In [36]:

```
import pandas as pd
import numpy as np
import os
import math
import matplotlib.pyplot as plt
import re
```

Data preprocessing

1. transform the data into correct format
2. seperate the test/train data
3. transform male/female/infants into numbers -1/1/0
4. transform rings in to binary: if rings <=9: rings = 1, else rings = -1

In [2]:

```
##### readin data and make some changes #####
# Before read 'abalone.data', I manually added the name of each column based on the
# description of abalone.name to abalone.data
df = pd.read_csv('abalone.data')
# df.columns

# transforming sex in to numbers, so that libsvm can deal with it.
sex_map = {'M':-1, 'I': 0, 'F':1}
df['Sex'] = df['Sex'].map(sex_map)

# reordering the columns of dataframe so that Rings appears first
df = df[['Rings','Sex', 'Length', 'Diameter', 'Height', 'Whole weight', 'Shucked weight', \
         'Viscera weight', 'Shell weight']]

# Binarilize the Rings. if Ring > 9, Ring = -1; else Ring = 1
df['Rings'] = np.where(df['Rings']<=9,1,-1)

training = df[:3133]
testing = df[3133:]

# saving the preprocessed csv
training.to_csv('./train.csv', index=False, header=False)
testing.to_csv('./test.csv', index=False, header=False)
df.to_csv('./total.csv', index=False, header=False)
```

In [3]:

```
df
```

Out[3]:

	Rings	Sex	Length	Diameter	Height	Whole weight	Shucked weight	Viscera weight	Shell weight
0	-1	-1	0.455	0.365	0.095	0.5140	0.2245	0.1010	0.1500
1	1	-1	0.350	0.265	0.090	0.2255	0.0995	0.0485	0.0700
2	1	1	0.530	0.420	0.135	0.6770	0.2565	0.1415	0.2100
3	-1	-1	0.440	0.365	0.125	0.5160	0.2155	0.1140	0.1550
4	1	0	0.330	0.255	0.080	0.2050	0.0895	0.0395	0.0550
...	...	...	...	...	...	...	...	...	...
4172	-1	1	0.565	0.450	0.165	0.8870	0.3700	0.2390	0.2490
4173	-1	-1	0.590	0.440	0.135	0.9660	0.4390	0.2145	0.2605
4174	1	-1	0.600	0.475	0.205	1.1760	0.5255	0.2875	0.3080
4175	-1	1	0.625	0.485	0.150	1.0945	0.5310	0.2610	0.2960
4176	-1	-1	0.710	0.555	0.195	1.9485	0.9455	0.3765	0.4950

4177 rows × 9 columns

In [4]:

```
##### Formatting data #####

# -1 1:1 2:-0.749474 3:-0.181429 5:-0.538462 6:-0.25 7:-0.888772 8:-1 9:-1 10:-1 11:1 13:-0.9
# a example of preprocessed data
datass = ['test','train','total']
for data in datass:
    with open(data + '.intermediate', mode='w') as f:
        for line in open('./'+ data + '.csv'):
            class_and_attr = line.split(',')
            output_string = class_and_attr[0] + ' '
            rg = len(class_and_attr)
            for i in range(1,rg):
                output_string += str(i) + ':' + class_and_attr[i] + ' '
            f.write(output_string)
        f.close()
    # there will be weird space at begining of each line after the above code running,
    # so I add this step to remove the spaces
    with open(data + '.unscaled', mode='w') as f:
        for line in open(data+'.intermediate'):
            f.write(line.lstrip())


# using tools checking if the format is correct
os.system('python3 ./libsvm/tools/checkdata.py total.unscaled')
```

Out[4]:

```
0
```

In [5]:

```
##### scaling the data #####
# I will use the bash within this notebook
# so that to keep a record of how I did it.
os.system('./libsvm/svm-scale  -s scaling_parameter train.unscaled > train.scaled')
os.system('./libsvm/svm-scale  -r scaling_parameter test.unscaled > test.scaled')
os.system('./libsvm/svm-scale  -r scaling_parameter total.unscaled > total.scaled')
```

Out[5]:

```
0
```

Model Selection

1. For d = 1,2,3,4, plot the 'cross-validation accuracy ± std' as a function of C.
2. From plots in 1. find best parameters (C,*d*)

In [19]:

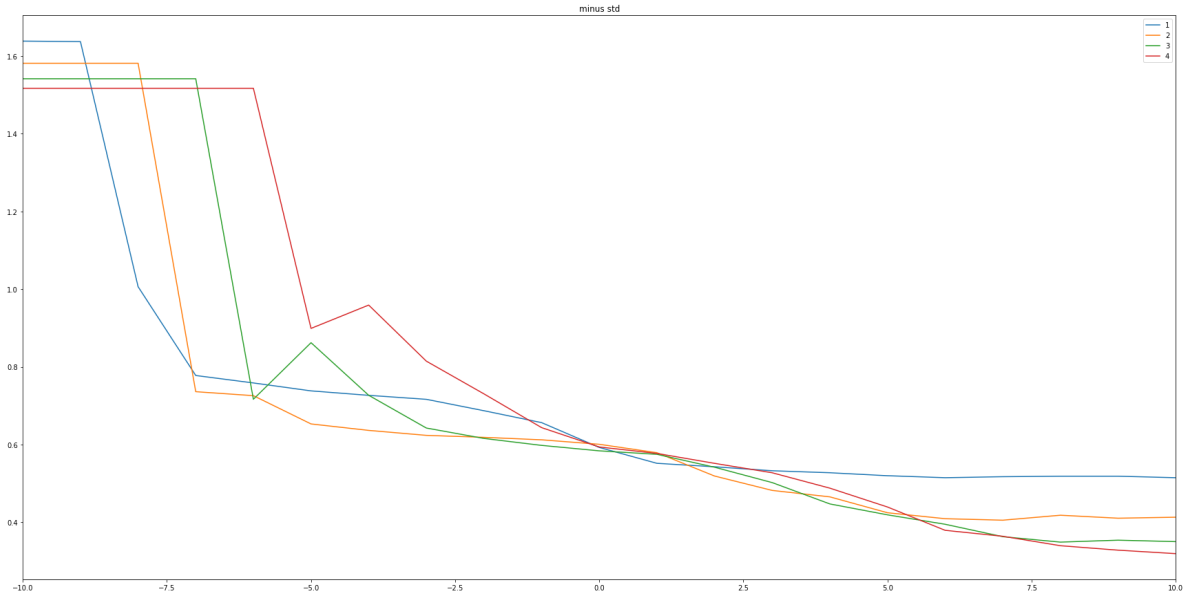
```
##### running cross-validation for best parameters#####
# a short helpful function
def read_mse(result):
    return float(result.split('\n')[-2].split('=')[-1])
# running cross-validation
range_of_c = 10
dic = {}
f = open('results_mse.txt', 'w')
error = open('errors.txt','w')
for d in range(1,5):
    subdic = {}
    for k in range(-range_of_c, range_of_c+1):
        c = 2**k
        result = os.popen('./libsvm/svm-train -t 1 -d '+ str(d) + \
                          ' -c ' + str(c) + ' -v 10 train.scaled').read()
        f.write("d, c = " + str(d) + ',' + str(c) + ':\n' + result + "!")
        subdic[k] = read_mse(result)
    dic[d] = subdic
```

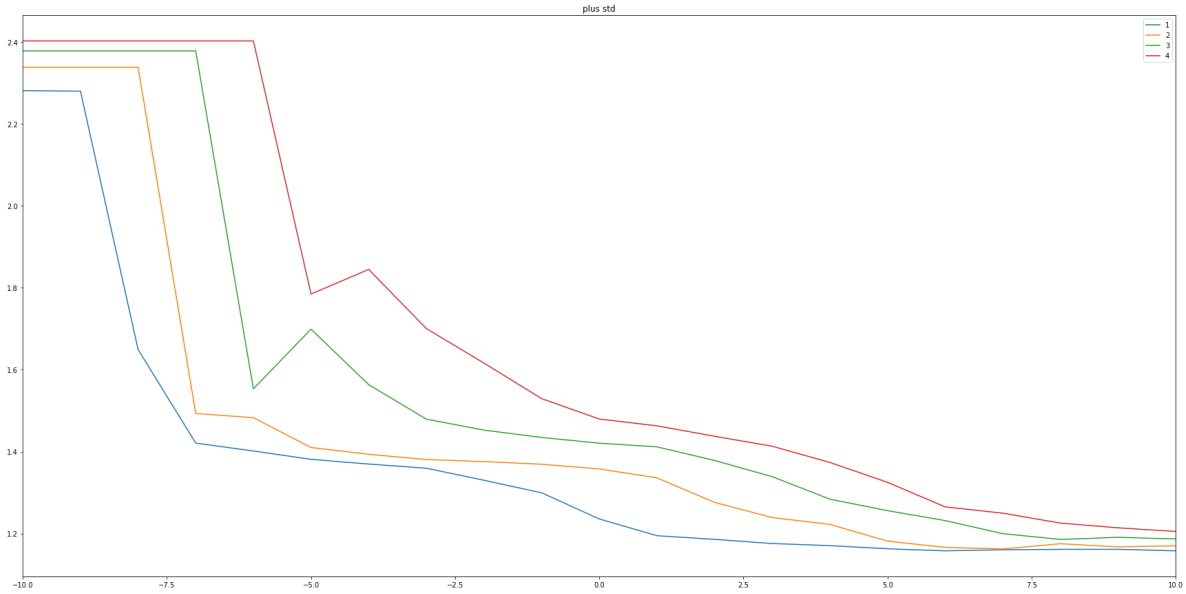
In [28]:

```
##### plotting the graph for cross-validation-error #####
msedf = pd.DataFrame(dic)
msedf_minus_std = pd.DataFrame()
msedf_plus_std = pd.DataFrame()
for d in range(1,5):
    msedf_minus_std[d] = msedf[d] - np.std(msedf[d])
    msedf_plus_std[d] =  msedf[d] + np.std(msedf[d])
msedf_minus_std.plot(figsize=(30,15),title='minus std')
msedf_plus_std.plot(figsize=(30,15),title='plus std')
```

Out[28]:

<AxesSubplot:title={'center':'plus std'}>





Selecting parameter

From the above graph, the color represends different value of d. The y-axis is cross-validation error. The x-axis is the log_2_ C.

I've intended to have a larger range of C, but libsvm will warn me: exceeding max iteration limit.

Selecting from 'minus std', the best pair I choose is (C*,d*) = (2^10=1024, 4)

Plotting stuff

Fix C = 1024

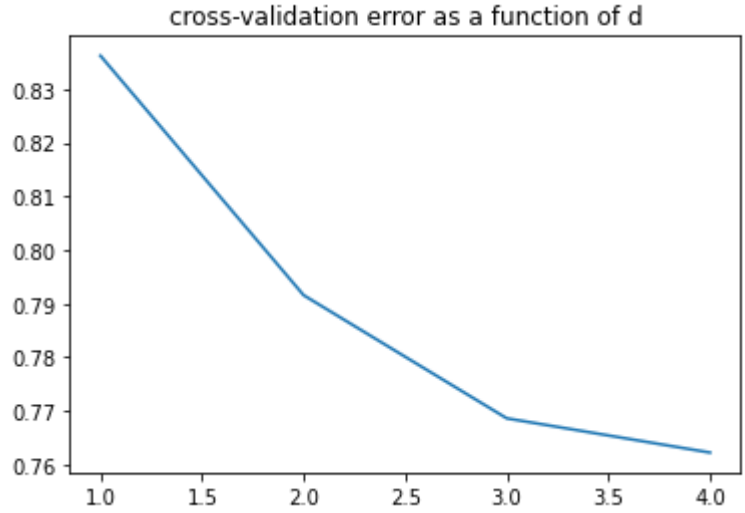
1. plot the ten-fold cross-validation error and the test error as a function of d.
2. plot the average number of support vectors as a function of d
3. How many support vectors lie in the margin hyperplane?

In [35]:

```
##### getting data needed for plotting the graph in C.5 #####
c = 1024

# test error #
## 1. I runned the command "./libsvm/svm-train -t 1 -d [i] -c 1024 train.scaled d[i].model"
##      where [i]=1,2,3,4 and get the model file
#
## 2. I runned the command "./libsvm/svm-predict test.scaled d[i].model d[i].output" in bash
##      where [i]=1,2,3,4 and get the test error
test_error = {1:0.911877, 2:0.881226, 3:0.858238, 4:0.854406}

# cross validation error #
mse = {}
for d in range(1,5):
    mse[d] = read_mse(os.popen('./libsvm/svm-train -t 1 -d '+ str(d) + ' -c ' \
                               + str(c) + ' -v 10 train.scaled').read())

# number of support vectors and those in the margin #
# in the process of calculating test error I also collected number of supp vect
# and those in the margin
num_supp_vec = {1:1533, 2:1466, 3:1457, 4:1473}
num_supp_mar = {1:1522, 2:1438, 3:1414, 4:1415}
```

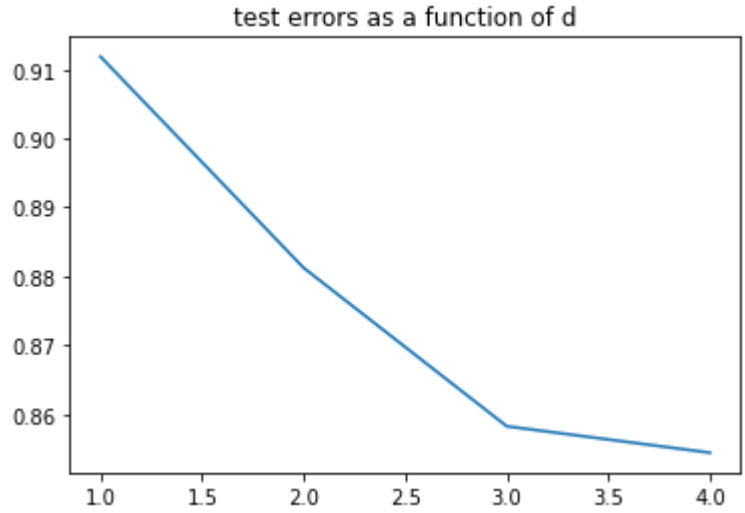
In [48]:

```
# tenfold cross-validation error as a function of d
plt.plot(list(mse.keys()),list(mse.values()))
plt.title('cross-validation error as a function of d')
plt.show()
```



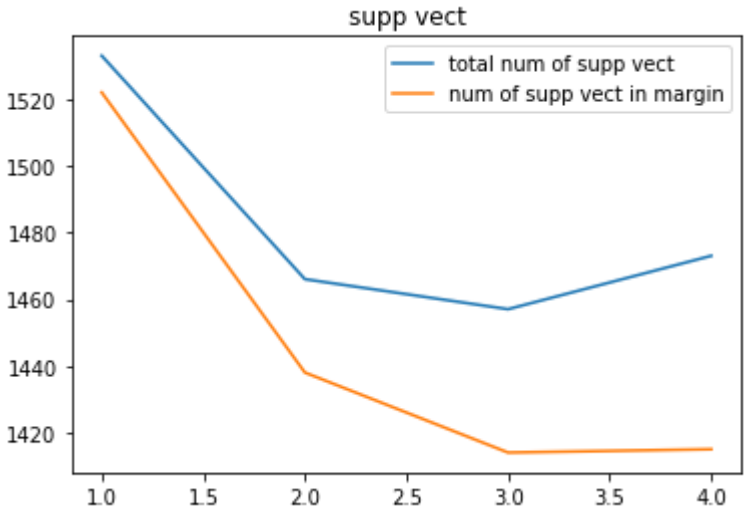
In [49]:

```
# test errors
plt.plot(list(test_error.keys()), list(test_error.values()))
plt.title('test errors as a function of d')
plt.show()
```



In [52]:

```
# total number of support vectors, and those on the plane
plt.plot(list(num_supp_vec.keys()),list(num_supp_vec.values()), label='total num of supp vect')
plt.plot(list(num_supp_mar.keys()),list(num_supp_mar.values()), label='num of supp vect in margin')
plt.title('supp vect')
plt.legend()
plt.show()
```



Remarks

The original libsvm returns cross-validation-accuracy instead of the cross-validation error. I made some change to the original program to make it return the cross-validation error. However, I believe these 2 notion should be equivalent in classification problem.