HW5_P1

October 30, 2018

In [53]:
```
import numpy as np
import matplotlib.pyplot as plt
import csv
import pandas as pd
from sklearn.model_selection import train_test_split
```

In [54]:
```
datax = pd.read_csv(filepath_or_buffer='C:/Users/weich/Google Drive/Rice University/3
datay = pd.read_csv(filepath_or_buffer='C:/Users/weich/Google Drive/Rice University/3
```

In [55]:
```
X_cor = list(datax['-1.1141'])
X_cor.insert(0, -1.1141)
Y_cor = list(datax['0.0113'])
Y_cor.insert(0, 0.0113)
X = np.zeros((len(X_cor), 2))
X[:,0] = X_cor
X[:,1] = Y_cor

y = list(datay['-1'])
y.insert(0, -1)
y = np.array(y)
```

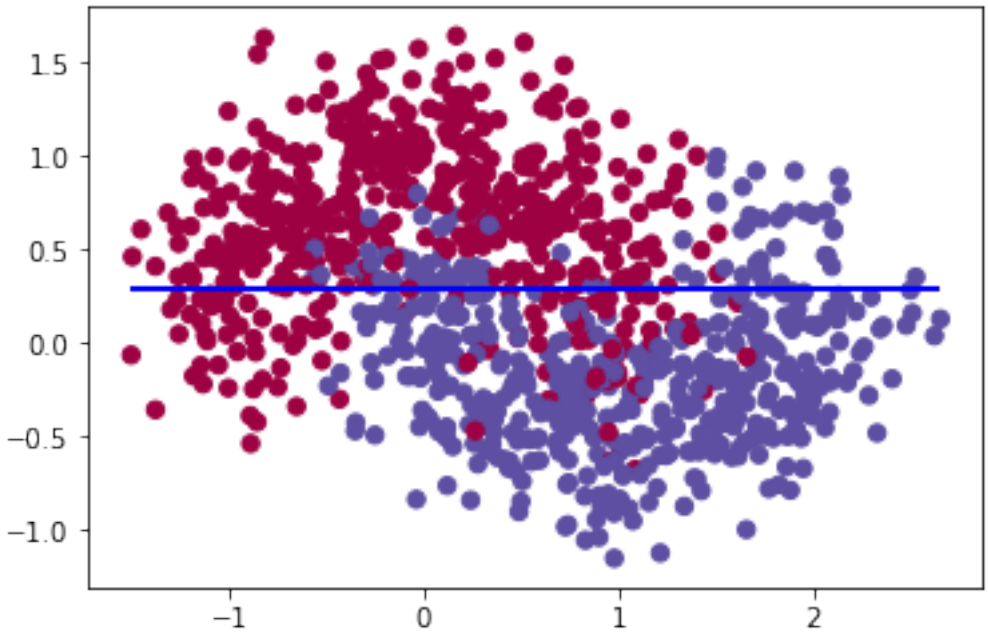
In [56]:
```
# Problem 1.1
prob = np.ones(len(X))/len(X)
min_error = 100000000
best_stump = {}
pnVSnp = 0 #pn = 0, np = 1
for d in range(len(X[0])):
    sorted_index = np.argsort(X[:,d])
    # 0th iteration
    th0 = X[sorted_index[0]][d]
    threshold = th0
    phi0 = np.zeros(len(X))
    errorpn0 = 0
    for i in range(len(X)):
        if X[i][d] >= th0:
            phi0[i] = 1
        else:
            phi0[i] = -1
```

```
            if phi0[i] != y[i]:
                errorpn0 += prob[i]
                errornp0 = 1 - errorpn0
        if min(errorpn0,errornp0) < min_error:
            min_error = min(errorpn0,errornp0)
            min_d = d
            min_threshold = th0
            if errorpn0 < errornp0:
                min_phi = phi0
                pnVSnp = 0
            else:
                min_phi = -phi0
                pnVSnp = 1

        for i in range(1, len(X)):
            th = (X[sorted_index[i-1]][d] + X[sorted_index[i]][d])/2
            phi = np.zeros(len(X))
            errorpn = 0
            for j in range(len(X)):
                if X[j][d] >= th:
                    phi[j] = 1
                else:
                    phi[j] = -1
                if phi[j] != y[j]:
                    errorpn += prob[j]
                    errornp = 1 - errorpn
            if min(errorpn,errornp) < min_error:
                min_d = d
                min_error = min(errorpn,errornp)
                min_threshold = th
                if errorpn < errornp:
                    min_phi = phi
                    pnVSnp = 0
                else:
                    min_phi = -phi
                    pnVSnp = 1

            #print(d, min_error, [errorpn,errornp], th, pnVSnp)
        best_stump['feature'] = min_d
        best_stump['error'] = min_error
        best_stump['threshold'] = min_threshold
        best_stump['phi'] = min_phi
        best_stump['pn vs. np'] = pnVSnp
```

In [57]:
```
best_stump['feature'], best_stump['threshold'], best_stump['error'], best_stump['pn vs
```

Out[57]:
```
(1, 0.2913, 0.1889999999999994, 1)
```

In [58]:
```
plt.scatter(X_cor, Y_cor, s=40, c=y, cmap=plt.cm.Spectral)
if best_stump['feature'] == 0:
    plt.vlines(x=best_stump['threshold'], ymin=min(Y_cor), ymax=max(Y_cor), linewidth=
else:
    plt.hlines(y=best_stump['threshold'], xmin=min(X_cor), xmax=max(X_cor), linewidth=
plt.show()
```



In [59]:
```
def weak_classifier(X, y, prob):
    best_stump = {}
    min_error = 100000000
    pnVSnp = 0 #pn = 0, np = 1
    for d in range(len(X[0])):
        sorted_index = np.argsort(X[:,d])
        # 0th iteration
        th0 = X[sorted_index[0]][d]
        threshold = th0
        phi0 = np.zeros(len(X))
        errorpn0 = 0
        for i in range(len(X)):
            if X[i][d] >= th0:
                phi0[i] = 1
            else:
                phi0[i] = -1
            if phi0[i] != y[i]:
                errorpn0 += prob[i]
                errornp0 = 1 - errorpn0
```

```
            if min(errorpn0,errornp0) < min_error:
                min_error = min(errorpn0,errornp0)
                min_d = d
                min_threshold = th0
                if errorpn0 < errornp0:
                    min_phi = phi0
                    pnVSnp = 0
                else:
                    min_phi = -phi0
                    pnVSnp = 1

            for i in range(1, len(X)):
                th = (X[sorted_index[i-1]][d] + X[sorted_index[i]][d])/2
                phi = np.zeros(len(X))
                errorpn = 0
                for j in range(len(X)):
                    if X[j][d] >= th:
                        phi[j] = 1
                    else:
                        phi[j] = -1
                    if phi[j] != y[j]:
                        errorpn += prob[j]
                        errornp = 1 - errorpn
                if min(errorpn,errornp) < min_error:
                    min_d = d
                    min_error = min(errorpn,errornp)
                    min_threshold = th
                    if errorpn < errornp:
                        min_phi = phi
                        pnVSnp = 0
                    else:
                        min_phi = -phi
                        pnVSnp = 1
            #print('[d, threshold, error]: ',[d, th, error])
            best_stump['feature'] = min_d
            best_stump['error'] = min_error
            best_stump['threshold'] = min_threshold
            best_stump['phi'] = min_phi
            best_stump['pn vs. np'] = pnVSnp
        print('[min_d, min_threshold, min_error, pn vs. np]: ',[min_d, min_threshold, min_
        return best_stump
```

In [60]:
```
def testing(X, y, th, pnVSnp):
    test_stump = {}
    error = np.zeros(len(X[0]))
    phi = np.zeros((len(X), len(X[0])))
    for d in range(len(X[0])):
        error[d] = 0
```

```
            for i in range(len(X)):
                if pnVSnp == 0:
                    if X[i][d] >= th:
                        phi[i][d] = 1
                    else:
                        phi[i][d] = -1
                else:
                    if X[i][d] >= th:
                        phi[i][d] = -1
                    else:
                        phi[i][d] = 1

                if phi[i][d] != y[i]:
                    error[d] += 1
        test_stump['error'] = min(error)
        test_stump['phi'] = phi[:,np.argmin(error)]
        return test_stump
```

In [91]:
```
# Problem 1.2
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20)
w = np.ones(len(X_train))
prob = w/sum(w)
step = 60 #int(np.log(len(X)))
theta = np.zeros(step)
phi_temp = np.zeros((step, len(X_train)))
theta_phi = np.zeros(len(X_train))
phi_test_temp = np.zeros((step, len(X_test)))
theta_phi_test = np.zeros(len(X_test))
error_train = np.zeros(step)
error_test = np.zeros(step)
threshold_temp = np.zeros(step)
dimension_temp = np.zeros(step)
pnVSnp_temp = np.zeros(step)
for t in range(step):
    best_stump = weak_classifier(X_train, y_train, prob)
    w_p = 0 # w+
    w_n = 0 # w-
    phi = best_stump['phi']
    threshold_temp[t] = best_stump['threshold']
    dimension_temp[t] = best_stump['feature']
    pnVSnp_temp[t] = best_stump['pn vs. np']
    for i in range(len(y_train)):
        if phi[i] == y_train[i]:
            w_p += w[i]
        else:
            w_n += w[i]
    #=w_n = sum(w) - w_p
    theta[t] = 0.5 * np.log(w_p/max(w_n,1e-16))
```

```
            phi_temp[t][:] = phi
            theta_phi[:] += theta[t] * phi
            print('[t, w_n, theta[t]]: ', [t, w_n, theta[t]])

            w *= np.exp(-y_train * theta[t] * phi)

            error_train[t] = sum((np.sign(theta_phi) != y_train)*np.ones(len(y_train)))/len(y

            test_stump = testing(X_test, y_test, best_stump['threshold'], pnVSnp_temp[t])
            phi_test = test_stump['phi']
            phi_test_temp[t][:] = phi_test
            theta_phi_test[:] += theta[t] * phi_test
            error_test[t] = sum((np.sign(theta_phi_test) != y_test)*np.ones(len(y_test)))/len

            prob = w/sum(w)
```

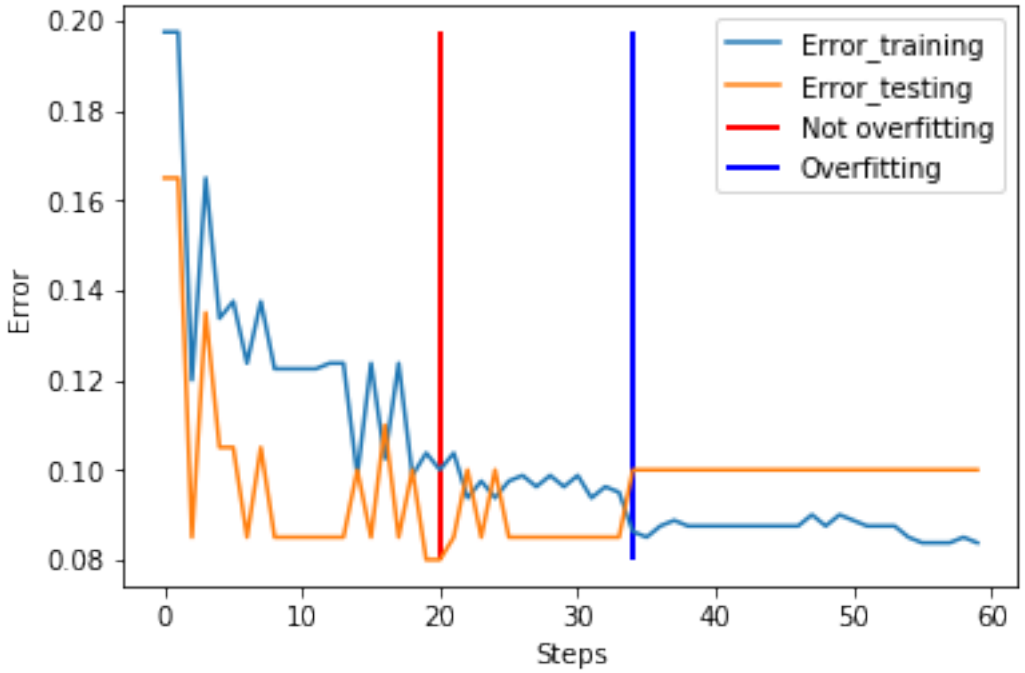
```
[min_d, min_threshold, min_error, pn vs. np]:  [1, 0.19495, 0.19750000000001156, 1]
[t, w_n, theta[t]]:  [0, 158.0, 0.7009966353314973]
[min_d, min_threshold, min_error, pn vs. np]:  [0, -0.2946, 0.300406167435626, 0]
[t, w_n, theta[t]]:  [1, 191.35283362519968, 0.42268223851104925]
[min_d, min_threshold, min_error, pn vs. np]:  [0, 1.4424000000000001, 0.24925368442503823, 0]
[t, w_n, theta[t]]:  [2, 145.57105984825313, 0.5512983042066759]
[min_d, min_threshold, min_error, pn vs. np]:  [1, 0.6854, 0.3187244465612875, 1]
[t, w_n, theta[t]]:  [3, 161.04462212189176, 0.3798199646503142]
[min_d, min_threshold, min_error, pn vs. np]:  [1, -0.3044, 0.3607204295837523, 1]
[t, w_n, theta[t]]:  [4, 169.8637378979352, 0.28611932303344667]
[min_d, min_threshold, min_error, pn vs. np]:  [0, -0.5637, 0.3608278762110066, 0]
[t, w_n, theta[t]]:  [5, 163.18896860410703, 0.2858863674889887]
[min_d, min_threshold, min_error, pn vs. np]:  [0, 1.6666, 0.37612435736551725, 0]
[t, w_n, theta[t]]:  [6, 163.38460173494636, 0.2530156157380246]
[min_d, min_threshold, min_error, pn vs. np]:  [0, 0.46995, 0.39936185585421324, 1]
[t, w_n, theta[t]]:  [7, 168.0703045742983, 0.20406237539886837]
[min_d, min_threshold, min_error, pn vs. np]:  [0, -0.35875, 0.3830708176579932, 0]
[t, w_n, theta[t]]:  [8, 157.9149383274345, 0.23826718266241845]
[min_d, min_threshold, min_error, pn vs. np]:  [0, 1.1886, 0.3892726670361741, 0]
[t, w_n, theta[t]]:  [9, 156.0217856170381, 0.22518527719737103]
[min_d, min_threshold, min_error, pn vs. np]:  [1, 0.4813, 0.40517216079967966, 1]
[t, w_n, theta[t]]:  [10, 158.36221240030977, 0.19197997630944771]
[min_d, min_threshold, min_error, pn vs. np]:  [1, 0.21505000000000002, 0.39394057058945514, 0]
[t, w_n, theta[t]]:  [11, 151.17783099359562, 0.2153889938707511]
[min_d, min_threshold, min_error, pn vs. np]:  [1, -0.10669999999999999, 0.3958746889867417, 1]
[t, w_n, theta[t]]:  [12, 148.46295224041182, 0.21134196287884802]
[min_d, min_threshold, min_error, pn vs. np]:  [0, 0.50125, 0.42056304463453276, 1]
[t, w_n, theta[t]]:  [13, 154.26373578555894, 0.16023123473396664]
[min_d, min_threshold, min_error, pn vs. np]:  [0, -0.5637, 0.4123689680359619, 0]
[t, w_n, theta[t]]:  [14, 149.3369774553091, 0.1770903761071526]
```

```
[min_d, min_threshold, min_error, pn vs. np]:  [1, -0.3044, 0.40705332753562895, 1]
[t, w_n, theta[t]]:  [15, 145.13028068167046, 0.18808013190890405]
[min_d, min_threshold, min_error, pn vs. np]:  [1, 0.9957, 0.4215944249612489, 1]
[t, w_n, theta[t]]:  [16, 147.69474847877567, 0.1581157679550992]
[min_d, min_threshold, min_error, pn vs. np]:  [0, 1.6666, 0.41241320985674534, 0]
[t, w_n, theta[t]]:  [17, 142.69094767346286, 0.1769990898604234]
[min_d, min_threshold, min_error, pn vs. np]:  [0, -0.5637, 0.42034581785513075, 0]
[t, w_n, theta[t]]:  [18, 143.18676487221526, 0.16067696985190172]
[min_d, min_threshold, min_error, pn vs. np]:  [0, 0.1632, 0.43047185225144957, 1]
[t, w_n, theta[t]]:  [19, 144.76339274950502, 0.13996313457944107]
[min_d, min_threshold, min_error, pn vs. np]:  [0, 1.1106, 0.43214261010186505, 0]
[t, w_n, theta[t]]:  [20, 143.9133412267952, 0.1365573316647032]
[min_d, min_threshold, min_error, pn vs. np]:  [1, 0.4813, 0.4301600268534501, 1]
[t, w_n, theta[t]]:  [21, 141.9277108645127, 0.14059913880873964]
[min_d, min_threshold, min_error, pn vs. np]:  [1, 0.18025, 0.42896261795975255, 0]
[t, w_n, theta[t]]:  [22, 140.14514947030008, 0.14304244883796322]
[min_d, min_threshold, min_error, pn vs. np]:  [1, -0.08015, 0.44504011688418954, 1]
[t, w_n, theta[t]]:  [23, 143.92285779468, 0.11036570002760589]
[min_d, min_threshold, min_error, pn vs. np]:  [0, -0.13774999999999998, 0.43425490996802585, 0
[t, w_n, theta[t]]:  [24, 139.5840264000685, 0.13225594697646048]
[min_d, min_threshold, min_error, pn vs. np]:  [0, 0.1632, 0.4405984211934276, 1]
[t, w_n, theta[t]]:  [25, 140.3933968241557, 0.1193668760169227]
[min_d, min_threshold, min_error, pn vs. np]:  [0, 1.4424000000000001, 0.45718425269948043, 0]
[t, w_n, theta[t]]:  [26, 144.64662881253793, 0.0858417251829608]
[min_d, min_threshold, min_error, pn vs. np]:  [1, 0.86355, 0.4332296155642993, 1]
[t, w_n, theta[t]]:  [27, 136.56425405177353, 0.13434318927491923]
[min_d, min_threshold, min_error, pn vs. np]:  [0, 1.6666, 0.4419745234336134, 0]
[t, w_n, theta[t]]:  [28, 138.07300423887997, 0.11657618857102263]
[min_d, min_threshold, min_error, pn vs. np]:  [0, -0.5637, 0.43646579080196923, 0]
[t, w_n, theta[t]]:  [29, 135.43077880647292, 0.1277590197211869]
[min_d, min_threshold, min_error, pn vs. np]:  [1, -0.3044, 0.443245420449362, 1]
[t, w_n, theta[t]]:  [30, 136.41956679742665, 0.11400045923227303]
[min_d, min_threshold, min_error, pn vs. np]:  [0, 0.9626, 0.4378885794921573, 1]
[t, w_n, theta[t]]:  [31, 133.8998377296631, 0.12486779704033772]
[min_d, min_threshold, min_error, pn vs. np]:  [0, 1.2879, 0.44693437719767576, 0]
[t, w_n, theta[t]]:  [32, 135.60734166457416, 0.10653244236082329]
[min_d, min_threshold, min_error, pn vs. np]:  [0, -0.13774999999999998, 0.4414760256413747, 0]
[t, w_n, theta[t]]:  [33, 133.194647925561, 0.1175869137061653]
[min_d, min_threshold, min_error, pn vs. np]:  [0, 0.441, 0.45298813973456775, 1]
[t, w_n, theta[t]]:  [34, 135.72846857695487, 0.09430227051728307]
[min_d, min_threshold, min_error, pn vs. np]:  [1, 0.36655000000000004, 0.46308267644375656, 1]
[t, w_n, theta[t]]:  [35, 138.13840432580693, 0.07396925890978436]
[min_d, min_threshold, min_error, pn vs. np]:  [1, 0.2624, 0.45205207817483284, 0]
[t, w_n, theta[t]]:  [36, 134.4798889517888, 0.0961914294350668]
[min_d, min_threshold, min_error, pn vs. np]:  [0, 1.6666, 0.4562189386881967, 0]
[t, w_n, theta[t]]:  [37, 135.09399796164547, 0.08778694101564788]
[min_d, min_threshold, min_error, pn vs. np]:  [1, 0.9957, 0.4405584534535927, 1]
[t, w_n, theta[t]]:  [38, 129.95559230911437, 0.11944795665430022]
```

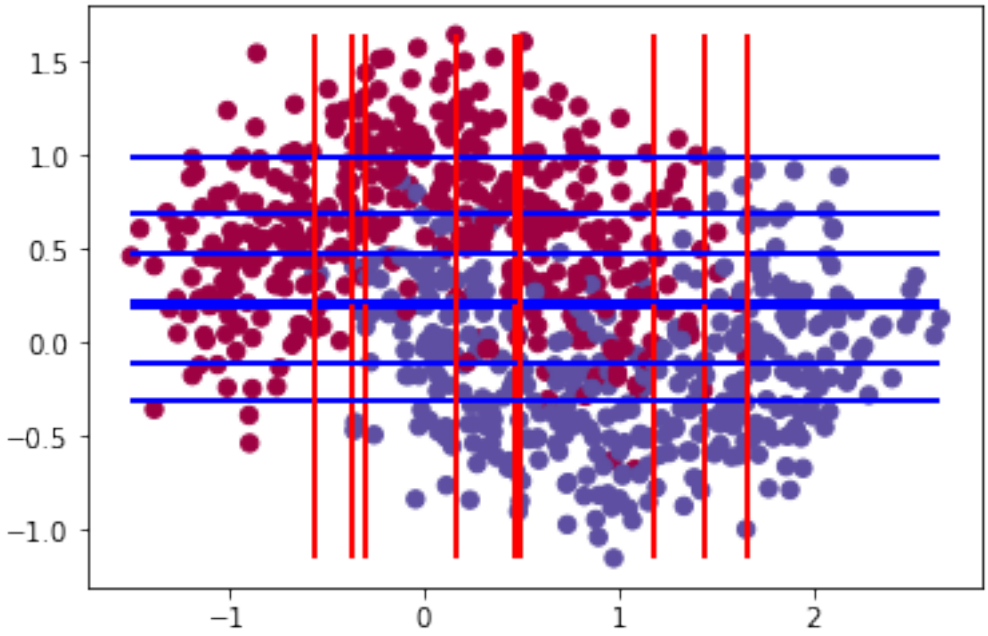
```
[min_d, min_threshold, min_error, pn vs. np]:  [0, 1.6666, 0.44687421151203394, 0]
[t, w_n, theta[t]]:  [39, 130.88378626284015, 0.10665414613900948]
[min_d, min_threshold, min_error, pn vs. np]:  [0, -0.5637, 0.4502341820088327, 0]
[t, w_n, theta[t]]:  [40, 131.1214117572471, 0.09986227510927234]
[min_d, min_threshold, min_error, pn vs. np]:  [1, -0.3044, 0.4503976419484177, 1]
[t, w_n, theta[t]]:  [41, 130.51768297738735, 0.09953209501745305]
[min_d, min_threshold, min_error, pn vs. np]:  [1, 0.9957, 0.45841454971319284, 1]
[t, w_n, theta[t]]:  [42, 132.18555048525522, 0.08336347590650918]
[min_d, min_threshold, min_error, pn vs. np]:  [0, 1.6666, 0.45533811227950793, 0]
[t, w_n, theta[t]]:  [43, 130.843538320054, 0.08956248287070682]
[min_d, min_threshold, min_error, pn vs. np]:  [0, -0.5637, 0.4569457594048069, 0]
[t, w_n, theta[t]]:  [44, 130.78062816197468, 0.08632225504300596]
[min_d, min_threshold, min_error, pn vs. np]:  [0, 1.6666, 0.46035917098446216, 0]
[t, w_n, theta[t]]:  [45, 131.26818880461565, 0.0794483977536264]
[min_d, min_threshold, min_error, pn vs. np]:  [1, 0.4813, 0.46106590566158645, 1]
[t, w_n, theta[t]]:  [46, 131.05587614754785, 0.07802614715507859]
[min_d, min_threshold, min_error, pn vs. np]:  [1, 0.61605, 0.4664406654354531, 0]
[t, w_n, theta[t]]:  [47, 132.18105846281225, 0.0672197304167339]
[min_d, min_threshold, min_error, pn vs. np]:  [1, 0.9957, 0.4602994602062608, 1]
[t, w_n, theta[t]]:  [48, 130.14660526697423, 0.07956857583756162]
[min_d, min_threshold, min_error, pn vs. np]:  [1, -0.68135, 0.46317450629319623, 1]
[t, w_n, theta[t]]:  [49, 130.54603351426874, 0.07378459500750303]
[min_d, min_threshold, min_error, pn vs. np]:  [0, 0.9626, 0.4628640759931031, 1]
[t, w_n, theta[t]]:  [50, 130.1042233072276, 0.07440887059751664]
[min_d, min_threshold, min_error, pn vs. np]:  [0, 1.09355, 0.45609902382578116, 0]
[t, w_n, theta[t]]:  [51, 127.84857587264045, 0.08802862887167269]
[min_d, min_threshold, min_error, pn vs. np]:  [0, -0.5637, 0.45476487586781783, 0]
[t, w_n, theta[t]]:  [52, 126.9822892650487, 0.0907182965011614]
[min_d, min_threshold, min_error, pn vs. np]:  [0, 1.6666, 0.4660035032027638, 0]
[t, w_n, theta[t]]:  [53, 129.58680583500393, 0.0680980634669951]
[min_d, min_threshold, min_error, pn vs. np]:  [0, 0.9626, 0.46364468138448045, 1]
[t, w_n, theta[t]]:  [54, 128.63248989637893, 0.072839181657529]
[min_d, min_threshold, min_error, pn vs. np]:  [0, 1.09355, 0.46601124839250047, 0]
[t, w_n, theta[t]]:  [55, 128.94684667841952, 0.06808250115862555]
[min_d, min_threshold, min_error, pn vs. np]:  [1, 0.9957, 0.4617617369082109, 1]
[t, w_n, theta[t]]:  [56, 127.47543992540237, 0.07662614662546184]
[min_d, min_threshold, min_error, pn vs. np]:  [1, -0.3044, 0.46344932739149536, 1]
[t, w_n, theta[t]]:  [57, 127.56663036887473, 0.07323197746972356]
[min_d, min_threshold, min_error, pn vs. np]:  [0, 0.9626, 0.46869997288641363, 1]
[t, w_n, theta[t]]:  [58, 128.6667264393856, 0.06268201870463133]
[min_d, min_threshold, min_error, pn vs. np]:  [0, 0.6745, 0.4687747110339161, 0]
[t, w_n, theta[t]]:  [59, 128.43484826960528, 0.06253195575095531]
```

```
In [96]: plt.plot(np.linspace(0, step-1, step), error_train)
         plt.plot(np.linspace(0, step-1, step), error_test)
         plt.vlines(x=20, ymin=min(min(error_train), min(error_test)), ymax=max(max(error_trai
         plt.vlines(x=34, ymin=min(min(error_train), min(error_test)), ymax=max(max(error_trai
```

```
plt.legend(('Error_training', 'Error_testing', 'Not overfitting', 'Overfitting'))
plt.ylabel('Error')
plt.xlabel('Steps')
plt.show()
```

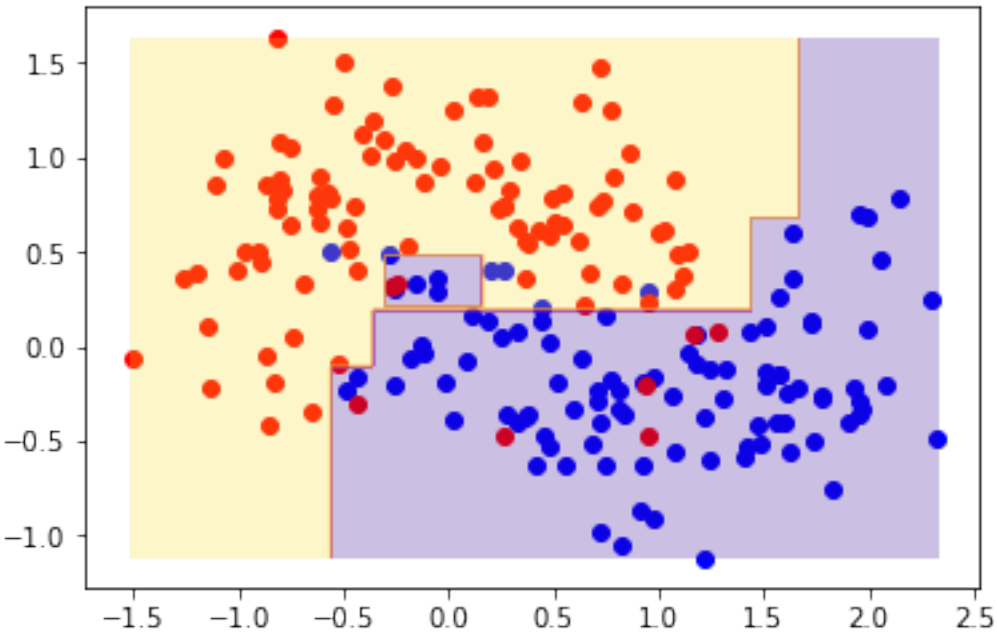


In [97]:
```
# small number of decision stumps
plt.scatter(X_train[:,0], X_train[:,1], s=40, c=y_train, cmap=plt.cm.Spectral)
for i in range(20):
    if dimension_temp[i] == 0:
        plt.vlines(x=threshold_temp[i], ymin=min(Y_cor), ymax=max(Y_cor), linewidth=2
    else:
        plt.hlines(y=threshold_temp[i], xmin=min(X_cor), xmax=max(X_cor), linewidth=2

plt.show()
```



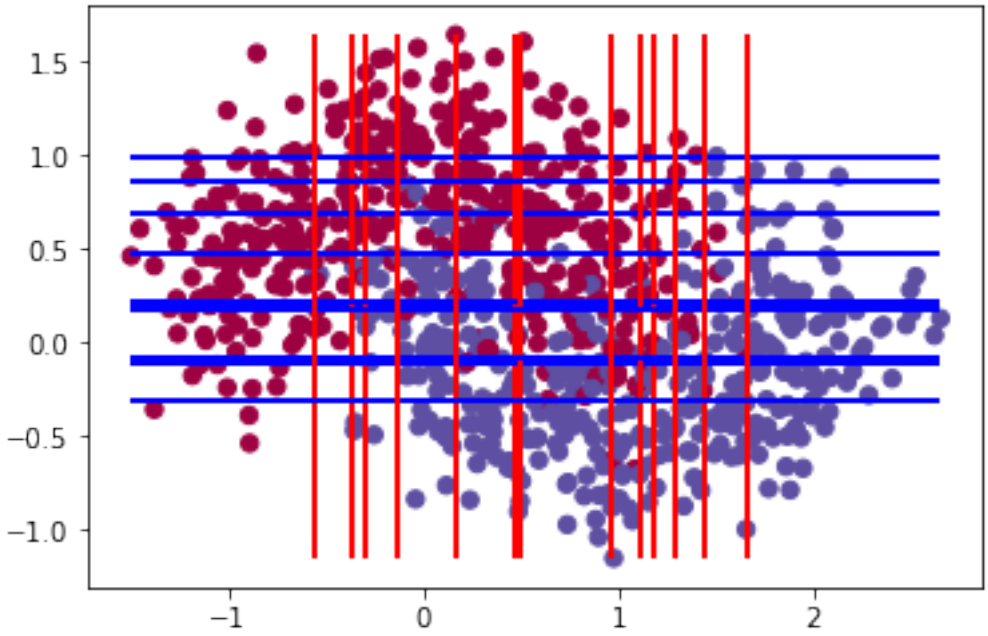

```
In [101]: # Small number of decision stumps
          X_feature0 = X_test[:,0]
          X_feature1 = X_test[:,1]
          X_p = np.where(y_test==1)[0]
          X_n = np.where(y_test==-1)[0]

          h = 0.01
          xx, yy = np.meshgrid(np.arange(min(X_test[:,0]), max(X_test[:,0]), h),
                               np.arange(min(X_test[:,1]), max(X_test[:,1]), h))
          f = 0*xx
          r = 0*xx+1
          for i in range(20):
              if dimension_temp[i] == 0:
                  if pnVSnp_temp[i] == 1:
                      f += theta[i]
                      f[:,np.where(xx[0,:]<=threshold_temp[i])[0]] -= 2*theta[i]
                      r = np.sign(f)
                  else:
                      f += theta[i]
                      f[:,np.where(xx[0,:]>threshold_temp[i])[0]] -= 2*theta[i]
                      r = np.sign(f)
              else:
                  if pnVSnp_temp[i] == 1:
                      f += theta[i]
                      f[np.where(yy[:,0]<=threshold_temp[i])[0],:] -= 2*theta[i]
                      r = np.sign(f)
```

```
            else:
                f += theta[i]
                f[np.where(yy[:,0]>threshold_temp[i])[0],:] -= 2*theta[i]
                r = np.sign(f)

        plt.scatter(X_feature0[X_p], X_feature1[X_p],color = 'b', label='y=1')
        plt.scatter(X_feature0[X_n], X_feature1[X_n],color = 'r', label='y=-1')
        plt.contourf(xx, yy, r, alpha=.25, cmap=plt.cm.plasma)
        plt.show()
        print('Refered from the link on Piazza ans collaborated with Chun-Yen Liu')
```



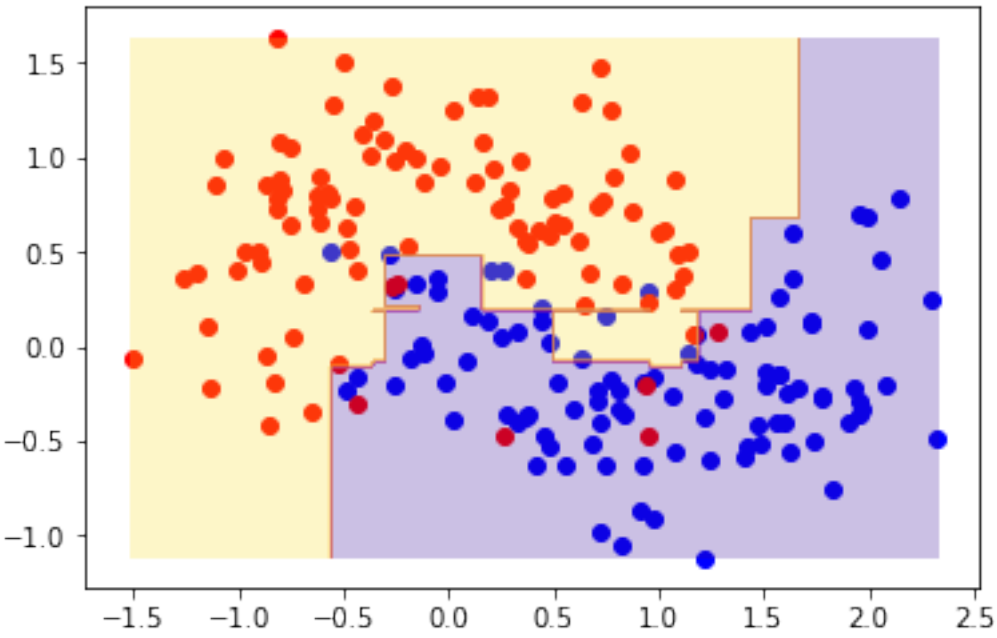

```
Refered from the link on Piazza ans collaborated with Chun-Yen Liu
```

```
In [99]: # large number of decision stumps that overfits
         plt.scatter(X_train[:,0], X_train[:,1], s=40, c=y_train, cmap=plt.cm.Spectral)
         for i in range(34):
             if dimension_temp[i] == 0:
                 plt.vlines(x=threshold_temp[i], ymin=min(Y_cor), ymax=max(Y_cor), linewidth=2
             else:
                 plt.hlines(y=threshold_temp[i], xmin=min(X_cor), xmax=max(X_cor), linewidth=2

         plt.show()
```



In [100]:
```
# large number of decision stumps that overfits
X_feature0 = X_test[:,0]
X_feature1 = X_test[:,1]
X_p = np.where(y_test==1)[0]
X_n = np.where(y_test==-1)[0]

h = 0.01
xx, yy = np.meshgrid(np.arange(min(X_test[:,0]), max(X_test[:,0]), h),
                     np.arange(min(X_test[:,1]), max(X_test[:,1]), h))
f = 0*xx
r = 0*xx+1
for i in range(34):
    if dimension_temp[i] == 0:
        if pnVSnp_temp[i] == 1:
            f += theta[i]
            f[:,np.where(xx[0,:]<=threshold_temp[i])[0]] -= 2*theta[i]
            r = np.sign(f)
        else:
            f += theta[i]
            f[:,np.where(xx[0,:]>threshold_temp[i])[0]] -= 2*theta[i]
            r = np.sign(f)
    else:
        if pnVSnp_temp[i] == 1:
            f += theta[i]
            f[np.where(yy[:,0]<=threshold_temp[i])[0],:] -= 2*theta[i]
            r = np.sign(f)
```

```
            else:
                f += theta[i]
                f[np.where(yy[:,0]>threshold_temp[i])[0],:] -= 2*theta[i]
                r = np.sign(f)

        plt.scatter(X_feature0[X_p], X_feature1[X_p],color = 'b', label='y=1')
        plt.scatter(X_feature0[X_n], X_feature1[X_n],color = 'r', label='y=-1')
        plt.contourf(xx, yy, r, alpha=.25, cmap=plt.cm.plasma)
        plt.show()
```



In [ ]: