In [50]:

```
train_data = 'train_data.txt'
train_label = 'train_label.txt'
test_data = 'test_data.txt'
test_label = 'test_label.txt'

import pandas as pd
import numpy as np
from cvxopt import matrix
from cvxopt import solvers
import matplotlib.pyplot as plt
```

In [51]:

```
mean = [0]
std = [0]


# -- helper functions
def getNumpyArray(file_name, preprocess = False, train = True):
    x = pd.read_csv(file_name, sep=" ", header=None)
    if preprocess:
        if train:
            mean[0] = x.mean(axis = 0)
        x = x - mean[0]
        if train:
            std[0] = x.std(axis = 0)
        x = x / std[0]
    x = x.to_numpy()
    return x

def I(n):
    return np.eye(n)
```

In [52]:

```
X_train = getNumpyArray(train_data, preprocess = True)
Y_train = getNumpyArray(train_label).reshape(-1)
X_test = getNumpyArray(test_data, preprocess = True, train = False)
Y_test = getNumpyArray(test_label).reshape(-1)
C = 0.5
```

In [53]:

```
def train_svm(X, Y, C, normalized = True):

    if not normalized:
        mean[0] = X.mean(axis = 0)
        X = X - mean[0]
        std[0] = X.std(axis = 0)
        X = X / std[0]

    N = X.shape[0]
    D = X.shape[1]

    assert len(Y.shape) == 1, 'dim error for label'
    assert Y.shape[0] == N, 'label shape[0] != N'

    temp = Y.reshape(-1,1) * X
    H = np.dot(temp , temp.T)
    P = matrix(H)*1.0 # - make it d type
    q = matrix(np.ones((N)) * -1)
    G = matrix(np.vstack((-I(N), I(N))))
    h = matrix(np.hstack((np.zeros(N), np.ones(N)*C)))
    A = Y.reshape(1,-1)
    A = A.astype('float')
    A = matrix(A)
    b = matrix(np.array([0.0]))

    solvers.options['show_progress'] = False # turn off the output

    sol = solvers.qp(P,q,G,h,A,b)

    alpha = sol['x']
    alpha = np.array(alpha).reshape(-1)
    assert len(alpha.shape) == 1, 'dim erro on alpha'
    assert alpha.shape[0] == N, 'alpha.shape[0] != N'

    w = np.dot((alpha * Y).T,X)
    w = w.reshape(-1)

    b = []
    for i in range(len(alpha)):
        if 10**-4 < alpha[i] < C:
            b.append(Y[i] - np.dot(w,X[i].T))

    assert len(w.shape) == 1, 'dim error on w'
    assert w.shape[0] == D, 'w.shape[0] != D'

    return w,b


def test_svm(X, Y, w, b, need_normalized = False):
    if need_normalized:
        X = X - mean[0]
        X = X / std[0]

    best = 0
    for b_ in b:
        pred_y = np.sign(np.dot(X,w)+b_)
        Y=Y.reshape(-1)

        acc = np.sum(pred_y==Y)/Y.shape[0]
        if acc > best:
            best = acc

    return best
```

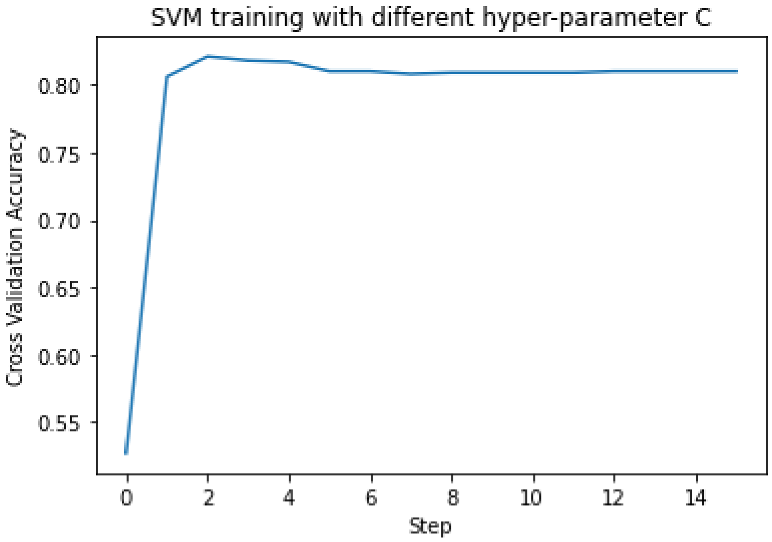
In [54]:

```
step = []
acc_list = []

X_train = getNumpyArray(train_data)
Y_train = getNumpyArray(train_label).reshape(-1)

best_C = np.inf
best_acc = 0
# - choose C with cross-validation
for i,c in enumerate(range(-6, 10)):
    step.append(i)
    C = 4**c
    total_acc = []
    for f in range(5):
        lo, hi = f*200, (f+1)*200
        validation_X = X_train[lo:hi]
        validation_Y = Y_train[lo:hi]
        if lo > 0:
            train_X = np.vstack((X_train[0:lo], X_train[hi:]))
            train_Y = np.hstack((Y_train[0:lo], Y_train[hi:]))
        else:
            train_X = X_train[hi:]
            train_Y = Y_train[hi:]
        w,b = train_svm(train_X, train_Y, C, normalized = False)
        acc_vali = test_svm(validation_X, validation_Y, w, b, need_normalized =
        if acc_vali > best_acc:
            best_C = C
            best_acc = acc_vali
        total_acc.append(acc_vali)
    total_acc = np.mean(total_acc)
    acc_list.append(total_acc)
    print('C = {%5.4f} validation acc: {%5.4f}' %(C,total_acc))
plt.title('SVM training with different hyper-parameter C')
plt.xlabel('Step')
plt.ylabel('Cross Validation Accuracy')
plt.plot(step, acc_list)
```

```
C = {0.0002} validation acc: {0.5270}
C = {0.0010} validation acc: {0.8060}
C = {0.0039} validation acc: {0.8210}
C = {0.0156} validation acc: {0.8180}
C = {0.0625} validation acc: {0.8170}
C = {0.2500} validation acc: {0.8100}
C = {1.0000} validation acc: {0.8100}
C = {4.0000} validation acc: {0.8080}
C = {16.0000} validation acc: {0.8090}
C = {64.0000} validation acc: {0.8090}
C = {256.0000} validation acc: {0.8090}
C = {1024.0000} validation acc: {0.8090}
C = {4096.0000} validation acc: {0.8100}
C = {16384.0000} validation acc: {0.8100}
C = {65536.0000} validation acc: {0.8100}
C = {262144.0000} validation acc: {0.8100}
```

Out[54]:

```
[<matplotlib.lines.Line2D at 0x7f8718087d30>]
```



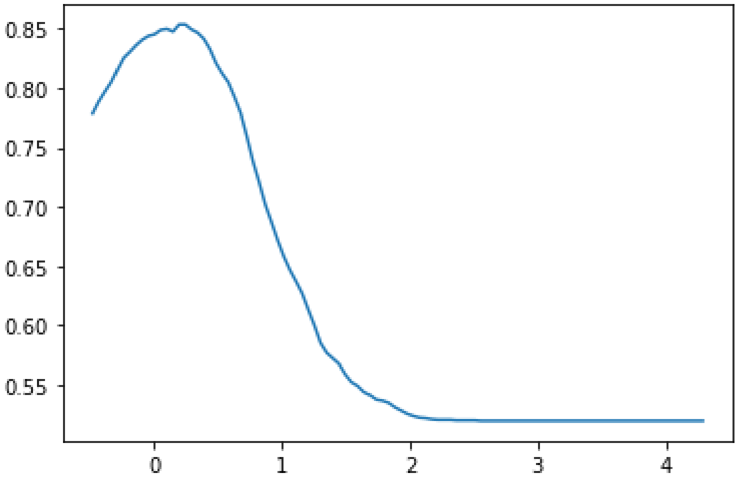
In [55]:

```
w,b = train_svm(X_train, Y_train, best_C)
B = np.linspace(min(b),max(b),100)
```

```
acc_list = []
for _b in B:
    acc = test_svm(X_test, Y_test, w, [_b])
    acc_list.append(acc)
plt.plot(B, acc_list)
```

Out[55]:

```
[<matplotlib.lines.Line2D at 0x7f87621c85b0>]
```



In [ ]: