In [1]:
```
import cvxopt
import numpy as np
from sklearn.preprocessing import PolynomialFeatures
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from data_gen import *
SEED=0
```

T1

Primal-SVM

In [2]:
```
class Primal_SVM:
    def __init__(self,transform_fn=None):
        self.transform_fn=transform_fn
    def fit(self,data):
        X=data[0]
        if self.transform_fn:
            X=self.transform_fn.fit_transform(X)
        y=data[1]
        N,d=X.shape #N: the num of samples  d: 不增广1的时候的维数
        X=np.hstack((np.ones((N,1)),X))
        y=y.reshape((N,1))

        Q=np.array((np.hstack((np.zeros((1+d,1)),np.vstack([np.zeros((1,
d)),np.eye(d)])))),dtype=np.float)
        p=np.zeros((d+1,1),dtype=np.float)
        A=-y*X
        c=np.full(N,-1.0).reshape((N,1))#注意类型要是小数类型, 不能是整数

        Q=cvxopt.matrix(Q)
        p=cvxopt.matrix(p)
        A=cvxopt.matrix(A)
        c=cvxopt.matrix(c)

        solution=cvxopt.solvers.qp(Q,p,A,c)
        weight=solution['x']
        self.b=weight[0]
        self.w=np.array(weight[1:]).flatten()

    def decision_fn(self,X):
        if self.transform_fn:
            X=self.transform_fn.fit_transform(X)
        return self.b+X@self.w

    def predict(self,X):
        return np.sign(self.decision_fn(X))

    def eval(self,X,y):
        pred=self.predict(X)
        mistake_indices = np.where(pred!=y)[0]
        accuracy = (X.shape[0]-len(mistake_indices))/X.shape[0]
        return accuracy
#X=np.array([[1,1],[2,2],[2,0],[0,0],[1,0],[0,1]])
```

```
# y=np.array([1]*3+[-1]*3,dtype=np.float)
# data=(X,y)
# model=Primal_SVM()
# model.fit(data)
# model.eval(data[0],data[1])
```

In [3]:
```
# SGD法求SVM的解
# X=np.array([[1,1],[2,2],[2,0],[0,0],[1,0],[0,1]])
# y=np.array([1]*3+[-1]*3)
# X=np.hstack((np.ones((6,1)),X))
# w0=np.array([0,0,0])
# w=w0
# lr=0.1
# for e in range(100):
#     g=((1-y*(X@w)>0)*(-y*X.T)).T
#     print('epoch ',e)
#     for i in range(6):
#         w=w-lr*g[i]
#         print(w)
```

dual SVM

In [4]:
```
class Dual_SVM:
    def __init__(self,epsilon=1e-9,transform_fn=None):
        self.epsilon=epsilon
        self.transform_fn=transform_fn

    def fit(self,data):
        X=data[0]#注意此处的X也是不增广1的X
        if self.transform_fn:
            X=self.transform_fn.fit_transform(X)
        y=data[1]
        N,d=X.shape

        Q=np.zeros((N,N))
        for i in range(N):
            for j in range(N):
                Q[i,j]=y[i]*y[j]*X[i,:]@X[j,:].T

        Q=np.array((Q),dtype=np.float)
        p=np.full((N,1),-1,dtype=np.float)
        A=-np.eye(N,dtype=np.float)#ppt上的公式有问题
        c=np.full((N,1),0.0)#注意类型要是小数类型，不能是整数
        r=y.reshape((1,N))
        v=0.0

        Q=cvxopt.matrix(Q)
        p=cvxopt.matrix(p)
        A=cvxopt.matrix(A)
        c=cvxopt.matrix(c)
        r=cvxopt.matrix(r)
        v=cvxopt.matrix(v)

        solution=cvxopt.solvers.qp(Q,p,A,c,r,v)
        alpha=np.array(solution['x'])
        w=np.sum(alpha*y.reshape((N,1))*X,axis=0)
        idx=np.where(alpha>self.epsilon)[0][0]#先取index的list再取list的第一个元
```

```
素
        b=y[idx]-X[idx,:]@w

        self.alpha=alpha
        self.w=w
        self.b=b

        self.SV=X[np.where(self.alpha.flatten()>self.epsilon)]

    def decision_fn(self,X):
        if self.transform_fn:
            X=self.transform_fn.fit_transform(X)
        return self.b+X@self.w

    def predict(self,X):
        return np.sign(self.decision_fn(X))

    def eval(self,X,y):
        pred=self.predict(X)
        mistake_indices = np.where(pred!=y)[0]
        accuracy = (X.shape[0]-len(mistake_indices))/X.shape[0]
        return accuracy

# X=np.array([[2,2],[-2,-2],[2,-2],[-2,2]],dtype=np.float)
# y=np.array([1]*2+[-1]*2,dtype=np.float)
# fn=PolynomialFeatures(degree=2,include_bias=False)
# model=Dual_SVM(transform_fn=fn)
# model.fit((X,y))
# hat_y=model.predict(X)
```

kernel SVM

In [5]:

```
class Kernel_SVM():
    def __init__(self,kernel_name,epsilon=1e-7):
        linear_kernel=lambda x1,x2,gamma:x1.T@x2
        square_kernel=lambda x1,x2,gamma: (1+gamma*x1.T@x2)**2
        cubic_kernel=lambda x1,x2,gamma: (1+gamma*x1.T@x2)**3
        quartic_kernel=lambda x1,x2,gamma: (1+gamma*x1.T@x2)**4
        rbf_kernel=lambda x1,x2,gamma: np.exp(-gamma*np.linalg.norm(x1-x
2)**2)

        self.epsilon=epsilon

        if kernel_name=='square':
            self.kernel_fn=square_kernel
            self.transform_fn=PolynomialFeatures(degree=2,include_bias=F
alse)
        elif kernel_name=='cubic':
            self.kernel_fn=cubic_kernel
            self.transform_fn=PolynomialFeatures(degree=3,include_bias=F
alse)
        elif kernel_name=='quartic':
            self.kernel_fn=quartic_kernel
            self.transform_fn=PolynomialFeatures(degree=4,include_bias=F
alse)
        elif kernel_name=='rbf':
            self.kernel_fn=rbf_kernel
        else :
```

```
            self.kernel_fn=linear_kernel
            self.transform_fn=PolynomialFeatures(degree=1,include_bias=F
alse)

        self.kernel_name=kernel_name

    def fit(self,data,gamma=1):
        X=data[0]#注意此处的X是不增广1不升维的X
        y=data[1]
        N,d=X.shape
        self.X=X
        self.y=y.reshape((N,1))
        self.gamma=gamma

        Q=np.zeros((N,N))
        for i in range(N):
            for j in range(N):
                Q[i,j]=y[i]*y[j]*self.kernel_fn(X[i,:].T,X[j,:].T,gamma)

        Q=np.array(Q,dtype=np.float)
        p=np.full((N,1),-1,dtype=np.float)
        A=-np.eye(N,dtype=np.float)
        c=np.full((N,1),0.0)#注意类型要是小数类型, 不能是整数
        r=y.reshape((1,N))
        v=0.0

        Q=cvxopt.matrix(Q)
        p=cvxopt.matrix(p)
        A=cvxopt.matrix(A)
        c=cvxopt.matrix(c)
        r=cvxopt.matrix(r)
        v=cvxopt.matrix(v)

        solution=cvxopt.solvers.qp(Q,p,A,c,r,v)
        self.alpha=np.array(solution['x'])
        # w=np.sum(alpha*y.reshape((N,1))*transform_fn.fit_transform(X),axis=0)
        idx=np.where(self.alpha>self.epsilon)[0][0]#先取index的list再取list的第
一个元素
        # b=y[idx]-transform_fn.fit_transform(X[idx,:].reshape((1,-1)))@w
        # pred=np.sign(transform_fn.fit_transform(X)@w+b)

        if self.kernel_name!='rbf':
            self.b=y[idx]-(self.alpha*self.y).T@self.kernel_fn(self.X.T,
self.X[idx,:],self.gamma) # (1,n)@ # (n,d).T.T@(d,)
        else:
            self.b=y[idx]-(self.alpha*self.y).T@[self.kernel_fn(self.X[n
,:],self.X[idx,:],self.gamma) for n in range(self.X.shape[0])]#(1,N)@ (N,)

        self.SV=self.X[np.where(self.alpha.flatten()>self.epsilon)]

        # print('alpha= \n ',solution['x'])
        # print('w= ',w)
        # print('b= ',b)
        # return b,w,transform_fn

    def predict(self,X):
        return np.sign(self.decision_fn(X))

    def decision_fn(self,X):
        if self.kernel_name!='rbf':
```

```
            s=self.kernel_fn(X.T,self.X.T,self.gamma)@(self.alpha*self.y
)+self.b #(m,d).T.T@(d,n)=(m,n) #(m,n)@(n,1)
        else:
            s=np.zeros((X.shape[0],1))
            for m in range(X.shape[0]):
                tmp=np.zeros(self.X.shape[0]) #(N,)
                for n in range(self.X.shape[0]):
                    tmp[n]=self.kernel_fn(self.X[n,:],X[m,:],self.gamma)
                s[m]=(self.alpha*self.y).T@tmp + self.b #(N,1).T@(N,)
        # return np.sign(s)
        return s

    def eval(self,X,y):
        pred=self.predict(X).flatten()#(N,1)与(N,)向量不能直接比较是否对应元素相
等
        mistake_indices = np.where(pred!=y)[0]
        accuracy = (X.shape[0]-len(mistake_indices))/X.shape[0]
        return accuracy


# X=np.array([[1,1],[2,2],[2,0],[0,0],[1,0],[0,1]])
# y=np.array([1]*3+[-1]*3,dtype=np.float).reshape((X.shape[0],1))
# model=Kernel_SVM('rbf')
# model.fit((X,y),gamma=0.1)
```

T2

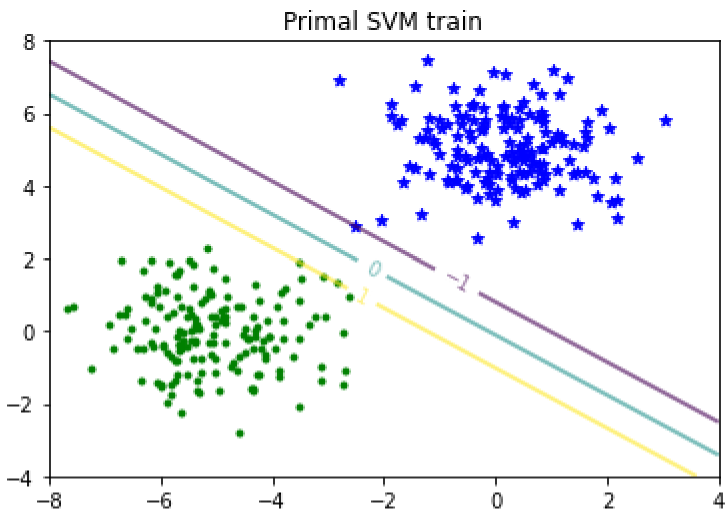
In [6]:
```
def plot_svm_res(data,model,axes,plot_sv=True):
    X=data[0]
    y=data[1]
    def plot_predictions(model,axes):
        x0s = np.linspace(axes[0], axes[1], 30)
        x1s = np.linspace(axes[2], axes[3], 30)
        x0, x1 = np.meshgrid(x0s, x1s)
        X_ = np.c_[x0.ravel(), x1.ravel()]
        pred=model.decision_fn(X_)
        y_pred = pred.reshape(x0.shape)
        # ax=plt.contourf(x0, x1, np.sign(y_pred), cmap=plt.cm.brg, alpha=0.2)
        cs=plt.contour(x0,x1,y_pred,linewidths=2,levels=[-1,0,1],alpha=0
.6)
        # plt.colorbar()
        plt.clabel(cs)
    plt.scatter(X[(y==1).flatten(),0],X[(y==1).flatten(),1],marker='.',c
olor='green')
    plt.scatter(X[(y==-1).flatten(),0],X[(y==-1).flatten(),1],marker='*'
,color='blue')
    if plot_sv:
        plt.scatter(model.SV[:,0],model.SV[:,1],marker='o',color='pink',
alpha=0.6)
    plot_predictions(model,axes)

data=data_generator([-5,0],np.eye(2),[0,5],np.eye(2),400,seed=SEED)
X_train,X_test,y_train,y_test=train_test_split(data[0],data[1],train_siz
e=0.8,test_size=0.2)
train_data=(X_train,y_train)
test_data=(X_test,y_test)
axes=[-8, 4, -4, 8]
```

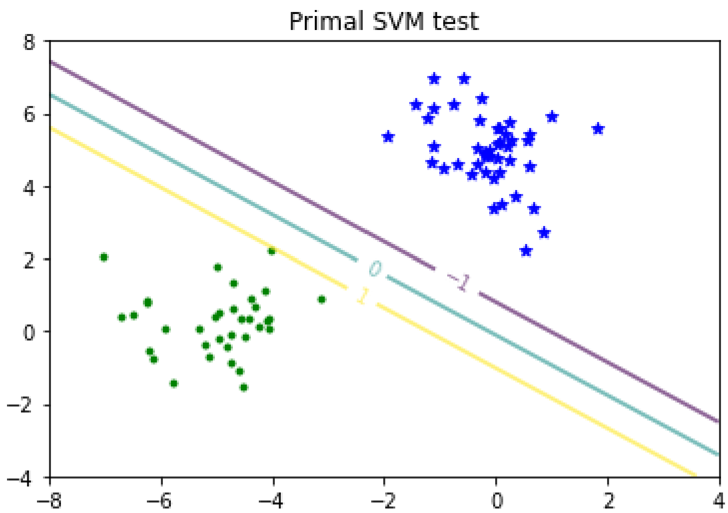
In [7]:
```
#fn=PolynomialFeatures(degree=2,include_bias=False)
def algorithm(train_data,test_data,axes):
    # primal svm
    model = Primal_SVM()
    model.fit(train_data)

    plt.figure()
    plot_svm_res(train_data,model,axes,plot_sv=False)
    plt.title('Primal SVM train')

    plt.figure()
    plot_svm_res(test_data,model,axes,plot_sv=False)
    plt.title('Primal SVM test')

    print('primal svm train accuracy:',model.eval(test_data[0],test_data
[1]))
    print('primal svm test accuracy:',model.eval(test_data[0],test_data[
1]))


    # dual svm
    model = Dual_SVM()
    model.fit(train_data)

    plt.figure()
    plot_svm_res(train_data,model,axes)
    plt.title('Dual SVM train')

    plt.figure()
    plot_svm_res(test_data,model,axes,plot_sv=False)
    plt.title('Dual SVM test')

    print('Dual SVM train accuracy:',model.eval(test_data[0],test_data[1
]))
    print('Dual SVM test accuracy:',model.eval(test_data[0],test_data[1]
))


    # kernel svm
    # quartic polynomial feature
    model = Kernel_SVM('quartic')
    model.fit(train_data)

    plt.figure()
    plot_svm_res(train_data,model,axes)
    plt.title('quartic kernel SVM train')

    plt.figure()
    plot_svm_res(test_data,model,axes,plot_sv=False)
    plt.title('quartic kernel SVM test')

    print('quartic kernel SVM train accuracy:',model.eval(test_data[0],t
est_data[1]))
    print('quartic kernel SVM test accuracy:',model.eval(test_data[0],te
st_data[1]))


    #kernel svm
    # rbf kernel
    model=Kernel_SVM('rbf')
```

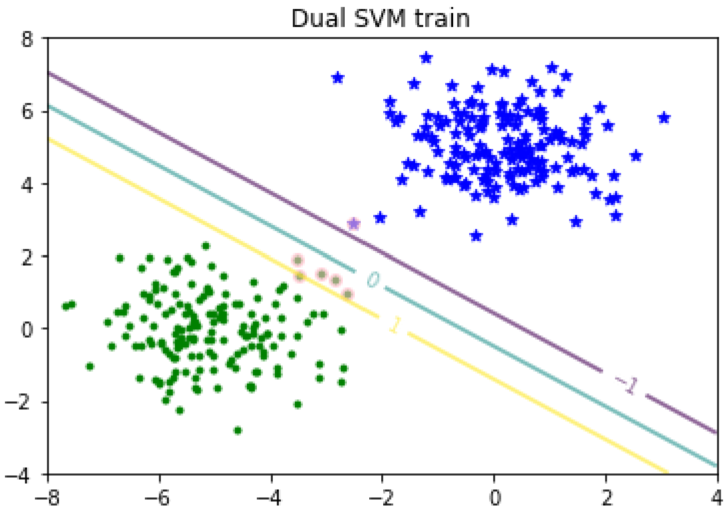
```
    model.fit(data,gamma=0.1)

    plt.figure()
    plot_svm_res(train_data,model,axes)
    plt.title('rbf kernel SVM train')

    plt.figure()
    plot_svm_res(test_data,model,axes,plot_sv=False)
    plt.title('rbf kernel SVM test')

    print('rbf kernel SVM train accuracy:',model.eval(test_data[0],test_
data[1]))
    print('rbf kernel SVM test accuracy:',model.eval(test_data[0],test_d
ata[1]))


algorithm(train_data,test_data,axes)
```

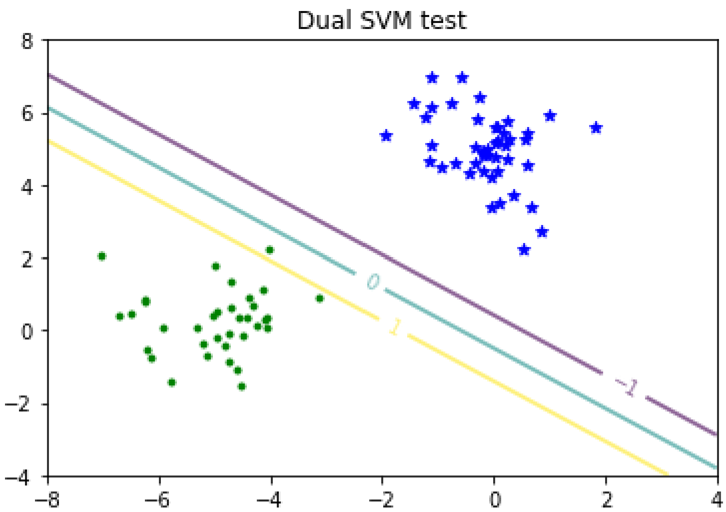
```
     pcost       dcost       gap    pres   dres
 0:  3.2951e-02  5.4758e+01  1e+03  2e+00  2e+03
 1:  1.7874e-01 -2.3700e+02  4e+02  9e-01  8e+02
 2:  2.6884e-01 -2.4907e+02  4e+02  7e-01  7e+02
 3:  5.6325e-01 -1.8593e+02  3e+02  4e-01  4e+02
 4:  1.0002e+00 -1.6451e+02  2e+02  2e-01  2e+02
 5:  1.3382e+00 -1.4064e+02  1e+02  1e-01  1e+02
 6:  1.4172e+00 -6.8217e+00  8e+00  6e-03  6e+00
 7:  1.0807e+00  6.0763e-01  5e-01  1e-15  1e-14
 8:  1.0370e+00  8.5081e-01  2e-01  9e-16  1e-14
 9:  1.0156e+00  1.0128e+00  3e-03  1e-15  6e-15
10:  1.0150e+00  1.0150e+00  3e-05  1e-15  1e-14
11:  1.0150e+00  1.0150e+00  3e-07  1e-15  1e-14
Optimal solution found.
primal svm train accuracy: 1.0
primal svm test accuracy: 1.0
     pcost       dcost       gap    pres   dres
 0: -2.1124e+01 -4.0993e+01  1e+03  3e+01  2e+00
 1: -2.4881e+01 -2.1501e+01  4e+02  1e+01  9e-01
 2: -6.0128e+01 -4.5100e+01  4e+02  1e+01  7e-01
 3: -7.5132e+01 -2.9223e+01  3e+02  6e+00  4e-01
 4: -2.2307e+01 -5.3029e+00  2e+02  3e+00  2e-01
 5: -1.5639e+01 -3.4772e+00  1e+02  2e+00  1e-01
 6: -1.4709e+00 -1.4357e+00  8e+00  9e-02  6e-03
 7: -6.0763e-01 -1.0807e+00  5e-01  2e-16  7e-15
 8: -8.5081e-01 -1.0370e+00  2e-01  2e-16  5e-15
 9: -1.0128e+00 -1.0156e+00  3e-03  1e-16  6e-15
10: -1.0150e+00 -1.0150e+00  3e-05  4e-16  7e-15
11: -1.0150e+00 -1.0150e+00  3e-07  1e-15  7e-15
Optimal solution found.
Dual SVM train accuracy: 1.0
Dual SVM test accuracy: 1.0
     pcost       dcost       gap    pres   dres
 0: -5.8227e+00 -1.0956e+01  7e+02  2e+01  2e+00
 1: -8.2417e+00 -4.0070e+00  2e+02  5e+00  5e-01
 2: -4.9164e+00 -4.2288e-01  2e+01  7e-01  6e-02
 3: -5.0339e-01 -5.2295e-02  2e+00  6e-02  6e-03
 4: -6.3088e-02 -2.0048e-02  5e-01  1e-02  1e-03
 5: -2.2613e-02 -9.9491e-03  4e-01  9e-03  8e-04
 6: -5.7690e-03 -2.6900e-03  4e-02  9e-04  9e-05
 7: -1.8284e-03 -1.9895e-03  9e-03  2e-04  2e-05
 8: -1.2040e-03 -1.7026e-03  6e-03  1e-04  9e-06
 9: -1.3449e-03 -1.2561e-03  3e-03  4e-05  4e-06
```

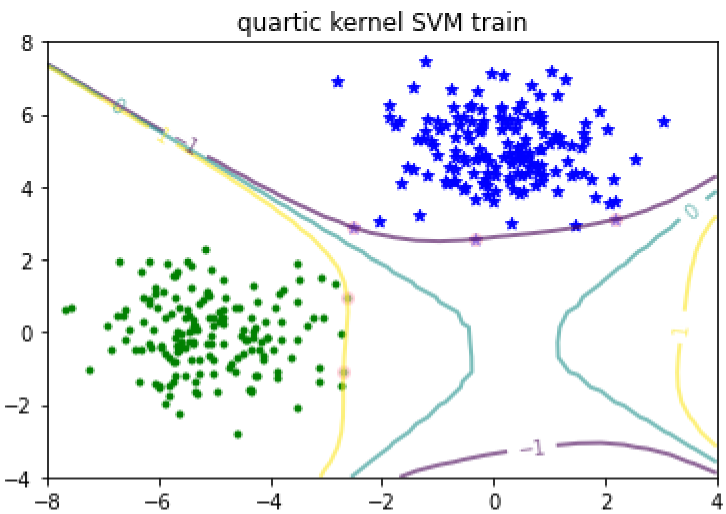
```
10: -1.3642e-03 -8.1355e-04  1e-03  2e-05  2e-06
11: -1.1212e-03 -5.1432e-04  8e-04  1e-05  1e-06
12: -4.8107e-04 -2.9114e-04  1e-04  2e-06  2e-07
13: -2.7510e-04 -2.7675e-04  4e-06  9e-09  9e-10
14: -2.7627e-04 -2.7629e-04  5e-08  1e-10  1e-11
Optimal solution found.
quartic kernel SVM train accuracy: 1.0
quartic kernel SVM test accuracy: 1.0
     pcost       dcost       gap    pres   dres
 0: -4.6282e+00 -1.4718e+01  8e+02  2e+01  2e+00
 1: -3.1388e+00 -1.4768e+01  1e+02  3e+00  3e-01
 2: -1.5980e+00 -9.7116e+00  8e+00  1e-15  7e-16
 3: -4.7590e+00 -7.1307e+00  2e+00  9e-16  1e-15
 4: -5.7651e+00 -6.5798e+00  8e-01  1e-15  8e-16
 5: -6.1179e+00 -6.4824e+00  4e-01  2e-15  9e-16
 6: -6.3886e+00 -6.4314e+00  4e-02  2e-15  1e-15
 7: -6.4269e+00 -6.4275e+00  6e-04  9e-16  1e-15
 8: -6.4274e+00 -6.4274e+00  6e-06  1e-15  1e-15
Optimal solution found.
rbf kernel SVM train accuracy: 1.0
rbf kernel SVM test accuracy: 1.0
```

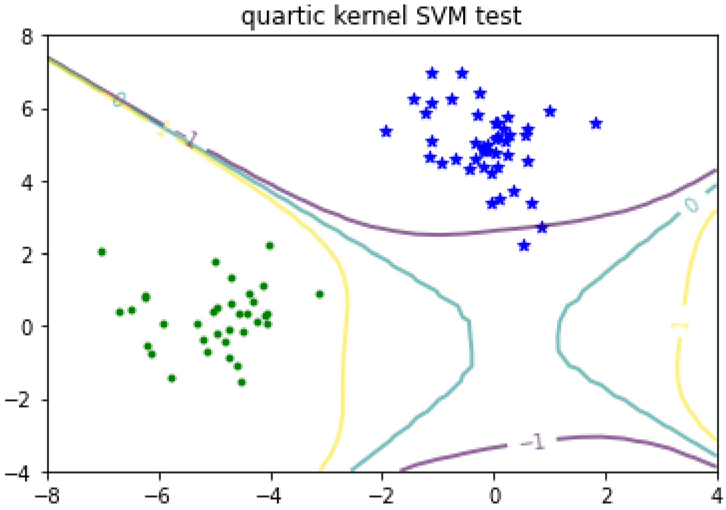


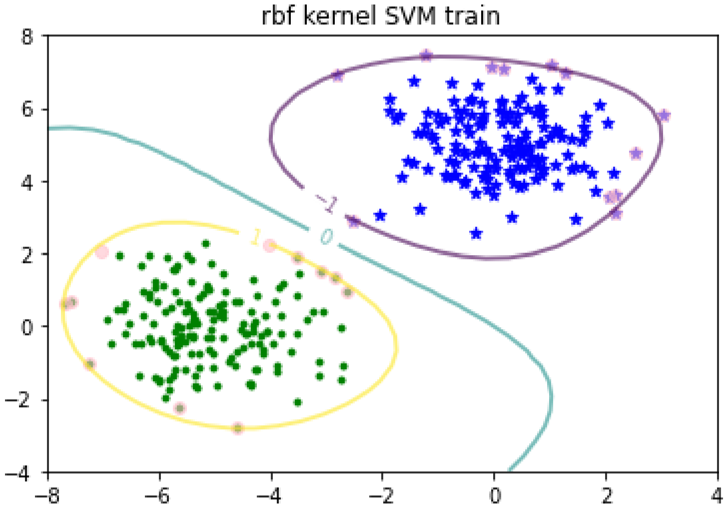


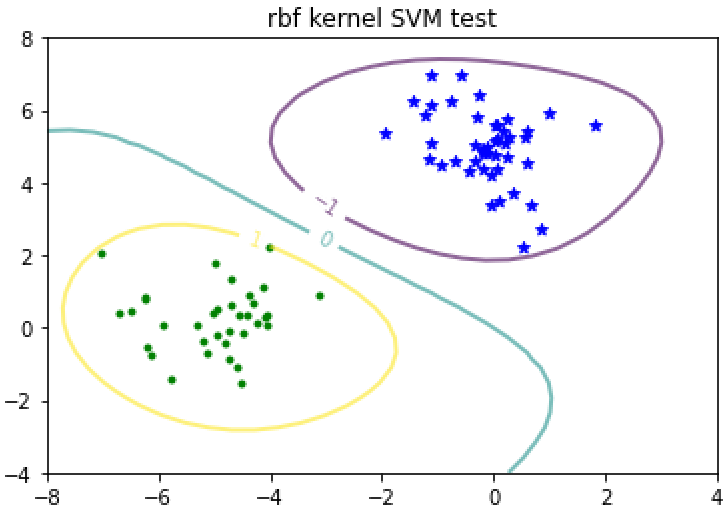

Dual SVM train
0
1
−1


Dual SVM test
0
1
−1


quartic kernel SVM train
0
1
−1







从结果图可以看出, 存在边界上的点不是支撑向量的情况, 即对应的alpha=0

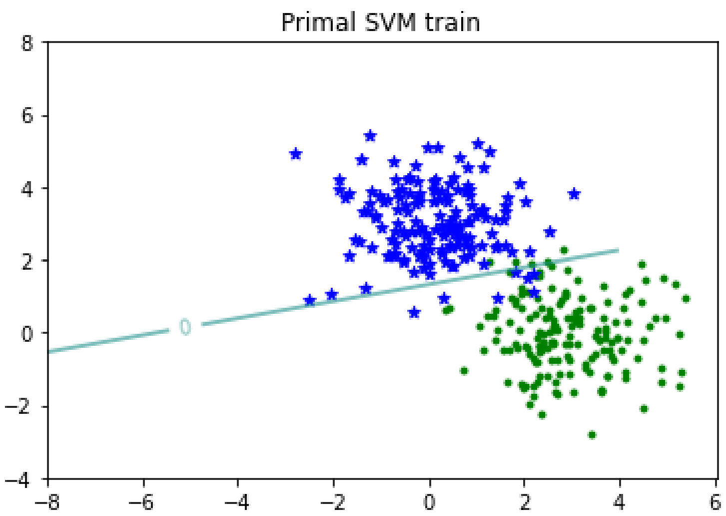
T3

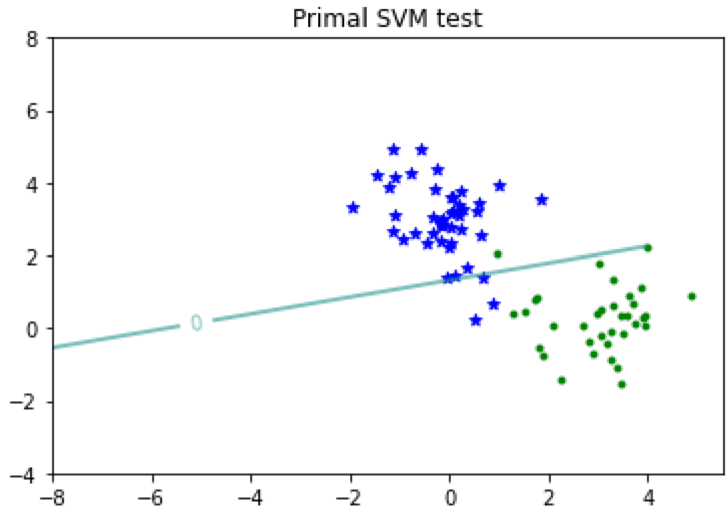
In [8]:
```
data=data_generator([3,0],np.eye(2),[0,3],np.eye(2),400,seed=SEED)
X_train,X_test,y_train,y_test=train_test_split(data[0],data[1],train_size=0.8,test_size=0.2)
train_data=(X_train,y_train)
test_data=(X_test,y_test)
algorithm(train_data, test_data, axes)
```

```
     pcost       dcost       gap    pres   dres
 0:  7.4657e-02  1.9787e+02  1e+03  2e+00  1e+03
 1:  3.2303e-01 -1.5991e+02  1e+03  2e+00  1e+03
 2:  2.5075e-01  2.0363e+02  1e+03  2e+00  9e+02
 3:  1.1282e-02  3.0301e+03  1e+03  1e+00  8e+02
 4:  3.4503e-03  5.8550e+03  2e+03  1e+00  8e+02
 5:  3.8524e-04  1.6948e+04  3e+03  1e+00  7e+02
 6:  5.5445e-05  7.7693e+04  6e+03  1e+00  6e+02
 7:  1.9968e-06  1.0570e+06  6e+03  1e+00  6e+02
 8:  2.4319e-07  1.2147e+08  3e+05  1e+00  6e+02
 9:  6.7450e-09  4.8418e+10  2e+07  1e+00  6e+02
10:  2.1476e-12  1.3483e+14  9e+08  1e+00  6e+02
11:  2.2197e-16  2.1133e+19  1e+12  1e+00  1e+04
12:  2.2199e-20  3.2583e+26  2e+17  1e+00  6e+10
13:  2.2819e-24  5.0218e+35  3e+24  1e+00  3e+20
14:  4.9784e-23  2.9081e+44  2e+33  1e+00  1e-11
15:  4.3267e-23  2.3552e+52  2e+41  1e+00  6e+36
16:  4.7364e-23  2.5868e+61  2e+50  1e+00  1e+46
17:  3.8847e-23  8.1857e+67  5e+56  1e+00  5e+52
18:  4.2051e-23  8.5827e+76  6e+65  1e+00  2e+61
19:  3.5830e-23  2.6548e+83  2e+72  1e+00  7e+67
20:  3.9451e-23  2.7709e+92  2e+81  1e+00  8e+76
21:  3.4281e-23  8.4488e+98  5e+87  1e+00  3e+83
22:  3.8200e-23  8.6492e+107  6e+96  1e+00  2e+92
23:  3.3450e-23  2.5837e+114  2e+103  1e+00  2e+99
24:  3.7497e-23  2.6326e+123  2e+112  1e+00 2e+108
25:  3.2912e-23  7.7302e+129  5e+118  1e+00 2e+114
26:  3.6922e-23  7.8190e+138  5e+127  1e+00 3e+123
27:  3.2443e-23  2.2513e+145  1e+134  1e+00 1e+130
28:  3.6602e-23  2.3085e+154  1e+143  1e+00 1e+139
29:  3.2110e-23  6.5488e+160  4e+149  1e+00 4e+145
30:  3.5984e-23  6.5703e+169  4e+158  1e+00    inf
31:  3.1625e-23  1.8324e+176  1e+165  1e+00    inf
32:  3.5653e-23  1.7822e+185  1e+174  1e+00    inf
33:  3.1298e-23  4.8916e+191  3e+180  1e+00    inf
34:  3.5343e-23  5.0104e+200  3e+189  1e+00    inf
35:  3.0977e-23  1.3522e+207  8e+195  1e+00    inf
36:  3.4772e-23  1.3219e+216  8e+204  1e+00    inf
37:  3.0528e-23  3.5131e+222  2e+211  1e+00    inf
38:  3.4849e-23  3.5589e+231  2e+220  1e+00    inf
39:  3.0415e-23  9.3151e+237  6e+226  1e+00    inf
40:  3.4410e-23  8.6216e+246  5e+235  1e+00    inf
41:  3.0054e-23  2.2174e+253  1e+242  1e+00    inf
42:  3.3188e-23  2.2132e+262  1e+251  1e+00    inf
43:  2.9271e-23  5.6039e+268  3e+257  1e+00    inf
44:  3.2667e-23  5.0685e+277  3e+266  1e+00    inf
45:  2.8860e-23  1.2705e+284  8e+272  1e+00    inf
46:  3.5049e-23  1.3828e+293  8e+281  1e+00    inf
Terminated (singular KKT matrix).
primal svm train accuracy: 0.95
primal svm test accuracy: 0.95
     pcost       dcost       gap    pres   dres
 0: -6.4272e+01 -1.4729e+02  1e+03  4e+01  2e+00
 1: -1.4493e+02 -2.3172e+02  1e+03  3e+01  2e+00
 2: -4.0438e+02 -6.3441e+02  1e+03  3e+01  2e+00
 3: -2.6892e+03 -3.6951e+03  1e+03  2e+01  1e+00
 4: -5.5084e+03 -7.0514e+03  2e+03  2e+01  1e+00
 5: -1.6602e+04 -1.9271e+04  3e+03  2e+01  1e+00
 6: -7.7373e+04 -8.3511e+04  6e+03  2e+01  1e+00
 7: -1.0567e+06 -1.0624e+06  6e+03  2e+01  1e+00
```

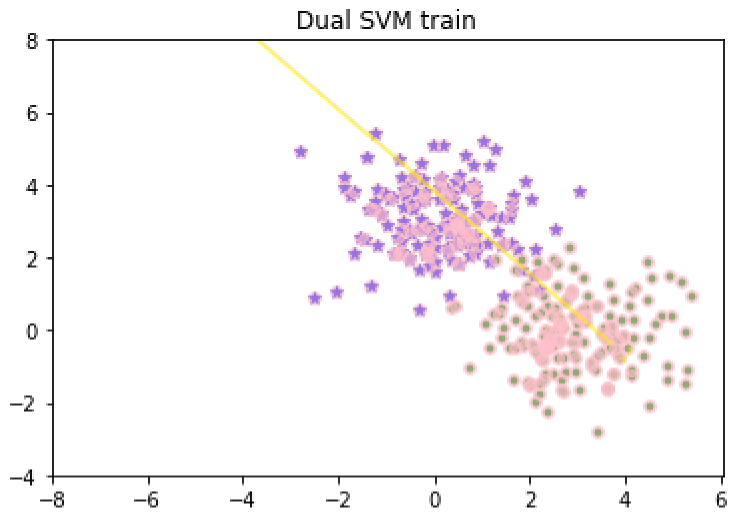
```
 8: -1.2148e+08 -1.2174e+08  3e+05  2e+01  1e+00
 9: -4.9713e+10 -4.9730e+10  2e+07  2e+01  1e+00
10: -6.3116e+10 -6.3136e+10  2e+07  2e+01  1e+00
11: -7.9467e+10 -7.9492e+10  3e+07  2e+01  1e+00
12: -9.9312e+10 -9.9343e+10  3e+07  2e+01  1e+00
13: -1.2352e+11 -1.2355e+11  4e+07  2e+01  1e+00
14: -1.7642e+11 -1.7647e+11  5e+07  2e+01  1e+00
15: -2.1079e+11 -2.1084e+11  5e+07  2e+01  1e+00
16: -2.3565e+11 -2.3570e+11  5e+07  2e+01  1e+00
Terminated (singular KKT matrix).
Dual SVM train accuracy: 0.4375
Dual SVM test accuracy: 0.4375
     pcost       dcost       gap    pres   dres
 0: -3.8183e+01 -9.3256e+01  1e+03  2e+01  2e+00
 1: -9.3605e+01 -1.6118e+02  8e+02  2e+01  2e+00
 2: -3.9782e+02 -6.6038e+02  8e+02  2e+01  2e+00
 3: -9.7267e+02 -1.2435e+03  7e+02  1e+01  1e+00
 4: -1.1061e+03 -1.4005e+03  8e+02  1e+01  1e+00
 5: -2.2110e+03 -2.6123e+03  9e+02  1e+01  1e+00
 6: -8.2203e+03 -8.8377e+03  1e+03  1e+01  1e+00
 7: -1.7884e+04 -1.9071e+04  2e+03  1e+01  1e+00
 8: -4.8835e+04 -5.1640e+04  4e+03  1e+01  1e+00
 9: -1.0190e+05 -1.0729e+05  6e+03  1e+01  1e+00
10: -4.3017e+05 -4.4923e+05  2e+04  1e+01  1e+00
11: -1.5937e+06 -1.6562e+06  6e+04  9e+00  1e+00
12: -4.4762e+06 -4.6398e+06  2e+05  9e+00  1e+00
13: -1.1140e+07 -1.1526e+07  4e+05  9e+00  1e+00
14: -3.8137e+07 -3.9350e+07  1e+06  9e+00  1e+00
15: -4.1402e+07 -4.2716e+07  1e+06  9e+00  1e+00
16: -3.9149e+08 -4.0067e+08  9e+06  9e+00  1e+00
17: -4.4263e+09 -4.5038e+09  8e+07  9e+00  1e+00
18: -4.8202e+09 -4.9046e+09  8e+07  9e+00  1e+00
19: -6.1600e+09 -6.2663e+09  1e+08  9e+00  1e+00
20: -9.1620e+09 -9.3128e+09  2e+08  9e+00  1e+00
21: -1.4223e+10 -1.4444e+10  2e+08  9e+00  1e+00
22: -1.7469e+10 -1.7730e+10  3e+08  9e+00  1e+00
23: -1.9100e+10 -1.9380e+10  3e+08  9e+00  1e+00
24: -2.3889e+10 -2.4219e+10  3e+08  9e+00  1e+00
25: -2.4541e+10 -2.4879e+10  3e+08  9e+00  1e+00
26: -3.2259e+10 -3.2659e+10  4e+08  9e+00  1e+00
27: -4.2712e+10 -4.3154e+10  4e+08  9e+00  1e+00
28: -5.6100e+10 -5.6505e+10  4e+08  9e+00  1e+00
Terminated (singular KKT matrix).
quartic kernel SVM train accuracy: 0.875
quartic kernel SVM test accuracy: 0.875
     pcost       dcost       gap    pres   dres
 0: -3.8609e+01 -1.1100e+02  1e+03  2e+01  3e+00
 1: -1.1051e+02 -2.3525e+02  1e+03  2e+01  2e+00
 2: -4.6601e+02 -8.6857e+02  1e+03  1e+01  2e+00
 3: -9.5889e+02 -1.4260e+03  1e+03  1e+01  1e+00
 4: -2.9550e+03 -3.5294e+03  1e+03  9e+00  1e+00
 5: -9.5078e+03 -1.0709e+04  2e+03  9e+00  1e+00
 6: -1.2848e+04 -1.4370e+04  2e+03  9e+00  1e+00
 7: -4.4837e+04 -4.8841e+04  4e+03  9e+00  1e+00
 8: -4.8860e+04 -5.3148e+04  5e+03  9e+00  1e+00
 9: -8.1598e+04 -8.8222e+04  7e+03  9e+00  1e+00
10: -1.8903e+05 -2.0265e+05  1e+04  9e+00  1e+00
11: -4.7406e+05 -5.0497e+05  3e+04  9e+00  1e+00
12: -1.2056e+06 -1.2776e+06  7e+04  9e+00  1e+00
13: -3.6054e+06 -3.8060e+06  2e+05  9e+00  1e+00
14: -5.9002e+06 -6.2125e+06  3e+05  9e+00  1e+00
```

```
15: -1.1276e+07 -1.1859e+07  6e+05  9e+00  1e+00
16: -1.9916e+07 -2.0972e+07  1e+06  9e+00  1e+00
17: -2.5791e+07 -2.7187e+07  1e+06  9e+00  1e+00
18: -3.9544e+07 -4.1811e+07  2e+06  9e+00  1e+00
19: -3.9660e+07 -4.1934e+07  2e+06  9e+00  1e+00
20: -4.2646e+07 -4.5121e+07  2e+06  9e+00  1e+00
21: -6.2123e+07 -6.6058e+07  4e+06  9e+00  1e+00
22: -1.4641e+08 -1.5977e+08  1e+07  8e+00  1e+00
23: -1.5449e+08 -1.6896e+08  1e+07  8e+00  1e+00
24: -2.0586e+08 -2.2853e+08  2e+07  8e+00  1e+00
25: -3.7849e+08 -4.4222e+08  6e+07  8e+00  9e-01
26: -6.7627e+08 -8.3339e+08  2e+08  6e+00  8e-01
27: -9.8746e+08 -1.1817e+09  2e+08  3e+00  3e-01
28: -1.0340e+09 -1.0576e+09  2e+07  2e-01  3e-02
29: -1.0348e+09 -1.0365e+09  2e+06  5e-03  6e-04
30: -1.0358e+09 -1.0358e+09  2e+04  6e-05  7e-06
31: -1.0358e+09 -1.0358e+09  2e+02  6e-07  2e-07
32: -1.0358e+09 -1.0358e+09  2e+00  4e-07  2e-07
33: -1.0358e+09 -1.0358e+09  2e-02  4e-07  2e-07
34: -1.0358e+09 -1.0358e+09  2e-04  4e-07  2e-07
35: -1.0358e+09 -1.0358e+09  2e-06  2e-07  1e-07
36: -1.0358e+09 -1.0358e+09  2e-08  2e-07  9e-08
37: -1.0358e+09 -1.0358e+09  2e-10  2e-07  8e-08
38: -1.0358e+09 -1.0358e+09  2e-12  1e-07  8e-08
39: -1.0358e+09 -1.0358e+09  2e-14  3e-07  7e-08
40: -1.0358e+09 -1.0358e+09  2e-16  2e-07  8e-08
41: -1.0358e+09 -1.0358e+09  2e-18  4e-07  8e-08
42: -1.0358e+09 -1.0358e+09  2e-20  2e-07  7e-08
43: -1.0358e+09 -1.0358e+09  2e-22  2e-07  8e-08
44: -1.0358e+09 -1.0358e+09  2e-24  1e-07  8e-08
45: -1.0358e+09 -1.0358e+09  2e-26  2e-07  9e-08
46: -1.0358e+09 -1.0358e+09  2e-28  1e-07  7e-08
47: -1.0358e+09 -1.0358e+09  2e-30  1e-07  9e-08
48: -1.0358e+09 -1.0358e+09  2e-32  3e-07  7e-08
49: -1.0358e+09 -1.0358e+09  2e-34  7e-08  7e-08
Optimal solution found.
rbf kernel SVM train accuracy: 1.0
rbf kernel SVM test accuracy: 1.0
```

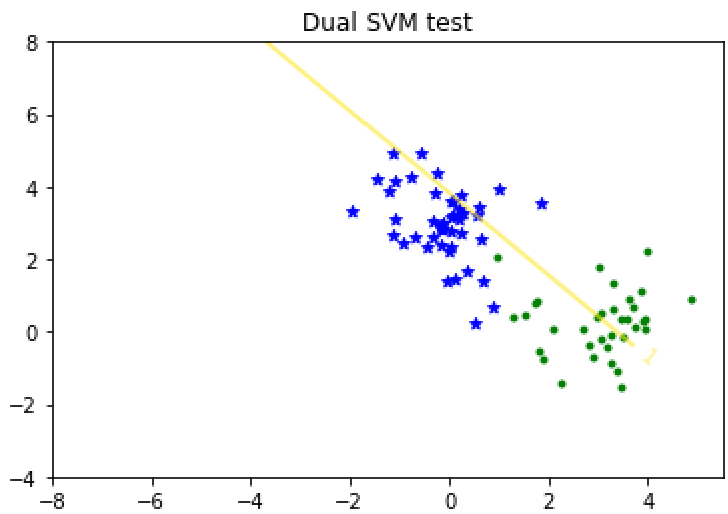




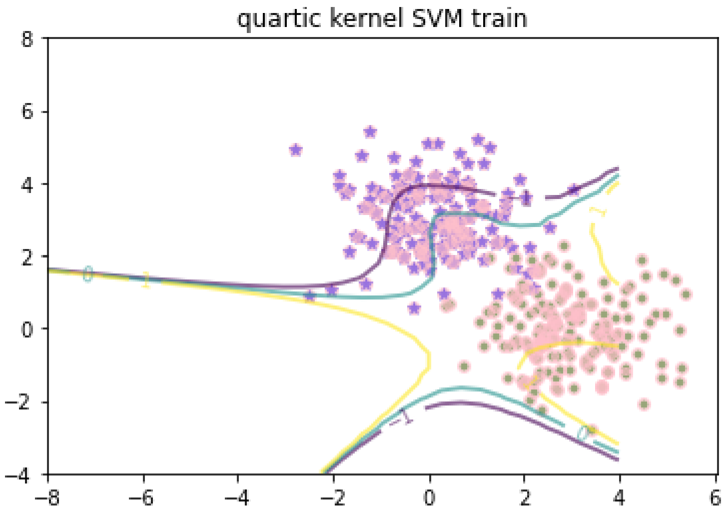
Primal SVM test

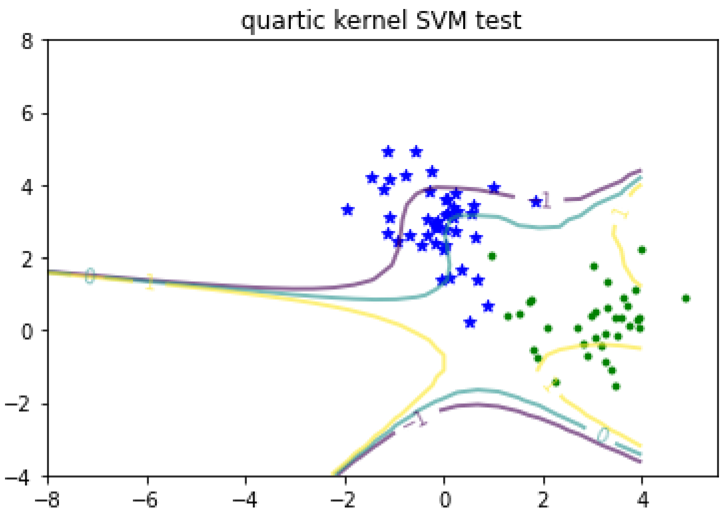


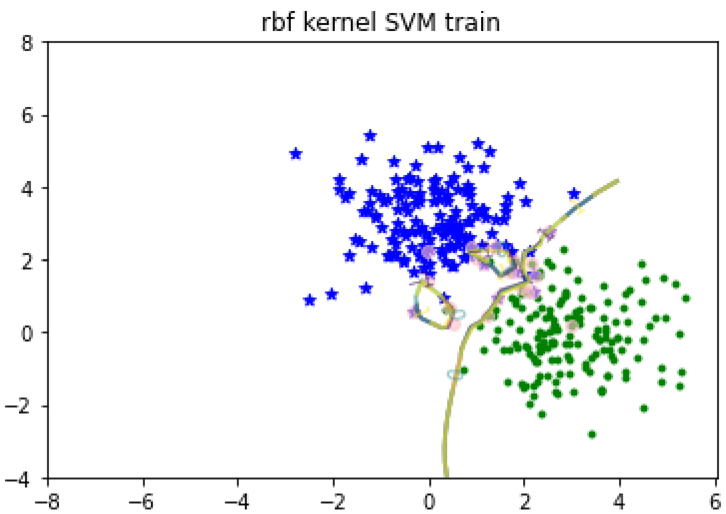
Dual SVM train

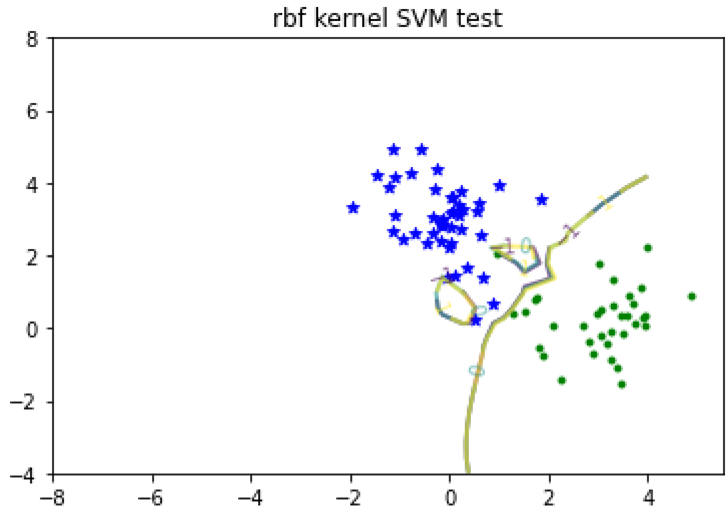


Dual SVM test


quartic kernel SVM train
8
6
4
2
0
−2
−4
−8
−6
−4
−2
0
2
4
6


quartic kernel SVM test
8
6
4
2
0
−2
−4
−8
−6
−4
−2
0
2
4


rbf kernel SVM train
8
6
4
2
0
−2
−4
−8
−6
−4
−2
0
2
4
6



由结果可见, 当数据难以线性区分时, 除了rbf核函数SVM外, 其他几种SVM的结果并不理想

T4

尝试调节核函数中gamma的值

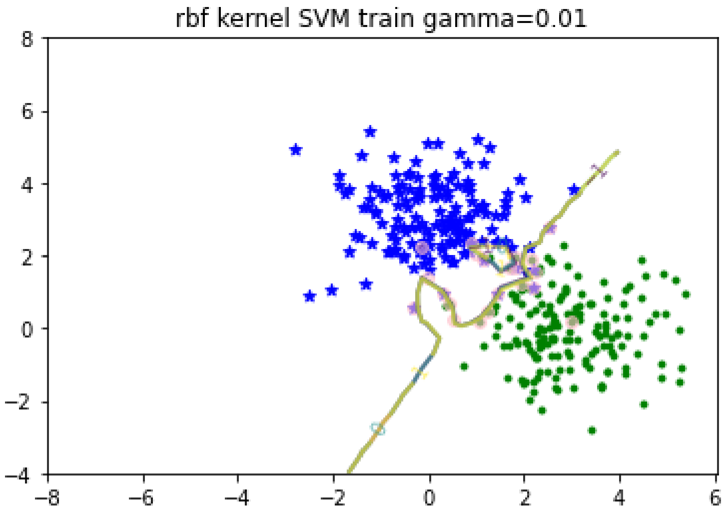
In [9]:
```
#kernel svm
# rbf kernel
model=Kernel_SVM('rbf')
model.fit(data,gamma=0.01)

plt.figure()
plot_svm_res(train_data,model,axes)
plt.title('rbf kernel SVM train gamma=0.01')

plt.figure()
plot_svm_res(test_data,model,axes,plot_sv=False)
plt.title('rbf kernel SVM test gamma=0.01')

print('rbf kernel SVM train accuracy:',model.eval(test_data[0],test_data
[1]))
print('rbf kernel SVM test accuracy:',model.eval(test_data[0],test_data[
1]))

model=Kernel_SVM('rbf')
model.fit(data,gamma=1)

plt.figure()
plot_svm_res(train_data,model,axes)
plt.title('rbf kernel SVM train gamma=1')

plt.figure()
plot_svm_res(test_data,model,axes,plot_sv=False)
plt.title('rbf kernel SVM test gamma=1')

print('rbf kernel SVM train accuracy:',model.eval(test_data[0],test_data
[1]))
print('rbf kernel SVM test accuracy:',model.eval(test_data[0],test_data[
1]))
```

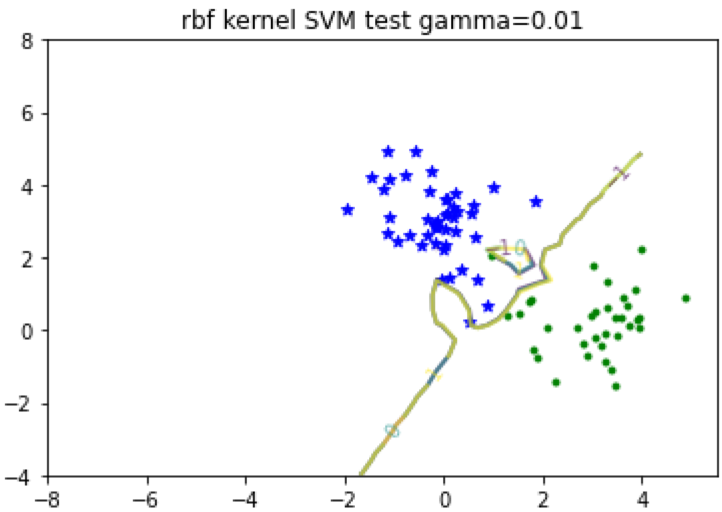
```
     pcost       dcost       gap    pres   dres
```

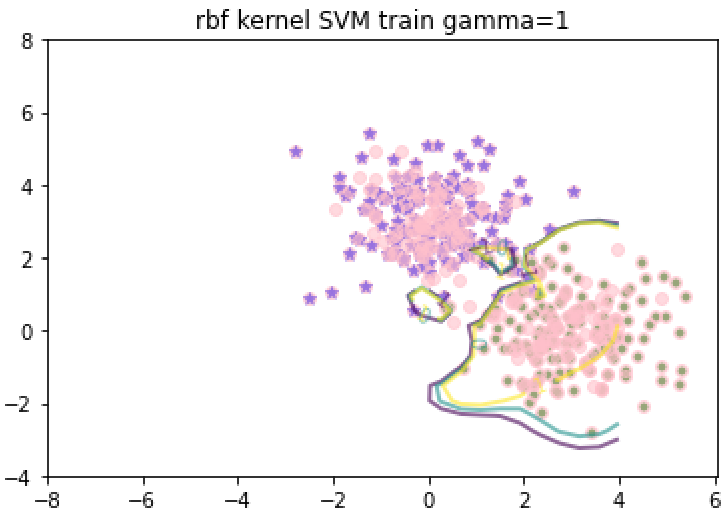
```
 0: -7.1142e+01 -1.8268e+02  1e+03  3e+01  2e+00
 1: -1.5578e+02 -3.0196e+02  1e+03  2e+01  2e+00
 2: -1.1833e+03 -1.9352e+03  1e+03  2e+01  2e+00
 3: -4.1296e+03 -4.9244e+03  9e+02  2e+01  1e+00
 4: -7.5381e+03 -8.7442e+03  1e+03  2e+01  1e+00
 5: -2.5191e+04 -2.7838e+04  3e+03  1e+01  1e+00
 6: -8.9304e+04 -9.6229e+04  7e+03  1e+01  1e+00
 7: -4.0831e+05 -4.3056e+05  2e+04  1e+01  1e+00
 8: -1.2216e+06 -1.2749e+06  5e+04  1e+01  1e+00
 9: -1.3904e+06 -1.4505e+06  6e+04  1e+01  1e+00
10: -1.5782e+06 -1.6457e+06  7e+04  1e+01  1e+00
11: -4.6577e+06 -4.8387e+06  2e+05  1e+01  1e+00
12: -5.2983e+06 -5.5033e+06  2e+05  1e+01  1e+00
13: -1.2870e+07 -1.3345e+07  5e+05  1e+01  1e+00
14: -1.3043e+07 -1.3525e+07  5e+05  1e+01  1e+00
15: -2.3606e+07 -2.4455e+07  8e+05  1e+01  1e+00
16: -2.6564e+07 -2.7510e+07  9e+05  1e+01  1e+00
17: -5.4571e+07 -5.6444e+07  2e+06  1e+01  1e+00
18: -5.5944e+07 -5.7861e+07  2e+06  1e+01  1e+00
19: -1.4933e+08 -1.5416e+08  5e+06  1e+01  1e+00
20: -6.0276e+08 -6.2082e+08  2e+07  1e+01  1e+00
21: -6.0926e+08 -6.2749e+08  2e+07  1e+01  1e+00
22: -6.4480e+08 -6.6402e+08  2e+07  1e+01  1e+00
23: -9.1190e+08 -9.3863e+08  3e+07  1e+01  1e+00
24: -2.0920e+09 -2.1513e+09  6e+07  1e+01  1e+00
25: -4.1977e+09 -4.3138e+09  1e+08  1e+01  1e+00
26: -7.6412e+09 -7.8502e+09  2e+08  1e+01  1e+00
27: -2.0341e+10 -2.0889e+10  5e+08  1e+01  1e+00
28: -5.8657e+10 -6.0236e+10  2e+09  1e+01  1e+00
29: -1.2488e+11 -1.2829e+11  3e+09  1e+01  1e+00
30: -2.4585e+11 -2.5275e+11  7e+09  1e+01  1e+00
31: -2.5046e+11 -2.5748e+11  7e+09  1e+01  1e+00
32: -4.7551e+11 -4.8887e+11  1e+10  1e+01  1e+00
33: -7.5805e+11 -7.7928e+11  2e+10  1e+01  1e+00
34: -8.2171e+11 -8.4470e+11  2e+10  1e+01  1e+00
35: -1.3864e+12 -1.4268e+12  4e+10  1e+01  1e+00
36: -2.2505e+12 -2.3210e+12  7e+10  1e+01  1e+00
37: -2.3113e+12 -2.3838e+12  7e+10  1e+01  1e+00
38: -2.8943e+12 -2.9902e+12  1e+11  1e+01  1e+00
39: -5.4372e+12 -5.6680e+12  2e+11  1e+01  1e+00
40: -5.8142e+12 -6.0685e+12  3e+11  1e+01  1e+00
41: -5.8222e+12 -6.0769e+12  3e+11  1e+01  1e+00
42: -6.7897e+12 -7.1094e+12  3e+11  1e+01  1e+00
43: -1.5970e+13 -1.7260e+13  1e+12  1e+01  1e+00
44: -2.0742e+13 -2.2724e+13  2e+12  1e+01  1e+00
45: -3.5444e+13 -4.0364e+13  5e+12  1e+01  9e-01
46: -6.4977e+13 -7.8599e+13  1e+13  1e+01  8e-01
47: -1.0067e+14 -1.2239e+14  2e+13  6e+00  4e-01
48: -1.1018e+14 -1.1426e+14  4e+12  3e-01  4e-02
49: -1.1110e+14 -1.1499e+14  4e+12  2e-01  4e-02
50: -1.1522e+14 -1.1647e+14  1e+12  4e-02  3e-02
51: -1.0903e+14 -1.0908e+14  5e+10  2e-02  4e-02
52: -1.1429e+14 -1.1431e+14  2e+10  2e-02  3e-02
53: -1.0963e+14 -1.0963e+14  8e+08  3e-02  4e-02
54: -1.1121e+14 -1.1121e+14  6e+09  2e-02  4e-02
55: -1.1580e+14 -1.1580e+14  2e+08  3e-02  3e-02
56: -1.1136e+14 -1.1136e+14  3e+07  6e-02  3e-02
57: -1.1396e+14 -1.1397e+14  2e+08  8e-02  4e-02
58: -1.1413e+14 -1.1413e+14  3e+06  2e-02  3e-02
59: -1.0972e+14 -1.0972e+14  4e+05  3e-02  3e-02
60: -1.0984e+14 -1.0984e+14  2e+06  2e-02  4e-02
```

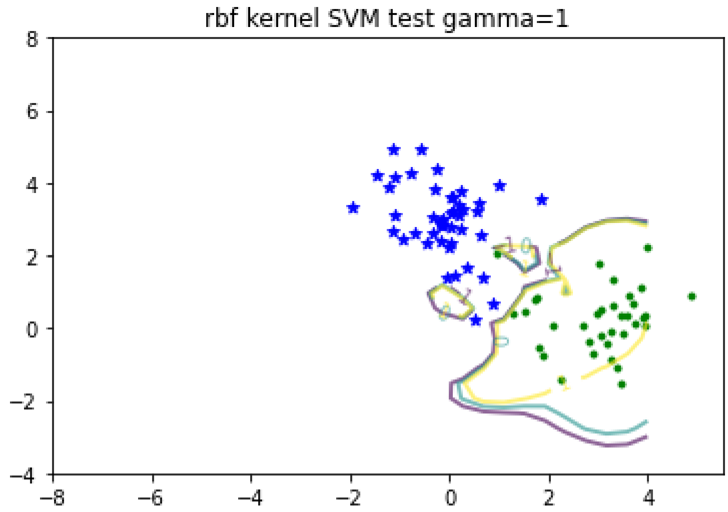
```
61: -1.1231e+14 -1.1231e+14  5e+05  4e-02  4e-02
62: -1.1320e+14 -1.1320e+14  2e+04  4e-02  4e-02
63: -1.0905e+14 -1.0905e+14  2e+04  5e-02  3e-02
64: -1.1345e+14 -1.1345e+14  1e+04  3e-02  3e-02
65: -1.1316e+14 -1.1316e+14  6e+02  3e-01  2e-02
66: -1.1157e+14 -1.1157e+14  5e+01  2e-02  2e-02
67: -1.1111e+14 -1.1111e+14  7e+00  3e-02  2e-02
68: -1.1219e+14 -1.1219e+14  6e-01  6e-03  2e-02
69: -1.1226e+14 -1.1226e+14  4e-02  3e-02  1e-02
70: -1.1199e+14 -1.1199e+14  4e-04  2e-02  1e-02
71: -1.1237e+14 -1.1237e+14  2e-05  2e-02  1e-02
72: -1.1351e+14 -1.1351e+14  4e-07  2e-02  1e-02
73: -1.1335e+14 -1.1335e+14  5e-08  3e-02  1e-02
74: -1.1288e+14 -1.1288e+14  9e-09  2e-02  1e-02
75: -1.1363e+14 -1.1363e+14  6e-10  3e-02  2e-02
76: -1.1129e+14 -1.1129e+14  7e-12  2e-02  1e-02
77: -1.1282e+14 -1.1282e+14  1e-12  4e-02  2e-02
78: -1.1157e+14 -1.1157e+14  7e-14  2e-02  1e-02
79: -1.1216e+14 -1.1216e+14  8e-15  2e-02  1e-02
80: -1.1260e+14 -1.1260e+14  1e-16  8e-03  1e-02
81: -1.1280e+14 -1.1280e+14  4e-18  2e-02  1e-02
82: -1.1329e+14 -1.1329e+14  5e-19  1e-02  1e-02
83: -1.1413e+14 -1.1413e+14  9e-20  3e-02  1e-02
84: -1.1197e+14 -1.1197e+14  6e-21  3e-01  1e-02
85: -1.1215e+14 -1.1215e+14  6e-22  2e-02  2e-02
86: -1.1325e+14 -1.1325e+14  1e-23  2e-02  1e-02
87: -1.1377e+14 -1.1377e+14  1e-25  3e-02  1e-02
88: -1.1217e+14 -1.1217e+14  2e-27  2e-02  2e-02
89: -1.1164e+14 -1.1164e+14  1e-28  1e-02  1e-02
90: -1.1245e+14 -1.1245e+14  1e-30  2e-02  1e-02
91: -1.1079e+14 -1.1079e+14  2e-31  1e-02  1e-02
92: -1.1299e+14 -1.1299e+14  3e-32  6e-02  2e-02
93: -1.1219e+14 -1.1219e+14  4e-34  6e-03  2e-02
94: -1.1297e+14 -1.1297e+14  1e-34  3e-02  1e-02
95: -1.1318e+14 -1.1318e+14  4e-36  3e-02  1e-02
96: -1.1348e+14 -1.1348e+14  1e-37  3e-02  1e-02
97: -1.1385e+14 -1.1385e+14  7e-39  2e-02  2e-02
98: -1.1118e+14 -1.1118e+14  7e-41  2e-02  2e-02
99: -1.1164e+14 -1.1164e+14  9e-42  2e-02  1e-02
100: -1.1262e+14 -1.1262e+14  3e-43  2e-02  1e-02
Terminated (maximum number of iterations reached).
rbf kernel SVM train accuracy: 1.0
rbf kernel SVM test accuracy: 1.0
     pcost       dcost       gap    pres   dres
 0: -3.7254e+01 -1.2568e+02  1e+03  2e+01  3e+00
 1: -1.0413e+02 -2.5491e+02  8e+02  1e+01  1e+00
 2: -3.2674e+02 -5.4193e+02  7e+02  1e+01  1e+00
 3: -7.7744e+02 -1.0551e+03  8e+02  9e+00  1e+00
 4: -1.9665e+03 -2.3733e+03  9e+02  9e+00  1e+00
 5: -4.8297e+03 -5.6123e+03  1e+03  8e+00  1e+00
 6: -8.1976e+03 -9.5012e+03  2e+03  8e+00  1e+00
 7: -1.5717e+04 -1.8844e+04  4e+03  7e+00  9e-01
 8: -2.5731e+04 -3.1793e+04  7e+03  5e+00  6e-01
 9: -2.9725e+04 -3.4706e+04  6e+03  3e+00  3e-01
10: -2.9567e+04 -3.4736e+04  6e+03  2e+00  3e-01
11: -3.1004e+04 -3.2442e+04  2e+03  5e-01  6e-02
12: -3.1121e+04 -3.1235e+04  1e+02  4e-03  5e-04
13: -3.1192e+04 -3.1206e+04  1e+01  4e-04  5e-05
14: -3.1199e+04 -3.1203e+04  4e+00  5e-05  7e-06
15: -3.1202e+04 -3.1203e+04  8e-01  8e-06  1e-06
16: -3.1203e+04 -3.1203e+04  1e-01  1e-06  1e-07
```

```
17: -3.1203e+04 -3.1203e+04  4e-03  2e-08  3e-09
Optimal solution found.
rbf kernel SVM train accuracy: 1.0
rbf kernel SVM test accuracy: 1.0
```









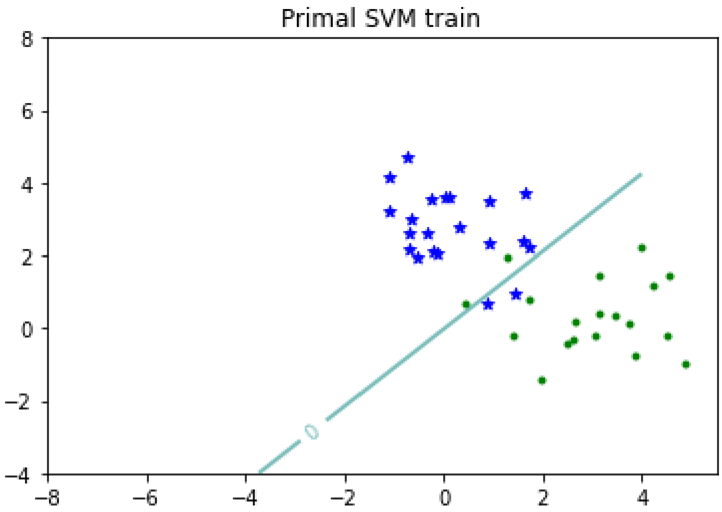
可见随着gamma的变小, 分类面也会变得更加平滑

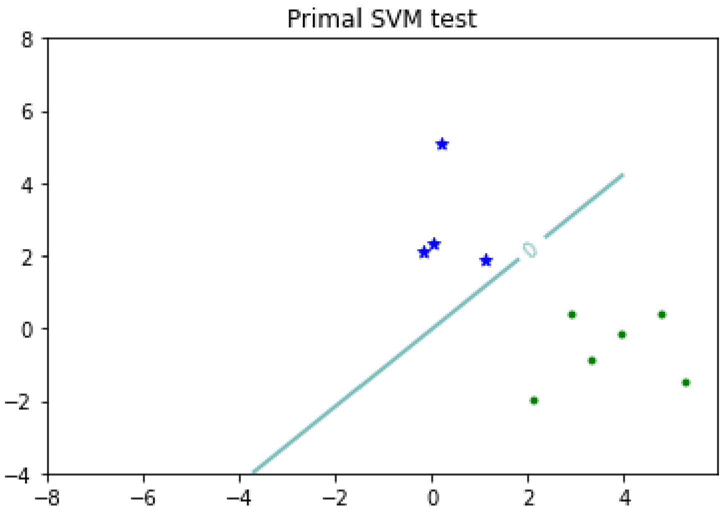
尝试调节数据量的大小

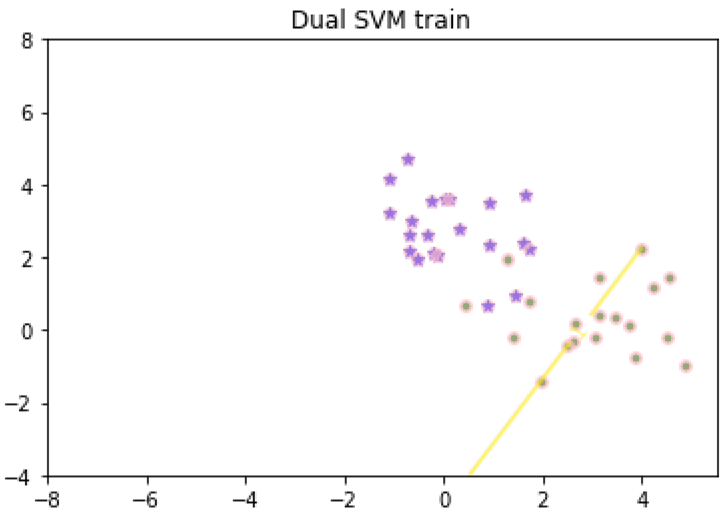
In [10]:
```
data=data_generator([3,0],np.eye(2),[0,3],np.eye(2),50,seed=SEED)
X_train,X_test,y_train,y_test=train_test_split(data[0],data[1],train_siz
e=0.8,test_size=0.2)
train_data=(X_train,y_train)
test_data=(X_test,y_test)
algorithm(train_data, test_data, axes)
```

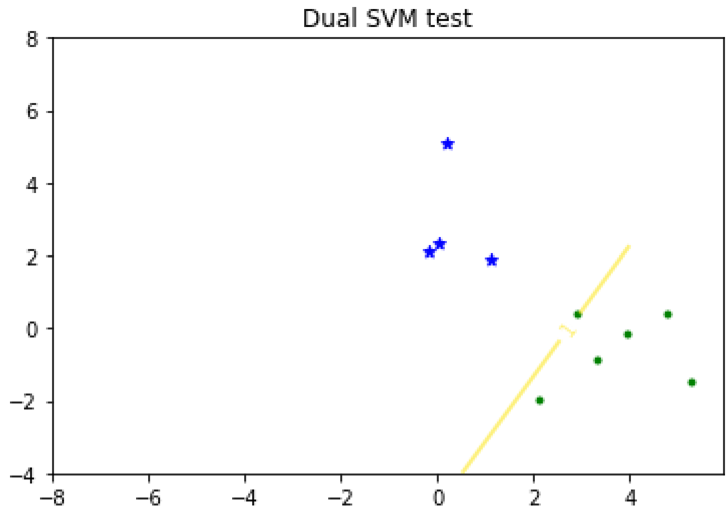
```
     pcost       dcost       gap    pres   dres
 0:  7.8487e-02  2.7079e+01  1e+02  2e+00  1e+02
 1:  3.2838e-01  1.4790e+01  9e+01  1e+00  8e+01
 2:  1.1135e-01  1.6841e+02  1e+02  1e+00  7e+01
 3:  1.0036e-03  7.0007e+02  1e+02  1e+00  6e+01
 4:  1.8775e-05  4.5441e+03  1e+02  1e+00  6e+01
 5:  9.7222e-06  2.9304e+04  5e+02  1e+00  6e+01
 6:  4.5461e-06  4.0878e+05  6e+03  1e+00  6e+01
 7:  2.2353e-07  2.2334e+07  8e+04  1e+00  5e+01
 8:  1.7189e-10  6.2370e+09  7e+05  1e+00  5e+01
 9:  2.4154e-14  5.8939e+13  7e+07  1e+00  5e+01
10:  2.4771e-18  4.6078e+19  6e+11  1e+00  1e+04
11:  1.2155e-21  3.5317e+27  6e+17  1e+00  2e+12
12:  1.7563e-23  1.9073e+37  1e+26  1e+00  5e+21
13:  2.8916e-23  8.0318e+45  4e+34  1e+00  1e+30
14:  1.8859e-23  3.4688e+54  2e+43  1e+00  4e+38
15:  3.8696e-23  1.5033e+63  8e+51  1e+00  4e+47
16:  3.0973e-24  2.2194e+71  1e+60  1e+00  2e+55
17:  3.4292e-23  5.1003e+79  3e+68  1e+00  1e+64
18:  4.9469e-23  8.4373e+85  5e+74  1e+00  5e+71
19:  7.0671e-23  3.2841e+92  2e+81  1e+00  9e+77
20:  7.1782e-23  1.9549e+97  1e+86  1e+00  1e+85
21:  1.2097e-22  8.7799e+99  5e+88  1e+00  3e+93
22:  1.2151e-22  8.0980e+101  4e+90  1e+00  3e+94
23:  1.2153e-22  2.0618e+102  1e+91  1e+00  1e+97
24:  1.2137e-22  5.3177e+102  3e+91  1e+00  4e+99
25:  5.8670e-23  8.9676e+104  5e+93  1e+00 1e+103
26:  6.1819e-23  9.2673e+106  5e+95  1e+00 1e+103
27:  8.8301e-23  3.2968e+107  2e+96  1e+00 3e+105
Terminated (singular KKT matrix).
primal svm train accuracy: 1.0
primal svm test accuracy: 1.0
```

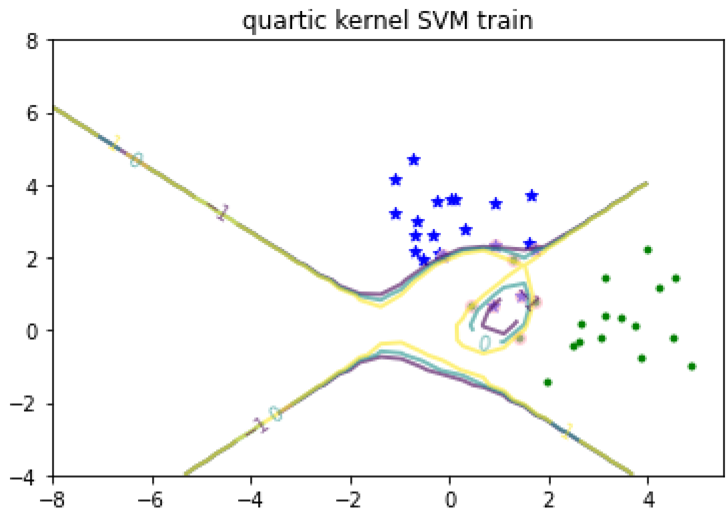
```
     pcost       dcost       gap    pres   dres
 0: -1.0305e+01 -2.2997e+01  1e+02  1e+01  2e+00
 1: -3.5690e+01 -5.0675e+01  9e+01  6e+00  1e+00
 2: -1.6331e+02 -2.1850e+02  1e+02  6e+00  1e+00
 3: -6.6625e+02 -7.9203e+02  1e+02  6e+00  1e+00
 4: -4.5144e+03 -4.6097e+03  1e+02  5e+00  1e+00
 5: -2.9275e+04 -2.9823e+04  5e+02  5e+00  1e+00
 6: -4.0875e+05 -4.1459e+05  6e+03  5e+00  1e+00
 7: -2.2334e+07 -2.2412e+07  8e+04  5e+00  1e+00
 8: -6.2733e+09 -6.2738e+09  5e+05  5e+00  1e+00
 9: -1.9640e+10 -1.9642e+10  2e+06  5e+00  1e+00
10: -3.1642e+10 -3.1644e+10  3e+06  5e+00  1e+00
11: -5.8218e+10 -5.8223e+10  5e+06  5e+00  1e+00
12: -1.3766e+11 -1.3767e+11  1e+07  5e+00  1e+00
13: -1.5225e+11 -1.5227e+11  1e+07  5e+00  1e+00
14: -2.8001e+11 -2.8003e+11  2e+07  5e+00  1e+00
15: -5.4205e+11 -5.4209e+11  3e+07  5e+00  1e+00
Terminated (singular KKT matrix).
Dual SVM train accuracy: 0.6
Dual SVM test accuracy: 0.6
     pcost       dcost       gap    pres   dres
 0: -7.2386e+00 -1.8240e+01  1e+02  8e+00  2e+00
 1: -3.0904e+01 -4.4719e+01  8e+01  5e+00  1e+00
 2: -7.9603e+01 -9.3657e+01  7e+01  4e+00  1e+00
 3: -1.6715e+02 -1.7102e+02  8e+01  3e+00  1e+00
 4: -1.5435e+02 -1.4343e+02  1e+02  3e+00  8e-01
 5: -1.1636e+02 -9.4123e+01  1e+02  2e+00  5e-01
 6: -7.4749e+01 -6.0731e+01  8e+01  9e-01  3e-01
 7: -4.5498e+01 -4.1897e+01  4e+01  3e-01  1e-01
 8: -4.2536e+01 -3.4953e+01  4e+01  3e-01  8e-02
 9: -3.3688e+01 -2.0691e+01  2e+01  1e-01  4e-02
10: -2.1367e+01 -1.0195e+01  1e+01  6e-02  2e-02
11: -1.4498e+01 -7.7981e+00  1e+01  3e-02  1e-02
12: -6.4133e+00 -4.4834e+00  5e+00  1e-02  3e-03
13: -4.8758e+00 -4.1304e+00  3e+00  5e-03  2e-03
14: -3.4972e+00 -3.7719e+00  3e-01  1e-15  8e-13
15: -3.6928e+00 -3.7039e+00  1e-02  1e-15  5e-13
16: -3.7012e+00 -3.7014e+00  1e-04  2e-15  5e-13
17: -3.7013e+00 -3.7013e+00  1e-06  1e-15  4e-13
Optimal solution found.
quartic kernel SVM train accuracy: 0.9
quartic kernel SVM test accuracy: 0.9
     pcost       dcost       gap    pres   dres
 0: -9.3912e+00 -2.7905e+01  1e+02  8e+00  2e+00
 1: -3.2496e+01 -5.9922e+01  9e+01  5e+00  1e+00
 2: -1.3176e+02 -1.9302e+02  1e+02  4e+00  1e+00
 3: -3.1376e+02 -4.0521e+02  1e+02  4e+00  1e+00
 4: -8.8910e+02 -1.0351e+03  2e+02  3e+00  1e+00
 5: -3.7251e+03 -4.0846e+03  4e+02  3e+00  1e+00
 6: -8.7254e+03 -9.6064e+03  9e+02  3e+00  1e+00
 7: -1.9540e+04 -2.2248e+04  3e+03  3e+00  1e+00
 8: -4.0927e+04 -4.9583e+04  9e+03  3e+00  9e-01
 9: -7.7206e+04 -9.8250e+04  2e+04  8e-01  3e-01
10: -7.8398e+04 -7.8856e+04  5e+02  2e-02  7e-03
11: -7.8399e+04 -7.8404e+04  5e+00  2e-04  7e-05
12: -7.8399e+04 -7.8399e+04  5e-02  2e-06  7e-07
13: -7.8399e+04 -7.8399e+04  5e-04  2e-08  7e-09
Optimal solution found.
rbf kernel SVM train accuracy: 1.0
rbf kernel SVM test accuracy: 1.0
```

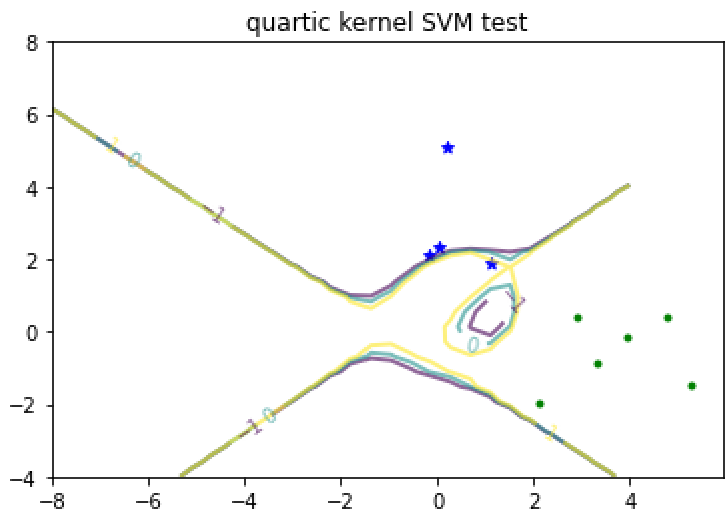

Primal SVM train
8
6
4
2
0
−2
−4
−8
−6
−4
−2
0
2
4
0

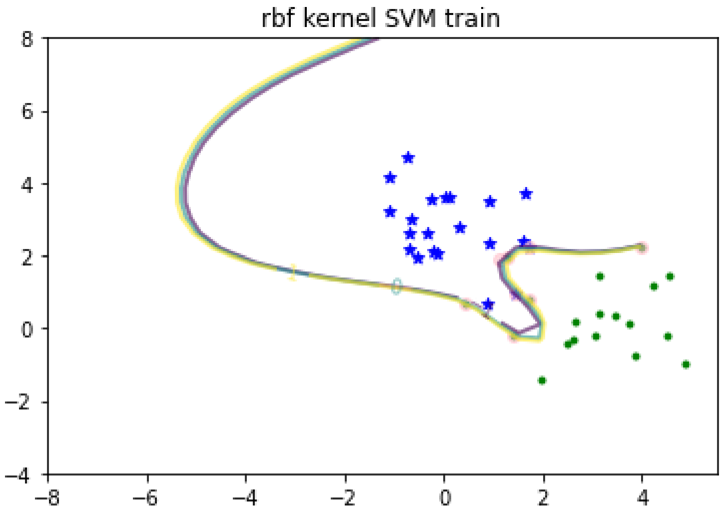

Primal SVM test
8
6
4
2
0
−2
−4
−8
−6
−4
−2
0
2
4
0

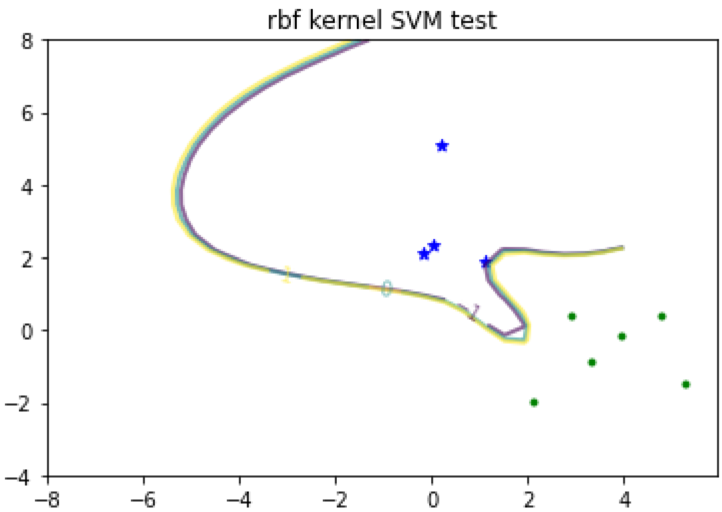

Dual SVM train
8
6
4
2
0
−2
−4
−8
−6
−4
−2
0
2
4


Dual SVM test


quartic kernel SVM train


quartic kernel SVM test





可见随着样本数量的减少, 数据变得更加线性可分后, 分类效果也有所好转, 同时rbf核SVM与四次核SVM效果均较为理想

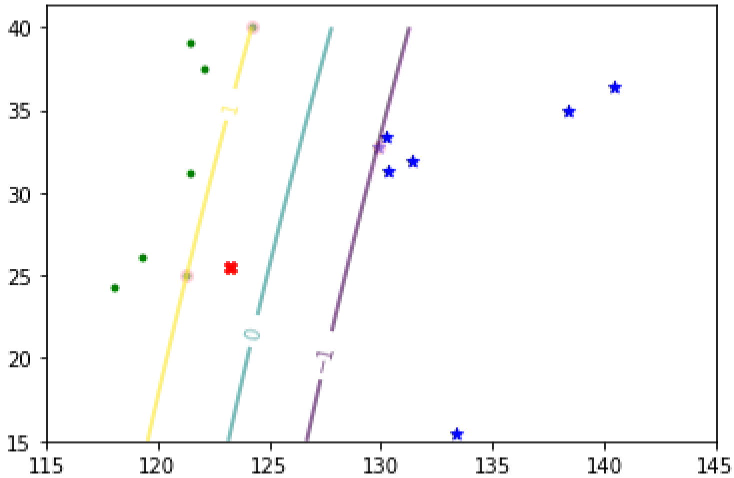
T5

In [11]:

```
## 仅仅使用沿海城市

x1=[[119.28,26.08],#福州
[121.31,25.03],#台北
[121.47,31.23],#上海
[118.06,24.27],#厦门
[121.46,39.04],#大连
[122.10,37.50],#威海
[124.23,40.07]]#丹东

x2=[[129.87,32.75],#长崎
[130.33,31.36],#鹿儿岛
[131.42,31.91],#宫崎
[130.24,33.35],#福冈
[133.33,15.43],#鸟取
[138.38,34.98],#静冈
[140.47,36.37]]#水户
X1=np.vstack((x1,x2))
y1=np.array([1.0]*7+[-1.0]*7)
data1=(X1,y1)
```

```python
## 钓鱼岛坐标
x=np.array([123.28,25.45]).reshape((1,2))

axes=[115,145,15,40]

model=Kernel_SVM('square')
model.fit(data1)
plot_svm_res(data1,model,axes)
plt.scatter(x[0,0],x[0,1],color='red',marker='X')
label2desc={1:'China',-1:'Japan'}
print('the predict label is ',label2desc[int(model.predict(x))])
```

```
     pcost       dcost       gap    pres   dres
 0: -1.3230e+00 -2.3742e+00  3e+01  5e+00  1e+00
 1: -9.9076e-01 -5.2166e-01  5e+00  1e+00  3e-01
 2: -1.1995e-02 -1.7384e-04  2e-01  3e-02  9e-03
 3: -1.4269e-04 -3.0382e-05  2e-03  4e-04  1e-04
 4: -2.4644e-05 -1.6925e-05  3e-04  5e-05  1e-05
 5: -1.2292e-06 -2.2903e-06  3e-05  3e-06  1e-06
 6:  2.9410e-07 -1.7126e-06  2e-06  4e-22  1e-13
 7: -2.6646e-07 -8.2611e-07  6e-07  1e-22  4e-14
 8: -6.1633e-07 -8.4720e-07  2e-07  8e-22  3e-14
 9: -6.9868e-07 -7.2017e-07  2e-08  2e-22  2e-14
Optimal solution found.
the predict label is  China
```



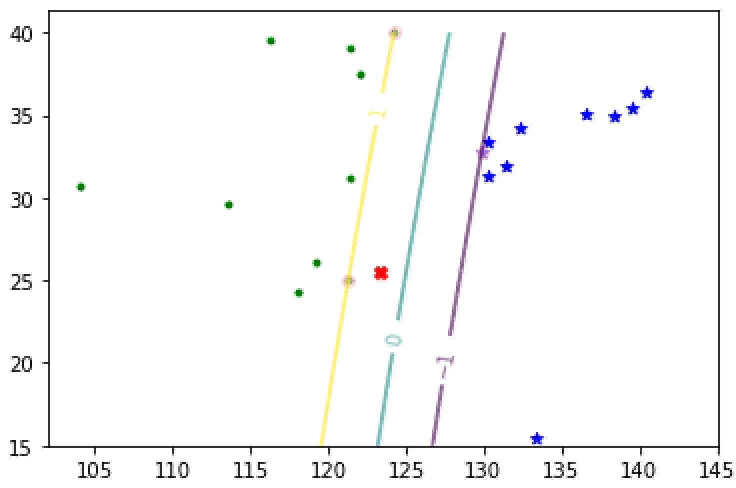
从预测结果可见, 钓鱼岛是属于中国的

增加内陆城市

In [12]:
```python
#添加内陆城市

xp1=[[119.28,26.08],#福州
[121.31,25.03],#台北
[121.47,31.23],#上海
[118.06,24.27],#厦门
[113.53,29.58],#武汉
[104.06,30.67],#成都
[116.25,39.54],#北京
[121.46,39.04],#大连
[122.10,37.50],#威海
[124.23,40.07]]#丹东
```

```
xp2=[[129.87,32.75],#长崎
[130.33,31.36],#鹿儿岛
[131.42,31.91],#宫崎
[130.24,33.35],#福冈
[136.54,35.10],#名古屋
[132.27,34.24],#广岛
[139.46,35.42],#东京
[133.33,15.43],#鸟取
[138.38,34.98],#静冈
[140.47,36.37]]#水户
X2=np.vstack((xp1,xp2))
y2=np.array([1.0]*10+[-1.0]*10)
data2=(X2,y2)
model=Kernel_SVM('square')
model.fit(data2)
plot_svm_res(data2,model,axes)
plt.scatter(x[0,0],x[0,1],color='red',marker='X')
label2desc={1:'China',-1:'Japan'}
print('the predict label is ',label2desc[int(model.predict(x))])
```

```
     pcost       dcost       gap    pres   dres
 0: -2.6873e+00 -5.2040e+00  4e+01  6e+00  2e+00
 1: -2.9112e+00 -1.6327e+00  1e+01  1e+00  4e-01
 2: -3.2298e-01 -2.7071e-02  1e+00  2e-01  5e-02
 3: -3.9486e-03 -2.5802e-05  2e-02  2e-03  6e-04
 4: -1.1727e-04 -2.0515e-05  5e-04  6e-05  2e-05
 5: -9.5005e-06 -1.8276e-06  4e-05  4e-06  1e-06
 6: -1.3364e-07 -1.5160e-06  1e-06  1e-09  3e-10
 7: -4.7164e-07 -8.7776e-07  4e-07  3e-10  9e-11
 8: -6.0351e-07 -7.9785e-07  2e-07  5e-11  1e-11
 9: -7.0618e-07 -7.1683e-07  1e-08  2e-12  6e-13
Optimal solution found.
the predict label is  China
```



从图中标出的支撑向量可以看出，增加的内陆城市对于预测并没有造成影响，支撑向量未发生改变，而且，增加内陆城市并不影响分类结果，钓鱼岛依旧是中国的