In [47]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as mp
np.random.seed(100)
```

Using cvxopt library to solve the optimization problem of SVM

In [48]:
```
from cvxopt import matrix, solvers
```

In [49]:
```
x = np.random.uniform(low =0, high = 1, size=(100,2))
```

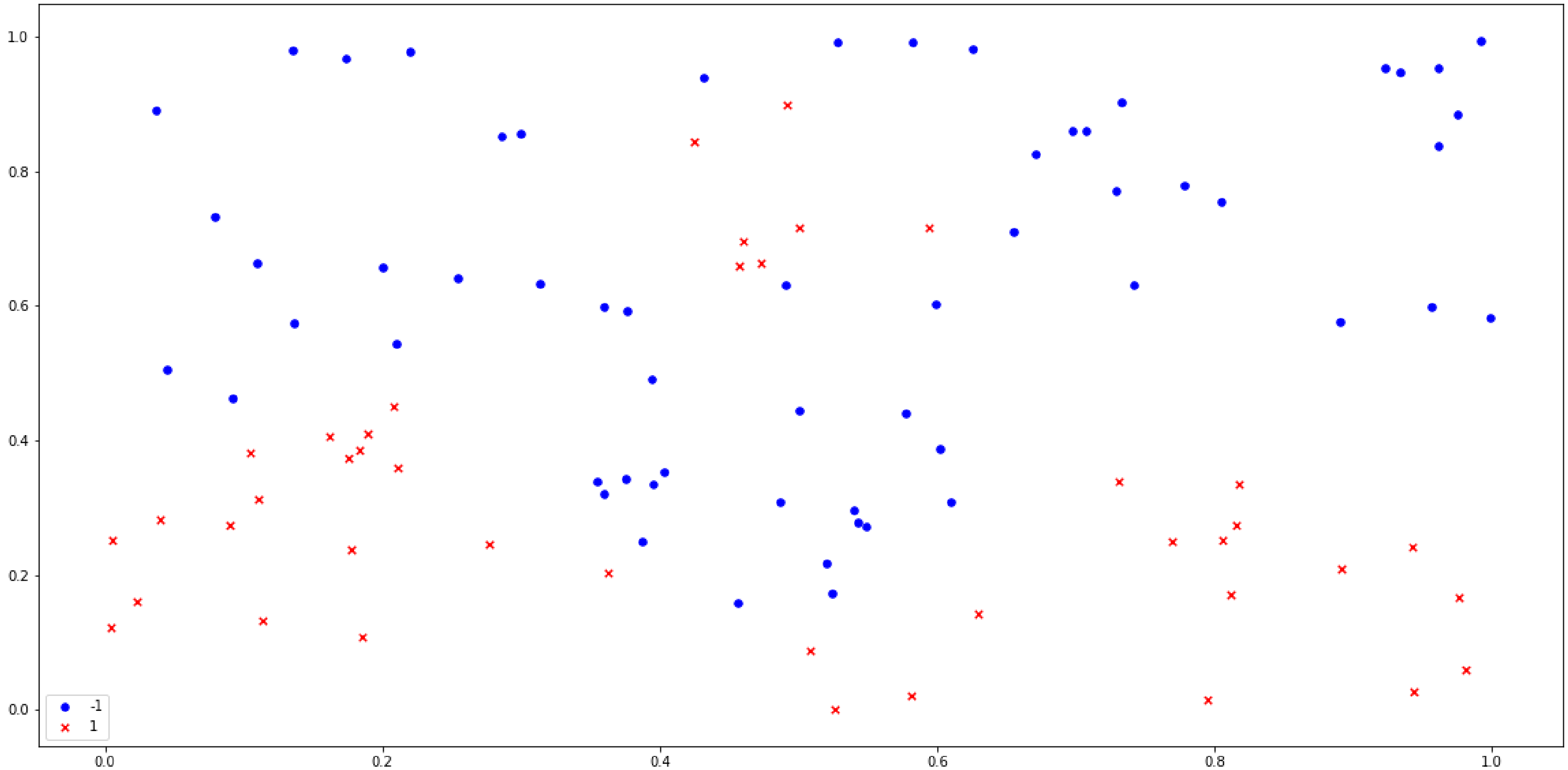
In [50]:
```
d =[]
colors =[]
for i in range(len(x)):
    if (x[i][1] < 0.2*(np.sin(10*x[i][0])) + 0.3) or ((x[i][1] - 0.8)**2 + (x[i][0]-0.5)**2 < 0.0225):
        d.append(1)
        colors.append("r")
    else:
        d.append(-1)
        colors.append("b")
d = np.asarray(d)
```

In [51]:
```
d.reshape(1,-1).shape
```

Out[51]:
```
(1, 100)
```

In [52]:
```
fig, ax = mp.subplots(figsize = (20,10))
cdict = {1: 'red', -1: 'blue'}
markers = {1:'x', -1:'o'}
for g in np.unique(d):
    ix = np.where(d == g)
    ax.scatter(x[:,0][ix], x[:,1][ix], c = cdict[g],marker = markers[g],label = g, s = 30)

ax.legend(loc = 'lower left')
```

Out[52]: <matplotlib.legend.Legend at 0x23f13959d68>



Kernel Function

I am using a polynomial kernel of degree 3

In [62]:
```
def kernel(X, Y):
#     K = np.zeros((X.shape[0], Y.shape[0]))

#     for i, x in enumerate(X):
#         for j, y in enumerate(Y):
#             K[i, j] = np.exp(-0.5*np.linalg.norm(x - y) ** 2)

    return (1+X.dot(Y.T))**3
```

Finding Alpha values using cvxopt library to solve the optimization problem of SVM

In [63]:
```
def train_svm():
    n, k = x.shape
    y_matrix = d.reshape(1, -1) * 1.
    H = np.dot(y_matrix.T, y_matrix) * kernel(x, x)
    P = matrix(H)
    q = matrix(-np.ones((n, 1)))
    G = matrix(np.diag(np.ones(n) * -1))
    h = matrix(np.zeros(n))
    A = matrix(y_matrix)
    b = matrix(np.zeros(1))

#     solvers.options['abstol'] = 1e-10
#     solvers.options['reltol'] = 1e-10
#     solvers.options['feastol'] = 1e-10

    return solvers.qp(P, q, G, h, A, b)
```

In [64]:
```
parameters = train_svm()
```

```
     pcost       dcost       gap    pres   dres
 0: -6.4862e+01 -1.9051e+02  4e+02  1e+01  3e+00
 1: -2.3482e+02 -3.8691e+02  2e+02  6e+00  2e+00
 2: -3.1394e+02 -4.9169e+02  2e+02  5e+00  1e+00
 3: -5.3124e+02 -7.5929e+02  3e+02  5e+00  1e+00
 4: -1.4735e+03 -1.7590e+03  3e+02  4e+00  1e+00
 5: -2.1726e+03 -2.5225e+03  4e+02  4e+00  1e+00
 6: -6.5390e+03 -7.2515e+03  7e+02  4e+00  1e+00
 7: -2.8736e+04 -3.0840e+04  2e+03  4e+00  1e+00
 8: -5.0928e+04 -5.4486e+04  4e+03  4e+00  1e+00
 9: -1.1426e+05 -1.2310e+05  9e+03  4e+00  1e+00
10: -2.5077e+05 -2.7840e+05  3e+04  4e+00  1e+00
11: -4.4253e+05 -5.0925e+05  7e+04  3e+00  1e+00
12: -7.9750e+05 -9.6717e+05  2e+05  3e+00  8e-01
13: -1.2592e+06 -1.5351e+06  3e+05  1e+00  4e-01
14: -1.3478e+06 -1.3794e+06  3e+04  9e-02  2e-02
15: -1.3483e+06 -1.3486e+06  3e+02  9e-04  2e-04
16: -1.3483e+06 -1.3483e+06  3e+00  9e-06  2e-06
17: -1.3483e+06 -1.3483e+06  3e-02  9e-08  2e-08
Optimal solution found.
```

Finding Bias using the alpha values

In [65]:
```
def bias(alphas):

# Threshold to find the support vectors
# Instead of zero tolerance, I am using some floating point tolerance

    threshold = 1e-5
    sv = (alphas > threshold).reshape(-1, )
    b = []
    for i in range(len(d[sv])):
        sum = 0
        for j in range(len(x)):
#             print(x[sv][i])
#             print(kernel(x[j].reshape(1,2), x[sv][i].reshape(1,2)))
#             print(f)
            sum += alphas[j]*d[j]*kernel(x[j].reshape(1,2), x[sv][i].reshape(1,2))
#             print(sum)
        b.append(d[sv][i] - sum)

#     print(b)
#     print(f)
    b = np.mean(b)
#     print(b)
    return b, sv
```

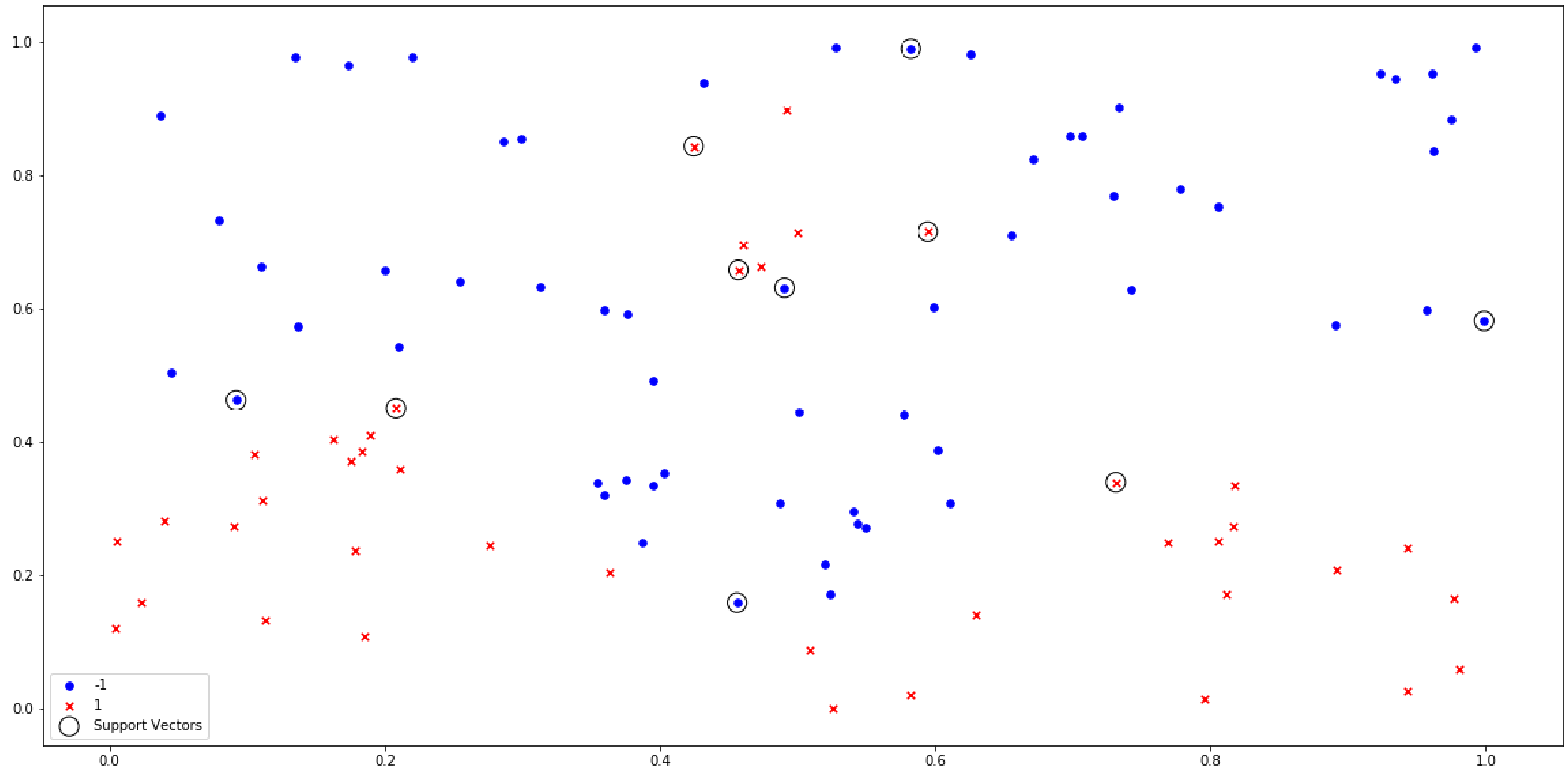
In [68]:
```
alphas = np.array(parameters['x'])[:, 0]
print(len(alphas))
b, sv = bias(alphas)

print('Alpha values of suppport vectors:', alphas[sv])
print('bias = ', b)
```

```
100
Alpha values of suppport vectors: [141624.83651654  97252.79769487 219453.89958003 114956.89032222
 105263.05297173  44391.42974999 551375.68098238 443111.91242398
  57475.12259771 921738.6130395 ]
bias =  228.77513944812785
```

Plotting support vectors

In [73]:
```
fig, ax = mp.subplots(figsize = (20,10))
cdict = {1: 'red', -1: 'blue'}
markers = {1:'x', -1:'o'}
for g in np.unique(d):
    ix = np.where(d == g)
    ax.scatter(x[:,0][ix], x[:,1][ix], c = cdict[g],marker = markers[g],label = g, s = 30)

support_vectors = x[sv]
ax.scatter(support_vectors[:,0], support_vectors[:,1], s=200, facecolors='none', edgecolors='black', label =
"Support Vectors")
ax.legend(loc='lower left')
```

Out[73]: <matplotlib.legend.Legend at 0x23f142d9da0>

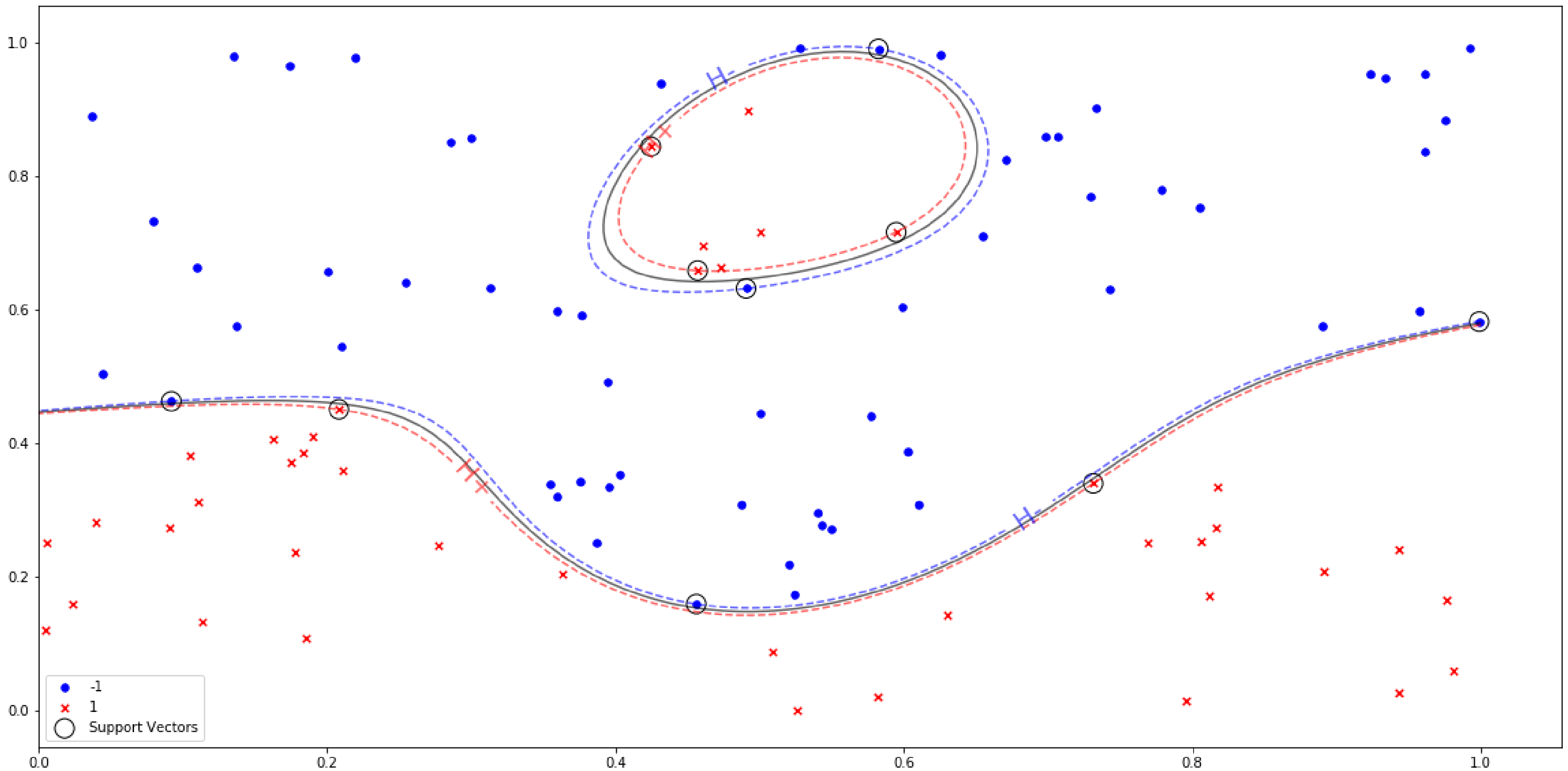


Finding the three hyperplanes

In [74]:
```
def decision_function(x, y):
    p =[]
    for i in x:
        for j in y:
            p.append(kernel(i,j))
    return np.array(p).reshape(x.shape[0], y.shape[0])
```

In [76]:
```
xx =  np.linspace(0,1,100)
yy = np.linspace(0,1,100)
XX, YY = np.meshgrid(xx, yy)
# print(XX.shape)
xy = np.c_[XX.ravel(), YY.ravel()]
#print(xy)

df = decision_function(x,xy)
#print(((alphas*d).reshape(1,XX.shape[0])).dot(kxy))
Z = ((((alphas*d).reshape(1,XX.shape[0])).dot(df)) +b).reshape(XX.shape)

fig, ax = mp.subplots(figsize = (20,10))
contour  = ax.contour(XX, YY, Z, colors=['blue', 'black', 'red'], levels = [-1,0,1], alpha = 0.6, linestyles
= ['--','-','--'])

fmt = {}
strs = ['H-', 'H', 'H+']
for l, s in zip(contour.levels, strs):
    fmt[l] = s

# Label every other level using strings
ax.clabel(contour, contour.levels[::2], inline = True,inline_spacing = 10, fmt=fmt, fontsize=20)



cdict = {1: 'red', -1: 'blue'}
markers = {1:'x', -1:'o'}
for g in np.unique(d):
    ix = np.where(d == g)
    ax.scatter(x[:,0][ix], x[:,1][ix], c = cdict[g],marker = markers[g],label = g, s = 30)

# ax.scatter(x[:,0], x[:,1], c= colors, s=10, label = ('blue'))
# ax.scatter(x[:,0], x[:,1], c= colors, s=10, label = ('red'))

# ax.legend()
support_vectors = x[sv]
ax.scatter(support_vectors[:,0], support_vectors[:,1], s=200, facecolors='none', edgecolors='black', label =
"Support Vectors")
ax.legend(loc='lower left')
```

Out[76]: <matplotlib.legend.Legend at 0x23f14971ba8>



In [45]:
```
# def af(x):
#     if x>= 0:
#         return 1
#     else:
#         return -1
```

In [ ]:

```
# def predict(x,y):

#     f =[]
#     for k in range(len(x)):
#         p =[]
#         for i in range(len(x[0])):
#             q = np.asarray([x[0][i], y[0][i]])
#             sum = 0
#             for j in range(len(x[0])):
#                 r = np.asarray([x[0][j], y[0][j]])
#                 sum += alphas[j] * d[j] * kernel(r.reshape(1,2), q.reshape(1,2))
#             p.append(af(sum+b))
#         f.append(p)

#     return np.asarray(f).T
```

In [ ]:

```
# u = []
# v = []
# for i in range(len(x)):
#     if predict(b, x[i]) == 1:
#         u.append(x[i])
#     elif predict(b,x[i]) == -1:
#         v.append(x[i])
# u = np.asarray(u)
# v = np.asarray(v)
```

In [119]:
```
# import numpy as np
# import matplotlib.pyplot as plt
# from sklearn import svm
# from sklearn.datasets import make_blobs


# # we create 40 separable points
# X, y = make_blobs(n_samples=40, centers=2, random_state=6)

# # fit the model, don't regularize for illustration purposes
# clf = svm.SVC(kernel='linear', C=1000)
# clf.fit(X, y)

# plt.scatter(X[:, 0], X[:, 1], c=y, s=30, cmap=plt.cm.Paired)

# # plot the decision function
# ax = plt.gca()
# xlim = ax.get_xlim()
# ylim = ax.get_ylim()

# # create grid to evaluate model
# xx = np.linspace(xlim[0], xlim[1], 30)
# yy = np.linspace(ylim[0], ylim[1], 30)
# YY, XX = np.meshgrid(yy, xx)
# # print(xx)
# # print(XX)
# # print(o)
# xy = np.vstack([XX.ravel(), YY.ravel()]).T
# Z = clf.decision_function(xy).reshape(XX.shape)
# # print(Z)
# # print(f)

# # plot decision boundary and margins
# ax.contour(XX, YY, Z, colors='k', levels=[-1, 0, 1], alpha=0.5,
#            linestyles=['--', '-', '--'])
# # plot support vectors
# ax.scatter(clf.support_vectors_[:, 0], clf.support_vectors_[:, 1], s=100,
#            linewidth=1, facecolors='none', edgecolors='k')
# plt.show()
```

In [ ]:

In [ ]: