Yong-Chan Shin

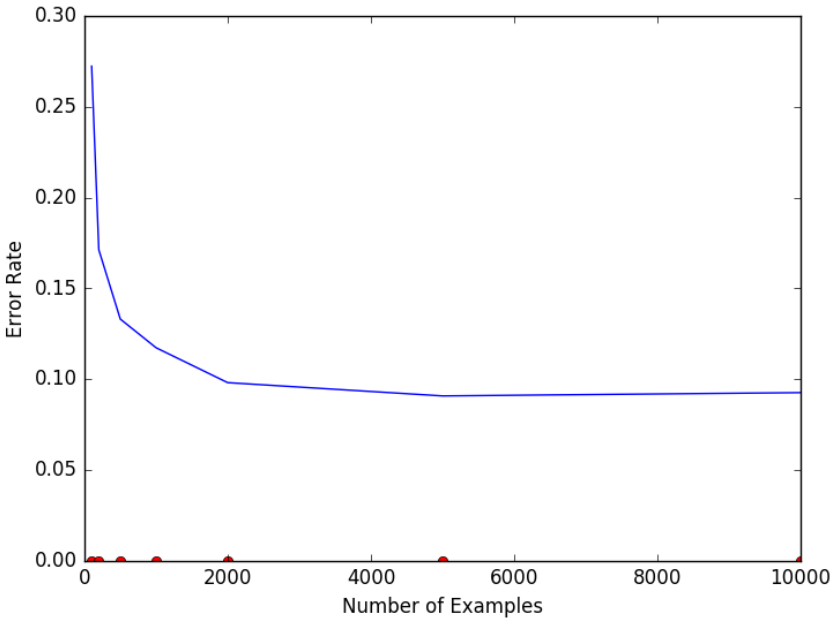
CS189

30 Jan 2017

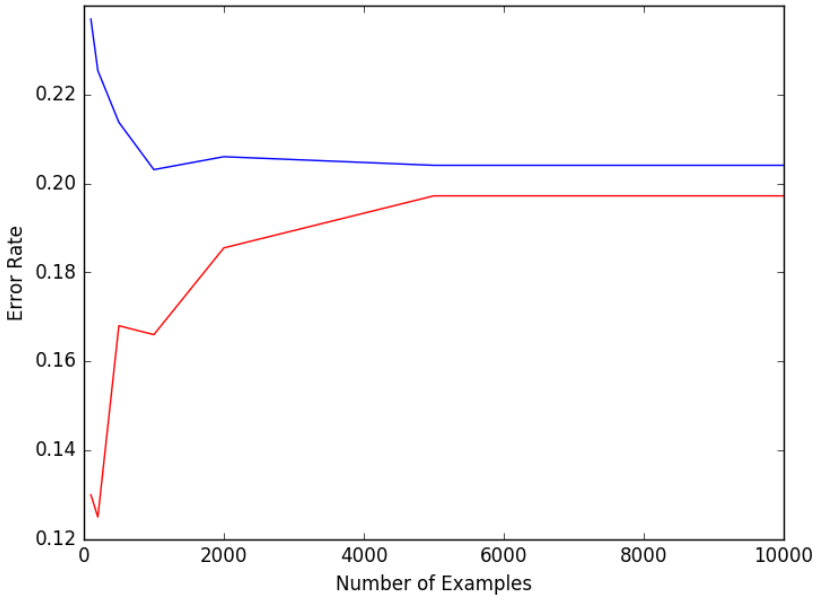
HW 1: Write-up

Problem 1: In prob1_2_digits function, it loads the matrix of train.mat file in mnist folder. Then use shuffle function of numpy array to shuffle the rows of the training matrix. Copy the first 10000 rows of the matrix as validation set and copy the rest as training set. In prob1_2_spam() function, load spam_data.mat in spam folder which contains training data and its labels. Different with the mnist data, the labels are stored separately, so I used concatenate function of numpy to make the training data and its labels have the same structure with MNIST data set. Copy the first 1/5 rows of the matrix as validation set and copy the rest as training set. In prob1_2_CIFAR function, do the same things with prob1_2_digits function but copy the first 5000 rows of the matrix as validation set and copy the rest as training set.

Problem 2: Most of code for this problem is errors function. At this point, since we do not consider the value of C for support vector machine, we use the default value 1.0. Train with training set, and compute the scores of training set and validation set. For all three graphs below, red line (or red dot) is training set's error rate, and blue line is validation set's error rate.

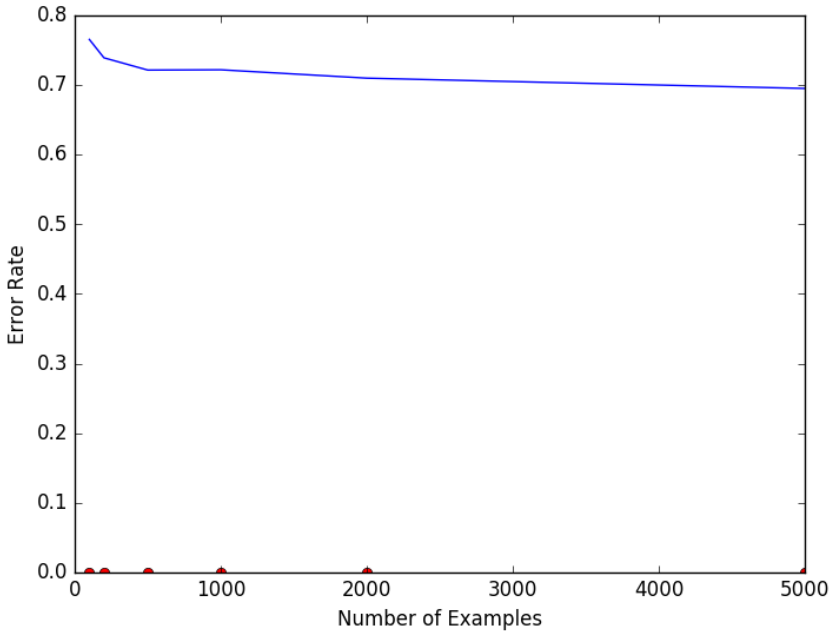


Above graph is error rate of MNIST data set. The accuracy of training set is 100% and the accuracy of validation set becomes 90% as more samples are used for training.



The graph above is spam dataset result. The error rate of validation set is decreasing as the

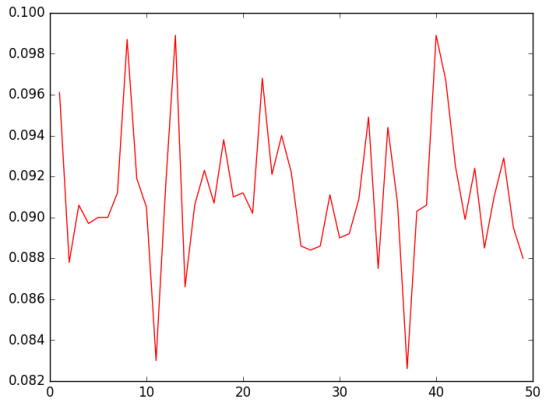
number of examples increase; the more samples we have, the more accurate the validation set score is. In contrast, the training set has higher accuracy with less samples because we test the accuracy with the trained data; for instance, as an extreme case, if we train the support vector machine with a single datum and test it with the datum, the accuracy is expected to be very high while the validation set accuracy will be extremely low.

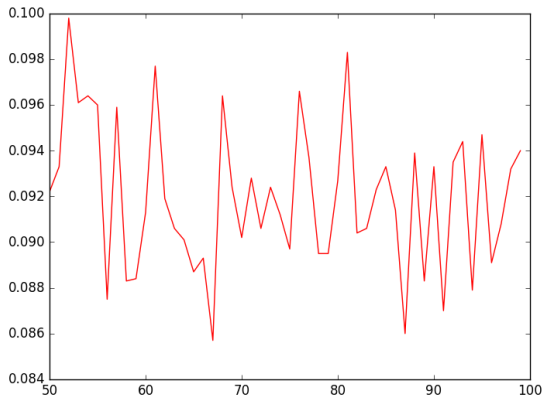


Above graph is error rate of CIFAR-10 data set. The accuracy of training set is 100% and the accuracy of validation set becomes around 30% as more samples are used for training.

Problem 3: In this problem, I tried various values for C: First, 0.001, 0.01, 0.1, 1.0, 10.0, 100.0, 1000.0. When I initially run the code, I thought 0.1 has highest accuracy, and I tried several values between 0.01 and 0.19. I iterate the first list 10 times and found that 10.0 and 100.0 result in highest

accuracy, not 0.1. Then I run values with 1 to 10, 1 to 49, and 50 to 99.





The graphs above are the result for C values between 1 and 99 to see which value results in the most stable and lowest error rate of validation set. The C value I choose is 11. The values I tried and corresponding validation set error with 10000 examples written in page 16 to 27 of write-up.

Problem 4: In this problem, I tried lots of different values written at the last part (from page 27) of this write-up. Used KFold function for k-fold cross-validation. This time, the code runs 10 times for each C value list to find the most stable C value, and the final choice is 97.

Problem 5: Using the C values I got from the previous two problems, trained the whole training data this time (instead of dividing it into the training set and validation set). Saved the result as a .csv file. The file saved right after running hw01.py has its first column as category and second column as ID, so I open the saved file with Excel and manually swapped the columns. The file for MNIST digits is "kaggle_ycls_hw1_digits.csv" and the file for spam is "kaggle_ycls_hw1_spam.csv". The MNIST digit test set's accuracy according to Kaggle was 0.91280. The spam test set's accuracy according to Kaggle was 0.84085 (User name: YONG-CHAN_SHIN).

Code:

```
import scipy.io
#from random import shuffle
import numpy as np
from sklearn import svm
from sklearn.cross_validation import KFold
import matplotlib.pyplot as plt
import csv
import pandas as pd


def main():
	prob1_2_digits()
	#prob1_2_spam()
	#prob1_2_CIFAR()
	#prob3_digits()
	#prob4_spam()
	#prob5()



# This function does the same with score function. It was not used.
def compute_score(svc,data_set,labels):
	pred = svc.predict(data_set)
	error_count = 0
```

```
        for tup in zip(pred,labels):
                if tup[0]!=tup[1]:
                        error_count+=1
        return float(labels.shape[0]-error_count)/float(labels.shape[0])


def errors(training_set,validation_set,nums_example,C_val=1.0):
        t_error=[]
        v_error=[]
        svc_t = svm.SVC(kernel='linear', C=C_val)
        #svc_t = svm.LinearSVC(C=C_val)
        #print "current C_val:",C_val
        for num_example in nums_example:
                t_set = training_set[:num_example,:training_set.shape[1]-1]
                t_label = training_set[:num_example,training_set.shape[1]-1]
                v_set = validation_set[:,:validation_set.shape[1]-1]
                v_label = validation_set[:,validation_set.shape[1]-1]
                svc_t = svc_t.fit(t_set, t_label)
                #svc_t = svm.LinearSVC(C=C_val).fit(t_set,t_label)
                score_t = svc_t.score(t_set,t_label)
                #score_t = compute_score(svc_t,t_set,t_label)
                #print 'score_t:',score_t
                t_error.append(1.0-score_t)
                score_v = svc_t.score(v_set,v_label)
```

```
                #score_v = compute_score(svc_v,v_set,v_label)
                #print 'score_v:',score_v
                v_error.append(1.0-score_v)
        return [t_error,v_error,svc_t]


def prob1_2_digits(problem3=False,c_val=1.0):
        if problem3==False:
                print "Now in prob1_2_digits"
        # problem 1
        nums_example = []
        if problem3==False:
                nums_example = [100, 200, 500, 1000, 2000, 5000, 10000]
        else:
                nums_example = [10000]
        mat_contents = scipy.io.loadmat('mnist/train.mat')
        # print mat_contents
        # print 'mat_contents type:',type(mat_contents)
        # print 'matshape:',mat_contents.keys()
        # print mat_contents['trainX']
        # print 'mat shape:',mat_contents['trainX'].shape
        training_set = mat_contents['trainX']
        np.random.shuffle(training_set)
        validation_set = training_set[:10000,:]
```

```
        training_set = training_set[10000:,:]
        print 'validation shape', validation_set.shape
        print 'training shape', training_set.shape

        # problem 2
        error = errors(training_set,validation_set,nums_example,c_val)
        t_error = error[0]
        v_error = error[1]
        #print "t_error:",t_error
        svc = error[2]
        #print "v_error:",v_error
        if problem3==False:
                plt.plot(nums_example,t_error,'ro',nums_example,v_error,'b-')
                plt.xlabel('Number of Examples')
                plt.ylabel('Error Rate')
                plt.show()
        return [v_error,svc]

# print 'training_data.shape:',training_data.shape
# print 'validation_set.shape:',validation_set.shape
# print 'validation_labels.shape:',validation_labels.shape
# print 'training_set.shape:',training_set.shape
# print 'training_labels.shape:',training_labels.shape
```

```
# print 'training_data_labels.shape:',training_data_labels.shape
def prob1_2_spam():
        print "Now in prob1_2_spam"
        # problem 1
        nums_example= [100, 200, 500, 1000, 2000, 5000, 10000]
        mat_contents = scipy.io.loadmat('spam/spam_data.mat')
        training_data = mat_contents['training_data']
        #test_data = mat_contents['test_data']
        training_data_labels = mat_contents['training_labels']
        training_data = np.concatenate((training_data,training_data_labels.T),axis=1)
        #print training_data[0]
        np.random.shuffle(training_data)
        #print training_data[0]
        validation_set = training_data[:training_data.shape[0]//5,:]
        #validation_labels = training_data[:training_data.shape[0]//5,training_data.shape[1]-1]
        training_set = training_data[training_data.shape[0]//5:,:]
        print 'validation shape', validation_set.shape
        print 'training shape', training_set.shape

        # problem 2
        error = errors(training_set,validation_set,nums_example)
        t_error = error[0]
        v_error = error[1]
```

```
        svc = error[2]
        plt.plot(nums_example,t_error,'r-',nums_example,v_error,'b-')
        plt.xlabel('Number of Examples')
        plt.ylabel('Error Rate')
        plt.show()
        return svc


def prob1_2_CIFAR():
        # problem 1
        print "Now in prob1_2_CIFAR"
        nums_example= [100, 200, 500, 1000, 2000, 5000]
        mat_contents1 = scipy.io.loadmat('cifar/train.mat')
        #mat_contents2 = scipy.io.loadmat('cifar/test.mat')
        #print mat_contents1
        #print mat_contents2
        training_data = mat_contents1['trainX']
        #test_data = mat_contents2['testX']
        #print 'training_data.shape:',training_data.shape
        #print 'test_data.shape:',test_data.shape
        np.random.shuffle(training_data)
        validation_set = training_data[:5000]
        training_set = training_data[5000:]
        print 'validation shape', validation_set.shape
```

```
        print 'training shape', training_set.shape


        # problem 2
        error = errors(training_set,validation_set,nums_example)
        t_error = error[0]
        v_error = error[1]
        svc = error[2]
        plt.plot(nums_example,t_error,'ro',nums_example,v_error,'b-')
        plt.xlabel('Number of Examples')
        plt.ylabel('Error Rate')
        plt.show()
        return svc


def prob3_digits():
        print "Now in prob3_digits"
        C_list = [11]
        v_errors = []
        for C_val in C_list:
                #print "C_val:",C_val
                v_error = prob1_2_digits(problem3=True, c_val=C_val)[0]
                #print "v_error:",v_error
                v_errors.append(v_error[0])
        plt.plot(C_list,v_errors,'r-')
```

```
        plt.show()



def prob4_spam():
        print "Now in prob4_spam"
        #nums_example= [100, 200, 500, 1000, 2000, 5000, 10000]
        nums_example= [10000]
        mat_contents = scipy.io.loadmat('spam/spam_data.mat')
        training_data = mat_contents['training_data']
        #test_data = mat_contents['test_data']
        training_data_labels = mat_contents['training_labels']
        training_data = np.concatenate((training_data,training_data_labels.T),axis=1)
        np.random.shuffle(training_data)

# Final choice of C-value
        C_list = [97]


        num_folds = 5
        kf = KFold(training_data.shape[0],n_folds=num_folds)
        t_error_tot = 0
        v_error_tot = 0
        best_C_val = 1.0
        lowest_error = 1.0
```

```
best_svc = svm.SVC()
for C_val in C_list:
        for train_index, validation_index in kf:
                #print "train_index:",train_index
                #print "validation_index,",validation_index
                training_set = training_data[train_index]
                validation_set = training_data[validation_index]
                #print "training_set.shape:",training_set.shape
                #print "validation_set.shape:",validation_set.shape
                error = errors(training_set,validation_set,nums_example,C_val)
                t_error = error[0]
                v_error = error[1]
                svc = error[2]
                t_error_tot += t_error[len(t_error)-1]
                v_error_tot += v_error[len(v_error)-1]
        t_error = t_error_tot/float(num_folds)
        v_error = v_error_tot/float(num_folds)
        if v_error < lowest_error:
                best_C_val = C_val
                lowest_error = v_error
                best_svc = svc
        #print "current C_val:",C_val
        #print "t_error:",t_error
```

```
                #print "v_error:",v_error
                t_error_tot = 0
                v_error_tot = 0
        print "best_C_val:",best_C_val
        print "lowest_error:",lowest_error
        #plt.plot(nums_example,t_error,'r-',nums_example,v_error,'b-',label="Number of Training Examples vs. Error Rate for Training and Validation Sets")
        #plt.show()
        return svc


def prob5_svc(training_set, C_val=1.0):
        svc_t = svm.SVC(kernel='linear', C=C_val)
        t_set = training_set[:,:training_set.shape[1]-1]
        t_label = training_set[:,training_set.shape[1]-1]
        svc_t = svc_t.fit(t_set, t_label)
        return svc_t


def prob5():
        print "Now in prob5"
        mat_contents_digits_train = scipy.io.loadmat('mnist/train.mat')
        mat_contents_digits_test = scipy.io.loadmat('mnist/test.mat')
        mat_contents_spam = scipy.io.loadmat('spam/spam_data.mat')
        #mat_contents_cifar_test = scipy.io.loadmat('cifar/test.mat')
        #mat_contents_cifar_train = scipy.io.loadmat('cifar/train.mat')
```

```
#print "mat_contents_digits",mat_contents_digits_test

#print "mat_contents_spam",mat_contents_spam

#print "mat_contents_cifar",mat_contents_cifar_test


digits_train_set = mat_contents_digits_train['trainX']

digits_test_set = mat_contents_digits_test['testX']

spam_test_set = mat_contents_spam['test_data']

#cifar_test_set = mat_contents_cifar_test['testX']

#cifar_train_set = mat_contents_cifar_train['trainX']

spam_train_data = mat_contents_spam['training_data']

spam_training_data_labels = mat_contents_spam['training_labels']

spam_train_set = np.concatenate((spam_train_data,spam_training_data_labels.T),axis=1)


np.random.shuffle(digits_train_set)

np.random.shuffle(spam_train_set)


digits_C_val = 11

digits_svc = prob5_svc(digits_train_set[:20000,:],digits_C_val)

digits_result = digits_svc.predict(digits_test_set)

digits_table = {"Category" : digits_result,

                                        "Id" : np.arange(0,len(digits_result))

                                        }
```

```
    digits_output = pd.DataFrame(data=digits_table)
    digits_output.to_csv("kaggle_ycls_hw1_digits.csv", index=False)


    spam_C_val = 97
    spam_svc = prob5_svc(spam_train_set[:,:],spam_C_val)
    spam_result = [int(x) for x in spam_svc.predict(spam_test_set)]
    spam_table = {"Category" : spam_result,
                                        "Id" : np.arange(0,len(spam_result))
                                        }
    spam_output = pd.DataFrame(data=spam_table)
    spam_output.to_csv("kaggle_ycls_hw1_spam.csv", index=False)


main()
```

Tried C values for problem 3:

C_val: 0.001

v_error: [0.09099999999999997]

C_val: 0.01

v_error: [0.091300000000000048]

C_val: 0.1

v_error: [0.09650000000000003]

C_val: 1.0

v_error: [0.093999999999999972]

C_val: 10.0

```
# v_error: [0.086300000000000043]

# C_val: 100.0

# v_error: [0.092799999999999994]

# C_val: 1000.0

# v_error: [0.088999999999999968]


# C_val: 0.01

# v_error: [0.090099999999999958]

# C_val: 0.03

# v_error: [0.089400000000000035]

# C_val: 0.05

# v_error: [0.089899999999999998]

# C_val: 0.07

# v_error: [0.093400000000000039]

# C_val: 0.09

# v_error: [0.092400000000000038]

# C_val: 0.1

# v_error: [0.087400000000000033]

# C_val: 0.11

# v_error: [0.090099999999999958]

# C_val: 0.13

# v_error: [0.091400000000000037]

# C_val: 0.15

# v_error: [0.096099999999999963]

# C_val: 0.17
```

v_error: [0.092300000000000049]

C_val: 0.19

v_error: [0.086699999999999999]

C_val: 1

v_error: [0.089400000000000035]

C_val: 2

v_error: [0.089700000000000002]

C_val: 3

v_error: [0.091600000000000015]

C_val: 4

v_error: [0.089300000000000046]

C_val: 5

v_error: [0.092199999999999949]

C_val: 6

v_error: [0.087400000000000033]

C_val: 7

v_error: [0.093099999999999961]

C_val: 8

v_error: [0.092700000000000005]

C_val: 9

v_error: [0.088099999999999956]

C_val: 10

v_error: [0.086899999999999977]

```
# C_val: 1
# v_error: [0.088500000000000023]
# C_val: 2
# v_error: [0.093999999999999972]
# C_val: 3
# v_error: [0.094400000000000039]
# C_val: 4
# v_error: [0.092999999999999972]
# C_val: 5
# v_error: [0.08550000000000002]
# C_val: 6
# v_error: [0.088300000000000045]
# C_val: 7
# v_error: [0.088999999999999968]
# C_val: 8
# v_error: [0.09650000000000003]
# C_val: 9
# v_error: [0.091700000000000004]
# C_val: 10
# v_error: [0.092400000000000038]

# C_val: 1
# v_error: [0.096099999999999963]
# C_val: 2
# v_error: [0.087799999999999989]
```

```
# C_val: 3
# v_error: [0.0906000000000000014]
# C_val: 4
# v_error: [0.089700000000000002]
# C_val: 5
# v_error: [0.089999999999999969]
# C_val: 6
# v_error: [0.089999999999999969]
# C_val: 7
# v_error: [0.091199999999999948]
# C_val: 8
# v_error: [0.09870000000000001]
# C_val: 9
# v_error: [0.091899999999999982]
# C_val: 10
# v_error: [0.090500000000000025]
# C_val: 11
# v_error: [0.082999999999999963]
# C_val: 12
# v_error: [0.091600000000000015]
# C_val: 13
# v_error: [0.098899999999999988]
# C_val: 14
# v_error: [0.08660000000000001]
# C_val: 15
```

```
# v_error: [0.090600000000000014]

# C_val: 16

# v_error: [0.092300000000000049]

# C_val: 17

# v_error: [0.090700000000000003]

# C_val: 18

# v_error: [0.093799999999999994]

# C_val: 19

# v_error: [0.09099999999999997]

# C_val: 20

# v_error: [0.091199999999999948]

# C_val: 21

# v_error: [0.090199999999999947]

# C_val: 22

# v_error: [0.096799999999999997]

# C_val: 23

# v_error: [0.09209999999999996]

# C_val: 24

# v_error: [0.093999999999999972]

# C_val: 25

# v_error: [0.092199999999999949]

# C_val: 26

# v_error: [0.088600000000000012]

# C_val: 27

# v_error: [0.088400000000000034]
```

```
# C_val: 28
# v_error: [0.088600000000000012]
# C_val: 29
# v_error: [0.091099999999999959]
# C_val: 30
# v_error: [0.088999999999999968]
# C_val: 31
# v_error: [0.089199999999999946]
# C_val: 32
# v_error: [0.090899999999999981]
# C_val: 33
# v_error: [0.094899999999999984]
# C_val: 34
# v_error: [0.087500000000000022]
# C_val: 35
# v_error: [0.094400000000000039]
# C_val: 36
# v_error: [0.090700000000000003]
# C_val: 37
# v_error: [0.082600000000000007]
# C_val: 38
# v_error: [0.090300000000000047]
# C_val: 39
# v_error: [0.090600000000000014]
# C_val: 40
```

```
# v_error: [0.0988999999999999988]

# C_val: 41

# v_error: [0.096700000000000008]

# C_val: 42

# v_error: [0.092500000000000027]

# C_val: 43

# v_error: [0.08989999999999998]

# C_val: 44

# v_error: [0.092400000000000038]

# C_val: 45

# v_error: [0.088500000000000023]

# C_val: 46

# v_error: [0.09099999999999997]

# C_val: 47

# v_error: [0.092899999999999983]

# C_val: 48

# v_error: [0.089500000000000024]

# C_val: 49

# v_error: [0.087999999999999967]

# C_val: 50

# v_error: [0.092199999999999949]

# C_val: 51

# v_error: [0.09330000000000005]

# C_val: 52

# v_error: [0.0998]
```

```
# C_val: 53
# v_error: [0.096099999999999963]
# C_val: 54
# v_error: [0.096400000000000041]
# C_val: 55
# v_error: [0.095999999999999974]
# C_val: 56
# v_error: [0.087500000000000022]
# C_val: 57
# v_error: [0.095899999999999985]
# C_val: 58
# v_error: [0.088300000000000045]
# C_val: 59
# v_error: [0.088400000000000034]
# C_val: 60
# v_error: [0.091300000000000048]
# C_val: 61
# v_error: [0.097700000000000009]
# C_val: 62
# v_error: [0.091899999999999982]
# C_val: 63
# v_error: [0.090600000000000014]
# C_val: 64
# v_error: [0.090099999999999958]
# C_val: 65
```

```
# v_error: [0.088700000000000001]

# C_val: 66

# v_error: [0.089300000000000046]

# C_val: 67

# v_error: [0.085699999999999998]

# C_val: 68

# v_error: [0.096400000000000041]

# C_val: 69

# v_error: [0.092400000000000038]

# C_val: 70

# v_error: [0.090199999999999947]

# C_val: 71

# v_error: [0.092799999999999994]

# C_val: 72

# v_error: [0.090600000000000014]

# C_val: 73

# v_error: [0.092400000000000038]

# C_val: 74

# v_error: [0.091199999999999948]

# C_val: 75

# v_error: [0.089700000000000002]

# C_val: 76

# v_error: [0.096600000000000019]

# C_val: 77

# v_error: [0.093700000000000006]
```

```
# C_val: 78
# v_error: [0.089500000000000024]
# C_val: 79
# v_error: [0.089500000000000024]
# C_val: 80
# v_error: [0.092700000000000005]
# C_val: 81
# v_error: [0.098300000000000054]
# C_val: 82
# v_error: [0.090400000000000036]
# C_val: 83
# v_error: [0.090600000000000014]
# C_val: 84
# v_error: [0.092300000000000049]
# C_val: 85
# v_error: [0.09330000000000005]
# C_val: 86
# v_error: [0.091400000000000037]
# C_val: 87
# v_error: [0.085999999999999965]
# C_val: 88
# v_error: [0.093899999999999983]
# C_val: 89
# v_error: [0.088300000000000045]
# C_val: 90
```

v_error: [0.09330000000000005]

C_val: 91

v_error: [0.086999999999999966]

C_val: 92

v_error: [0.093500000000000028]

C_val: 93

v_error: [0.094400000000000039]

C_val: 94

v_error: [0.087899999999999978]

C_val: 95

v_error: [0.094700000000000006]

C_val: 96

v_error: [0.089099999999999957]

C_val: 97

v_error: [0.090799999999999992]

C_val: 98

v_error: [0.09319999999999995]

C_val: 99

v_error: [0.093999999999999972]

Tried C values for problem 4:

#C_list = [0.001,0.01,0.1,1.0,10.0,100.0,1000.0] #Best Val: 10, 100

#C_list = [x*0.1 for x in range(1,11)]

best_C_val: 1.0

lowest_error: 0.195285696932

best_C_val: 0.7

```
# lowest_error: 0.193350152777
# best_C_val: 0.9
# lowest_error: 0.192961997402
# best_C_val: 1.0
# lowest_error: 0.193150748932
# best_C_val: 1.0
# lowest_error: 0.192382100375
# best_C_val: 0.9
# lowest_error: 0.193156542296
# best_C_val: 0.8
# lowest_error: 0.191803324643
# best_C_val: 1.0
# lowest_error: 0.191806688532
# best_C_val: 0.9
# lowest_error: 0.193158971771
# best_C_val: 1.0
# lowest_error: 0.193350526542


            #C_list = range(1,100)
# best_C_val: 77
# lowest_error: 0.17807660322
# best_C_val: 62
# lowest_error: 0.181356955307
# best_C_val: 82
# lowest_error: 0.17884749437
# best_C_val: 97
# lowest_error: 0.179235649744
# best_C_val: 85
# lowest_error: 0.178073986862
# best_C_val: 93
```

```
# lowest_error: 0.181162410413
# best_C_val: 97
# lowest_error: 0.181936478569
# best_C_val: 97
# lowest_error: 0.179231538325
# best_C_val: 83
# lowest_error: 0.180777058279
# best_C_val: 79
# lowest_error: 0.17904316056

            #C_list = range(62,150)
# best_C_val: 84
# lowest_error: 0.179815733655
# best_C_val: 118
# lowest_error: 0.179232285856
# best_C_val: 92
# lowest_error: 0.180389837319
# best_C_val: 74
# lowest_error: 0.180005606481
# best_C_val: 122
# lowest_error: 0.175562470216
# best_C_val: 95
# lowest_error: 0.181166148067
# best_C_val: 95
# lowest_error: 0.178659303488
# best_C_val: 127
# lowest_error: 0.177107616405
# best_C_val: 75
# lowest_error: 0.177105186929
# best_C_val: 88
```

```
# lowest_error: 0.177302535064


            #C_list = range(150,250)
# best_C_val: 153
# lowest_error: 0.182716900737
# best_C_val: 165
# lowest_error: 0.181170259487
# best_C_val: 226
# lowest_error: 0.183484054233
# best_C_val: 151
# lowest_error: 0.180391145498
# best_C_val: 175
# lowest_error: 0.183291378166
# best_C_val: 206
# lowest_error: 0.182325568357
# best_C_val: 205
# lowest_error: 0.180007849074
# best_C_val: 168
# lowest_error: 0.185230286211
# best_C_val: 168
# lowest_error: 0.182138872537
# best_C_val: 181
# lowest_error: 0.180004672068
```