 1.2 (Exam Style)

In [27]:
```
from matplotlib import pyplot as plt
p = np.linspace(0,1,100)
gini = 1-(p**2+(1-p)**2)
error_rate = list(map(lambda p: 1-np.max([p, 1-p]),p))
plt.plot(p, gini)
plt.plot(p, error_rate)
```

Out[27]:
```
[<matplotlib.lines.Line2D at 0x7f920ee289d0>]
```



Part 2: Decision Trees as Partitions

In [28]:
```
from urllib.request import urlretrieve
from os import path as osp
import pickle
import numpy as np
from matplotlib import pyplot as plt
from matplotlib import cm
from matplotlib.colors import ListedColormap
from matplotlib.patches import Rectangle
%matplotlib inline
```

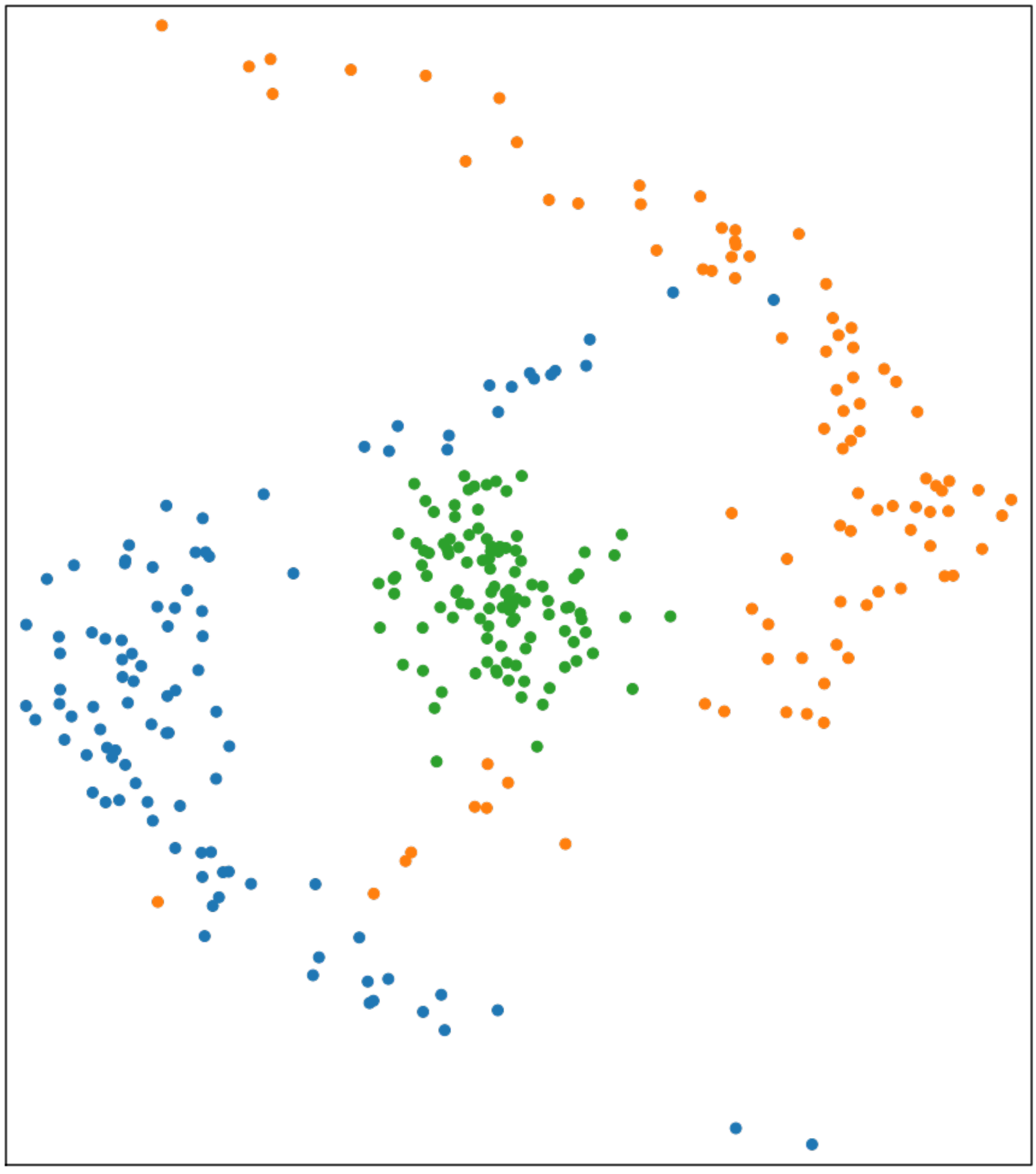
In [29]:
```
def bounding_box(xs, margin=0.5):
    mn, mx = np.min(xs, axis=0) - margin, np.max(xs, axis=0) + margin
    return {'left': mn[0], 'right': mx[0], 'bottom': mn[1], 'top': mx[1]}


def shade_box(box, color, alpha=0.2):
    pale_color = color.copy()
    pale_color[3] = alpha
    corner = box['left'], box['bottom']
    width, height = box['right'] - corner[0], box['top'] - corner[1]
    rectangle = Rectangle(corner, width, height,
                          edgecolor='none', facecolor=pale_color)
    plt.gca().add_patch(rectangle)
```

In [30]:
```
def retrieve(file_name, semester='fall21', course='371d', homework=8):
    if osp.exists(file_name):
        print('Using previously downloaded file {}'.format(file_name))
    else:
        fmt = 'https://www2.cs.duke.edu/courses/{}/compsci{}/homework/{}/{}'
        url = fmt.format(semester, course, homework, file_name)
        urlretrieve(url, file_name)
        print('Downloaded file {}'.format(file_name))
```

In [31]:
```
def plot_data(data):
    box = bounding_box(data['x'])
    plt.figure(figsize=(15, 15), tight_layout=True)
    plt.plot((box['left'], box['right'], box['right'], box['left'], box['left']),
             (box['bottom'], box['bottom'], box['top'], box['top'], box['bottom']), 'k')
    colormap = ListedColormap(cm.tab10(range(len(np.unique(data['y'])))))
    plt.scatter(data['x'][:, 0], data['x'][:, 1], s=80, c=data['y'], cmap=colormap)
    plt.axis('equal')
    plt.axis('off')
    return box, colormap.colors
```

In [32]:
```
small_set_name = 'small_set.pickle'
retrieve(small_set_name)
with open(small_set_name, 'rb') as file:
    small_set = pickle.load(file)
plot_data(small_set)
plt.show()
```

```
Downloaded file small_set.pickle
```

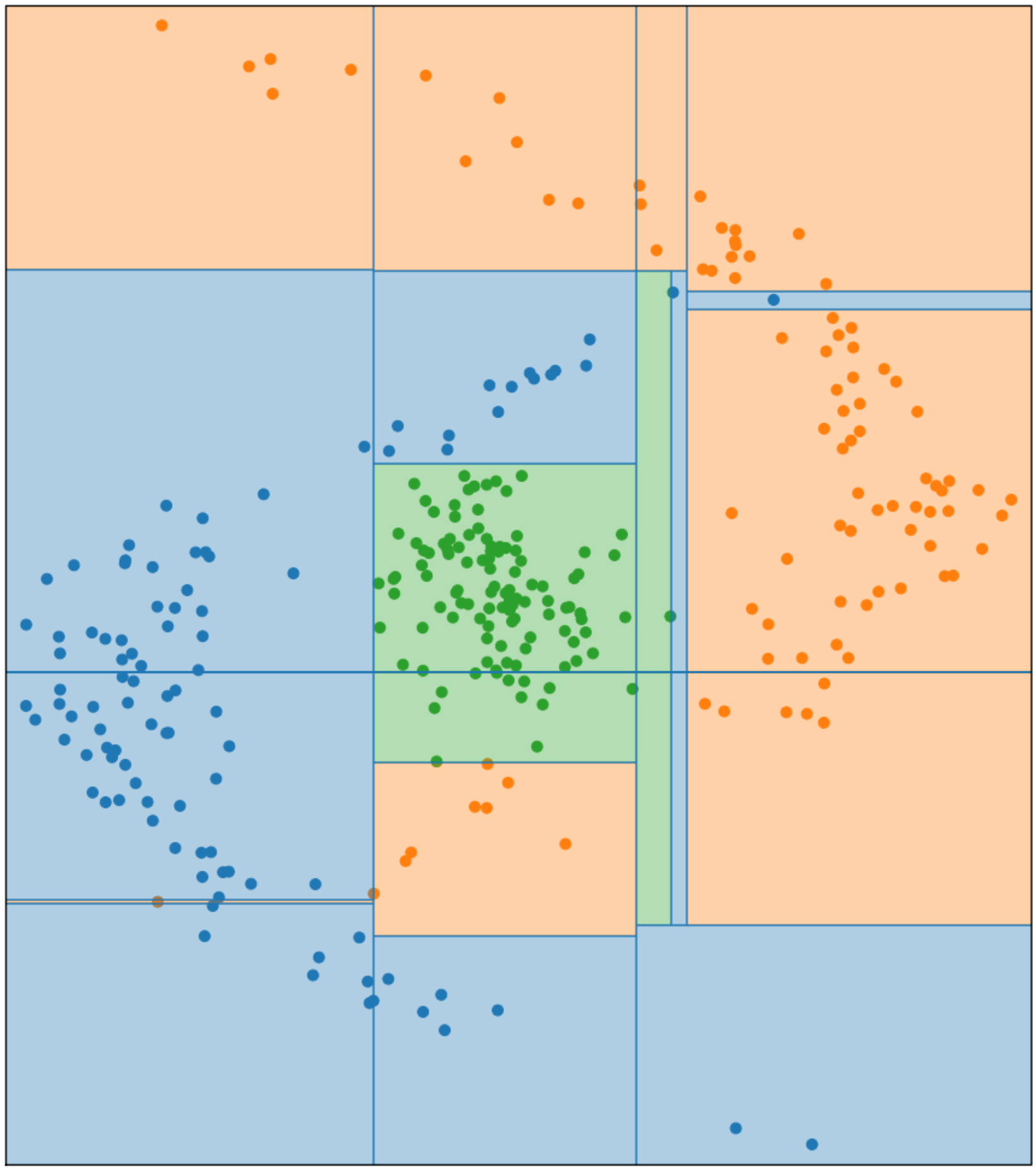


Problem 2.1

In [139]:
```
def draw_tree(t, b, colors):
    def draw(node, box):
        if (t.feature[node] == 0): #vertical line
            plt.vlines(t.threshold[node],box['bottom'],box['top'])
            boxL = replace_side(box, 'right', t.threshold[node])
            boxR = replace_side(box, 'left', t.threshold[node])
        else: #horizontal line
            plt.hlines(t.threshold[node], box['left'], box['right'])
            boxL = replace_side(box, 'top', t.threshold[node])
            boxR = replace_side(box, 'bottom', t.threshold[node])

        if (t.children_left[node] >= 0): draw(t.children_left[node], boxL)
        else: shade_box(box, colors[np.argmax(t.value[node])])
        if (t.children_right[node] >= 0): draw(t.children_right[node], boxR)
        else: shade_box(box, colors[np.argmax(t.value[node])])
    draw(0,b)
```

In [99]:
```
def replace_side(box, side, value):
    new = box.copy()
    new[side] = value
    return new
```

In [140]:
```
from sklearn.tree import DecisionTreeClassifier as DTC
dtc = DTC()
h = dtc.fit(small_set['x'],small_set['y'])
t = h.tree_
bx, class_colors = plot_data(small_set)
draw_tree(t,bx,class_colors)
plt.show()
```



Problem 2.2 (Exam Style)

There are obvious places in the drawing where the tree overfits, for example, the orange slice on the bottom left of the graph where the slice is orange because of one orange sample amidst majority blue samples. We can see more of these overfitted examples on the right side of the plot where due to the plot always seeking out lowest gini index, it goes as far as slicing out a horizontal blue region admist all orange samples due to one blue sample, ensuring each sample is classified correctly during training.

Part 3: Trees and Forests

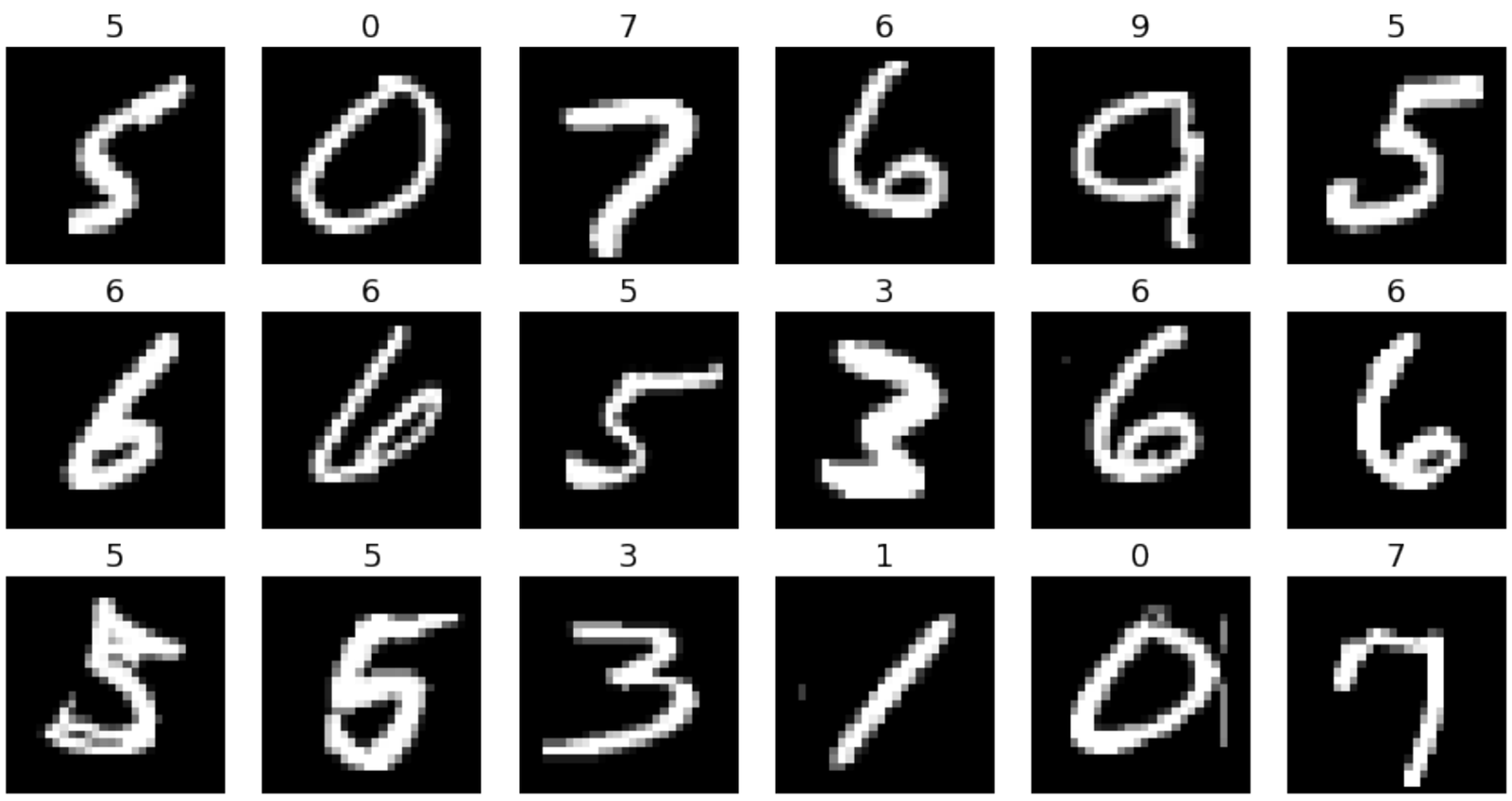
In [148]:
```
from torchvision.datasets import MNIST
from sklearn.preprocessing import StandardScaler
```

In [149]:
```
def standardize(data, stats=None):
    if stats is None:
        scaler = StandardScaler().fit(data)
        data = scaler.transform(data).astype(np.float32)
        return data, {'mean': scaler.mean_, 'std': scaler.scale_}
    else:
        data -= stats['mean']
        data /= stats['std']
        return data
```

In [150]:
```
def load_mnist():
    data = {}
    for which in ('train', 'test'):
        is_train = True if which == 'train' else False
        d = MNIST('.', train=is_train, download=True)
        ds = {'x': d.data.numpy(), 'y': d.targets.numpy()}
        n, shape = ds['x'].shape[0], ds['x'].shape[1:]
        ds['x'] = ds['x'].reshape((n, -1)).astype(np.float32)
        ds['y'] = ds['y'].astype(np.uint8)
        data[which] = ds

    data['train']['x'], stats = standardize(data['train']['x'])
    stats['max'], stats['shape'] = np.max(data['train']['x']), shape
    data['test']['x'] = standardize(data['test']['x'], stats)

    return data, stats
```

In [151]:
```
def x_to_image(x, stats):
    x = np.round(x * stats['std'] + stats['mean'])
    x = np.clip(x * 255. / stats['max'], 0., 255.).astype(np.uint8)
    return np.reshape(x, stats['shape'])


def show_random_training_images(data, stats, rows=3, columns=6):
    xs, ys = data['train']['x'], data['train']['y']
    rng = np.random.default_rng()
    indices = rng.integers(low=0, high=len(ys), size=rows * columns)
    plt.figure(figsize=(2 * columns, 2.1 * rows), tight_layout=True)
    for plot, index in enumerate(indices):
        image = x_to_image(xs[index], stats)
        plt.subplot(rows, columns, plot + 1)
        plt.imshow(image, cmap='gray')
        plt.axis('off')
        plt.title(ys[index], fontsize=18)
    plt.show()
```

In [152]:
```
digits, image_stats = load_mnist()
show_random_training_images(digits, image_stats)
```



Problem 3.1

In [273]:
```
from sklearn.tree import DecisionTreeClassifier as DTC
digitstree = DTC()
h = digitstree.fit(digits['train']['x'], digits['train']['y'])
```

In [214]:
```
def evaluate(h, data):
    def error_rate(predictor, samples):
        x, y = samples['x'], samples['y']
        return (1 - predictor.score(x, y)) * 100

    e_train = error_rate(h, data['train'])
    e_test = error_rate(h, data['test'])
    return e_train, e_test
```

In [235]:
```
from sklearn.metrics import confusion_matrix
def evaluate_classifiers(data, h, classify_string):
    train_eval, test_eval = evaluate(h, data)
    print("\n Error statistics for the " + classify_string + " classifier (percent):")
    print("Training: " + str(round(train_eval,3)))
    print("Testing: " + str(round(test_eval,3)))
    print("Confusion Matrix:")
    print(confusion_matrix(data['test']['y'], h.predict(data['test']['x'])))
```

In [263]:
```
%time evaluate_classifiers(digits, h, "Decision Tree Classifier")
```

```
 Error statistics for the Decision Tree Classifier classifier (percent):
Training: 0.0
Testing: 12.23
Confusion Matrix:
[[ 914    0   11    8    6   12   10    3    9    7]
 [   3 1088    9    9    4    5    6    4    5    2]
 [  13    8  889   32   11   14   11   26   20    8]
 [   7    7   28  860   10   49    3    7   22   17]
 [   5    3   10    6  858    4   18   10   24   44]
 [  15    7    3   43    6  746   26    4   23   19]
 [  17    2   12   10   16   21  847    1   24    8]
 [   4   10   24   18    6    6    1  931    7   21]
 [  13    7   26   38   22   31   19    6  782   30]
 [  11    2    8   24   43    8    6   21   24  862]]
CPU times: user 87.2 ms, sys: 1.77 ms, total: 89 ms
Wall time: 88 ms
```

Problem 3.2 (Exam Style)

Pair: (3, 5); for numbers known to be 3, it is wrongly classified as 5 for 49 times; for number known to be 5, it is wrongly classified as 3 for 43 times. I think the confusion matrix here is roughly symmetrical, most of the opposing digits are within 5 from each other.

The decision tree does not underfit: training risk is zero so it is classifying all training samples correctly.

The decision tree does overfit: we can tell from the zero training risk. The reason might be that the decision tree algorithm always goes for the optimal threshold and dimensionality that give the lowest gini index, which would lead to splitting the dataset until it cannot be more pure, classifiying each training sample correctly lead to zero impurity.

Problem 3.3

In [229]:
```
from sklearn.ensemble import RandomForestClassifier as RFC
randomForest = RFC(n_estimators = 100, oob_score = True)
r = randomForest.fit(digits['train']['x'], digits['train']['y'])
```

In [247]:
```
%time evaluate_classifiers(digits, r, "Random Forest Classifier")
```

```
 Error statistics for the Random Forest Classifier classifier (percent):
Training: 0.0
Testing: 3.05
Confusion Matrix:
[[ 969    0    0    0    0    4    3    1    3    0]
 [   0 1124    3    3    0    1    3    0    1    0]
 [   7    0  998    7    3    0    3    8    6    0]
 [   0    0    9  975    0    7    0    8    9    2]
 [   1    0    1    0  957    0    6    0    2   15]
 [   3    0    0   12    2  858    7    2    6    2]
 [   7    3    1    0    3    4  937    0    3    0]
 [   2    4   19    2    1    0    0  987    2   11]
 [   3    0    6   11    2    6    2    4  931    9]
 [   7    5    2   12   12    1    1    4    6  959]]
CPU times: user 3.41 s, sys: 545 ms, total: 3.95 s
Wall time: 4.41 s
```

In [245]:
```
print("Out of Bag Error Rate: " + str(round((1-r.oob_score_)*100,3)))
```

```
Out of Bag Error Rate: 3.537
```

Problem 3.4 (Exam Style)

The pair of digits that correspond(s) to the largest nondiagonal entry of the confusion matrix is 7 and 2. The matrix doesn't look symmetric, the lower half of the matrix under the large diagonal numbers seem to have larger values This means that the larger numbers are misclassified more often.

The random forest does not underfit.

The random forest overfits less than the decision tree, we can deduce this from the lower testing error.

The out of bag error rate can act as an unbiased estimate of the random forest's statistical risk, and it is sufficiently close to the testing rate. So it is a reasonable estimate of the test error rate.

Problem 3.5 (Exam Style)

Running time ratio (random forest:decision tree): 3.95/(0.001*91) = 43.41

This ratio would be lower just for training on this dataset, roughly n/784*100 : n, or 0.12, where n is runtime of decision tree, and 784 is the number of total dimensions of the dataset. We divide by 784 because random decision forest skips the loop over all dimensions that the decision tree does, and then multiply by 100 because there are 100 trees in the forest.

The time taken to train the random forest much less than 100 times the time taken to train a single tree because the random forest isn't looping through all possible dimensions and thresholds to find the optimal parameters that yield the least gini index; picking a random set of parameters is much faster resulting in the time difference. In addition, bagging results in repeated datapoints which decreases the depth of the tree.