Problem 1: VC Dimension

1. Consider a binary classification problem for data points in R. Let H be the hypothesis space of all intervals in R. Given an interval in H, points inside the interval are classified as '+' and the remaining points are classified as '−'.

Consider the boosted hypothesis space H' that takes a pair of hypotheses from H and takes the sign of their weighted combination (similar to what would be produced by two rounds of boosting). Specifically,

H' = {f|f(x) = sign(α1h1(x) + α2h2(x)) for some h1, h2 ∈ H and α1, α2 ∈ R}.

To break ties, if α1h1(x)+α2h2(x) = 0, the hypothesis should return a '+'. What is VC(H')? Prove it.

VC(H') = 2

Consider 3 points in one dimension placed as +, -, +.

2 Rounds of Boosting cannot shatter the points in 1 dimension as after first round one point would remain misclassified and increase the weight of the misclassified point. But in the next iteration, previously low weight point would be misclassified which would again increase weight

In [35]:
```
import pandas as pd
import numpy as np
from tqdm.notebook import tqdm
import math
```

In [42]:
```
data = {
    'x': [0,1,2],
    'y': [1,-1,1]
}
df_3p = pd.DataFrame.from_dict(data)
y_train = df_3p['y']
X_train = df_3p['x']
```

In [28]:
```
def gen_hypotheses():
    y_val = [1,-1]
    H = []
    for i in y_val:
        for j in y_val:
            for k in y_val:
                if not (i == k == 1 and j == -1):
                    h = [i,j,k]
                    H.append(h)
    return H
H = np.array(gen_hypotheses())
print(H)
```

```
[[ 1  1  1]
 [ 1  1 -1]
 [ 1 -1 -1]
 [-1  1  1]
 [-1  1 -1]
 [-1 -1  1]
 [-1 -1 -1]]
```

In [154]:

```
def adaboost(H, y_train, m, T):
    n = 1
    w = np.array([1/m] * m)
    alphas = [0] * T
    epsilons = [0] * T
    selected_H = [None] * T
    y_predictions = [None] * T
    print("Running Adaboost")
    for t in range(T):
        e_t = 1
        h_t = None
        y_t = None
        best_i = 0
        h_i = 0
        tq = tqdm(H)
        tq.set_description(f"Round {t+1}")
        for h in tq:
            h_i += 1
            y_pred = h
            mask = (y_pred != y_train).astype(np.float64)
            e_h = np.sum(mask * w)
            if e_h < e_t:
                e_t = e_h
                h_t = h
                y_t = y_pred
                best_i = h_i
        print(f"Round {t+1} - Best hypothesis index {best_i} {h_t}")
        selected_H[t] = h_t
        y_predictions[t] = y_t
        epsilons[t] = e_t

        a_t = 0.5 * math.log((1-e_t)/e_t) # Log base e
        alphas[t] = a_t

        # Weight update
        normalize = 2 * np.sqrt(e_t * (1-e_t))
        w = w * np.exp(-1 * y_train * y_t * a_t)/normalize
        print("Weights: ")
        print(y_train.ravel())
        print(w.ravel())
    return np.array(alphas), np.array(epsilons), selected_H, np.array(y_predic
tions)
```

In [155]:
```
a, e, h_, y_ = adaboost(H, y_train, 3, 2)
```

```
Running Adaboost


Round 1 - Best hypothesis index 1 [1 1 1]
Weights:
[ 1 -1  1]
[0.25 0.5  0.25]


Round 2 - Best hypothesis index 3 [ 1 -1 -1]
Weights:
[ 1 -1  1]
[0.16666667 0.33333333 0.5       ]
```

In [156]:
```
def boosting_predict(a, H=None, x=None, h_x=None):
    if h_x is None:
        h_x = []
        for h in H:
            y_pred = h
            h_x.append(y_pred)
        h_x = np.array(h_x)
    res = a.dot(h_x)
    res[res == 0] = 1
    return np.sign(res)

def accuracy(y_truth, y_pred):
    return np.mean(y_truth == y_pred)
```

In [157]:
```
print("Alpha: ")
print(a)
print("Epsilon: ")
print(e)
```

```
Alpha:
[0.34657359 0.54930614]
Epsilon:
[0.33333333 0.25      ]
```

In [158]:
```
pred = boosting_predict(np.array(a), h_x=y_)
print("Train accuracy: ", accuracy(y_train.ravel(), pred.flatten()))
```

```
Train accuracy:  0.6666666666666666
```