In [41]:
```
import numpy as np
import random
import matplotlib.pyplot as plt
import sklearn.svm as svm
```

In [42]:
```
def get_points(num, minval = -1.0, maxval = 1.0, dim = 2):
    points = np.asarray([[None for i in range(dim)] for j in range(num)])
    for i in range(dim):
        for j in range(num):
            points[j,i] = random.uniform(minval, maxval)
    return points
```

In [43]:
```
def get_target_fn(points):
    # y = mx + b
    #returns m, b
    x1 = points[0,0]
    y1 = points[0,1]
    x2 = points[1,0]
    y2 = points[1,1]
    m = (y2-y1)/(x2-x1)
    b = y1 - m*(x1)
    return m, b
```

In [44]:
```
def get_y(x,m,b):
    return m*x + b
```

In [45]:
```
def init_weights(dim = 2):
    w = np.zeros(dim+1)
    return w
```

In [46]:
```
def eval_target_fn(datapoints,m,b):
    y = np.zeros(len(datapoints))
    for i in range(len(datapoints)):
        if datapoints[i,1] < datapoints[i,0]*m + b:
            y[i] = -1
        else:
            y[i] = 1
    return y
```

In [47]:
```
def get_sign(vec):
    sign = np.zeros(len(vec))
    for i in range(len(vec)):
        if vec[i] == 0:
            sign[i] = 0.0
        else:
            if vec[i] < 0:
                sign[i] = -1.0
            else:
                sign[i] = 1.0
    return sign
```

In [48]:
```
def get_misclassified_point_idx(sign, y):
    idx = []
    for i in range(len(sign)):
        if sign[i] != y[i]:
            idx.append(i)
    return idx
```

In [49]:
```
def update_weights(w,x,y,idx):
    w = w + y[idx]*x[idx]
    return w
```

In [50]:
```
def add_bias_to_x(x):
    bias = np.ones(len(x[:,0]))
    x = np.c_[x,bias]
    return x
```

In [51]:
```
def get_target(N,minval = -1.0, maxval = 1.0, dim = 2):

    target_fn_points = get_points(dim)
    m, b  = get_target_fn(target_fn_points)
    x = get_points(N)
    y = eval_target_fn(x,m,b)
    return m,b,x,y
```

In [52]:
```
def estimate_disagreement(w, m, b, num_test_points = 100):
    test_x = get_points(num_test_points)
    test_y = eval_target_fn(test_x,m,b)
    test_x = add_bias_to_x(test_x)
    vec = w.dot(test_x.transpose())
    sign = get_sign(vec)
    idx = get_misclassified_point_idx(sign,test_y)
    disagreement_prob = float(len(idx))/float(num_test_points)
    return disagreement_prob
```

In [64]:
```
def estimate_svm_disagreement(clf, m, b, num_test_points = 100):
    test_x = get_points(num_test_points)
    test_y = eval_target_fn(test_x,m,b)
    sign = clf.predict(test_x)

    idx = get_misclassified_point_idx(sign,test_y)
    disagreement_prob = float(len(idx))/float(num_test_points)
    return disagreement_prob
```

In [53]:
```
def plot_target(m,b,x,y):
    plt.plot((-1,1), (get_y(-1,m,b), get_y(1,m,b)))
    plt.xlim([-1,1])
    plt.ylim([-1,1])
    neg_x = [x[i,0] for i in range(len(x[:,0])) if y[i] < 0]
    pos_x = [x[i,0] for i in range(len(x[:,0])) if y[i] > 0]
    neg_y = [x[i,1] for i in range(len(x[:,1])) if y[i] < 0]
    pos_y = [x[i,1] for i in range(len(x[:,1])) if y[i] > 0]
    plt.plot(neg_x, neg_y,'r*')
    plt.plot(pos_x, pos_y,'b*')
    plt.title('Target data')
    plt.show()
```

In [54]:
```
def plot_lr(m,b,x,y, iteration):
    plt.plot((-1,1), (get_y(-1,m,b), get_y(1,m,b)), '--')
    plt.xlim([-1,1])
    plt.ylim([-1,1])
    neg_x = [x[i,0] for i in range(len(data[:,0])) if y[i] < 0]
    pos_x = [x[i,0] for i in range(len(data[:,0])) if y[i] > 0]
    neg_y = [x[i,1] for i in range(len(data[:,1])) if y[i] < 0]
    pos_y = [x[i,1] for i in range(len(data[:,1])) if y[i] > 0]
    plt.plot(neg_x, neg_y,'r*')
    plt.plot(pos_x, pos_y,'b*')

    plt.title('PLA iteration '+str(iteration))
    plt.show()
```

In [55]:
```
def PLA(m,b,x,y,w,threshold = 0.1, minval = -1.0, maxval = 1.0, dim = 2):
    iter_count = 0
    idx = list(range(len(y)))
    while len(idx) > 0: #while there are misclassified points
        iter_count += 1
        vec = w.dot(x.transpose())
        sign = get_sign(vec)
        idx = get_misclassified_point_idx(sign,y)
        if len(idx) > 0:
            w = update_weights(w,x,y,random.choice(idx))
    disagreement_prob = estimate_disagreement(w,m,b)
    return iter_count, disagreement_prob
```

In [74]:
```
N = 10
iterations = 1000
num_iters = []
disagreement_probs = []
svm_disagree_probs = []
n_support_vecs = []
for i in range(iterations):
    m,b,x,y = get_target(N)
    while len(np.unique(y)) <2:
        m,b,x,y = get_target(N)
    #do PLA
    x_with_bias = add_bias_to_x(x)
    w = init_weights()
    iter_count, disagreement = PLA(m,b,x_with_bias,y,w)
    num_iters.append(iter_count)
    disagreement_probs.append(disagreement)
    #do SVM
    clf = svm.SVC(C = 10e20, kernel = 'linear')
    clf.fit(x, y)
    svm_disagree_probs.append(estimate_svm_disagreement(clf,m,b))
    n_support_vecs.append(len(clf.support_))

print(disagreement_probs[0])
print(np.mean(num_iters))
print(np.mean(disagreement_probs))
print(np.mean(svm_disagree_probs))
print(sum([1 for i in range(len(svm_disagree_probs)) if svm_disagree_probs[i
print(np.mean(n_support_vecs))
```

```
0.13
11.934
0.10742
0.09079
0.373
2.849
```

In [75]:
```
N = 100
iterations = 1000
num_iters = []
disagreement_probs = []
svm_disagree_probs = []
n_support_vecs = []
for i in range(iterations):
    m,b,x,y = get_target(N)
    while len(np.unique(y)) <2:
        m,b,x,y = get_target(N)
    #do PLA
    x_with_bias = add_bias_to_x(x)
    w = init_weights()
    iter_count, disagreement = PLA(m,b,x_with_bias,y,w)
    num_iters.append(iter_count)
    disagreement_probs.append(disagreement)
    #do SVM
    clf = svm.SVC(C = 10e20, kernel = 'linear')
    clf.fit(x, y)
    svm_disagree_probs.append(estimate_svm_disagreement(clf,m,b))
    n_support_vecs.append(len(clf.support_))

print(disagreement_probs[0])
print(np.mean(num_iters))
print(np.mean(disagreement_probs))
print(np.mean(svm_disagree_probs))
print(sum([1 for i in range(len(svm_disagree_probs)) if svm_disagree_probs[i
print(np.mean(n_support_vecs))
```

```
0.01
109.9
0.01374
0.01043
0.271
3.001
```

In [ ]: