Problem 8.13 - (a) Plot the classification regions for the final hypothesis in X space

In [93]:

```
#import libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import random
%matplotlib inline
```

In [4]:

```
#input data points
X1 = [-0.494,0.311,0.0064,-0.0089,0.0014,-0.189,0.085,0.171,0.142]
X1
```

Out[4]:

```
[-0.494, 0.311, 0.0064, -0.0089, 0.0014, -0.189, 0.085, 0.171, 0.142]
```

In [5]:

```
y1=[0.363,-0.101,0.374,-0.173,0.138,0.718,0.32208,-0.302,0.568]
y1
```

Out[5]:

```
[0.363, -0.101, 0.374, -0.173, 0.138, 0.718, 0.32208, -0.302, 0.568]
```

In [6]:

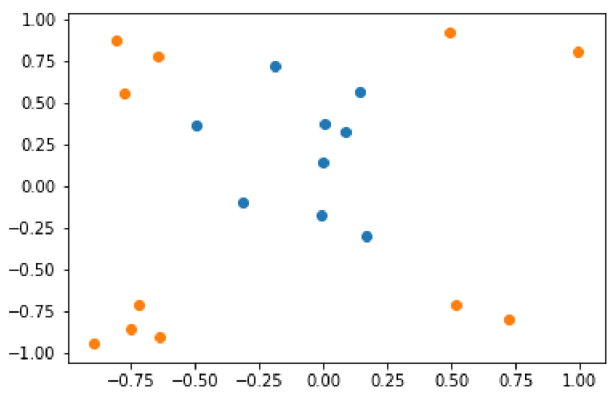
```
x2 = [0.491, -0.892, -0.721, 0.519, -0.775, -0.646,-0.803, 0.994, 0.724, -0.748,-0.635]
y2 = [0.920, -0.946, -0.710, -0.715, 0.551, 0.773, 0.875, 0.801, -0.795,-0.853, -0.905]
```

In [23]:

```
plt.scatter(x1,y1)
plt.scatter(x2,y2)
```

Out[23]:

```
<matplotlib.collections.PathCollection at 0x217b9584470>
```



In [7]:

```
#data points with bias
X1 = np.array([[1,-0.494,0.363],
[1,-0.311,-0.101],
[1,0.0064,0.374],
[1,-0.0089,-0.173],
[1,0.0014,0.138],
[1,-0.189,0.718],
[1,0.085,0.32208],
[1,0.171,-0.302],
[1,0.142,0.568],
[1,0.491,  0.920],
[1,-0.892, -0.946],
[1,-0.721, -0.710],
[1,0.519, -0.715],
[1,-0.775, 0.551],
[1,-0.646,0.773],
[1,-0.803, 0.875],
[1,0.994, 0.801],
[1,0.724,-0.795],
[1,-0.748,-0.853],
[1,-0.635,-0.905]])
y1 = [1,1,1,1,1,1,1,1,1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1]
```

In [8]:

```
from numpy.linalg import inv
```

In [94]:

```
data = np.array([[1,-0.494,0.363,1],
[1,-0.311,-0.101,1],
[1,0.0064,0.374,1],
[1,-0.0089,-0.173,1],
[1,0.0014,0.138,1],
[1,-0.189,0.718,1],
[1,0.085,0.32208,1],
[1,0.171,-0.302,1],
[1,0.142,0.568,1],
[1,0.491,  0.920,-1],
[1,-0.892, -0.946,-1],
[1,-0.721, -0.710,-1],
[1,0.519, -0.715,-1],
[1,-0.775, 0.551,-1],
[1,-0.646,0.773,-1],
[1,-0.803, 0.875,-1],
[1,0.994, 0.801,-1],
[1,0.724,-0.795,-1],
[1,-0.748,-0.853,-1],
[1,-0.635,-0.905,-1]])
data
```

Out[94]:

```
array([[ 1.     , -0.494  ,  0.363  ,  1.     ],
       [ 1.     , -0.311  , -0.101  ,  1.     ],
       [ 1.     ,  0.0064 ,  0.374  ,  1.     ],
       [ 1.     , -0.0089 , -0.173  ,  1.     ],
       [ 1.     ,  0.0014 ,  0.138  ,  1.     ],
       [ 1.     , -0.189  ,  0.718  ,  1.     ],
       [ 1.     ,  0.085  ,  0.32208,  1.     ],
       [ 1.     ,  0.171  , -0.302  ,  1.     ],
       [ 1.     ,  0.142  ,  0.568  ,  1.     ],
       [ 1.     ,  0.491  ,  0.92   , -1.     ],
       [ 1.     , -0.892  , -0.946  , -1.     ],
       [ 1.     , -0.721  , -0.71   , -1.     ],
       [ 1.     ,  0.519  , -0.715  , -1.     ],
       [ 1.     , -0.775  ,  0.551  , -1.     ],
       [ 1.     , -0.646  ,  0.773  , -1.     ],
       [ 1.     , -0.803  ,  0.875  , -1.     ],
       [ 1.     ,  0.994  ,  0.801  , -1.     ],
       [ 1.     ,  0.724  , -0.795  , -1.     ],
       [ 1.     , -0.748  , -0.853  , -1.     ],
       [ 1.     , -0.635  , -0.905  , -1.     ]])
```

In [124]:

```
# feature transformation - for third order polynomial
```

```
def transform(data):
    result = []
    for i in data:
        x1 = i[1]
        x2 = i[2]
        flag = i[-1]
        x = [1, x1, x2, x1 * x2, x1**2, x2**2, x1**3, (x1**2)*x2, x1*(x2**2), x2**3, flag]
        result.append(x)
    return np.array(result)
newdata = transform(data)
def f(x,y,w):
    return w[0] + w[1]*x + w[2]*y + w[3]*x*y + w[4]*(x**2) + w[5]*(y**2) + w[6]*(x**3) + w[7]*(x**2)*y
+ w[8]*x*(y**2) + w[9]*y**3
```

In [95]:

```
# number of data points
n = 1000
x = np.linspace(-1, 1, n)
y = np.linspace(-1, 1, n)
X, Y = np.meshgrid(x, y)
```

In [253]:

```
# pre-processing data
Xnew = newdata[:,:-1]
Ynew = newdata[:,-1]

#weights after transformation to third order
w1 = inv(Xnew.T.dot(Xnew)).dot(Xnew.T).dot(Ynew)
w1
```

Out[253]:

```
array([ 1.19974401,  0.40820538,  0.86460535,  0.09728382, -0.85414543,
       -2.52854105,  2.04534985, -2.01173905, -2.46045065,  0.44509924])
```

In [96]:

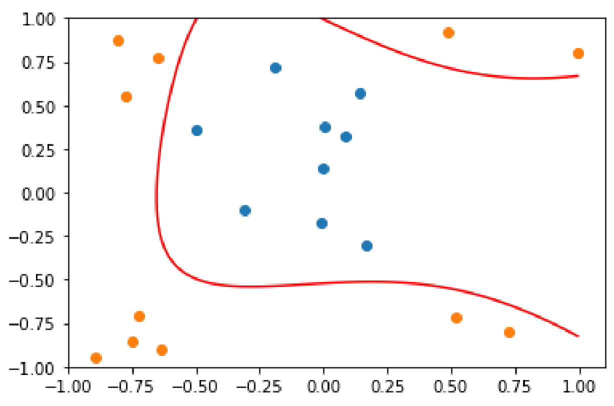
```
A1 = [-0.494,-0.311,0.0064,-0.0089,0.0014,-0.189,0.085,0.171,0.142]
A2=[0.363,-0.101,0.374,-0.173,0.138,0.718,0.32208,-0.302,0.568]
B1 = [0.491, -0.892, -0.721, 0.519, -0.775, -0.646,-0.803, 0.994, 0.724, -0.748,-0.635]
B2 = [0.920, -0.946, -0.710, -0.715, 0.551, 0.773, 0.875, 0.801, -0.795,-0.853, -0.905]
```

In [256]:

```
plt.scatter(A1,A2)
plt.scatter(B1,B2)
plt.contour(X, Y, f(X, Y,w1), 1, colors = 'red')
```

Out[256]:

```
<matplotlib.contour.QuadContourSet at 0x217e9b6a2e8>
```



The above plot shows the Third order classification by final hypothesis

In [264]:

```
# feature transformation - for second oder polynomial
def transform(data):
    result = []
    for i in data:
        x1 = i[1]
        x2 = i[2]
        flag = i[-1]
        x = [1, x1, x2, x1**2, x1*x2, x2**2, flag]
        result.append(x)
    return np.array(result)

newdata = transform(data)
def f(x,y,w):
    return w[0] + w[1]*x + w[2]*y + w[3]*x**2 + w[4]*(x*y) + w[5]*(y**2)
```

In [265]:

```
# pre-processing data
Xnew = newdata[:,:-1]
Ynew = newdata[:,-1]

#weights for transformed second order data
w2 = inv(Xnew.T.dot(Xnew)).dot(Xnew.T).dot(Ynew)
w2
```

Out[265]:

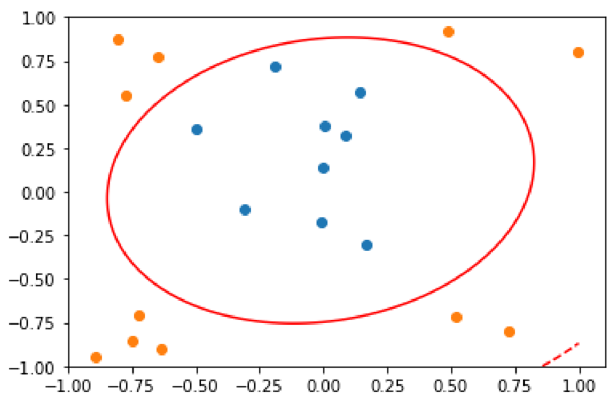
```
array([ 1.15098545, -0.0627625 ,  0.23103786, -1.68671156,  0.43419817,
       -1.75129786])
```

In [266]:

```
plt.scatter(A1,A2)
plt.scatter(B1,B2)
plt.contour(X, Y, f(X, Y,w2), 1, colors = 'red')
```

Out[266]:

```
<matplotlib.contour.QuadContourSet at 0x21780595f28>
```



The above plot shows the Second order classification by final hypothesis

Problem 8.13(b) - The third order polynomial seems to be overfitted because of the higher order nature of the final hypothesis

Problem 8.13(c) - pseudo inverse algorithm using lambda = 1

In [186]:

```
# feature transformation - for third orderr polynomial
def transform(data):
    result = []
    for i in data:
        x1 = i[1]
        x2 = i[2]
        flag = i[-1]
        x = [1, x1, x2, x1 * x2, x1**2, x2**2, x1**3, (x1**2)*x2, x1*(x2**2), x2**3, flag]
        result.append(x)
    return np.array(result)

newdata1 = transform(data)

def f(x,y,w):
    return w[0] + w[1]*x + w[2]*y + w[3]*x*y + w[4]*(x**2) + w[5]*(y**2) + w[6]*(x**3) + w[7]*(x**2)*y
+ w[8]*x*(y**2) + w[9]*y**3
```

In [187]:

```
# pre-processing data
Xnew = newdata1[:,:-1]
Ynew = newdata1[:,-1]

#Set lambda =1
#calculate new weights for transformed data for third order with lambda =1
l=1
w3 = inv(Xnew.T.dot(Xnew) + l*(Xnew.T.dot(Xnew))).dot(Xnew.T).dot(Ynew)
w3
```

Out[187]:

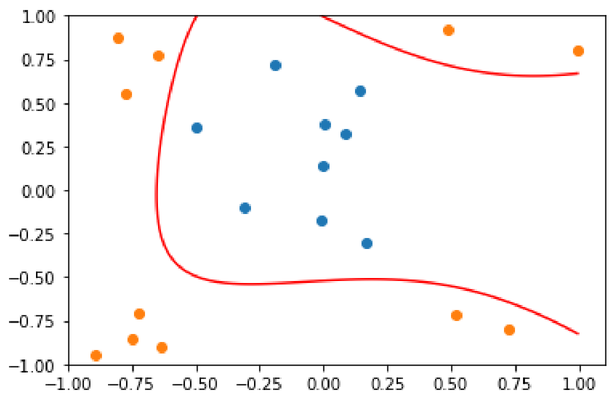
```
array([ 0.599872  ,  0.20410269,  0.43230267,  0.04864191, -0.42707272,
       -1.26427053,  1.02267492, -1.00586952, -1.23022533,  0.22254962])
```

In [189]:

```
plt.scatter(A1,A2)
plt.scatter(B1,B2)
plt.contour(X, Y, f(X, Y,w3), 1, colors = 'red')
```

Out[189]:

```
<matplotlib.contour.QuadContourSet at 0x17fcfd4c9e8>
```



From the above plots we can see that after regularization, the weights have changed -

CASE1 - [1.19974401, 0.40820538, 0.86460535, 0.09728382, -0.85414543, -2.52854105, 2.04534985, -2.01173905, -2.46045065, 0.44509924] CASE2- [0.599872 , 0.20410269, 0.43230267, 0.04864191, -0.42707272, -1.26427053, 1.02267492, -1.00586952, -1.23022533, 0.22254962]

In [66]:

```
# feature transformation - for 2nd order
def transform(data):
    result = []
    for i in data:
        x1 = i[1]
        x2 = i[2]
        flag = i[-1]
        x = [1, x1, x2, x1**2, x1*x2, x2**2, flag]
        result.append(x)
    return np.array(result)

newdata3 = transform(data)


def f(x,y,w):
    return w[0] + w[1]*x + w[2]*y + w[3]*x**2 + w[4]*(x*y) + w[5]*(y**2)
```

In [70]:

```
# pre-processing data
Xnew = newdata3[:,:-1]
Ynew = newdata3[:,-1]
l=1
w4 = inv(Xnew.T.dot(Xnew) + l*(Xnew.T.dot(Xnew))).dot(Xnew.T).dot(Ynew)
w4
```

Out[70]:

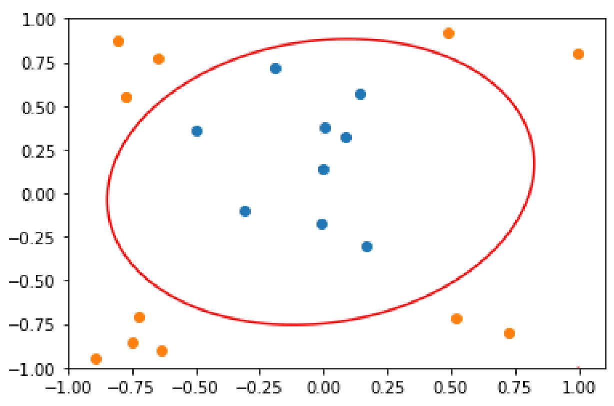
```
array([ 0.57549273, -0.03138125,  0.11551893, -0.84335578,  0.21709908,
       -0.87564893])
```

In [74]:

```
plt.scatter(A1,A2)
plt.scatter(B1,B2)
plt.contour(X, Y, f(X, Y,w4), 1, colors = 'red')
```

Out[74]:

```
<matplotlib.contour.QuadContourSet at 0x17fcc2b1470>
```



From the above plots we can see that after regularization, the weights have changed -

CASE1 - [1.15098545, -0.0627625 , 0.23103786, -1.68671156, 0.43419817,-1.75129786]

CASE2- [0.57549273, -0.03138125, 0.11551893, -0.84335578, 0.21709908,-0.87564893]

Problem 8.13(d) Use SVM to find the classifier, and compare the results with (a) and (c).

In [97]:

```
from sklearn.svm import SVC
```

In [191]:

```
model7 =SVC(decision_function_shape='ovo')
```

In [192]:

```
#take the x1 and y1 values from the data
a=X1[:,1:]
```

In [193]:

```
#labels
y1 = np.array(y1)
```

In [194]:

```
model7.fit(a, y1)
```

Out[194]:

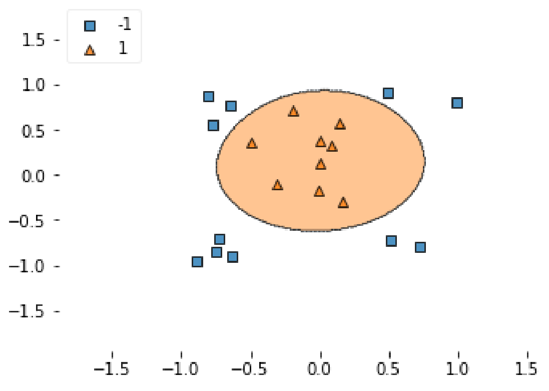
```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovo', degree=3, gamma='auto', kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
```

In [195]:

```
from mlxtend.plotting import plot_decision_regions
plot_decision_regions(X=a,
                      y=y1,
                      clf=model7,
                      legend=2)
```

Out[195]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x17fcfdb63c8>
```



The above plot is drawn for SVM with decision regions. Below, I have plotted for third order polynomial and second order polynomial

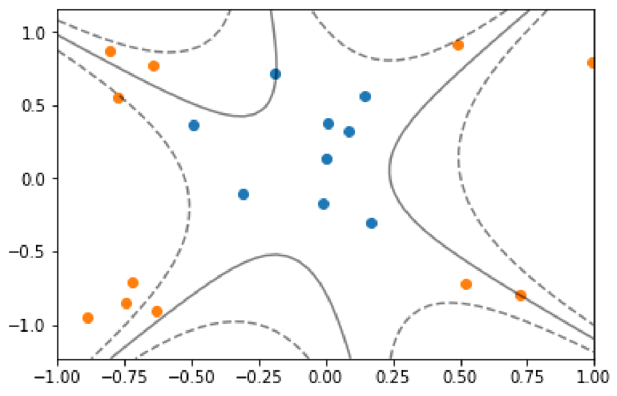
In [196]:

```
model7 =SVC(C=1000,kernel = 'poly',degree= 3)
```

```
model7.fit(a,y1)
xlim,xmax = plt.xlim(-1,1)
a = -w[0] / w[1]
xx = np.linspace(ymin, ymax)
yy = a * xx - (model7.intercept_[0]) / w[1]
#plt.plot(yy,xx)
YY, XX = np.meshgrid(yy, xx)
xy = np.vstack([XX.ravel(), YY.ravel()]).T
Z = model7.decision_function(xy).reshape(XX.shape)
plt.contour(xx, yy,Z,colors='k', levels=[-1, 0, 1], alpha=0.5,linestyles=['--', '-', '--'])
plt.scatter(A1,A2)
plt.scatter(B1,B2)
```

Out[196]:

```
<matplotlib.collections.PathCollection at 0x17fcfe8f7f0>
```

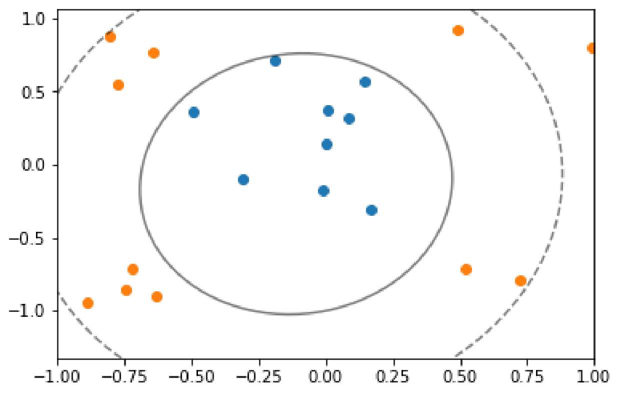


In [185]:

```
model7 =SVC(C=1,kernel = 'poly',degree= 2)
model7.fit(a,y1)
xlim,xmax = plt.xlim(-1,1)
a = -w[0] / w[1]
xx = np.linspace(ymin, ymax)
yy = a * xx - (model7.intercept_[0]) / w[1]
#plt.plot(yy,xx)
YY, XX = np.meshgrid(yy, xx)
xy = np.vstack([XX.ravel(), YY.ravel()]).T
Z = model7.decision_function(xy).reshape(XX.shape)
plt.contour(xx, yy,Z,colors='k', levels=[-1, 0, 1], alpha=0.5,linestyles=['--', '-', '--'])
plt.scatter(A1,A2)
plt.scatter(B1,B2)
```

Out[185]:

```
<matplotlib.collections.PathCollection at 0x17fcfd77a90>
```



In [306]:

```
# question 8.5
```

Problem 8.5 (a) Generate points

In [34]:

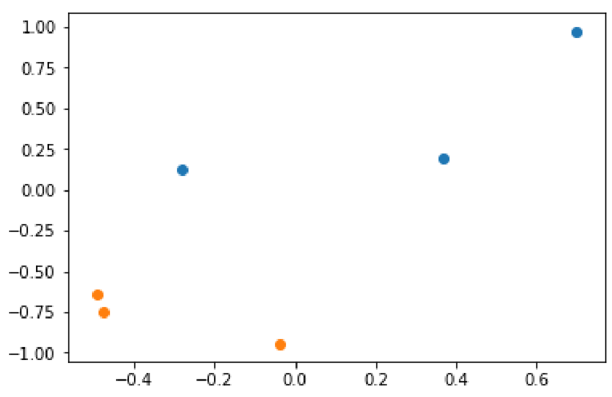
```
count =0
val_x1 = np.zeros(3)
val_y1 = np.zeros(3)
val_x = np.zeros(6)
val_x.reshape(3,2)
while(count<3):
    df_x = np.random.uniform(-1,1)
    df_y = np.random.uniform(0,1)
    val_x1[count] = df_x
    val_y1[count] = df_y
    count +=1
val_x1=val_x1.reshape(3,1)
val_y1=val_y1.reshape(3,1)
val_x=np.concatenate([val_x1,val_y1],axis=1)
count =0
val_x2 = np.zeros(3)
val_y2 = np.zeros(3)
while(count<3):
    df_x = np.random.uniform(-1,0)
    df_y = np.random.uniform(-1,0)
    val_x2[count] = df_x
    val_y2[count] = df_y
    count +=1
val_x2=val_x2.reshape(3,1)
val_y2=val_y2.reshape(3,1)
val_y=np.concatenate([val_x2,val_y2],axis=1)
```

In [35]:

```
plt.scatter(val_x1,val_y1)
plt.scatter(val_x2,val_y2)
```

Out[35]:

```
<matplotlib.collections.PathCollection at 0x2555aef72e8>
```



In [36]:

```
X = np.vstack([val_x,val_y])
y = np.array([1,1,1,-1,-1,-1])
label = X[:,1:2]
```

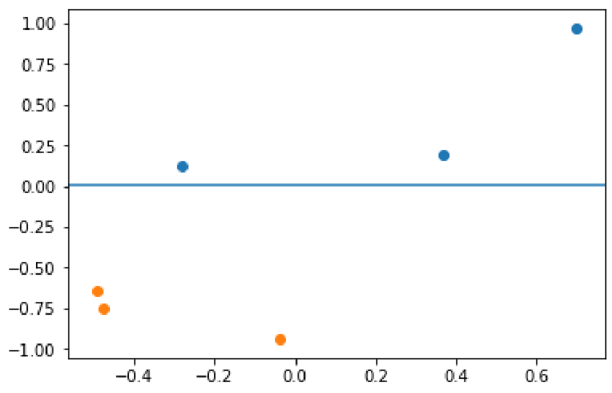
Problem 3.5(b) Create Plots

In [39]:

```
import random
check = True

def generate_random_points():
    while True:
        rand=random.uniform(-1,1)
        check=False
        for i in range(6):
            if((np.sign(y[i]-rand)*label[i])>0):
                check=True
            else:
                check = False
                break

        if(check==True):
            return rand
            break

random=generate_random_points()
plt.axhline(y=random)
plt.scatter(val_x1,val_y1)
plt.scatter(val_x2,val_y2)
```

Out[39]:

```
<matplotlib.collections.PathCollection at 0x2555c1c17b8>
```



The plot above shows a random hypothesis

In [40]:

```
model2 =SVC(kernel = 'linear', C=1000)
```

In [41]:

```
model2.fit(X,y)
```

Out[41]:

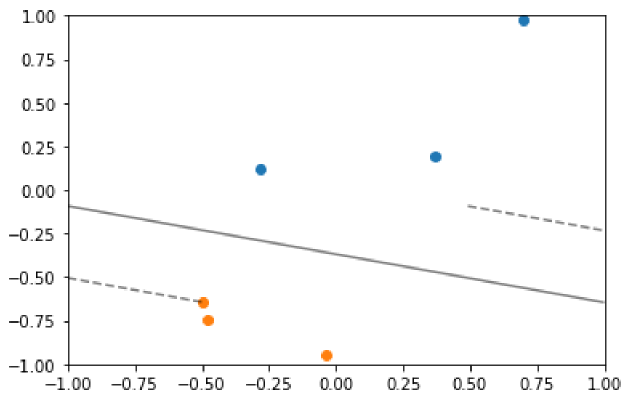
```
SVC(C=1000, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
```

In [55]:

```
w = model2.coef_[0]
xlim,xmax = plt.xlim(-1,1)
ymin,ymax = plt.ylim(-1,1)
a = -w[0] / w[1]
xx = np.linspace(ymin, ymax)
yy = a * xx - (model2.intercept_[0]) / w[1]
#plt.plot(yy,xx)
YY, XX = np.meshgrid(yy, xx)
xy = np.vstack([XX.ravel(), YY.ravel()]).T
Z = model2.decision_function(xy).reshape(XX.shape)
plt.contour(XX, YY, Z, colors='k', levels=[-1, 0, 1], alpha=0.5,linestyles=['--', '-', '--'])
plt.scatter(val_x1,val_y1)
plt.scatter(val_x2,val_y2)
```

Out[55]:

```
<matplotlib.collections.PathCollection at 0x2555d484048>
```



The plot above shows the hypothesis using SVM

Problem 8.5(c) -repeat for million points

In [57]:

```
#Generate data and run a million times and collect the alphas for random hypothesis as well as SVM
import random
```

In [63]:

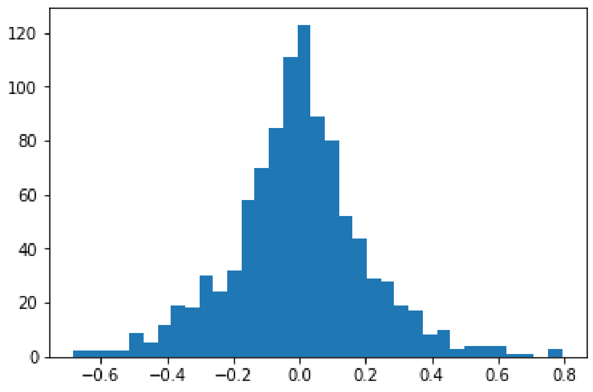
```
alpha_random = []
alpha_svm = []
num = 0
while(num<1000000):
    count =0
    val_x1 = np.zeros(3)
    val_y1 = np.zeros(3)
    val_x = np.zeros(6)
    val_x.reshape(3,2)
    while(count<3):
        df_x = np.random.uniform(-1,1)
        df_y = np.random.uniform(0,1)
        val_x1[count] = df_x
        val_y1[count] = df_y
        count +=1
    val_x1=val_x1.reshape(3,1)
    val_y1=val_y1.reshape(3,1)
    val_x=np.concatenate([val_x1,val_y1],axis=1)


    count =0
    val_x2 = np.zeros(3)
    val_y2 = np.zeros(3)
    while(count<3):
        df_x = np.random.uniform(-1,1)
        df_y = np.random.uniform(-1,0)
        val_x2[count] = df_x
        val_y2[count] = df_y
        count +=1
    val_x2=val_x2.reshape(3,1)
    val_y2=val_y2.reshape(3,1)
    val_y=np.concatenate([val_x2,val_y2],axis=1)
    val_y
    X = np.vstack([val_x,val_y])
    y = np.array([1,1,1,-1,-1,-1])

    score_random = generate_random_points()
    alpha_random.append(score_random)
    model2 =SVC(kernel = 'linear', C=1)
    model2.fit(X,y)
    score_svc=model2.intercept_[0]
    alpha_svm.append(score_svc)
    num +=1
```

# Alphas for hypothesis for random hypothesis and SVM are stored in alpha_random an alpha_svm respectively

Problem 8.5(d)- Plot histograms

In [113]:

```
plt.hist(alpha_random,bins='auto')
plt.show()
```

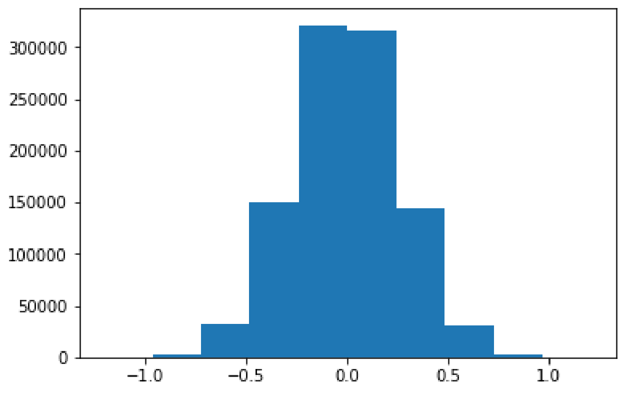


In [68]:

```
plt.hist(alpha_svm)
```

Out[68]:

```
(array([6.20000e+01, 2.53200e+03, 3.26550e+04, 1.49669e+05, 3.21178e+05,
        3.16662e+05, 1.44508e+05, 3.04830e+04, 2.20300e+03, 4.80000e+01]),
 array([-1.20163173, -0.96059429, -0.71955684, -0.47851939, -0.23748194,
         0.00355551,  0.24459295,  0.4856304 ,  0.72666785,  0.9677053 ,
         1.20874275]),
 <a list of 10 Patch objects>)
```



In [61]:

```
x = np.array(alpha_random)
y= np.array(alpha_svm)
```

In [13]:

```
x.shape
```

Out[13]:

```
(1000000,)
```

In [16]:

```
y.shape
```

Out[16]:

```
(1000000,)
```

From the above 2 plots we can infer that for a million points, the alphas for random hypothesis are mostly distributed randomly between -1 and 1 and for SVM also , the alphas are distributed from -1 to 1 quite uniformly like a normal distribution. SVM is centered toward the middle as it separates the data with equal margin.