COMPSCI 289A Homework 1 **Mitchell Negus**
1/30/2017

*Note: Code was worked on independently

Problem 1

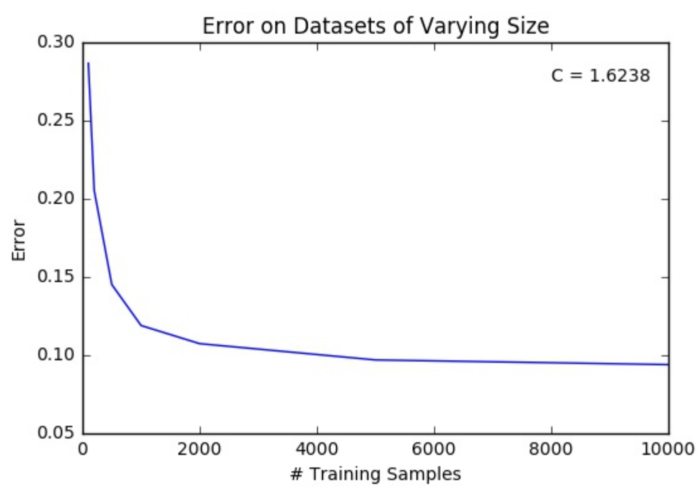
Data was partitioned as specified (10,000 validation images for MNIST, 20% as validation samples for spam, and 5,000 validation images for CIFAR-10). See code, provided in the appendix, for evidence. Partitioning was accomplished by calling the partition function defined in HW01_utils.py module.

Problem 2

The linear SVM was trained on all three datasets. The score (accuracy) of the method was calculated for a range of samples. For each data set–MNIST, spam, and CIFAR-10–the error (error = 1-accuracy) is plotted as a function of N samples used for training.

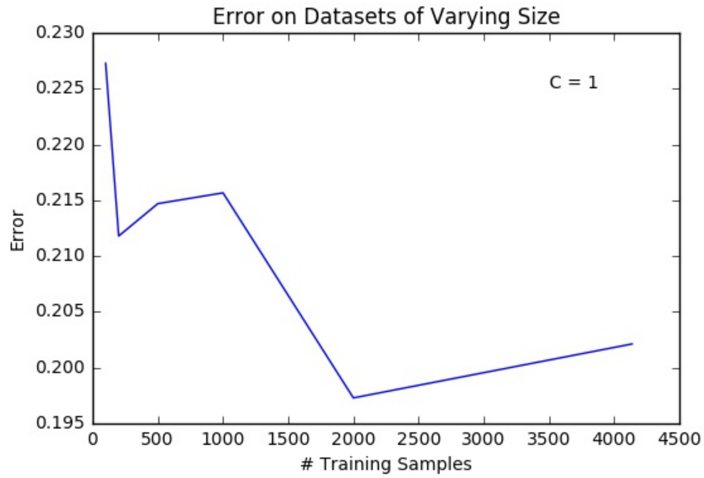
MNIST

Accuracy of MNIST training for 100, 200, 500, 1,000, 2,000, 5,000, and 10,000 training samples.



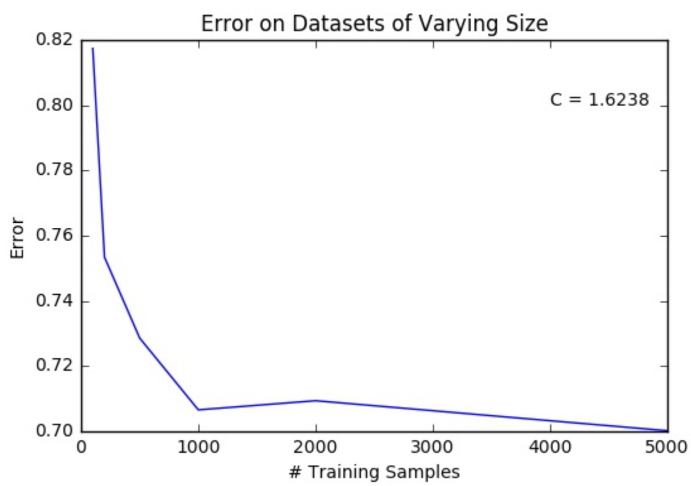
Spam

Accuracy of spam/ham training for 100, 200, 500, 1,000, 2,000, and all (4,132) training samples.



CIFAR-10

Accuracy of CIFAR-10 training for 100, 200, 500, 1,000, 2,000, and 5,000 training samples.



Problem 3

For the MNIST data set, the best value of C was found to be $7.84759970351 \times 10^{-7}$, giving an accuracy of 92.98%. All accuracies for a range of C values (all trained on 10,000 samples) are given below.

10,000 samples

C	**Accuracy**
1.0000E-08	0.8935
4.2813E-08	0.9147
1.8330E-07	0.9255
7.8476E-07	**0.9298**
3.3598E-06	0.9229
1.4384E-05	0.9118
6.1585E-05	0.9071

2.6367E-04	0.906
0.0011288	0.906
0.0048329	0.906
0.0206914	0.906
0.0885867	0.906
0.3792690	0.906
1.6237767	0.906
6.9519280	0.906
29.763514	0.906
127.42750	0.906
545.55948	0.906
2335.7215	0.906
10000.0	0.906

Full results for all sample counts are provided in the appendix.

Problem 4

For the spam/ham data sets, best value of C was found to be 100, giving an accuracy of 80.75%. This value was found using a K-Fold Cross-Validation scheme where $k = 5$. Below are all accuracies for a range of C values when the SVM was trained on 2,000 samples.

2,000 samples

C	**Accuracy**
1.0000E-08	0.710058
3.3598E-08	0.710058
1.1288E-07	0.710058
3.7927E-07	0.710058
1.2743E-06	0.710058
4.2813E-06	0.710058
1.4384E-05	0.710058
4.8329E-05	0.710058
0.00016238	0.717215
0.00054556	0.734429
0.00183298	0.750097
0.00615848	0.768279
0.02069138	0.779304
0.06951928	0.793617
0.23357215	0.8
0.78475997	0.802515
2.63665090	0.804836
8.85866790	0.805029
29.7635144	0.807350
100.0	**0.807544**

Problem 5

My Kaggle Leaderboard name: **mitch**
My Kaggle username: **mnegus**

Kaggle Scores:
MNIST: 0.93360 Spam: 0.84085

Appendix

Below are all accuracies tabulated for tested range of hyperparameter C for N training samples.

Table 1: **MNIST**

***N*\C**	1E-08	4E-08	1.8E-07	7.8E-07	3.36E-06	1.44E-05	6.16E-05	0.000264	0.001129	0.004833
100	0.1119	0.2301	0.6438	0.7169	0.7133	0.7133	0.7133	0.7133	0.7133	0.7133
200	0.0963	0.425	0.7747	0.8005	0.7947	0.7947	0.7947	0.7947	0.7947	0.7947
500	0.3171	0.7527	0.865	0.8635	0.8552	0.8548	0.8548	0.8548	0.8548	0.8548
1000	0.5782	0.85	0.8867	0.8909	0.8815	0.881	0.881	0.881	0.881	0.881
2000	0.7981	0.8824	0.9043	0.9086	0.895	0.8926	0.8926	0.8926	0.8926	0.8926
5000	0.8721	0.9038	0.9183	0.9202	0.9135	0.9041	0.903	0.903	0.903	0.903
10000	0.8935	0.9147	0.9255	0.9298	0.9229	0.9118	0.9071	0.906	0.906	0.906

***N*\C**	0.020691	0.088587	0.379269	1.623777	6.951928	29.76351	127.4275	545.5595	2335.721	10000
100	0.7133	0.7133	0.7133	0.7133	0.7133	0.7133	0.7133	0.7133	0.7133	0.7133
200	0.7947	0.7947	0.7947	0.7947	0.7947	0.7947	0.7947	0.7947	0.7947	0.7947
500	0.8548	0.8548	0.8548	0.8548	0.8548	0.8548	0.8548	0.8548	0.8548	0.8548
1000	0.881	0.881	0.881	0.881	0.881	0.881	0.881	0.881	0.881	0.881
2000	0.8926	0.8926	0.8926	0.8926	0.8926	0.8926	0.8926	0.8926	0.8926	0.8926
5000	0.903	0.903	0.903	0.903	0.903	0.903	0.903	0.903	0.903	0.903
10000	0.906	0.906	0.906	0.906	0.906	0.906	0.906	0.906	0.906	0.906

Table 2: **Spam**

***N*\C**	1E-08	3E-08	1.1E-07	3.8E-07	1.27E-06	4.28E-06	1.44E-05	4.83E-05	0.000162	0.000546
100	0.710058	0.710058	0.710058	0.710058	0.710058	0.710058	0.710058	0.709865	0.710445	0.710638
200	0.710058	0.710058	0.710058	0.710058	0.710058	0.710058	0.710058	0.710058	0.710251	0.711025
500	0.710058	0.710058	0.710058	0.710058	0.710058	0.710058	0.710058	0.710058	0.710058	0.713926
1000	0.710058	0.710058	0.710058	0.710058	0.710058	0.710058	0.710058	0.710058	0.711412	0.725725
2000	0.710058	0.710058	0.710058	0.710058	0.710058	0.710058	0.710058	0.710058	0.717215	0.734429
4138	0.710058	0.710058	0.710058	0.710058	0.710058	0.710058	0.710058	0.712186	0.725338	0.743327

***N*\C**	0.001833	0.006158	0.020691	0.069519	0.233572	0.78476	2.636651	8.858668	29.76351	100
100	0.711605	0.716828	0.728433	0.74236	0.754932	0.770406	0.777756	0.786074	0.775629	0.781044
200	0.716441	0.730174	0.745261	0.758801	0.767892	0.782592	0.789555	0.798066	0.792456	0.786847
500	0.731141	0.745261	0.763636	0.77176	0.783172	0.791876	0.793424	0.793037	0.792456	0.792456
1000	0.741199	0.761896	0.7706	0.784139	0.790135	0.792843	0.797099	0.798646	0.79942	0.798453
2000	0.750097	0.768279	0.779304	0.793617	0.8	0.802515	0.804836	0.805029	0.80735	0.807544
4138	0.763056	0.774855	0.789362	0.794971	0.79942	0.801354	0.802128	0.802901	0.802708	0.802515

Table 3: **CIFAR-10**

N\C	1E-08	4E-08	1.8E-07	7.8E-07	3.36E-06	1.44E-05	6.16E-05	0.000264	0.001129	0.004833
100	0.111	0.1658	0.1834	0.1838	0.1826	0.1826	0.1826	0.1826	0.1826	0.1826
200	0.1026	0.2272	0.2622	0.2466	0.2466	0.2466	0.2466	0.2466	0.2466	0.2466
500	0.2202	0.295	0.3008	0.2842	0.2708	0.2714	0.2714	0.2714	0.2714	0.2714
1000	0.2632	0.3164	0.3306	0.3148	0.2934	0.2934	0.2934	0.2934	0.2934	0.2934
2000	0.3	0.3476	0.345	0.321	0.3016	0.291	0.2906	0.2906	0.2906	0.2906
5000	0.353	0.3736	0.3752	0.3518	0.3216	0.3066	0.2992	0.2998	0.2998	0.2998

N\C	0.020691	0.088587	0.379269	1.623777	6.951928	29.76351	127.4275	545.5595	2335.721	10000
100	0.1826	0.1826	0.1826	0.1826	0.1826	0.1826	0.1826	0.1826	0.1826	0.1826
200	0.2466	0.2466	0.2466	0.2466	0.2466	0.2466	0.2466	0.2466	0.2466	0.2466
500	0.2714	0.2714	0.2714	0.2714	0.2714	0.2714	0.2714	0.2714	0.2714	0.2714
1000	0.2934	0.2934	0.2934	0.2934	0.2934	0.2934	0.2934	0.2934	0.2934	0.2934
2000	0.2906	0.2906	0.2906	0.2906	0.2906	0.2906	0.2906	0.2906	0.2906	0.2906
5000	0.2998	0.2998	0.2998	0.2998	0.2998	0.2998	0.2998	0.2998	0.2998	0.2998

Included on the following pages is the code used for this project: namely the 3 Jupyter notebooks and 2 python modules:

CS289A_HW01_MNIST

January 30, 2017

```
In [16]: %load_ext autoreload

The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload


In [17]: %autoreload 2

In [18]: import numpy as np
         import HW01_utils as utils
         import trainfunctions as tf

In [46]: _LOCAL_PATH = r"C:\Users\Mitch\Documents\Cal\2 - 2017 Spring\COMPSCI 289A
         _DATA_PATH = "Data\hw01_data"

         _DATA_DIR = _LOCAL_PATH + "\\" + _DATA_PATH
         trainpath = r"mnist\train.mat"

In [20]: valsetsize = 10000
         samples = np.array([100, 200, 500, 1000, 2000, 5000, 10000])
         hyperparams = np.logspace(-8,4,num=20)

In [21]: # Load MNIST training data
         mnist = utils.loaddata(trainpath,_DATA_DIR,'trainX')

         # Shuffle data before splitting
         np.random.shuffle(mnist)

In [22]: trainset,valset = utils.partition(valsetsize,mnist)
         trainsetarrays,trainsetlabels = utils.separatelabels(trainset)
         valsetarrays,valsetlabels = utils.separatelabels(valset)

In [23]: Accs = np.empty((len(samples),len(hyperparams)))
         i = 0 # sample index counter
         for nsamples in samples:
             print(nsamples,'samples')
             j = 0 # hyperparameter index counter
             for hp in hyperparams:
                 acc = tf.TrainAndScoreNsamples(trainsetarrays[:nsamples],trainset]
```

```
            print('\tC =',hp,'\tAccuracy:',acc)
            Accs[i,j] = acc
            j+=1
        i+=1
    print(Accs)
```

```
100 samples
	C = 1e-08 	Accuracy: 0.1119
	C = 4.28133239872e-08 	Accuracy: 0.2301
	C = 1.83298071083e-07 	Accuracy: 0.6438
	C = 7.84759970351e-07 	Accuracy: 0.7169
	C = 3.35981828628e-06 	Accuracy: 0.7133
	C = 1.43844988829e-05 	Accuracy: 0.7133
	C = 6.15848211066e-05 	Accuracy: 0.7133
	C = 0.000263665089873 	Accuracy: 0.7133
	C = 0.00112883789168 	Accuracy: 0.7133
	C = 0.00483293023857 	Accuracy: 0.7133
	C = 0.0206913808111 	Accuracy: 0.7133
	C = 0.088586679041 	Accuracy: 0.7133
	C = 0.379269019073 	Accuracy: 0.7133
	C = 1.62377673919 	Accuracy: 0.7133
	C = 6.95192796178 	Accuracy: 0.7133
	C = 29.7635144163 	Accuracy: 0.7133
	C = 127.42749857 	Accuracy: 0.7133
	C = 545.559478117 	Accuracy: 0.7133
	C = 2335.72146909 	Accuracy: 0.7133
	C = 10000.0 	Accuracy: 0.7133
200 samples
	C = 1e-08 	Accuracy: 0.0963
	C = 4.28133239872e-08 	Accuracy: 0.425
	C = 1.83298071083e-07 	Accuracy: 0.7747
	C = 7.84759970351e-07 	Accuracy: 0.8005
	C = 3.35981828628e-06 	Accuracy: 0.7947
	C = 1.43844988829e-05 	Accuracy: 0.7947
	C = 6.15848211066e-05 	Accuracy: 0.7947
	C = 0.000263665089873 	Accuracy: 0.7947
	C = 0.00112883789168 	Accuracy: 0.7947
	C = 0.00483293023857 	Accuracy: 0.7947
	C = 0.0206913808111 	Accuracy: 0.7947
	C = 0.088586679041 	Accuracy: 0.7947
	C = 0.379269019073 	Accuracy: 0.7947
	C = 1.62377673919 	Accuracy: 0.7947
	C = 6.95192796178 	Accuracy: 0.7947
	C = 29.7635144163 	Accuracy: 0.7947
	C = 127.42749857 	Accuracy: 0.7947
	C = 545.559478117 	Accuracy: 0.7947
	C = 2335.72146909 	Accuracy: 0.7947
	C = 10000.0 	Accuracy: 0.7947
```

```
500 samples
	C = 1e-08	Accuracy: 0.3171
	C = 4.28133239872e-08	Accuracy: 0.7527
	C = 1.83298071083e-07	Accuracy: 0.865
	C = 7.84759970351e-07	Accuracy: 0.8635
	C = 3.35981828628e-06	Accuracy: 0.8552
	C = 1.43844988829e-05	Accuracy: 0.8548
	C = 6.15848211066e-05	Accuracy: 0.8548
	C = 0.000263665089873	Accuracy: 0.8548
	C = 0.00112883789168	Accuracy: 0.8548
	C = 0.00483293023857	Accuracy: 0.8548
	C = 0.0206913808111	Accuracy: 0.8548
	C = 0.088586679041	Accuracy: 0.8548
	C = 0.379269019073	Accuracy: 0.8548
	C = 1.62377673919	Accuracy: 0.8548
	C = 6.95192796178	Accuracy: 0.8548
	C = 29.7635144163	Accuracy: 0.8548
	C = 127.42749857	Accuracy: 0.8548
	C = 545.559478117	Accuracy: 0.8548
	C = 2335.72146909	Accuracy: 0.8548
	C = 10000.0	Accuracy: 0.8548
1000 samples
	C = 1e-08	Accuracy: 0.5782
	C = 4.28133239872e-08	Accuracy: 0.85
	C = 1.83298071083e-07	Accuracy: 0.8867
	C = 7.84759970351e-07	Accuracy: 0.8909
	C = 3.35981828628e-06	Accuracy: 0.8815
	C = 1.43844988829e-05	Accuracy: 0.881
	C = 6.15848211066e-05	Accuracy: 0.881
	C = 0.000263665089873	Accuracy: 0.881
	C = 0.00112883789168	Accuracy: 0.881
	C = 0.00483293023857	Accuracy: 0.881
	C = 0.0206913808111	Accuracy: 0.881
	C = 0.088586679041	Accuracy: 0.881
	C = 0.379269019073	Accuracy: 0.881
	C = 1.62377673919	Accuracy: 0.881
	C = 6.95192796178	Accuracy: 0.881
	C = 29.7635144163	Accuracy: 0.881
	C = 127.42749857	Accuracy: 0.881
	C = 545.559478117	Accuracy: 0.881
	C = 2335.72146909	Accuracy: 0.881
	C = 10000.0	Accuracy: 0.881
2000 samples
	C = 1e-08	Accuracy: 0.7981
	C = 4.28133239872e-08	Accuracy: 0.8824
	C = 1.83298071083e-07	Accuracy: 0.9043
	C = 7.84759970351e-07	Accuracy: 0.9086
	C = 3.35981828628e-06	Accuracy: 0.895
```

```
        C = 1.43844988829e-05         Accuracy: 0.8926
        C = 6.15848211066e-05         Accuracy: 0.8926
        C = 0.000263665089873         Accuracy: 0.8926
        C = 0.00112883789168         Accuracy: 0.8926
        C = 0.00483293023857         Accuracy: 0.8926
        C = 0.0206913808111         Accuracy: 0.8926
        C = 0.088586679041         Accuracy: 0.8926
        C = 0.379269019073         Accuracy: 0.8926
        C = 1.62377673919         Accuracy: 0.8926
        C = 6.95192796178         Accuracy: 0.8926
        C = 29.7635144163         Accuracy: 0.8926
        C = 127.42749857         Accuracy: 0.8926
        C = 545.559478117         Accuracy: 0.8926
        C = 2335.72146909         Accuracy: 0.8926
        C = 10000.0         Accuracy: 0.8926
5000 samples
        C = 1e-08         Accuracy: 0.8721
        C = 4.28133239872e-08         Accuracy: 0.9038
        C = 1.83298071083e-07         Accuracy: 0.9183
        C = 7.84759970351e-07         Accuracy: 0.9202
        C = 3.35981828628e-06         Accuracy: 0.9135
        C = 1.43844988829e-05         Accuracy: 0.9041
        C = 6.15848211066e-05         Accuracy: 0.903
        C = 0.000263665089873         Accuracy: 0.903
        C = 0.00112883789168         Accuracy: 0.903
        C = 0.00483293023857         Accuracy: 0.903
        C = 0.0206913808111         Accuracy: 0.903
        C = 0.088586679041         Accuracy: 0.903
        C = 0.379269019073         Accuracy: 0.903
        C = 1.62377673919         Accuracy: 0.903
        C = 6.95192796178         Accuracy: 0.903
        C = 29.7635144163         Accuracy: 0.903
        C = 127.42749857         Accuracy: 0.903
        C = 545.559478117         Accuracy: 0.903
        C = 2335.72146909         Accuracy: 0.903
        C = 10000.0         Accuracy: 0.903
10000 samples
        C = 1e-08         Accuracy: 0.8935
        C = 4.28133239872e-08         Accuracy: 0.9147
        C = 1.83298071083e-07         Accuracy: 0.9255
        C = 7.84759970351e-07         Accuracy: 0.9298
        C = 3.35981828628e-06         Accuracy: 0.9229
        C = 1.43844988829e-05         Accuracy: 0.9118
        C = 6.15848211066e-05         Accuracy: 0.9071
        C = 0.000263665089873         Accuracy: 0.906
        C = 0.00112883789168         Accuracy: 0.906
        C = 0.00483293023857         Accuracy: 0.906
        C = 0.0206913808111         Accuracy: 0.906
```

```
        C = 0.088586679041         Accuracy: 0.906
        C = 0.379269019073         Accuracy: 0.906
        C = 1.62377673919         Accuracy: 0.906
        C = 6.95192796178         Accuracy: 0.906
        C = 29.7635144163         Accuracy: 0.906
        C = 127.42749857         Accuracy: 0.906
        C = 545.559478117         Accuracy: 0.906
        C = 2335.72146909         Accuracy: 0.906
        C = 10000.0          Accuracy: 0.906
[[ 0.1119  0.2301  0.6438  0.7169  0.7133  0.7133  0.7133  0.7133  0.7133
   0.7133  0.7133  0.7133  0.7133  0.7133  0.7133  0.7133  0.7133  0.7133
   0.7133  0.7133]
 [ 0.0963  0.425   0.7747  0.8005  0.7947  0.7947  0.7947  0.7947  0.7947
   0.7947  0.7947  0.7947  0.7947  0.7947  0.7947  0.7947  0.7947  0.7947
   0.7947  0.7947]
 [ 0.3171  0.7527  0.865   0.8635  0.8552  0.8548  0.8548  0.8548  0.8548
   0.8548  0.8548  0.8548  0.8548  0.8548  0.8548  0.8548  0.8548  0.8548
   0.8548  0.8548]
 [ 0.5782  0.85    0.8867  0.8909  0.8815  0.881   0.881   0.881   0.881
   0.881   0.881   0.881   0.881   0.881   0.881   0.881   0.881   0.881
   0.881   0.881 ]
 [ 0.7981  0.8824  0.9043  0.9086  0.895   0.8926  0.8926  0.8926  0.8926
   0.8926  0.8926  0.8926  0.8926  0.8926  0.8926  0.8926  0.8926  0.8926
   0.8926  0.8926]
 [ 0.8721  0.9038  0.9183  0.9202  0.9135  0.9041  0.903   0.903   0.903
   0.903   0.903   0.903   0.903   0.903   0.903   0.903   0.903   0.903
   0.903   0.903 ]
 [ 0.8935  0.9147  0.9255  0.9298  0.9229  0.9118  0.9071  0.906   0.906
   0.906   0.906   0.906   0.906   0.906   0.906   0.906   0.906   0.906
   0.906   0.906 ]]
```

```
In [28]: # Find the index of the maximum value in the accuracies table
         maxindex = np.array([int(len(Accs)*np.argmax(Accs)/(len(Accs.flatten()))),
         print('The index of the maximum accuracy ('+str(Accs[maxindex[0],maxindex[

         besthp = hyperparams[maxindex[1]]
         bestns = samples[maxindex[0]]
         # Determine which sample count-hyperparameter combination this corresponds
         print('This corresponds to a hyperparameter of C = '+ str(besthp) + ' wher
```

```
The index of the maximum accuracy (0.9298) is: [6 3]
This corresponds to a hyperparameter of C = 7.84759970351e-07 when training on 1000
```

```
In [29]: besthp = 7.84759970351e-07
         bestns = 10000

In [30]: # Load test data
         testpath = r"mnist\test.mat"
```

```
mnist_test = utils.loaddata(testpath,_DATA_DIR,'testX')
predictions = tf.TrainAndPredictNsamples(trainsetarrays[:bestns],trainsetl
```

```
In [27]: IDs = np.arange(len(predictions))
         numpycsv = np.c_[IDs,predictions]
         np.savetxt(_LOCAL_PATH+r'\MNIST_testpredictions.csv',numpycsv,fmt='%i',del
```

```
In [31]: from matplotlib import pyplot as plt
```

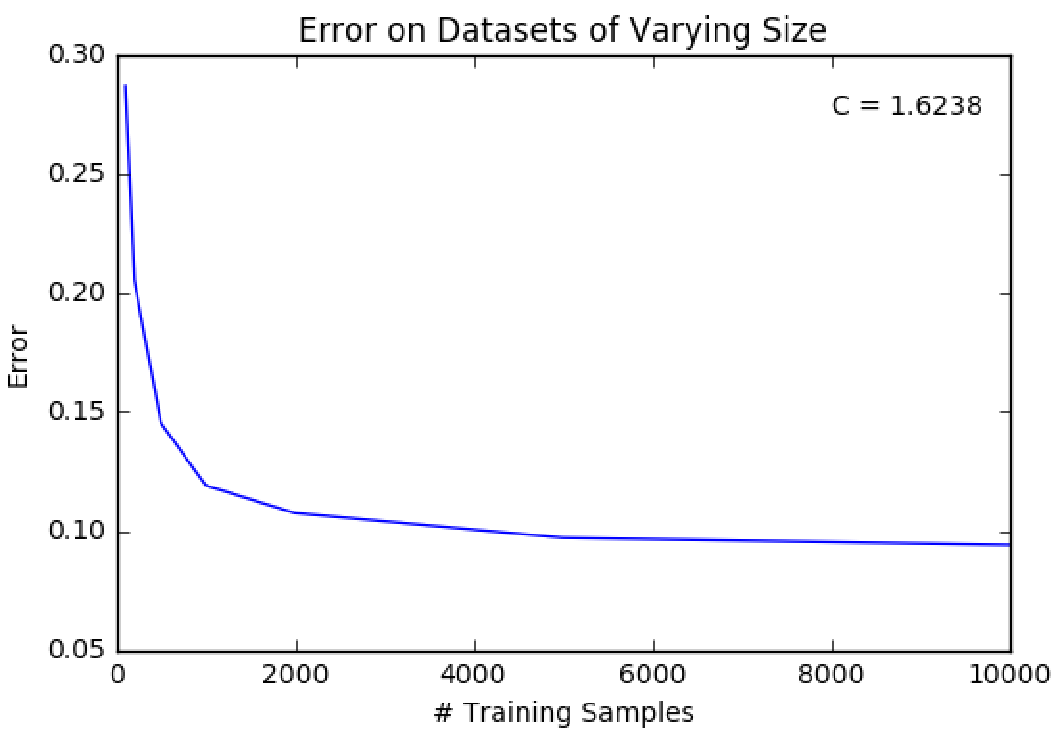
```
In [32]: hpC1 = 13
```

```
In [33]: errors = np.ones_like(Accs[:,hpC1])-Accs[:,hpC1]
```

```
In [42]: fig = plt.figure()
         plt.plot(samples,errors)
         plt.title('Error on Datasets of Varying Size')
         plt.xlabel('# Training Samples')
         plt.ylabel('Error')
         plt.text(8000,0.275,'C = '+str(round(hyperparams[hpC1],4)))
```

```
Out[42]: <matplotlib.text.Text at 0x23b99f887f0>
```

```
In [43]: plt.show()
```




```
In [47]: fig.savefig(_LOCAL_PATH+r'\Figures\MNIST_SampleAcc.jpg')
```

```
In [ ]: # Export data to csv files for report
        np.savetxt(_LOCAL_PATH+r'\MNIST_Accuracies.csv',Accs,fmt='%f',delimiter=','
        np.savetxt(_LOCAL_PATH+r'\MNIST_hyperparams.csv',hyperparams,fmt='%.8f',del
```

CS289A_HW01_spam

January 30, 2017

```
In [1]: %load_ext autoreload

In [2]: %autoreload 2

In [3]: from scipy import io as spio
        import numpy as np
        import HW01_utils as utils
        import trainfunctions as tf

In [10]: def loaddata_spam(shortpath,_DATA_DIR):
         # Load data
             data_dict = spio.loadmat(_DATA_DIR+"\\"+shortpath)
             data = np.array(data_dict['training_data'])
             labels = np.array(data_dict['training_labels'])
             return data,labels

In [5]: def kfoldPartition(k,data):
            # Partition the shuffled data into k sets
            subsetlen = int(len(data)/k)      #NOTE: This will neglect a few data po
            subsets = np.empty((k,subsetlen,len(data[0])))
            for i in range(k):
                subsets[i] = data[i*subsetlen:(i+1)*subsetlen]
            return subsets

In [76]: _LOCAL_PATH = r"C:\Users\Mitch\Documents\Cal\2 - 2017 Spring\COMPSCI 289A
         _DATA_PATH = "Data\hw01_data"

         _DATA_DIR = _LOCAL_PATH + "\\" + _DATA_PATH
         datafilepath = r"spam\spam_data.mat"

In [11]: # Load spam training data (w/ features extracted)
         spamdata,labels = loaddata_spam(datafilepath,_DATA_DIR)

         # Append labels to the corresponding data (to prevent loss of association

         spam = np.c_[spamdata,labels[0]]

         #spam_testdata = np.array(spam_dict['test_data'])

         # Shuffle data before splitting
         np.random.shuffle(spam)
```

```
In [12]: valsetpercent = 20
         valsetsize = int(len(spam)*valsetpercent/100)

In [13]: trainset,valset = utils.partition(valsetsize,spam)
         trainsetarrays,trainsetlabels = utils.separatelabels(trainset)
         valsetarrays,valsetlabels = utils.separatelabels(valset)

In [14]: samples = [100, 200, 500, 1000, 2000, len(trainset)]
         hyperparams = np.logspace(-8,2,num=20)    #100 was chosen as highest C valu

In [15]: def kfoldCrossVal(k,data,nsamples,hyperparam):
         # Implementation of k-fold Cross-Validation
             spamsets = kfoldPartition(k,data)
             scores = np.zeros(k)
             for i in range(k):
                 trainset = spamsets[np.arange(len(spamsets))!=i]
                 trainset = np.concatenate(trainset[:])
                 valset = spamsets[i]
                 scores[i] = tf.TrainAndScoreNsamples(trainset[:nsamples,:-1],trair
             average = np.sum(scores)/len(scores)
             return average

In [84]: AccsNoK = np.empty((len(samples)))
         i = 0 # sample index counter
         for nsamples in samples:
             print(nsamples,'samples')
             hp=1
             acc = tf.TrainAndScoreNsamples(trainsetarrays[:nsamples],trainsetlabel
             print('\tC =',hp,'\tAccuracy:',acc)
             AccsNoK[i] = acc
             i+=1
         print(AccsNoK)

100 samples
        C = 1          Accuracy: 0.772727272727
200 samples
        C = 1          Accuracy: 0.788201160542
500 samples
        C = 1          Accuracy: 0.785299806576
1000 samples
        C = 1          Accuracy: 0.784332688588
2000 samples
        C = 1          Accuracy: 0.802707930368
4138 samples
        C = 1          Accuracy: 0.797872340426
[ 0.77272727  0.78820116  0.78529981  0.78433269  0.80270793  0.79787234]


In [93]: Accs = np.empty((len(samples),len(hyperparams)))
         i = 0 # sample index counter
```

```
for nsamples in samples:
    print(nsamples,'samples')
    j = 0 # hyperparameter index counter
    for hp in hyperparams:
        acc = kfoldCrossVal(5,spam,nsamples,hp)

        print('\tC =',hp,'\tAccuracy:',acc)
        Accs[i,j] = acc
        j+=1
    i+=1
print(Accs)
```

```
100 samples
	C = 1e-08	Accuracy: 0.710058027079
	C = 3.35981828628e-08	Accuracy: 0.710058027079
	C = 1.12883789168e-07	Accuracy: 0.710058027079
	C = 3.79269019073e-07	Accuracy: 0.710058027079
	C = 1.2742749857e-06	Accuracy: 0.710058027079
	C = 4.28133239872e-06	Accuracy: 0.710058027079
	C = 1.43844988829e-05	Accuracy: 0.710058027079
	C = 4.83293023857e-05	Accuracy: 0.709864603482
	C = 0.000162377673919	Accuracy: 0.710444874275
	C = 0.000545559478117	Accuracy: 0.710638297872
	C = 0.00183298071083	Accuracy: 0.711605415861
	C = 0.00615848211066	Accuracy: 0.716827852998
	C = 0.0206913808111	Accuracy: 0.728433268859
	C = 0.0695192796178	Accuracy: 0.742359767892
	C = 0.233572146909	Accuracy: 0.754932301741
	C = 0.784759970351	Accuracy: 0.770406189555
	C = 2.63665089873	Accuracy: 0.777756286267
	C = 8.8586679041	Accuracy: 0.786073500967
	C = 29.7635144163	Accuracy: 0.775628626692
	C = 100.0	Accuracy: 0.781044487427
200 samples
	C = 1e-08	Accuracy: 0.710058027079
	C = 3.35981828628e-08	Accuracy: 0.710058027079
	C = 1.12883789168e-07	Accuracy: 0.710058027079
	C = 3.79269019073e-07	Accuracy: 0.710058027079
	C = 1.2742749857e-06	Accuracy: 0.710058027079
	C = 4.28133239872e-06	Accuracy: 0.710058027079
	C = 1.43844988829e-05	Accuracy: 0.710058027079
	C = 4.83293023857e-05	Accuracy: 0.710058027079
	C = 0.000162377673919	Accuracy: 0.710251450677
	C = 0.000545559478117	Accuracy: 0.711025145068
	C = 0.00183298071083	Accuracy: 0.716441005803
	C = 0.00615848211066	Accuracy: 0.730174081238
	C = 0.0206913808111	Accuracy: 0.745261121857
	C = 0.0695192796178	Accuracy: 0.758800773694
```

```
        C = 0.233572146909         Accuracy: 0.767891682785
        C = 0.784759970351         Accuracy: 0.782591876209
        C = 2.63665089873         Accuracy: 0.789555125725
        C = 8.8586679041         Accuracy: 0.798065764023
        C = 29.7635144163         Accuracy: 0.792456479691
        C = 100.0         Accuracy: 0.786847195358
500 samples
        C = 1e-08         Accuracy: 0.710058027079
        C = 3.35981828628e-08         Accuracy: 0.710058027079
        C = 1.12883789168e-07         Accuracy: 0.710058027079
        C = 3.79269019073e-07         Accuracy: 0.710058027079
        C = 1.2742749857e-06         Accuracy: 0.710058027079
        C = 4.28133239872e-06         Accuracy: 0.710058027079
        C = 1.43844988829e-05         Accuracy: 0.710058027079
        C = 4.83293023857e-05         Accuracy: 0.710058027079
        C = 0.000162377673919         Accuracy: 0.710058027079
        C = 0.000545559478117         Accuracy: 0.713926499033
        C = 0.00183298071083         Accuracy: 0.731141199226
        C = 0.00615848211066         Accuracy: 0.745261121857
        C = 0.0206913808111         Accuracy: 0.763636363636
        C = 0.0695192796178         Accuracy: 0.771760154739
        C = 0.233572146909         Accuracy: 0.783172147002
        C = 0.784759970351         Accuracy: 0.791876208897
        C = 2.63665089873         Accuracy: 0.793423597679
        C = 8.8586679041         Accuracy: 0.793036750484
        C = 29.7635144163         Accuracy: 0.792456479691
        C = 100.0         Accuracy: 0.792456479691
1000 samples
        C = 1e-08         Accuracy: 0.710058027079
        C = 3.35981828628e-08         Accuracy: 0.710058027079
        C = 1.12883789168e-07         Accuracy: 0.710058027079
        C = 3.79269019073e-07         Accuracy: 0.710058027079
        C = 1.2742749857e-06         Accuracy: 0.710058027079
        C = 4.28133239872e-06         Accuracy: 0.710058027079
        C = 1.43844988829e-05         Accuracy: 0.710058027079
        C = 4.83293023857e-05         Accuracy: 0.710058027079
        C = 0.000162377673919         Accuracy: 0.711411992263
        C = 0.000545559478117         Accuracy: 0.725725338491
        C = 0.00183298071083         Accuracy: 0.741199226306
        C = 0.00615848211066         Accuracy: 0.761895551257
        C = 0.0206913808111         Accuracy: 0.770599613153
        C = 0.0695192796178         Accuracy: 0.78413926499
        C = 0.233572146909         Accuracy: 0.790135396518
        C = 0.784759970351         Accuracy: 0.792843326886
        C = 2.63665089873         Accuracy: 0.797098646035
        C = 8.8586679041         Accuracy: 0.798646034816
        C = 29.7635144163         Accuracy: 0.799419729207
        C = 100.0         Accuracy: 0.798452611219
```

```
2000 samples
        C = 1e-08         Accuracy: 0.710058027079
        C = 3.35981828628e-08         Accuracy: 0.710058027079
        C = 1.12883789168e-07         Accuracy: 0.710058027079
        C = 3.79269019073e-07         Accuracy: 0.710058027079
        C = 1.2742749857e-06         Accuracy: 0.710058027079
        C = 4.28133239872e-06         Accuracy: 0.710058027079
        C = 1.43844988829e-05         Accuracy: 0.710058027079
        C = 4.83293023857e-05         Accuracy: 0.710058027079
        C = 0.000162377673919         Accuracy: 0.717214700193
        C = 0.000545559478117         Accuracy: 0.734429400387
        C = 0.00183298071083         Accuracy: 0.750096711799
        C = 0.00615848211066         Accuracy: 0.768278529981
        C = 0.0206913808111         Accuracy: 0.779303675048
        C = 0.0695192796178         Accuracy: 0.793617021277
        C = 0.233572146909         Accuracy: 0.8
        C = 0.784759970351         Accuracy: 0.80251450677
        C = 2.63665089873         Accuracy: 0.804835589942
        C = 8.8586679041         Accuracy: 0.80502901354
        C = 29.7635144163         Accuracy: 0.807350096712
        C = 100.0         Accuracy: 0.807543520309
4138 samples
        C = 1e-08         Accuracy: 0.710058027079
        C = 3.35981828628e-08         Accuracy: 0.710058027079
        C = 1.12883789168e-07         Accuracy: 0.710058027079
        C = 3.79269019073e-07         Accuracy: 0.710058027079
        C = 1.2742749857e-06         Accuracy: 0.710058027079
        C = 4.28133239872e-06         Accuracy: 0.710058027079
        C = 1.43844988829e-05         Accuracy: 0.710058027079
        C = 4.83293023857e-05         Accuracy: 0.712185686654
        C = 0.000162377673919         Accuracy: 0.725338491296
        C = 0.000545559478117         Accuracy: 0.74332688588
        C = 0.00183298071083         Accuracy: 0.763056092843
        C = 0.00615848211066         Accuracy: 0.774854932302
        C = 0.0206913808111         Accuracy: 0.789361702128
        C = 0.0695192796178         Accuracy: 0.79497098646
        C = 0.233572146909         Accuracy: 0.799419729207
        C = 0.784759970351         Accuracy: 0.801353965184
        C = 2.63665089873         Accuracy: 0.802127659574
        C = 8.8586679041         Accuracy: 0.802901353965
        C = 29.7635144163         Accuracy: 0.802707930368
        C = 100.0         Accuracy: 0.80251450677
[[ 0.71005803  0.71005803  0.71005803  0.71005803  0.71005803  0.71005803
   0.71005803  0.7098646   0.71044487  0.7106383   0.71160542  0.71682785
   0.72843327  0.74235977  0.7549323   0.77040619  0.77775629  0.7860735
   0.77562863  0.78104449]
 [ 0.71005803  0.71005803  0.71005803  0.71005803  0.71005803  0.71005803
   0.71005803  0.71005803  0.71025145  0.71102515  0.71644101  0.73017408
```

```
 0.74526112  0.75880077  0.76789168  0.78259188  0.78955513  0.79806576
 0.79245648  0.7868472 ]
[ 0.71005803  0.71005803  0.71005803  0.71005803  0.71005803  0.71005803
 0.71005803  0.71005803  0.71005803  0.7139265   0.7311412   0.74526112
 0.76363636  0.77176015  0.78317215  0.79187621  0.7934236   0.79303675
 0.79245648  0.79245648]
[ 0.71005803  0.71005803  0.71005803  0.71005803  0.71005803  0.71005803
 0.71005803  0.71005803  0.71141199  0.72572534  0.74119923  0.76189555
 0.77059961  0.78413926  0.7901354   0.79284333  0.79709865  0.79864603
 0.79941973  0.79845261]
[ 0.71005803  0.71005803  0.71005803  0.71005803  0.71005803  0.71005803
 0.71005803  0.71005803  0.7172147   0.7344294   0.75009671  0.76827853
 0.77930368  0.79361702  0.8         0.80251451  0.80483559  0.80502901
 0.8073501   0.80754352]
[ 0.71005803  0.71005803  0.71005803  0.71005803  0.71005803  0.71005803
 0.71005803  0.71218569  0.72533849  0.74332689  0.76305609  0.77485493
 0.7893617   0.79497099  0.79941973  0.80135397  0.80212766  0.80290135
 0.80270793  0.80251451]]
```

```
In [17]: # Find the index of the maximum value in the accuracies table
         maxindex = np.array([int(len(Accs)*np.argmax(Accs)/(len(Accs.flatten()))),
         print('The index of the maximum accuracy ('+str(Accs[maxindex[0],maxindex[

         besthp = hyperparams[maxindex[1]]
         bestns = samples[maxindex[0]]
         # Determine which sample count-hyperparameter combination this corresponds
         print('This corresponds to a hyperparameter of C = '+ str(besthp) + ' when
```

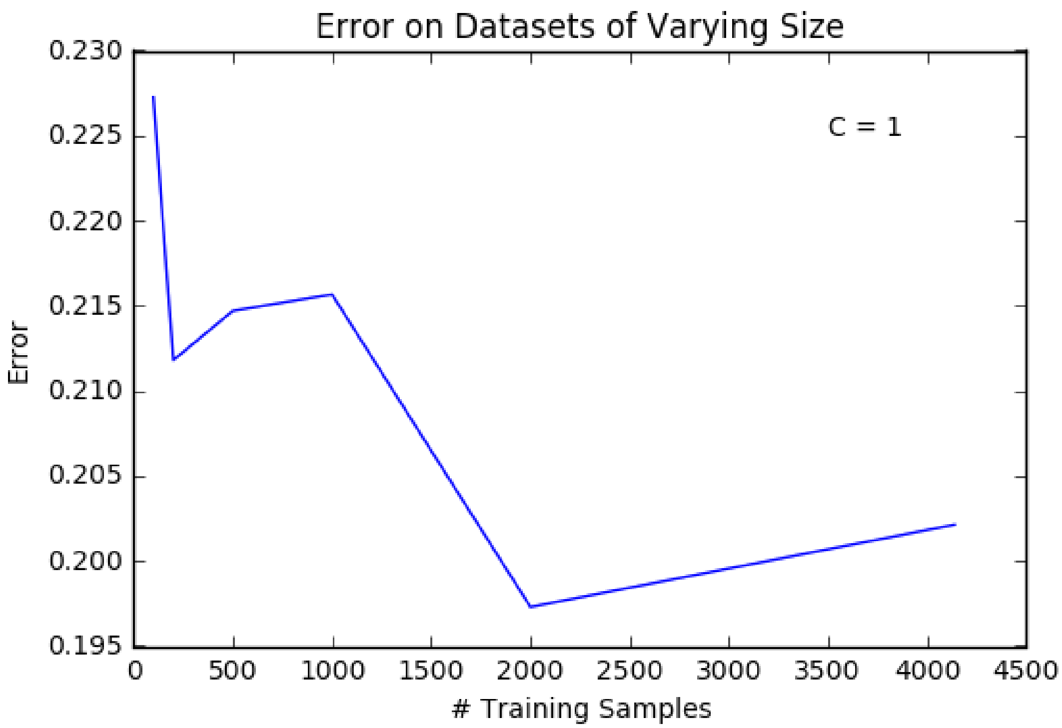
```
The index of the maximum accuracy (0.807543520309) is: [ 4 19]
This corresponds to a hyperparameter of C = 100.0 when training on 2000 samples.
```

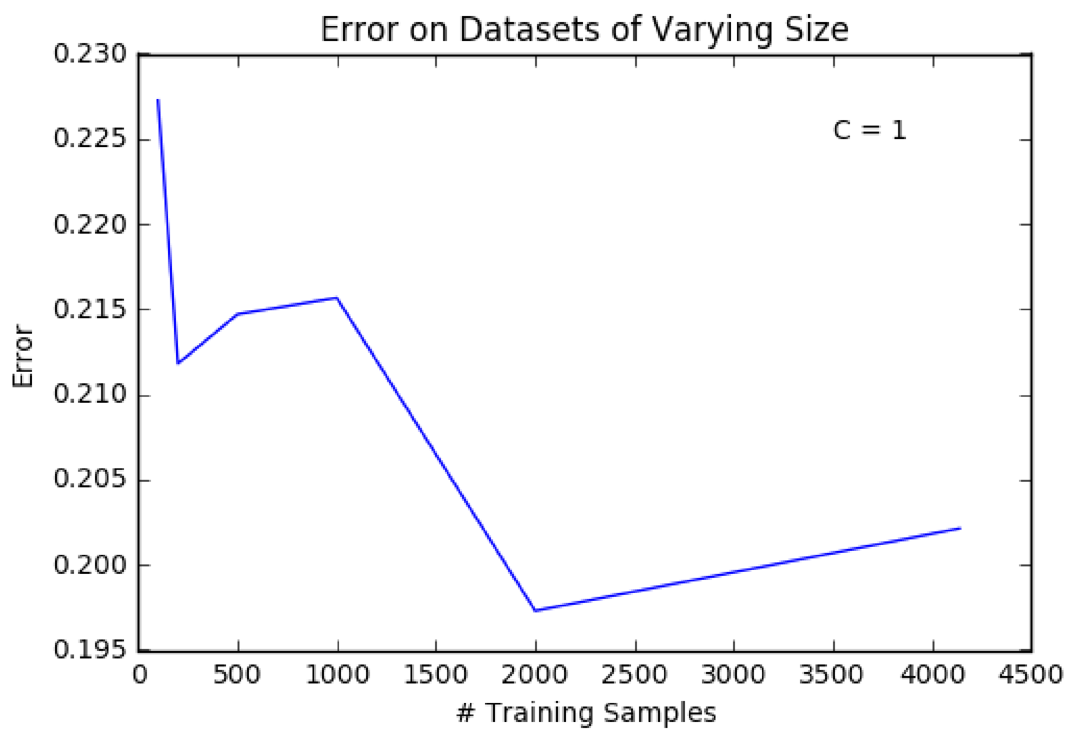
```
In [86]: # Load test data
         datafilepath = r"spam\spam_data.mat"
         spam_dict = spio.loadmat(_DATA_DIR+"\\"+datafilepath)
         spam_test = np.array(spam_dict['test_data'])
         predictions = tf.TrainAndPredictNsamples(spamdata,labels[0],spam_test,best

In [87]: IDs = np.arange(len(predictions))
         numpycsv = np.c_[IDs,predictions]
         np.savetxt(_LOCAL_PATH+r'\spam_testpredictions.csv',numpycsv,fmt='%i',deli

In [80]: from matplotlib import pyplot as plt

In [50]: hpC1 = 15

In [86]: errors = np.ones_like(AccsNoK)-AccsNoK
```

```
In [87]: fig = plt.figure()
         plt.plot(samples,errors)
         plt.title('Error on Datasets of Varying Size')
         plt.xlabel('# Training Samples')
         plt.ylabel('Error')
         plt.text(3500,0.225,'C = 1')
```

```
Out[87]: <matplotlib.text.Text at 0x1a6a6ae5b38>
```

```
In [88]: plt.show()
```






```
In [89]: fig.savefig(_LOCAL_PATH+r"\Figures\spam_SampleAcc.jpg")

In [91]: np.savetxt(_LOCAL_PATH+r'\spam_Accuracies.csv',Accs,fmt='%f',delimiter=','
         np.savetxt(_LOCAL_PATH+r'\spam_hyperparams.csv',hyperparams,fmt='%.8f',del

In [94]: print(Accs)

[[ 0.71005803  0.71005803  0.71005803  0.71005803  0.71005803  0.71005803
   0.71005803  0.7098646   0.71044487  0.7106383   0.71160542  0.71682785
   0.72843327  0.74235977  0.7549323   0.77040619  0.77775629  0.7860735
   0.77562863  0.78104449]
 [ 0.71005803  0.71005803  0.71005803  0.71005803  0.71005803  0.71005803
   0.71005803  0.71005803  0.71025145  0.71102515  0.71644101  0.73017408
   0.74526112  0.75880077  0.76789168  0.78259188  0.78955513  0.79806576
   0.79245648  0.7868472 ]
 [ 0.71005803  0.71005803  0.71005803  0.71005803  0.71005803  0.71005803
   0.71005803  0.71005803  0.71005803  0.7139265   0.7311412   0.74526112
   0.76363636  0.77176015  0.78317215  0.79187621  0.7934236   0.79303675
   0.79245648  0.79245648]
 [ 0.71005803  0.71005803  0.71005803  0.71005803  0.71005803  0.71005803
   0.71005803  0.71005803  0.71141199  0.72572534  0.74119923  0.76189555
   0.77059961  0.78413926  0.7901354   0.79284333  0.79709865  0.79864603
   0.79941973  0.79845261]
 [ 0.71005803  0.71005803  0.71005803  0.71005803  0.71005803  0.71005803
```

```
   0.71005803  0.71005803  0.7172147   0.7344294   0.75009671  0.76827853
   0.77930368  0.79361702  0.8         0.80251451  0.80483559  0.80502901
   0.8073501   0.80754352]
 [ 0.71005803  0.71005803  0.71005803  0.71005803  0.71005803  0.71005803
   0.71005803  0.71218569  0.72533849  0.74332689  0.76305609  0.77485493
   0.7893617   0.79497099  0.79941973  0.80135397  0.80212766  0.80290135
   0.80270793  0.80251451]]
```

In [95]:
```python
# Export data to csv files for report
np.savetxt(_LOCAL_PATH+r'\spam_Accuracies.csv',Accs,fmt='%f',delimiter=','
np.savetxt(_LOCAL_PATH+r'\spam_hyperparams.csv',hyperparams,fmt='%.8f',del
```

In [ ]:

CS289A_HW01_CIFAR-10

January 30, 2017

```
In [1]: %load_ext autoreload

In [2]: %autoreload 2

In [12]: from sklearn import svm
         from scipy import io as spio
         import numpy as np
         import HW01_utils as utils
         import trainfunctions as tf

In [26]: _LOCAL_PATH = r"C:\Users\Mitch\Documents\Cal\2 - 2017 Spring\COMPSCI 289A
         _DATA_PATH = "Data\hw01_data"

         _DATA_DIR = _LOCAL_PATH + "\\" + _DATA_PATH
         trainpath = r"cifar\train.mat"

In [8]: valsetsize = 5000
        samples = np.array([100, 200, 500, 1000, 2000, 5000])
        hyperparams = np.logspace(-8,4,num=20)

In [9]: # Load CIFAR-10 training data
        cifardata = spio.loadmat(_DATA_DIR+'\\'+trainpath)
        cifar = cifardata['trainX']

In [10]: # Shuffle data before splitting
         np.random.shuffle(cifar)

In [13]: trainset,valset = utils.partition(valsetsize,cifar)
         trainsetarrays,trainsetlabels = utils.separatelabels(trainset)
         valsetarrays,valsetlabels = utils.separatelabels(valset)

In [14]: Accs = np.empty((len(samples),len(hyperparams)))
         i = 0 # sample index counter
         for nsamples in samples:
             print(nsamples,'samples')
             j = 0 # hyperparameter index counter
             for hp in hyperparams:
                 acc = tf.TrainAndScoreNsamples(trainsetarrays[:nsamples],trainset
                 print('\tC =',hp,'\tAccuracy:',acc)
```

```
                Accs[i,j] = acc
                j+=1
            i+=1
        print(Accs)
```

```
100 samples
        C = 1e-08         Accuracy: 0.111
        C = 4.28133239872e-08         Accuracy: 0.1658
        C = 1.83298071083e-07         Accuracy: 0.1834
        C = 7.84759970351e-07         Accuracy: 0.1838
        C = 3.35981828628e-06         Accuracy: 0.1826
        C = 1.43844988829e-05         Accuracy: 0.1826
        C = 6.15848211066e-05         Accuracy: 0.1826
        C = 0.000263665089873         Accuracy: 0.1826
        C = 0.00112883789168         Accuracy: 0.1826
        C = 0.00483293023857         Accuracy: 0.1826
        C = 0.0206913808111         Accuracy: 0.1826
        C = 0.088586679041         Accuracy: 0.1826
        C = 0.379269019073         Accuracy: 0.1826
        C = 1.62377673919         Accuracy: 0.1826
        C = 6.95192796178         Accuracy: 0.1826
        C = 29.7635144163         Accuracy: 0.1826
        C = 127.42749857         Accuracy: 0.1826
        C = 545.559478117         Accuracy: 0.1826
        C = 2335.72146909         Accuracy: 0.1826
        C = 10000.0         Accuracy: 0.1826
200 samples
        C = 1e-08         Accuracy: 0.1026
        C = 4.28133239872e-08         Accuracy: 0.2272
        C = 1.83298071083e-07         Accuracy: 0.2622
        C = 7.84759970351e-07         Accuracy: 0.2466
        C = 3.35981828628e-06         Accuracy: 0.2466
        C = 1.43844988829e-05         Accuracy: 0.2466
        C = 6.15848211066e-05         Accuracy: 0.2466
        C = 0.000263665089873         Accuracy: 0.2466
        C = 0.00112883789168         Accuracy: 0.2466
        C = 0.00483293023857         Accuracy: 0.2466
        C = 0.0206913808111         Accuracy: 0.2466
        C = 0.088586679041         Accuracy: 0.2466
        C = 0.379269019073         Accuracy: 0.2466
        C = 1.62377673919         Accuracy: 0.2466
        C = 6.95192796178         Accuracy: 0.2466
        C = 29.7635144163         Accuracy: 0.2466
        C = 127.42749857         Accuracy: 0.2466
        C = 545.559478117         Accuracy: 0.2466
        C = 2335.72146909         Accuracy: 0.2466
        C = 10000.0         Accuracy: 0.2466
500 samples
```

```
        C = 1e-08         Accuracy: 0.2202
        C = 4.28133239872e-08         Accuracy: 0.295
        C = 1.83298071083e-07         Accuracy: 0.3008
        C = 7.84759970351e-07         Accuracy: 0.2842
        C = 3.35981828628e-06         Accuracy: 0.2708
        C = 1.43844988829e-05         Accuracy: 0.2714
        C = 6.15848211066e-05         Accuracy: 0.2714
        C = 0.000263665089873         Accuracy: 0.2714
        C = 0.00112883789168         Accuracy: 0.2714
        C = 0.00483293023857         Accuracy: 0.2714
        C = 0.0206913808111         Accuracy: 0.2714
        C = 0.088586679041         Accuracy: 0.2714
        C = 0.379269019073         Accuracy: 0.2714
        C = 1.62377673919         Accuracy: 0.2714
        C = 6.95192796178         Accuracy: 0.2714
        C = 29.7635144163         Accuracy: 0.2714
        C = 127.42749857         Accuracy: 0.2714
        C = 545.559478117         Accuracy: 0.2714
        C = 2335.72146909         Accuracy: 0.2714
        C = 10000.0         Accuracy: 0.2714
1000 samples
        C = 1e-08         Accuracy: 0.2632
        C = 4.28133239872e-08         Accuracy: 0.3164
        C = 1.83298071083e-07         Accuracy: 0.3306
        C = 7.84759970351e-07         Accuracy: 0.3148
        C = 3.35981828628e-06         Accuracy: 0.2934
        C = 1.43844988829e-05         Accuracy: 0.2934
        C = 6.15848211066e-05         Accuracy: 0.2934
        C = 0.000263665089873         Accuracy: 0.2934
        C = 0.00112883789168         Accuracy: 0.2934
        C = 0.00483293023857         Accuracy: 0.2934
        C = 0.0206913808111         Accuracy: 0.2934
        C = 0.088586679041         Accuracy: 0.2934
        C = 0.379269019073         Accuracy: 0.2934
        C = 1.62377673919         Accuracy: 0.2934
        C = 6.95192796178         Accuracy: 0.2934
        C = 29.7635144163         Accuracy: 0.2934
        C = 127.42749857         Accuracy: 0.2934
        C = 545.559478117         Accuracy: 0.2934
        C = 2335.72146909         Accuracy: 0.2934
        C = 10000.0         Accuracy: 0.2934
2000 samples
        C = 1e-08         Accuracy: 0.3
        C = 4.28133239872e-08         Accuracy: 0.3476
        C = 1.83298071083e-07         Accuracy: 0.345
        C = 7.84759970351e-07         Accuracy: 0.321
        C = 3.35981828628e-06         Accuracy: 0.3016
        C = 1.43844988829e-05         Accuracy: 0.291
```

```
        C = 6.15848211066e-05         Accuracy: 0.2906
        C = 0.000263665089873         Accuracy: 0.2906
        C = 0.00112883789168         Accuracy: 0.2906
        C = 0.00483293023857         Accuracy: 0.2906
        C = 0.0206913808111         Accuracy: 0.2906
        C = 0.088586679041         Accuracy: 0.2906
        C = 0.379269019073         Accuracy: 0.2906
        C = 1.62377673919         Accuracy: 0.2906
        C = 6.95192796178         Accuracy: 0.2906
        C = 29.7635144163         Accuracy: 0.2906
        C = 127.42749857         Accuracy: 0.2906
        C = 545.559478117         Accuracy: 0.2906
        C = 2335.72146909         Accuracy: 0.2906
        C = 10000.0         Accuracy: 0.2906
5000 samples
        C = 1e-08         Accuracy: 0.353
        C = 4.28133239872e-08         Accuracy: 0.3736
        C = 1.83298071083e-07         Accuracy: 0.3752
        C = 7.84759970351e-07         Accuracy: 0.3518
        C = 3.35981828628e-06         Accuracy: 0.3216
        C = 1.43844988829e-05         Accuracy: 0.3066
        C = 6.15848211066e-05         Accuracy: 0.2992
        C = 0.000263665089873         Accuracy: 0.2998
        C = 0.00112883789168         Accuracy: 0.2998
        C = 0.00483293023857         Accuracy: 0.2998
        C = 0.0206913808111         Accuracy: 0.2998
        C = 0.088586679041         Accuracy: 0.2998
        C = 0.379269019073         Accuracy: 0.2998
        C = 1.62377673919         Accuracy: 0.2998
        C = 6.95192796178         Accuracy: 0.2998
        C = 29.7635144163         Accuracy: 0.2998
        C = 127.42749857         Accuracy: 0.2998
        C = 545.559478117         Accuracy: 0.2998
        C = 2335.72146909         Accuracy: 0.2998
        C = 10000.0         Accuracy: 0.2998
[[ 0.111   0.1658  0.1834  0.1838  0.1826  0.1826  0.1826  0.1826  0.1826
   0.1826  0.1826  0.1826  0.1826  0.1826  0.1826  0.1826  0.1826  0.1826
   0.1826  0.1826]
 [ 0.1026  0.2272  0.2622  0.2466  0.2466  0.2466  0.2466  0.2466  0.2466
   0.2466  0.2466  0.2466  0.2466  0.2466  0.2466  0.2466  0.2466  0.2466
   0.2466  0.2466]
 [ 0.2202  0.295   0.3008  0.2842  0.2708  0.2714  0.2714  0.2714  0.2714
   0.2714  0.2714  0.2714  0.2714  0.2714  0.2714  0.2714  0.2714  0.2714
   0.2714  0.2714]
 [ 0.2632  0.3164  0.3306  0.3148  0.2934  0.2934  0.2934  0.2934  0.2934
   0.2934  0.2934  0.2934  0.2934  0.2934  0.2934  0.2934  0.2934  0.2934
   0.2934  0.2934]
 [ 0.3     0.3476  0.345   0.321   0.3016  0.291   0.2906  0.2906  0.2906
```

```
   0.2906  0.2906  0.2906  0.2906  0.2906  0.2906  0.2906  0.2906  0.2906
   0.2906  0.2906]
 [ 0.353   0.3736  0.3752  0.3518  0.3216  0.3066  0.2992  0.2998  0.2998
   0.2998  0.2998  0.2998  0.2998  0.2998  0.2998  0.2998  0.2998  0.2998
   0.2998  0.2998]]
```

In [15]:
```
# Find the index of the maximum value in the accuracies table
maxindex = np.array([int(len(Accs)*np.argmax(Accs)/(len(Accs.flatten()))),
print('The index of the maximum accuracy ('+str(Accs[maxindex[0],maxindex[

besthp = hyperparams[maxindex[1]]
bestns = samples[maxindex[0]]
# Determine which sample count-hyperparameter combination this corresponds
print('This corresponds to a hyperparameter of C = '+ str(besthp) + ' wher
```

```
The index of the maximum accuracy (0.3752) is: [5 2]
This corresponds to a hyperparameter of C = 1.83298071083e-07 when training on 5000
```

In [16]:
```
from matplotlib import pyplot as plt
```

In [17]:
```
hpC1 = 13
```

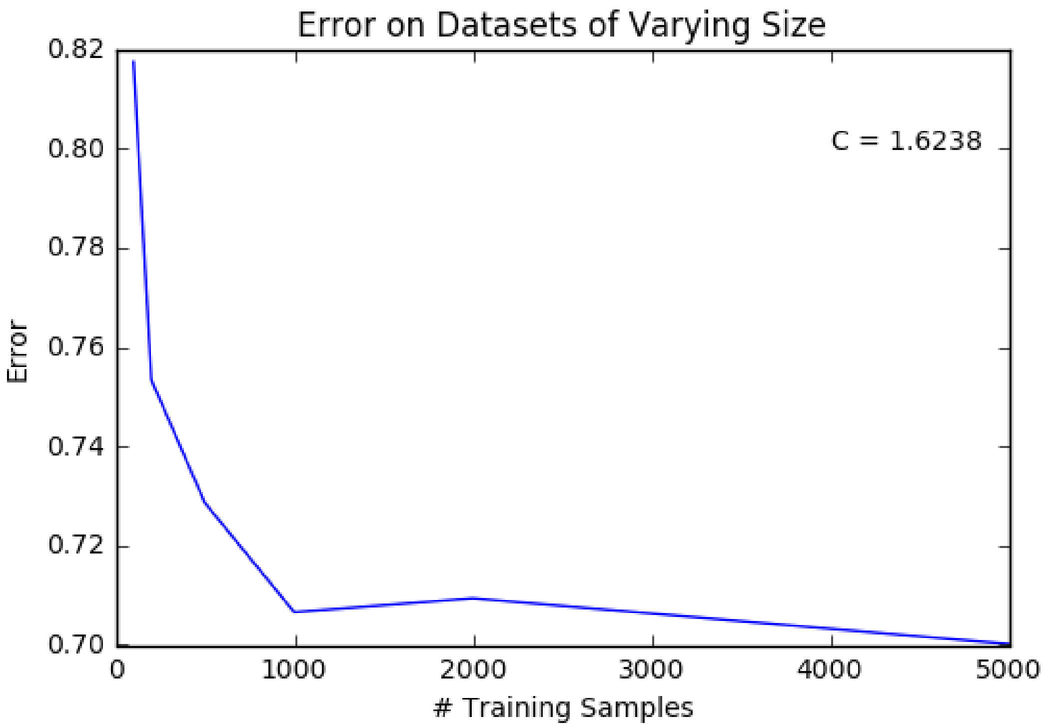
In [18]:
```
errors = np.ones_like(Accs[:,hpC1])-Accs[:,hpC1]
```

In [39]:
```
fig = plt.figure()
plt.plot(samples,errors)
plt.title('Error on Datasets of Varying Size')
plt.xlabel('# Training Samples')
plt.ylabel('Error')
plt.text(4000,0.80,'C = '+str(round(hyperparams[hpC1],4)))
```

Out[39]:
```
<matplotlib.text.Text at 0x170daddb2e8>
```

In [41]:
```
plt.show()
```

```
<matplotlib.figure.Figure at 0x170dac40630>
```




```
In [40]: plt.savefig(_LOCAL_PATH+r'\Figures\CIFAR10_SampleAcc.jpg')
```

```
C:\Users\Mitch\Documents\Cal\2 - 2017 Spring\COMPSCI 289A - Intro to Machine Learni
```

```
In [43]: # Export data to csv files for report
         np.savetxt(_LOCAL_PATH+r'\CIFAR_Accuracies.csv',Accs,fmt='%f',delimiter=',
         np.savetxt(_LOCAL_PATH+r'\CIFAR_hyperparams.csv',hyperparams,fmt='%.8f',de
```

```
# trainfunctions.py
#-----------------------------------------
# Python module for CS289A HW01
#-----------------------------------------
#-----------------------------------------


from sklearn import svm


def TrainAndScoreNsamples(trainsetarrays,trainsetlabels,valsetarrays,valsetlabels,hyperp
# Train the classifier and score on the validation set
    clf = svm.SVC(C=hyperparam,kernel='linear')
    clf.fit(trainsetarrays,trainsetlabels)
    return clf.score(valsetarrays,valsetlabels)


def TrainAndPredictNsamples(trainsetarrays,trainsetlabels,testarrays,hyperparam):
# Train the classifier and predict on the test array
    clf = svm.SVC(C=hyperparam,kernel='linear')
    clf.fit(trainsetarrays,trainsetlabels)
    return clf.predict(testarrays)
```

```python
# HW01_utils.py
#-----------------------------------------
# Python module for CS289A HW01
#-----------------------------------------
#-----------------------------------------


import numpy as np
from scipy import io as spio


def loaddata(shortpath,_DATA_DIR,dictkey):
#Load data
  data_dict = spio.loadmat(_DATA_DIR+"\\"+shortpath)
  data = np.array(data_dict[dictkey])
  return data


def partition(valsetsize,data):
# Separate <valsetsize> items for validation
  trainset = data[valsetsize:]
  valset = data[:valsetsize]
  return trainset,valset


def separatelabels(inset):
# Separate labels from sets
  setarrays = inset[:,:-1]
  setlabels = inset[:,-1]
  return setarrays,setlabels

```