Wajid Ghafoor

Benjamin Ostendorf

In [2]:
```
# Imports needed for Exercise 5
%pylab inline
import numpy as np
import matplotlib.pyplot as plt
import math as m
import random
%pylab inline
```

```
Populating the interactive namespace from numpy and matplotlib
```

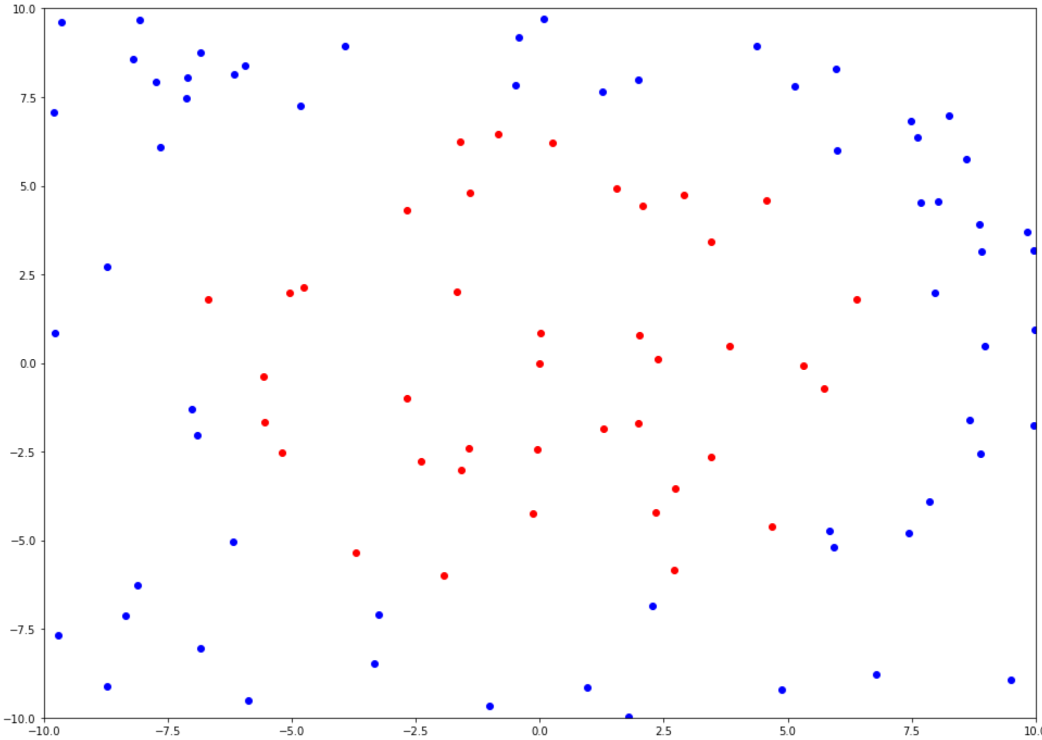
In [3]:
```
# We read the raw data of dataCircles
with open('dataCircle.txt') as f:
    read_data = f.read()
f.close()
#raw_data = [data_line.rstrip().split() for data_line in read_data ]
#data = [tuple(map(lambda x: float(x), line)) for line in raw_data]
```

In [4]:
```
# Bringin Data into correct shape for the ongoing Algorithmn
read_data_string = read_data.split('\r')
data_circle_string= list(map(lambda x: x.split(), read_data_string))
```

$$D = \{(x_i, y_i) : x_i \in R^d, y_i \in \{-1, +1\}, i = 1, 2, \ldots, m\} m, d = (102, 2)$$

In [ ]:
```
# We want the classes of {-1,+1}. The +1 is already
data_circle = []
for liste in data_circle_string:
    data_circle.append(list(map(lambda x: float(x), liste)))
    if data_circle[-1][2] == 0.0:
        data_circle[-1][2] = -1.0
```

In [6]:
```
# We made sure that we got the right representation of the Data by comparing
it to the erxercise sheet
#% matplotlib
fig = plt.figure(figsize=(36,12))
ax1 = fig.add_subplot(121)
ax1.plot([data_x[0] for data_x in data_circle[0:40]], [data_y[1] for data_y
in data_circle[0:40]], "ro")
ax1.plot([data_x[0] for data_x in data_circle[40:]], [data_y[1] for data_y i
n data_circle[40:]], "bo")
ax1.axis([-10, 10, -10, 10])

plt.show()
```



$$h_t : \mathbb{R}^d \rightarrow \{-1, +1\}$$

In [248]:
```
def get_weak_classifier(data):
    axis = random.choice([0, 1])
    val = random.uniform(-10, 10)
    corrects = 0
    faults = 0
    for point in data:
        if point[axis] < val and point[2] == 1:
            corrects += 1
        elif point[axis] >= val and point[2] == -1:
            corrects += 1
        else:
            faults += 1
    if corrects > faults:
        return [lambda point: 1.0 if point[axis] < val else -1.0, axis, val,
corrects/float(len(data))]
    else:
        return [lambda point: -1.0 if point[axis] < val else 1.0,  axis, val
,  faults/float(len(data))]
```

In [249]:
```
def error_of_weak_classifier(distribution, data, classifier):
    result = 0
    for index in range(len(data)):
        if classifier[0](data[index]) != data[index][2]:
            result += distribution[index]
    return result
```

In [250]:
```
def update_distribution(distribution, data, classifier):
    #print(error_of_weak_classifier(distribution, data, classifier))
    alph = alpha(error_of_weak_classifier(distribution, data, classifier))
    for i in range(len(data)):
        distribution[i] = 1/float(z(distribution, data, alph, classifier)) *
distribution[i] * m.exp(-alph * data[i][2] * classifier[0](data[i]))
    return distribution
```

In [251]:
```
def z(distribution, data, alpha, classifier):
    corrects = []
    faults = []
    for i in range(len(data)):
        if classifier[0](data[i]) == data[i][2]:
            corrects.append([data[i], i])
        else:
            faults.append([data[i], i])
    sum1 = 0
    for point in corrects:
        sum1 += distribution[point[1]]*m.exp(-alpha)
    sum2 = 0
    for point in faults:
        sum2 += distribution[point[1]]*m.exp(alpha)
    return sum1 + sum2
```

In [252]:
```
def alpha(error_b, error):
    return 0.5 * (m.log((1-error_b)/float(error)))
```

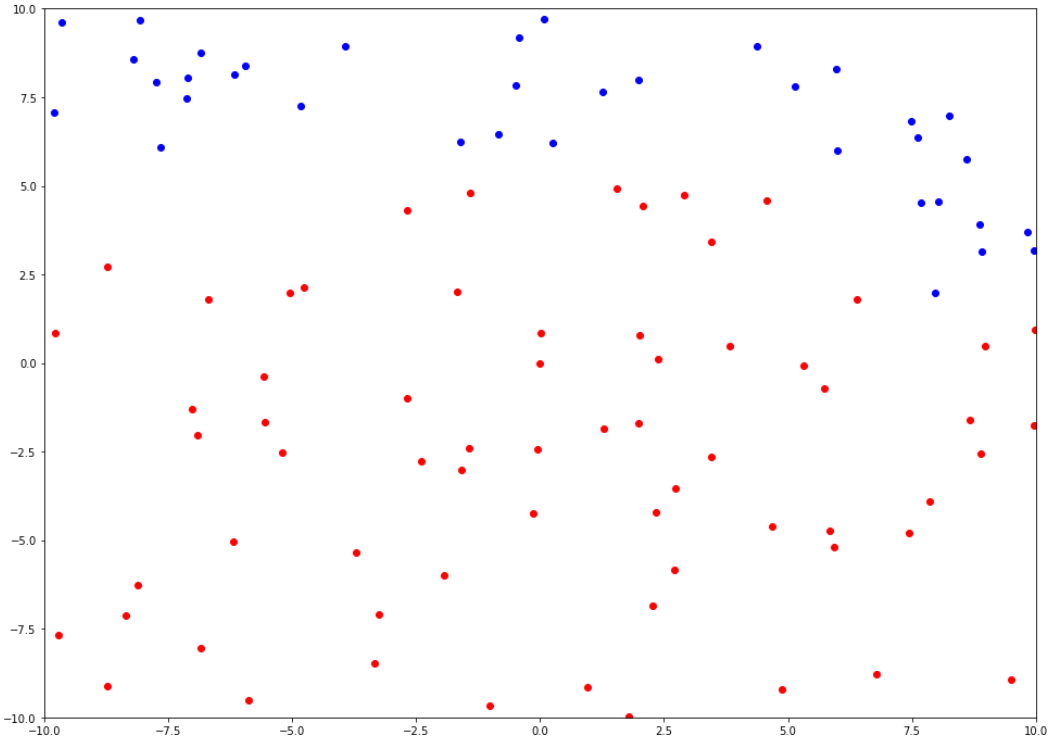
In [253]:
```
def ada_boost(list_of_classifiers, data):
    distribution = [1/float(len(data)) for i in range(len(data))]
    error_list=[]
    for i in range(100):
        error_list = list(map(lambda x: error_of_weak_classifier(distributio
n, data, x), list_of_classifiers))
        min_error_classifier_index = error_list.index(min(error_list))
        distribution = update_distribution(distribution, data, list_of_class
ifiers[min_error_classifier_index])
    result = []
    for i in range(len(error_list)):
        result.append([alpha(error_list[i]), list_of_classifiers[i]])
    return result
```

In [254]:
```
def classify(point, strong_classifier):
    result = 0
    for weighted_classifier in strong_classifier:
        result += weighted_classifier[0]*weighted_classifier[1][0](point)
    return np.sign(result)
```

In [255]:
```
list_of_cla = [get_weak_classifier(data_circle) for i in range(100)]
classifiers = ada_boost(list_of_cla, data_circle)
```

In [241]:
```
classified_point = []
for point in data_circle:
    classified_point.append(classify(point, classifiers))
```

In [242]:
```
#Plot of the predicted point using Ada boost
#% matplotlib
fig = plt.figure(figsize=(36,12))
ax1 = fig.add_subplot(121)
for point in range(len(classified_point)):
    if classified_point[point] == 1.0:
        ax1.plot([data_circle[point][0]], [data_circle[point][1]], "ro")
    else:
        ax1.plot([data_circle[point][0]], [data_circle[point][1]], "bo")
ax1.axis([-10, 10, -10, 10])

plt.show()
```



In [287]:
```
def empiric_error_rate(points_pos,points_neg):
    er = 0
    for point in points_pos:
        if point == -1:
            er += 1
    for point in points_neg:
        if point == 1:
            er += 1
    return (1.0/102) * er
```

In [290]:
```
empiric_error_rate(classified_point[0:40],classified_point[40:])
```

Out[290]: 0.3137254901960784

Above is our first attempt. Unfortunatly there is a problem that after some updates of the distribution some of the classifiers yield errors > 0.5 which leads to negative alpha values.

Second attempt to solve this Exercise.

In [11]:
```
rng = np.random.RandomState(0)
```

In [5]:
```
XY = np.asarray(data_circle)
X = XY[:,:2]
Y = np.squeeze(XY[:,2:], axis=1)
D = np.ones_like(np.ones(102)) / len(X)
```

In [12]:
```
ind = rng.choice(102, size=(102, ), p=D)
ind.sort()
X_ = X[ind]
Y_ = Y[ind]
```

In [40]:
```
predictor_1 = lambda x: 1 if x > 0 else -1
output_1 = np.asanyarray(map(predictor_1,X[:,1:]))
is_correct = np.asanyarray(output_1==Y).astype(np.float)
```

$0.5 \log \frac{1-\epsilon^*}{\epsilon^*}$

In [44]:
```
eps_1 = ((1.0 - is_correct) * D).sum()
alpha_1 = 0.5 * np.log( (1-eps_1) / eps_1 )
```

In [45]:
```
print("error:", eps_1, "weight:", alpha_1)
```

```
('error:', 0.4168799915271065, 'weight:', 0.16779731066390552)
```

Sample weight update equation: $D_{t+1}(x) \leftarrow D_t(x) * \exp(\pm\alpha_t)$ if it is correct it gives a - and reduce the weight by this process

In [23]:
```
sample_signs_1 = (0.5 - is_correct) * 2.0
D = D * np.exp(sample_signs_1 * alpha_1)
```

In [539]:
```
#Normalisation because D.sum() is not 1!
D /= D.sum()
```

Everything above was to understand the single steps of the Algorithmn.

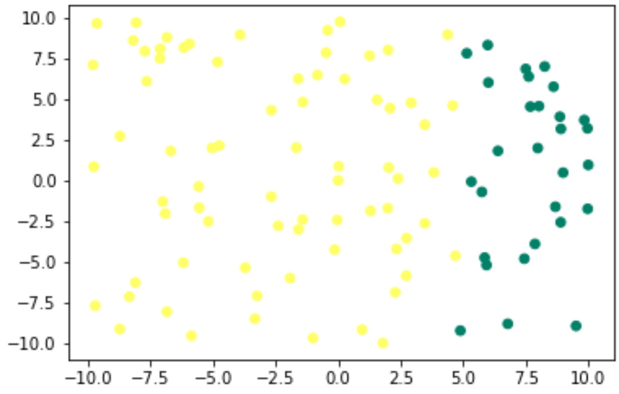
In [47]:
```
# We read the raw data of dataCircles
with open('dataCircle.txt') as f:
    read_data = f.read()
f.close()
#raw_data = [data_line.rstrip().split() for data_line in read_data ]
#data = [tuple(map(lambda x: float(x), line)) for line in raw_data]

# Bringin Data into correct shape for the ongoing Algorithmn
read_data_string = read_data.split('\r')
data_circle_string= list(map(lambda x: x.split(), read_data_string))

data_circle = []
for liste in data_circle_string:
    data_circle.append(list(map(lambda x: float(x), liste)))

XY = np.asarray(data_circle)
X = XY[:,:2]
Y = np.squeeze(XY[:,2:], axis=1)
D = np.ones_like(np.ones(102)) / len(X)
```

In [48]:
```
class FlatCut:
    def __init__(self):
        self.mode = 'Undetermined'
        self.th = None
        #gt for grater than
    def predict(self, data):
        if self.mode == 'horizontal_gt':
            pred = data[:, 0] >= self.th
        elif self.mode == 'horizontal_lt':
            pred = data[:, 0] < self.th
        elif self.mode == 'vertical_gt':
            pred = data[:, 1] >= self.th
        elif self.mode == 'vertical_lt':
            pred = data[:, 1] < self.th
        else:
            assert False, "Unknown mode"

        return pred.astype(np.float)

    def fit(self, data, targets):
        xmin, xmax = data[:,0].min(), data[:,0].max()
        ymin, ymax = data[:,1].min(), data[:,1].max()
        best_th = None
        best_mode = None
        best_accuracy = 0

        for self.mode in ['horizontal_gt', 'horizontal_lt']:
            for self.th in np.linspace(xmin, xmax,100):
                accu = np.count_nonzero(self.predict(data) ==targets) / floa
t(targets.size)
                if accu > best_accuracy:
                    best_mode = self.mode
                    best_th = self.th
                    best_accuracy = accu

        for self.mode in ['vertical_gt', 'vertical_lt']:
            for self.th in np.linspace(ymin, ymax,100):
                accu = np.count_nonzero(self.predict(data) ==targets) / floa
t(targets.size)
                if accu > best_accuracy:
                    best_mode = self.mode
                    best_th = self.th
                    best_accuracy = accu

        self.th = best_th
        self.mode = best_mode
        print("mode", self.mode, "Threshhold", self.th)
```

In [49]:
```
#First weak classifier. This is what we get!
wc = FlatCut()
wc.fit(X,Y)
p = wc.predict(X)
scatter(X[:,0], X[:,1], c= p, s=32, cmap='summer')
```

```
('mode', 'horizontal_lt', 'Threshhold', 4.794567070707069)
```

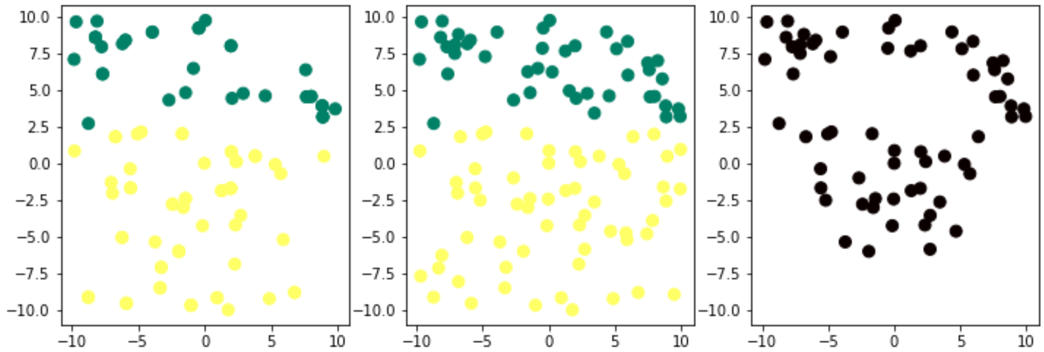
Out[49]:
```
<matplotlib.collections.PathCollection at 0x7f4ad07d75d0>
```



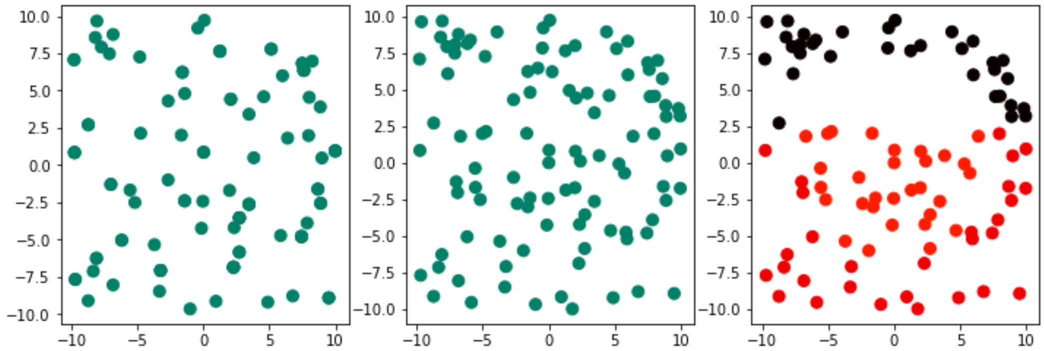
In [51]:
```
def ensamble_pred(X, ensemble):
    N = X.shape[0]
    A0  =np.zeros(N)
    A1  =np.zeros(N)
    for p, a in ensemble:
        pred = p.predict(X)
        A0[pred==0] +=a
        A1[pred==1] +=a
    return (A1>A0).astype(np.float)
```

In [72]:
```
#Adaboost
rng = np.random.RandomState(0)
N = X.shape[0]
subsample_size = N
T = 50
D = np.ones(N)/X.shape[0] # initial distribution
ind = rng.choice(X.shape[0], size=(subsample_size,), p=D)
X_ = X[ind]
y_ = Y[ind]
ensemble = []
for t in range(T):
    week_pred_t = FlatCut()
    week_pred_t.fit(X_,y_)

    pred = week_pred_t.predict(X)
    errors = (pred!= Y).astype(np.float)
    error_w = errors * D
    eps_t = max(error_w.sum(), 1e-6)


    alpha = 0.5 * np.log((1-eps_t) / eps_t)
    print(alpha)
    D *= np.exp( (errors - 0.5) * 2.0 * alpha)
    D /= D.sum()

    ensemble.append((week_pred_t,alpha))
    #visualise
    figure(1, (12,4))
    subplot(1,3,1)
    scatter(X_[:,0], X_[:,1], c=week_pred_t.predict(X_), cmap='summer', s=64
)
    subplot(1,3,2)
    scatter(X[:,0], X[:,1], c=ensamble_pred(X, ensemble), cmap='summer', s=6
4)
    subplot(1,3,3)
    scatter(X[:,0], X[:,1], c=D, cmap='hot', s=64)
    show()

    ind = rng.choice(N, size=(subsample_size,), p=D)
    X_ = X[ind]
    y_ = Y[ind]
```

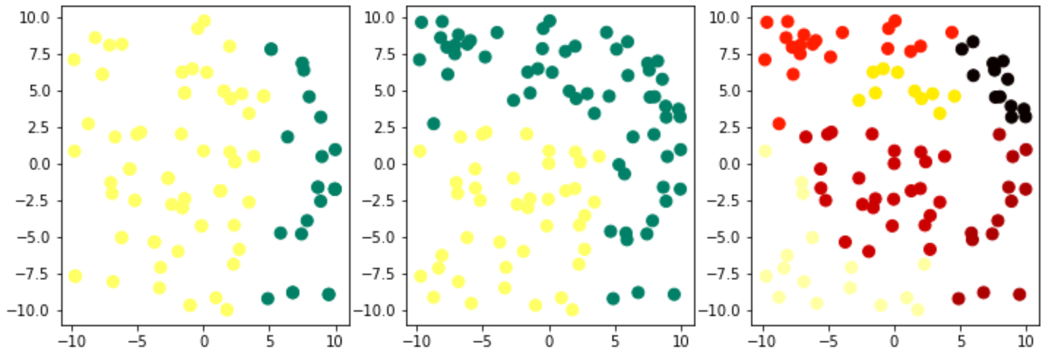
```
('mode', 'vertical_lt', 'Threshhold', 2.1635151515151527)
0.23978654013094314
```



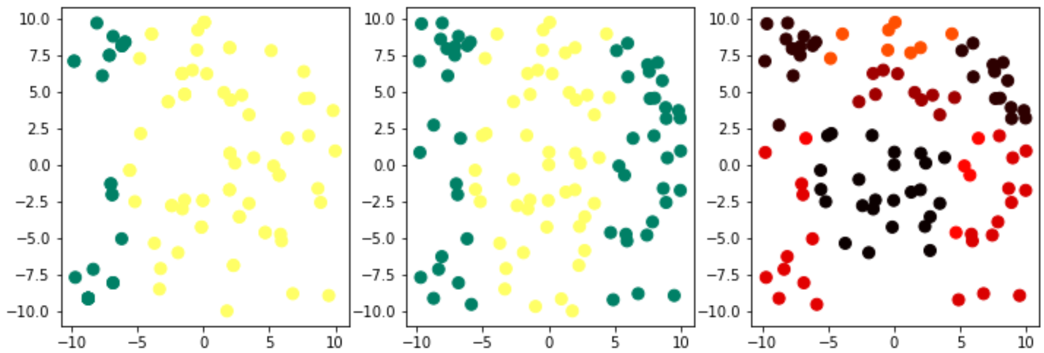
```
('mode', 'horizontal_lt', 'Threshhold', -9.79845)
0.2740607042548438
```



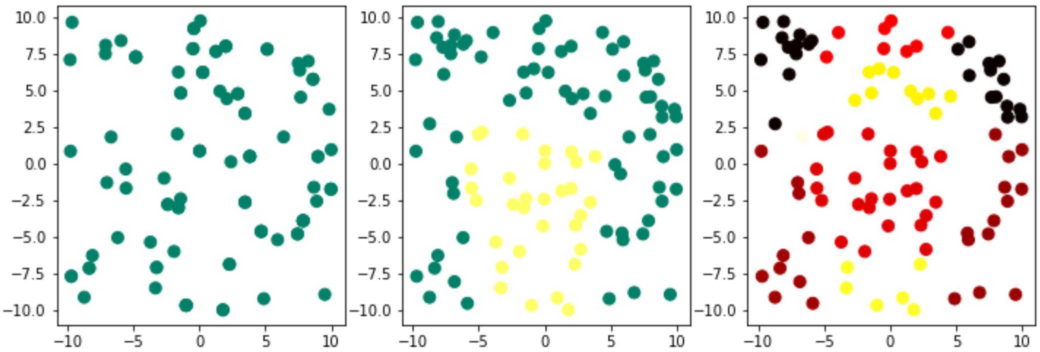
```
('mode', 'horizontal_lt', 'Threshhold', 4.594662727272727)
0.35313424285057937
```



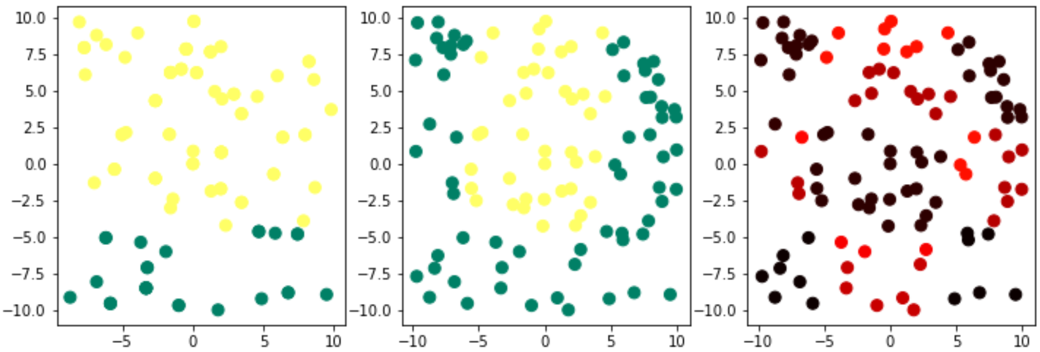
```
('mode', 'horizontal_gt', 'Threshhold', -5.800363131313132)
0.343442933772187
```



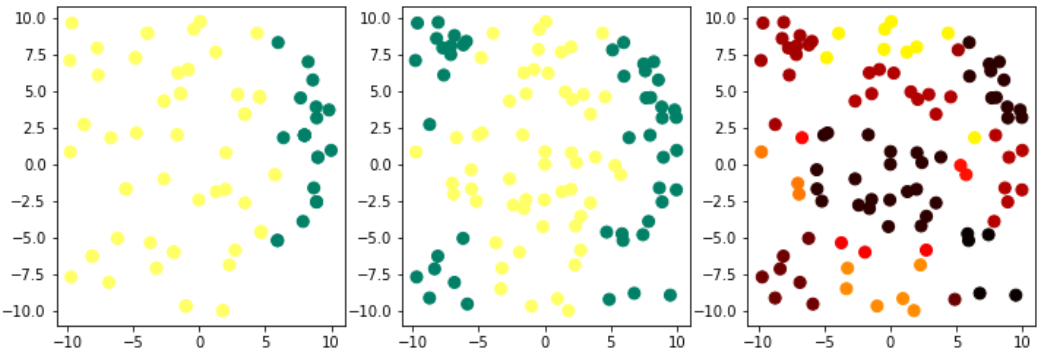
```
('mode', 'horizontal_lt', 'Threshhold', -9.79845)
0.39637012290624146
```



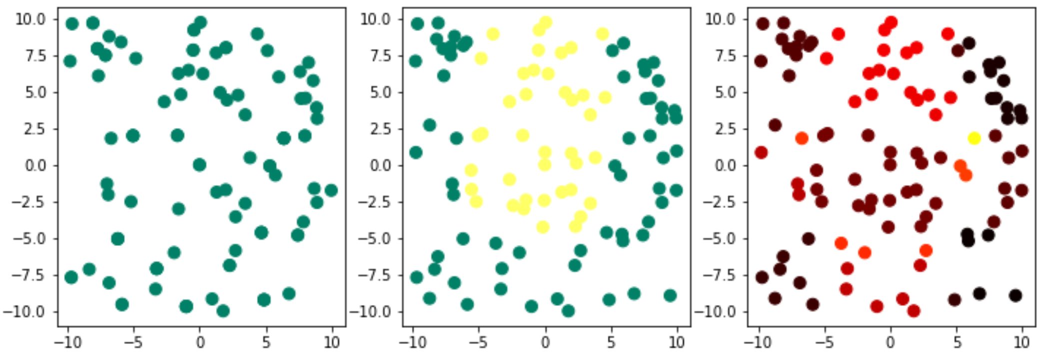

```
('mode', 'vertical_gt', 'Threshhold', -4.6003418181818185)
0.327775842906262
```



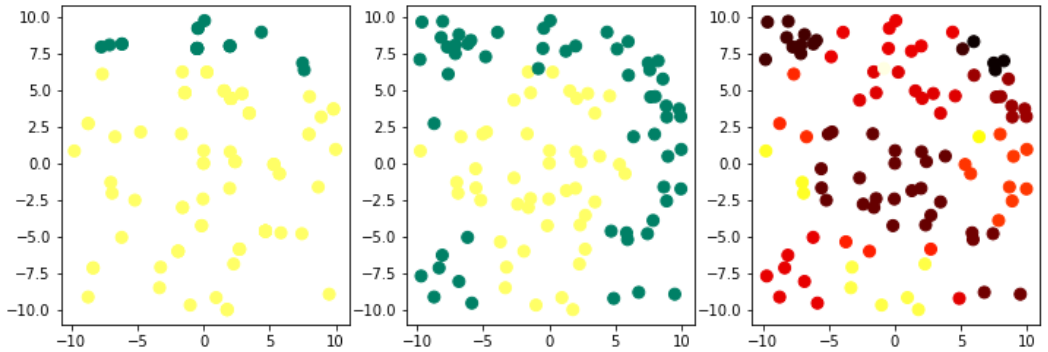

```
('mode', 'horizontal_lt', 'Threshhold', 5.794088787878787)
0.20982963459115844
```



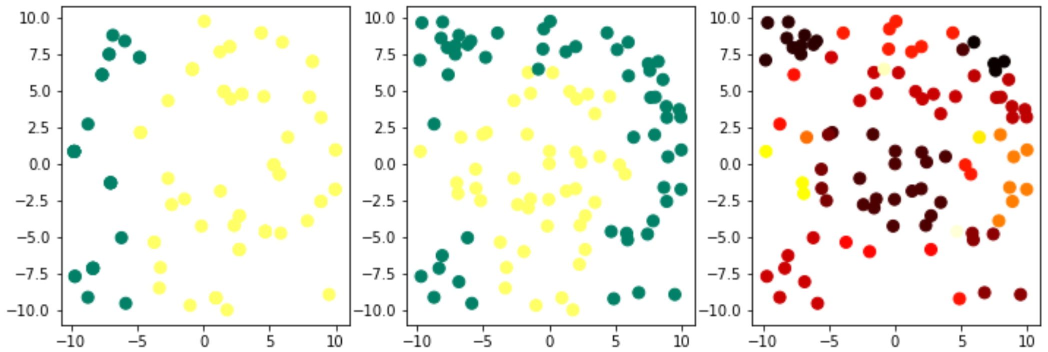

```
('mode', 'horizontal_lt', 'Threshhold', -9.79845)
0.3099929374241277
```



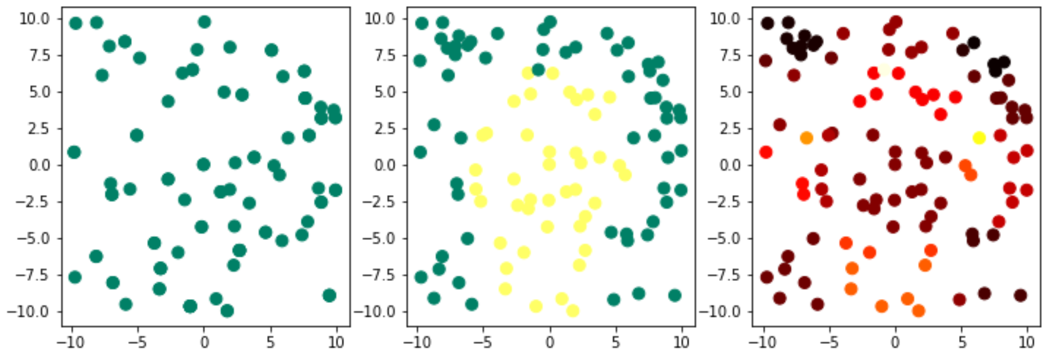
```
('mode', 'vertical_lt', 'Threshhold', 6.341191515151518)
0.42832236014061165
```



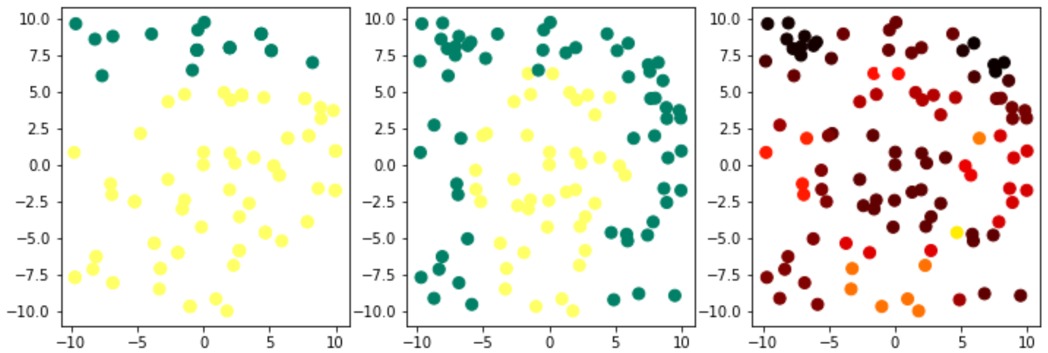

```
('mode', 'horizontal_gt', 'Threshhold', -4.777692121212122)
0.10349640600843424
```



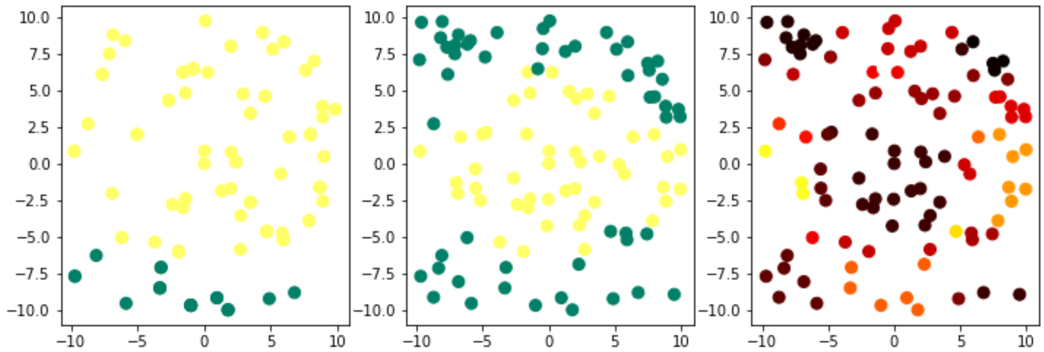

```
('mode', 'horizontal_lt', 'Threshhold', -9.76748)
0.31435731840297665
```



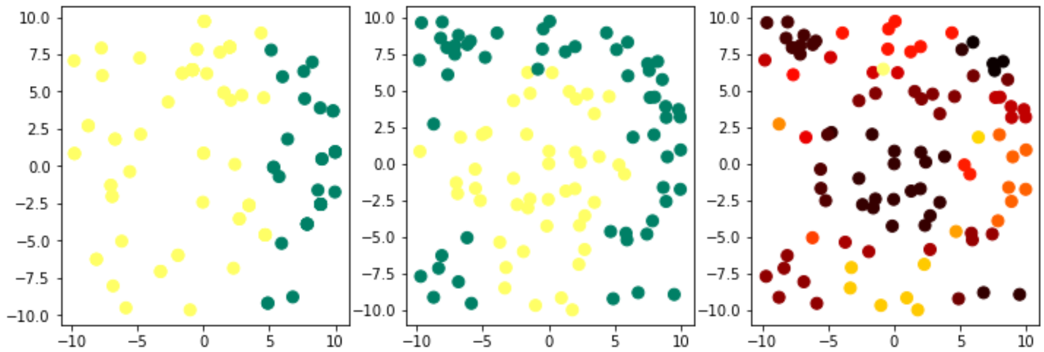

```
('mode', 'vertical_lt', 'Threshhold', 4.948632727272729)
0.16702781059674532
```



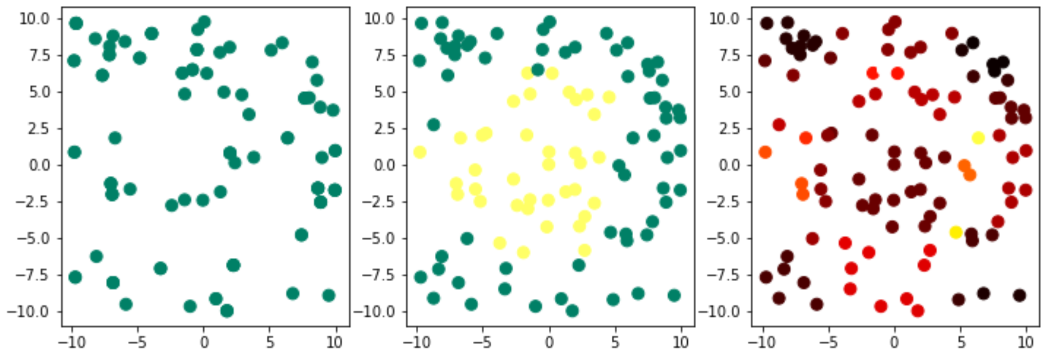
```
('mode', 'vertical_gt', 'Threshhold', -6.191837575757576)
0.2971141632107256
```



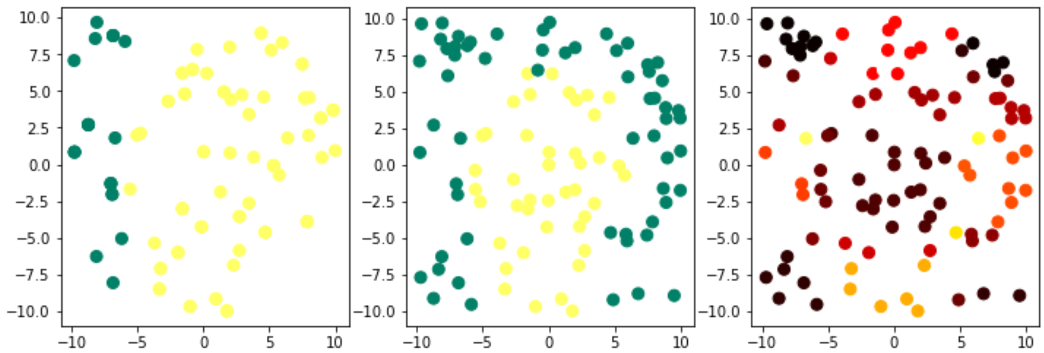
```
('mode', 'horizontal_lt', 'Threshhold', 4.794567070707069)
0.17261634487507485
```



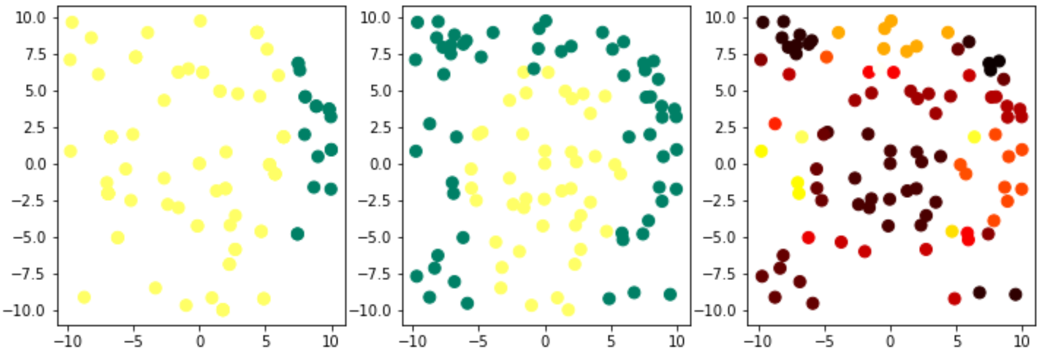
```
('mode', 'horizontal_lt', 'Threshhold', -9.79845)
0.4070948728990799
```



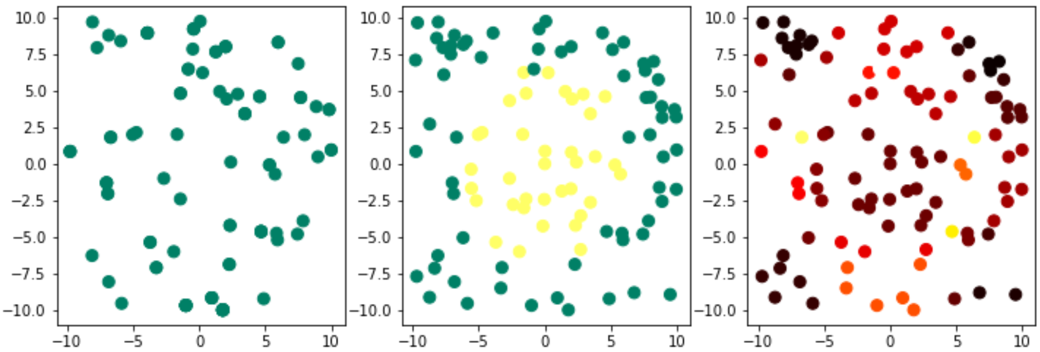
```
('mode', 'horizontal_gt', 'Threshhold', -5.800363131313132)
0.3109496535889911
```



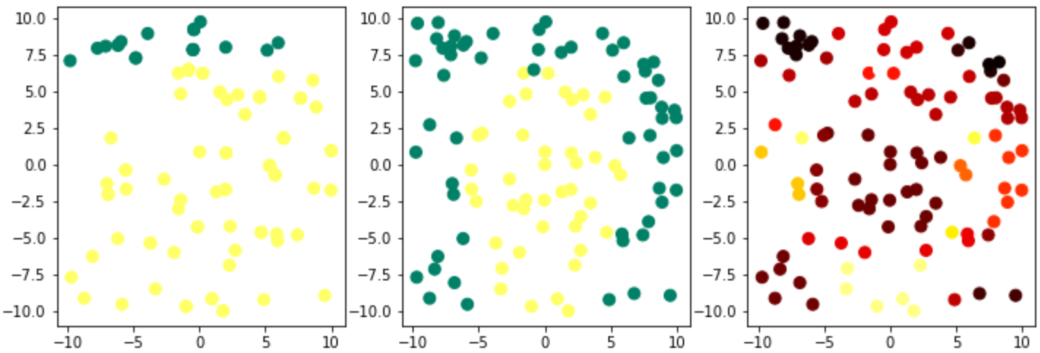
```
('mode', 'horizontal_lt', 'Threshhold', 6.393801818181819)
0.2132556970847048
```



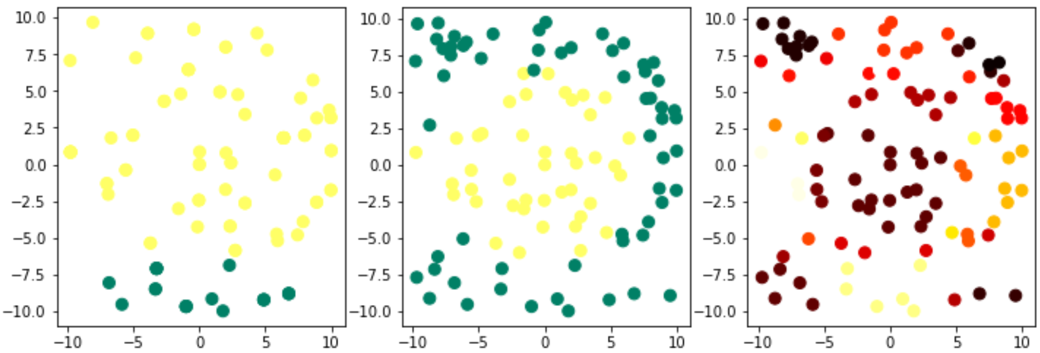
```
('mode', 'horizontal_lt', 'Threshhold', -9.76748)
0.3437631976523702
```



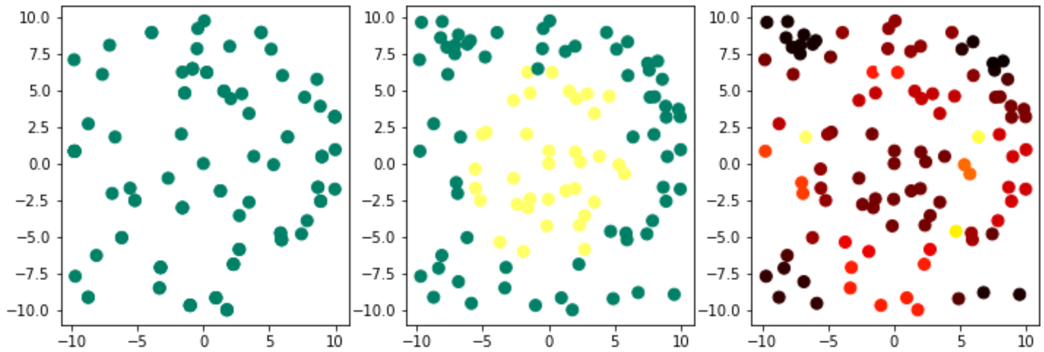
```
('mode', 'vertical_lt', 'Threshhold', 6.540128484848488)
0.27765406779438956
```



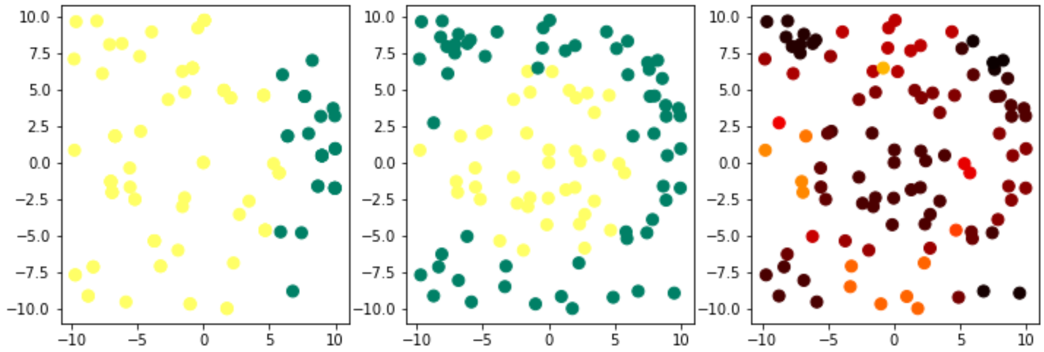
```
('mode', 'vertical_gt', 'Threshhold', -6.796530505050505)
0.18311543401918326
```



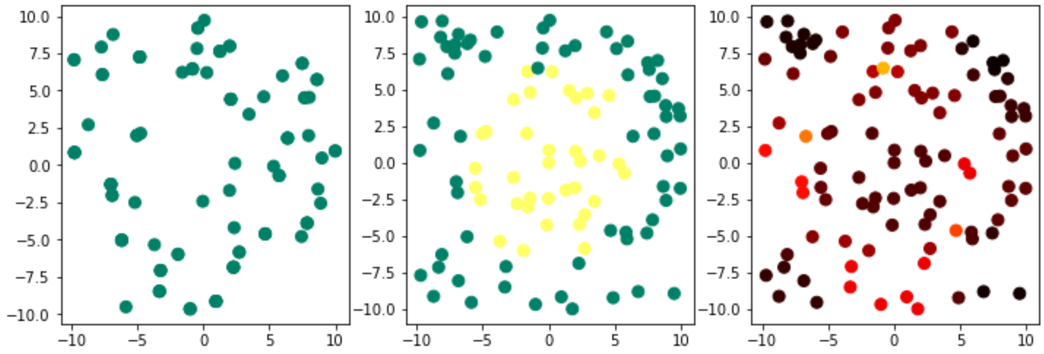
```
('mode', 'horizontal_lt', 'Threshhold', -9.79845)
0.36266968538396244
```



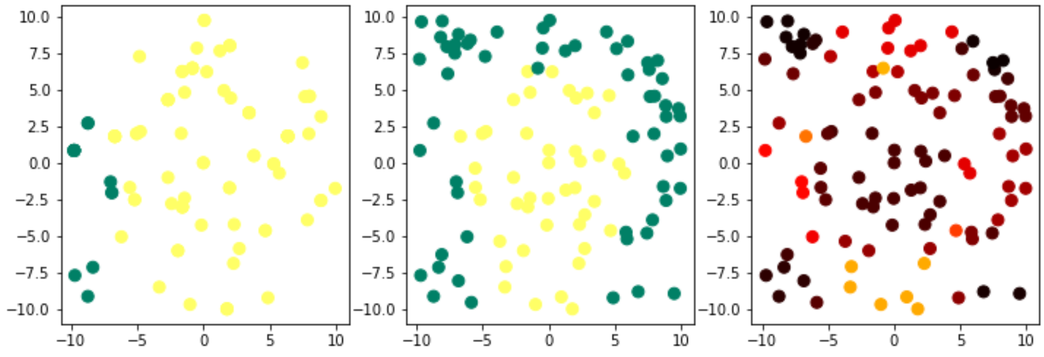
```
('mode', 'horizontal_lt', 'Threshhold', 5.794088787878787)
0.3215615602182444
```



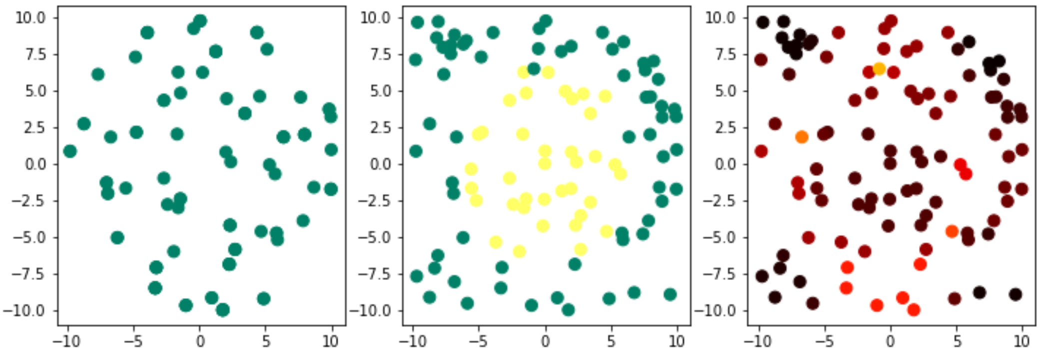
```
('mode', 'horizontal_lt', 'Threshhold', -9.79845)
0.19579680326343787
```



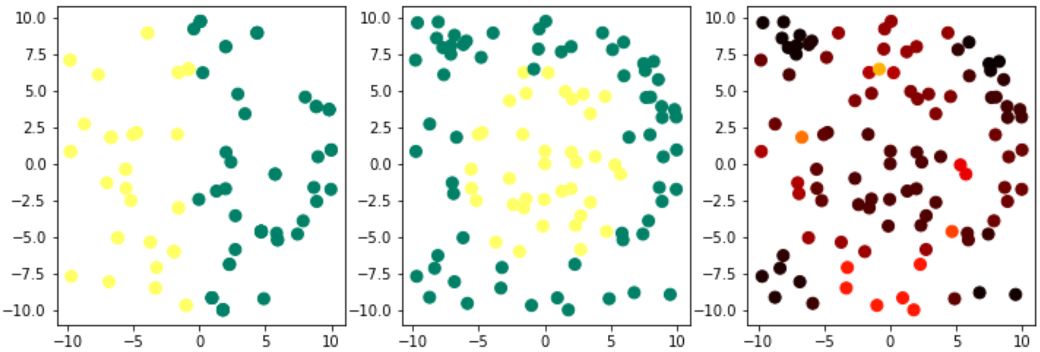
```
('mode', 'horizontal_gt', 'Threshhold', -6.777213333333334)
0.2899135854777691
```



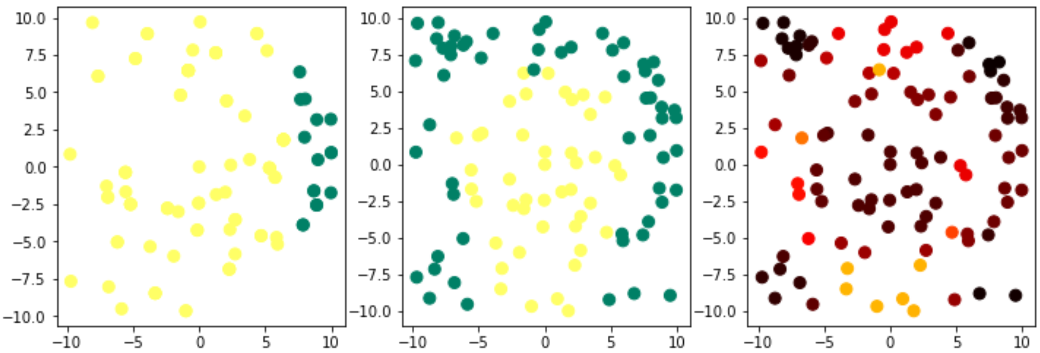
```
('mode', 'horizontal_lt', 'Threshhold', -9.76748)
0.20892994238719553
```



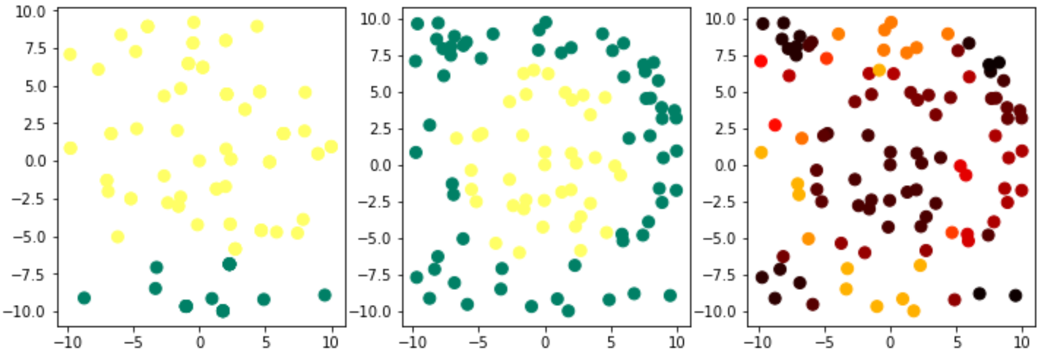
```
('mode', 'horizontal_lt', 'Threshhold', -0.8027545454545457)
0.003914202815402735
```



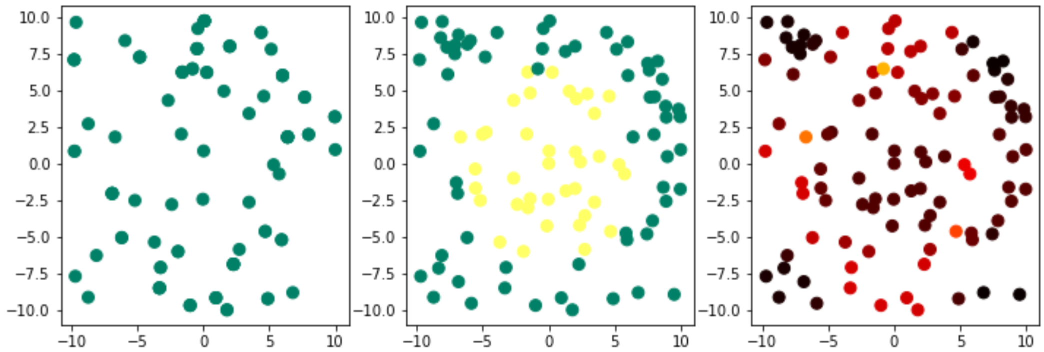
```
('mode', 'horizontal_lt', 'Threshhold', 6.3994327272727265)
0.21680159675836927
```



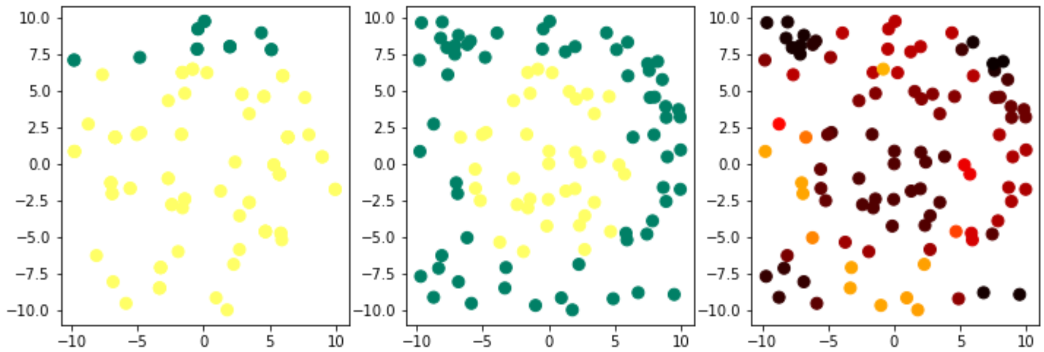
```
('mode', 'vertical_gt', 'Threshhold', -6.679523030303031)
0.24049170774929207
```



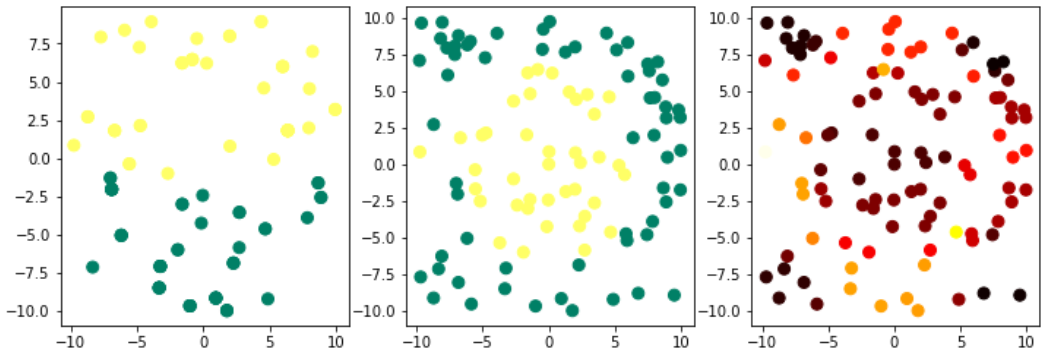
```
('mode', 'horizontal_lt', 'Threshhold', -9.79845)
0.3593171560741169
```



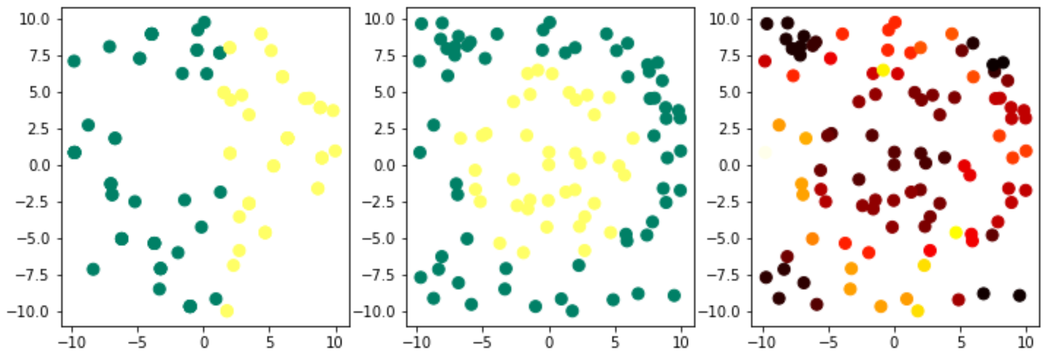
```
('mode', 'vertical_lt', 'Threshhold', 6.540128484848488)
0.33863143257691897
```



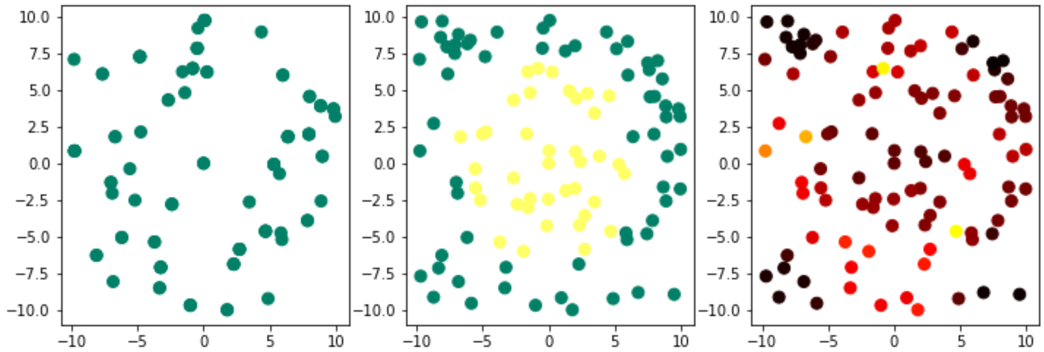
```
('mode', 'vertical_gt', 'Threshhold', -1.1858444444444451)
0.2324932394017141
```



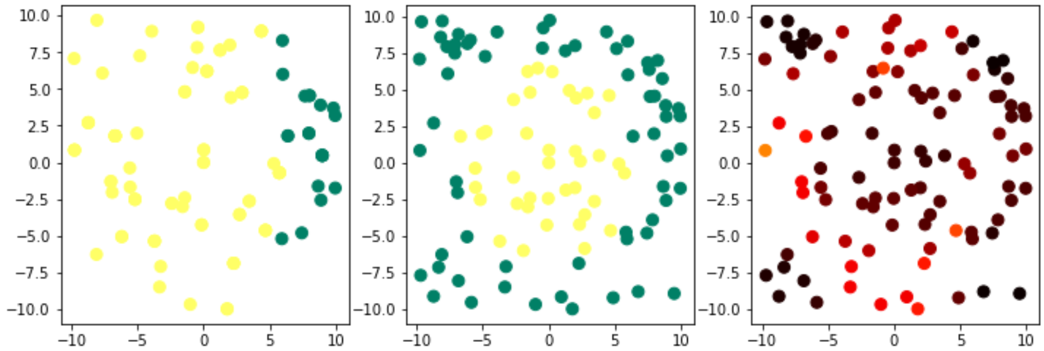
```
('mode', 'horizontal_gt', 'Threshhold', 1.3961932323232311)
0.07357585854973171
```



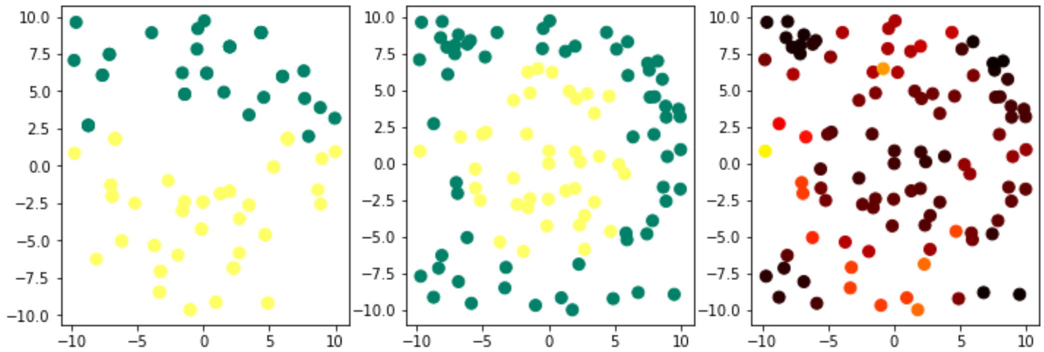
```
('mode', 'horizontal_lt', 'Threshhold', -9.79845)
0.2754270813375769
```



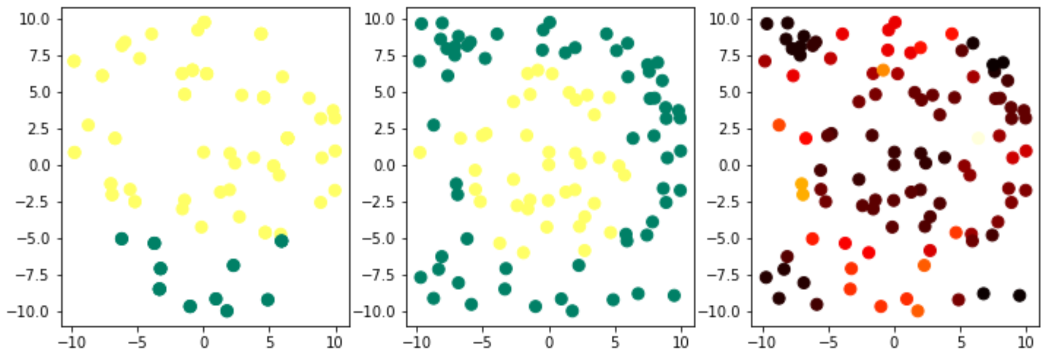

```
('mode', 'horizontal_lt', 'Threshhold', 5.775337272727274)
0.23357089084728277
```



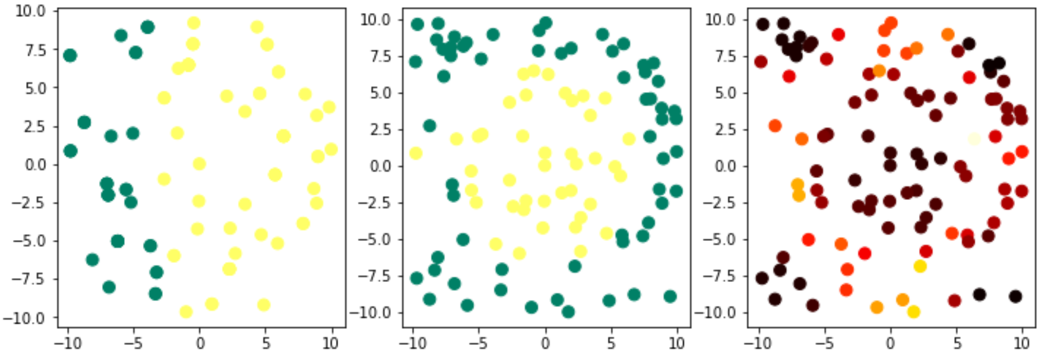

```
('mode', 'vertical_lt', 'Threshhold', 1.8904169696969682)
0.12997958073577406
```



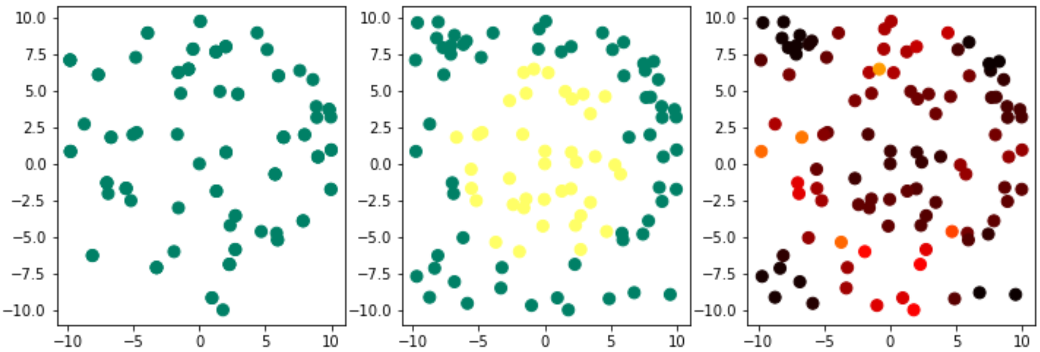

```
('mode', 'vertical_gt', 'Threshhold', -4.998215757575758)
0.17403653142976547
```



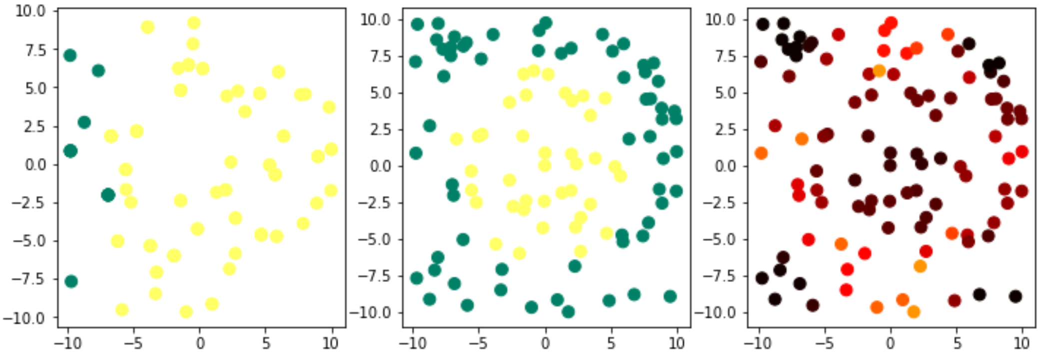
```
('mode', 'horizontal_gt', 'Threshhold', -3.2016066666666676)
0.16029219137598139
```



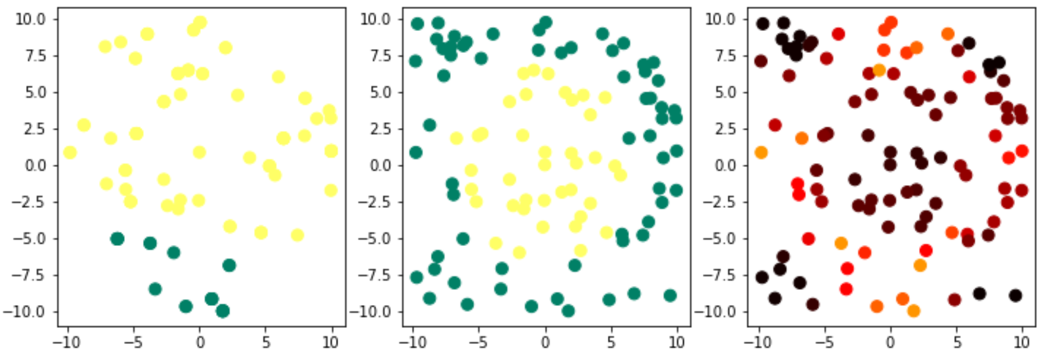
```
('mode', 'horizontal_lt', 'Threshhold', -9.79845)
0.3418302615478427
```



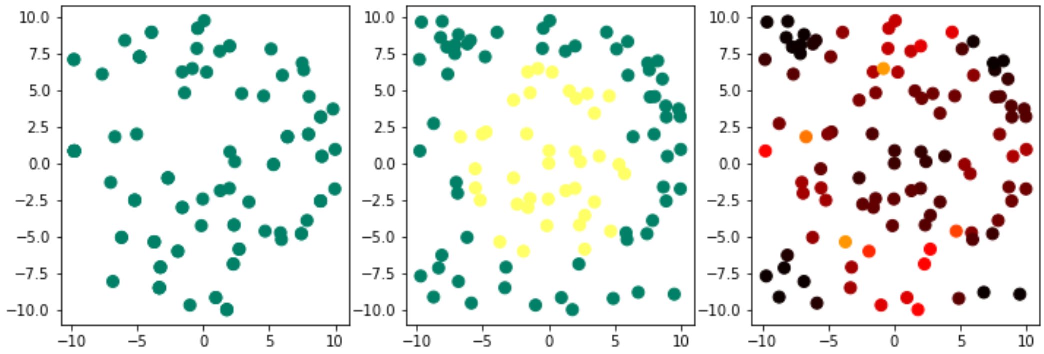
```
('mode', 'horizontal_gt', 'Threshhold', -6.799884848484849)
0.2393875393413601
```



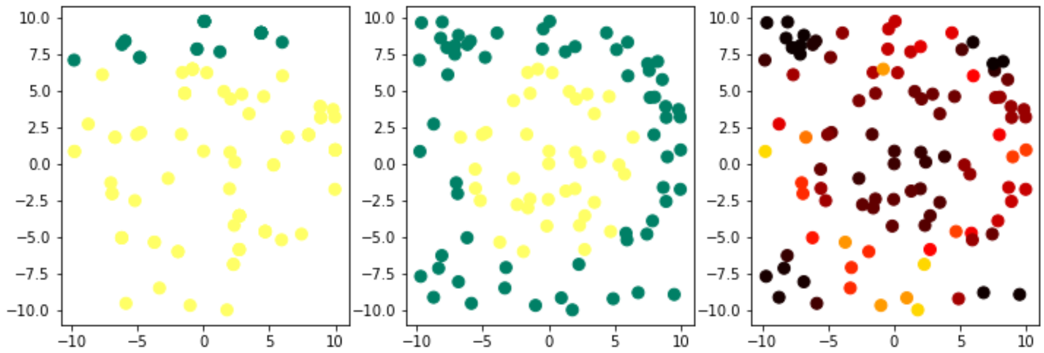
```
('mode', 'vertical_gt', 'Threshhold', -4.998215757575758)
0.06524488786892498
```



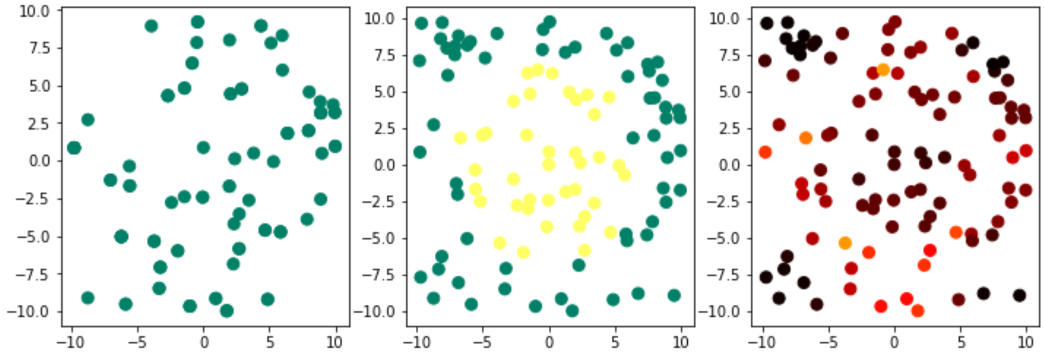
```
('mode', 'horizontal_lt', 'Threshhold', -9.79845)
0.22894555305885309
```



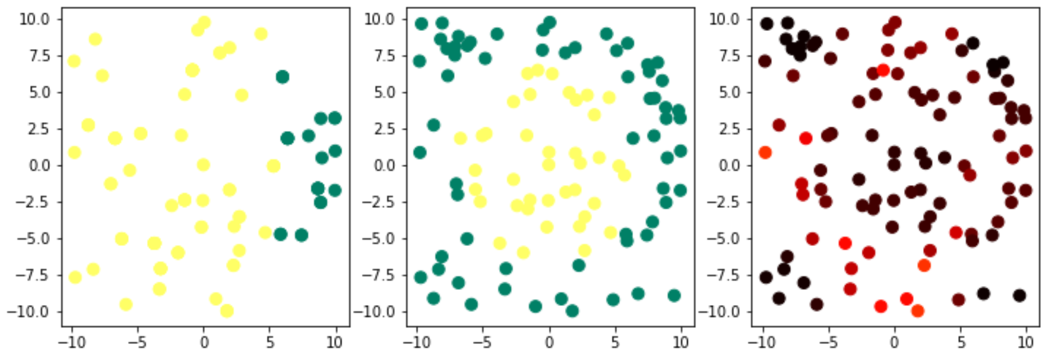

```
('mode', 'vertical_lt', 'Threshhold', 6.540128484848488)
0.30336652394391095
```



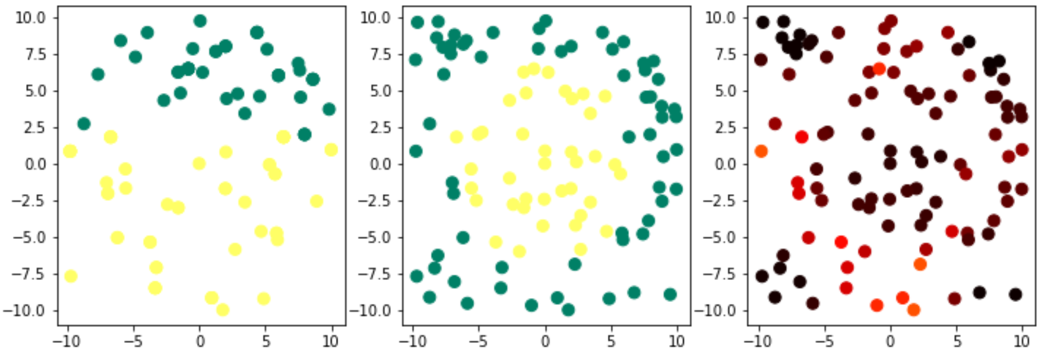

```
('mode', 'horizontal_lt', 'Threshhold', -9.76748)
0.2233018159630692
```



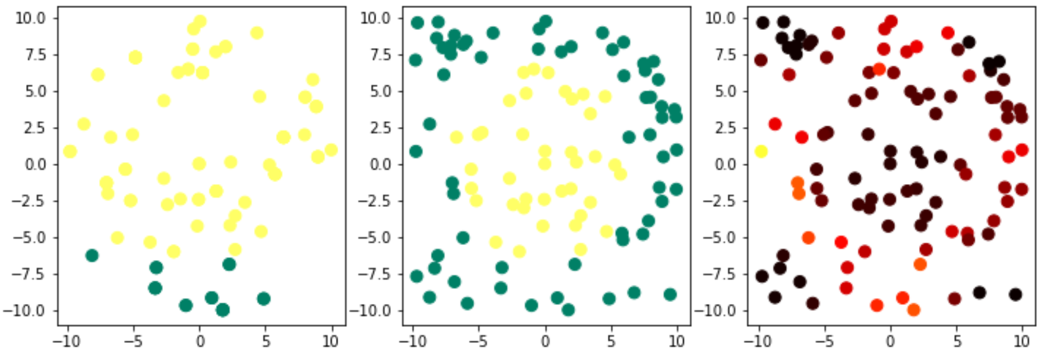

```
('mode', 'horizontal_lt', 'Threshhold', 5.3942801010101)
0.22387096230511988
```



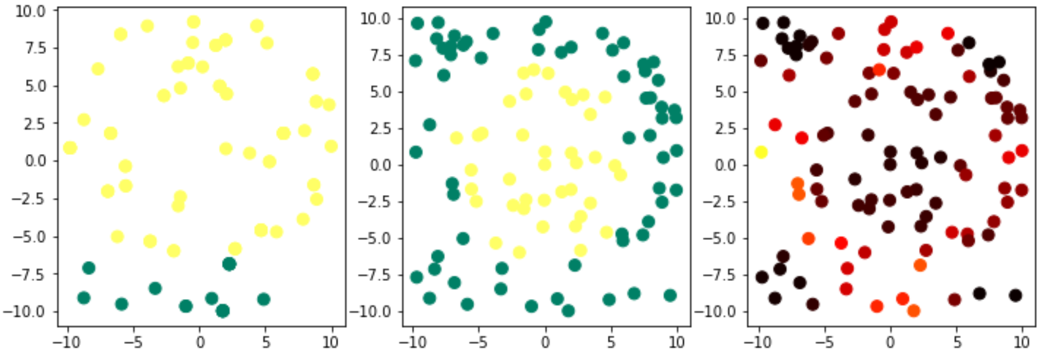
```
('mode', 'vertical_lt', 'Threshhold', 1.9645781818181831)
0.052087797583840516
```



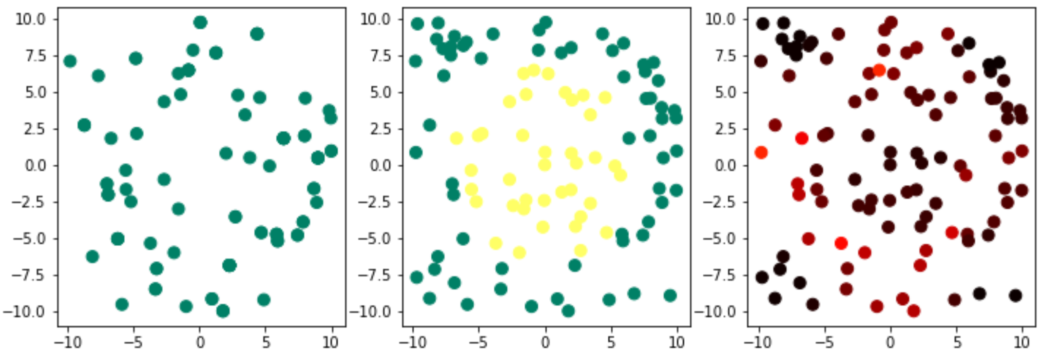
```
('mode', 'vertical_gt', 'Threshhold', -6.191837575757576)
0.24281624750972133
```



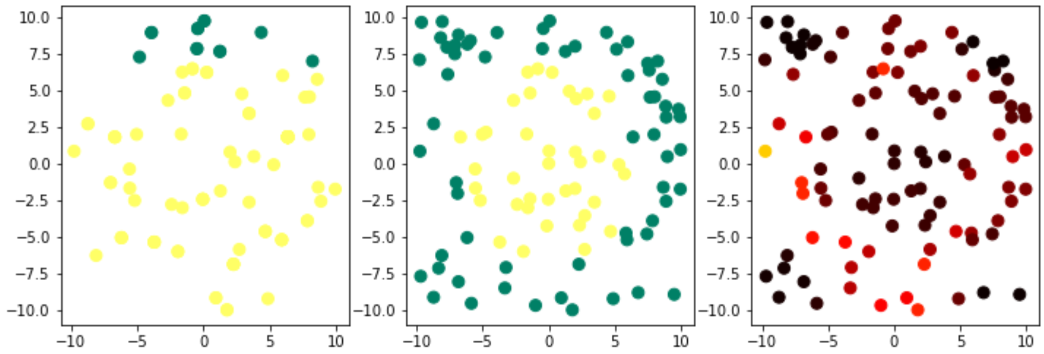
```
('mode', 'vertical_gt', 'Threshhold', -6.679523030303031)
-0.005945956907964739
```



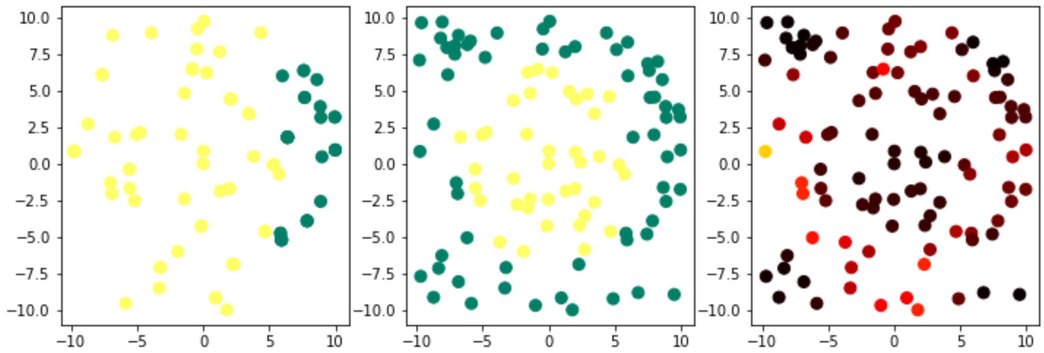
```
('mode', 'horizontal_lt', 'Threshhold', -9.79845)
0.31465670136904794
```



```
('mode', 'vertical_lt', 'Threshhold', 6.540128484848488)
0.23447456585357343
```



```
('mode', 'horizontal_lt', 'Threshhold', 5.800658181818182)
0.0737874776961314
```



on the left side is the cut of the weak classifier which yields the lowest error shown in each iteration. In the middle we see the stron classifier. On the right side the distribution is shown.