HW1Digits

February 4, 2016

```
In [1]: %pylab inline
        import scipy.io
        from sklearn import svm
        DEBUG=True
```

```
Populating the interactive namespace from numpy and matplotlib
```

1 Problem 1.

Train a linear SVM using raw pixels as features. Plot the error rate on a validation set versus the number of training ex- amples that you used to train your classifier. Make sure you set aside 10,000 training images as a validation set. The number of training ex- amples in your experiment should be 100, 200, 500, 1,000, 2,000, 5,000, and 10,000. At this stage, you should expect accuracies between 70% and 90%.

```
In [2]: digit_data_test = scipy.io.loadmat("data/digit-dataset/test.mat")
        digit_data_train = scipy.io.loadmat("data/digit-dataset/train.mat")

        test_img= digit_data_test['test_images']
        train_img= digit_data_train['train_images']
        train_label= digit_data_train['train_labels']
```

```
In [3]: #Flatten the 28x28 images into 784 pixel long vectors
        train_img_flat=[]
        for i in np.arange(shape(train_img)[2]):
            train_img_flat.append(train_img[:,:,i].flatten())
        train_img_flat1= np.array(train_img_flat)
```

To debug and verify that the data partitioning is preserves a fairly uniform number of sample for each digit, I plot the histogram of the labels to visually verify that the histogram is approximately flat (i.e. uniform).

Setting aside 10,000 images for validation

Since this data is sorted , we need to pick randomly from the sample

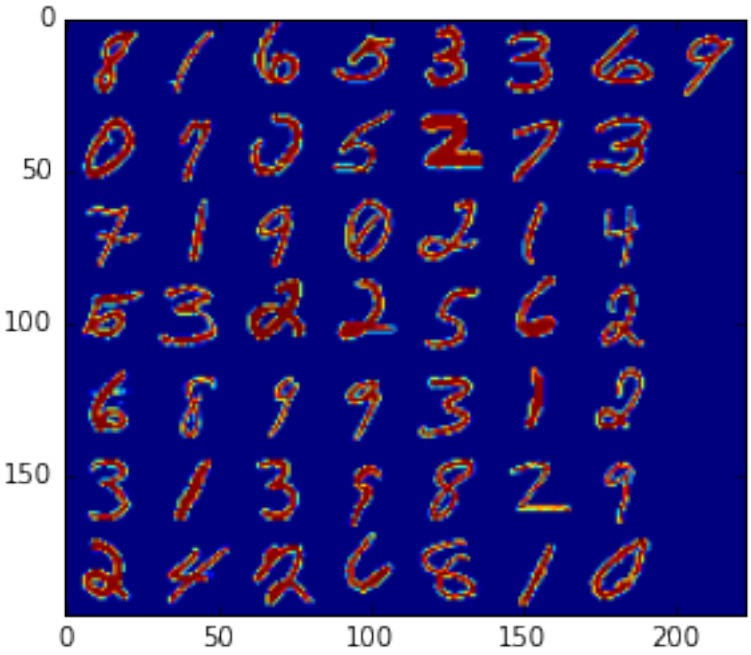
```
In [114]: #get a list of 10100 unique random numbers for indexing
          N=1000
          num_verification = 10000
          s = set()
          while len(s) < N+num_verification:
              s.add(random.randint(60000))
          rand_idx=np.array(list(s))
          np.random.shuffle(rand_idx)
```

In [115]:
```
train_subset = []
labels_subset = []
for i in rand_idx[:N]:
    train_subset.append(train_img_flat[i])
    labels_subset.append(train_label[:,0][i])
train_subset = np.array(train_subset)
labels_subset = np.array(labels_subset)
if (DEBUG) : print shape(train_subset)
if (DEBUG) :print shape(labels_subset)
```

```
(1000, 784)
(1000,)
```

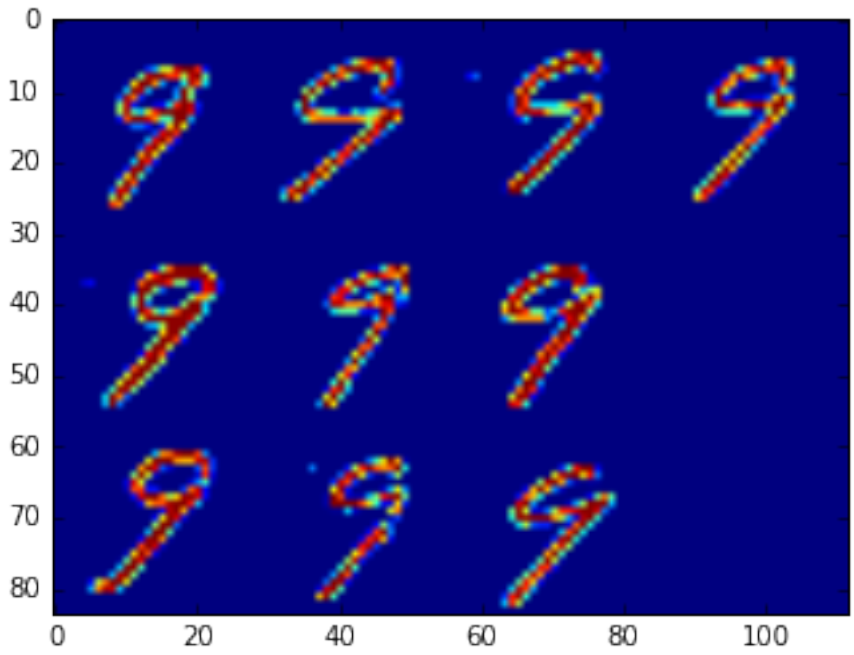
In [126]:
```
plt.imshow(montage_images(train_subset.T.reshape((28,28,1000))[:,:,:50]))
```

Out[126]: <matplotlib.image.AxesImage at 0x7f78e1e13490>

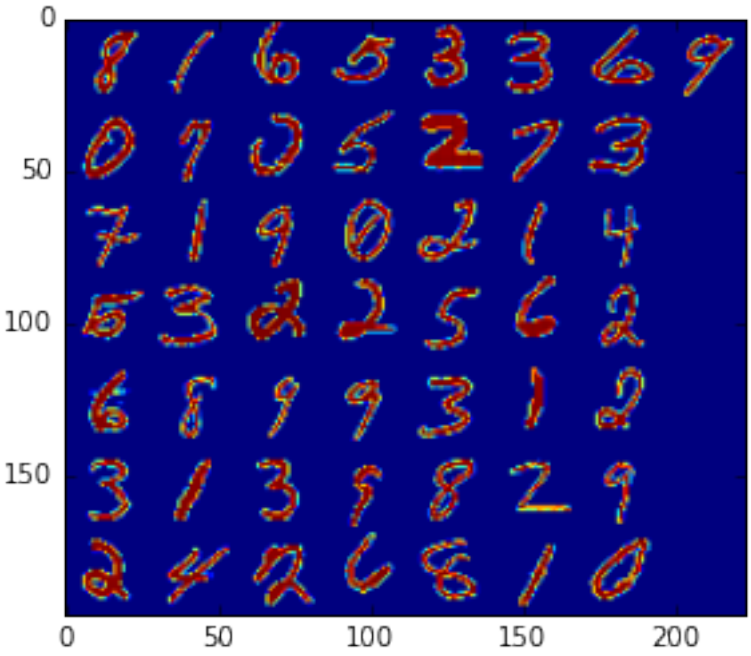


In [110]:
```
plt.imshow(montage_images(train_img[:,:,-10:]))
```

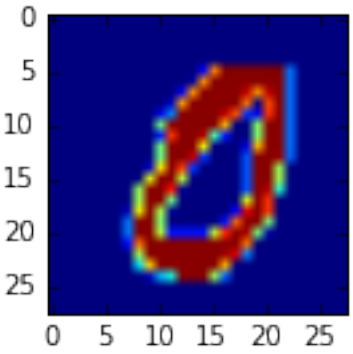
Out[110]: <matplotlib.image.AxesImage at 0x7f78e21c2610>

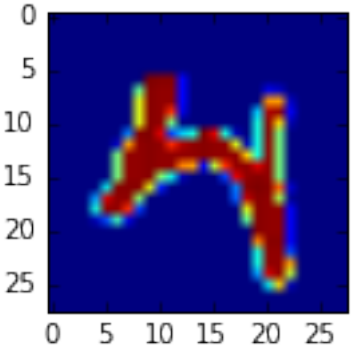


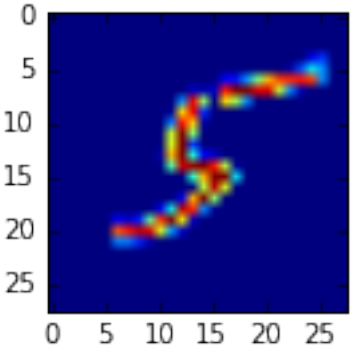

```
In [130]: #Creating 10000 verification subset
          verify_train_subset = []
          verify_labels_subset = []
          for i in rand_idx[N:num_verification+N]:
              verify_train_subset.append(train_img_flat[i])
              verify_labels_subset.append(train_label[:,0][i])
          verify_train_subset = np.array(verify_train_subset)
          verify_labels_subset = np.array(verify_labels_subset)
          if (DEBUG) : print shape(verify_train_subset)
          if (DEBUG) :print shape(verify_labels_subset)

(10000, 784)
(10000,)

In [57]: clf = svm.LinearSVC()
         clf.fit(train_subset,labels_subset)

Out[57]: LinearSVC(C=1.0, class_weight=None, dual=True, fit_intercept=True,
              intercept_scaling=1, loss='squared_hinge', max_iter=1000,
              multi_class='ovr', penalty='l2', random_state=None, tol=0.0001,
              verbose=0)

In [140]: plt.imshow(montage_images(train_subset.T.reshape((28,28,1000))[:,:,:50]))

Out[140]: <matplotlib.image.AxesImage at 0x7f78e1b702d0>
```

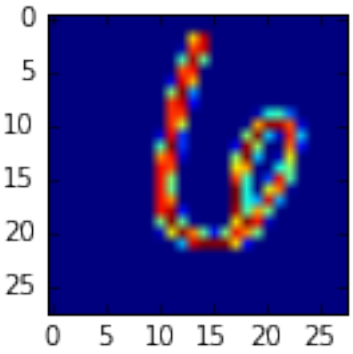


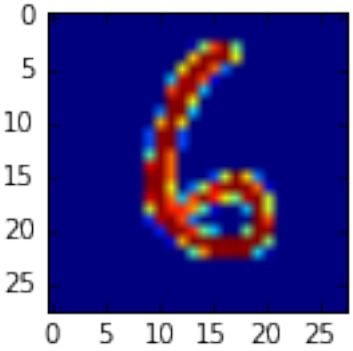

```
In [183]: def check_classifier(i):
              fig, ax = subplots(figsize=(2,2))
              ax.imshow(verify_train_subset[i].reshape((28,28)))
              print clf.predict(verify_train_subset[i])

In [167]: def check_classifier(i):
              fig, ax = subplots(figsize=(2,2))
              ax.imshow(train_img[:,:,i])
              print clf.predict(train_img[:,:,i].reshape(784,))

In [168]: check_classifier(10)
          check_classifier(30000)
          check_classifier(35000)
          check_classifier(38720)
          check_classifier(41000)
          check_classifier(51000)

[0]
[4]
[4]
[6]
[6]
[5]
```

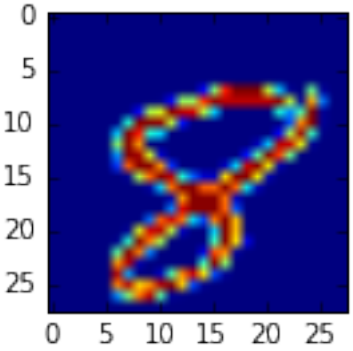

0
5
10
15
20
25
0 5 10 15 20 25


0
5
10
15
20
25
0 5 10 15 20 25


0
5
10
15
20
25
0 5 10 15 20 25





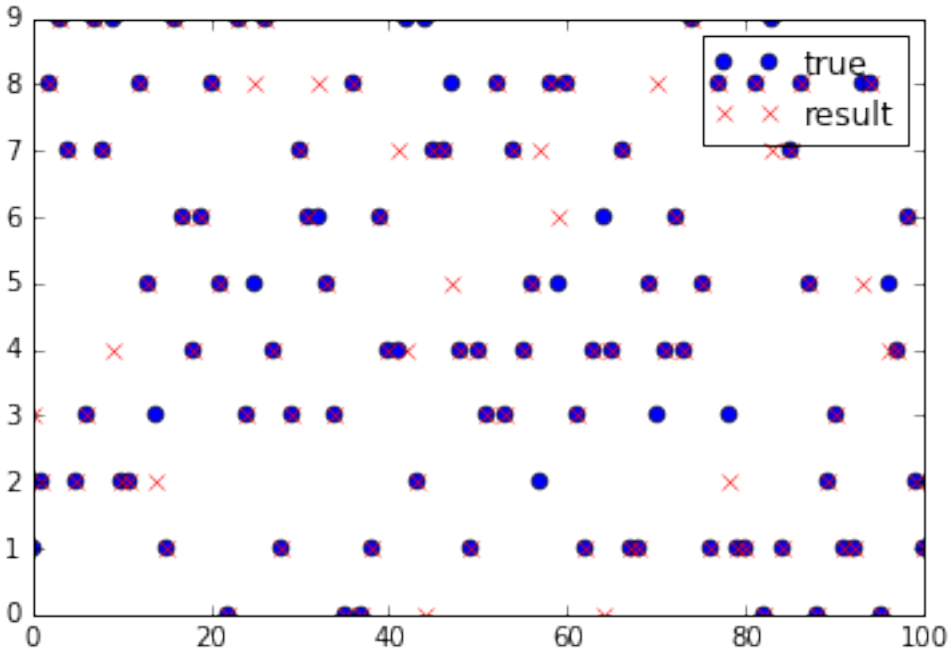



```
In [187]: result = clf.predict(verify_train_subset)
          print shape(verify_labels_subset)
          print shape(result)
```

```
(10000,)
(10000,)
```

In [188]:
```
plt.plot(verify_labels_subset,'o',label="true")
plt.plot(result,'x',color="red",label="result")
plt.legend()
plt.xlim(0,100)
```

Out[188]: (0, 100)



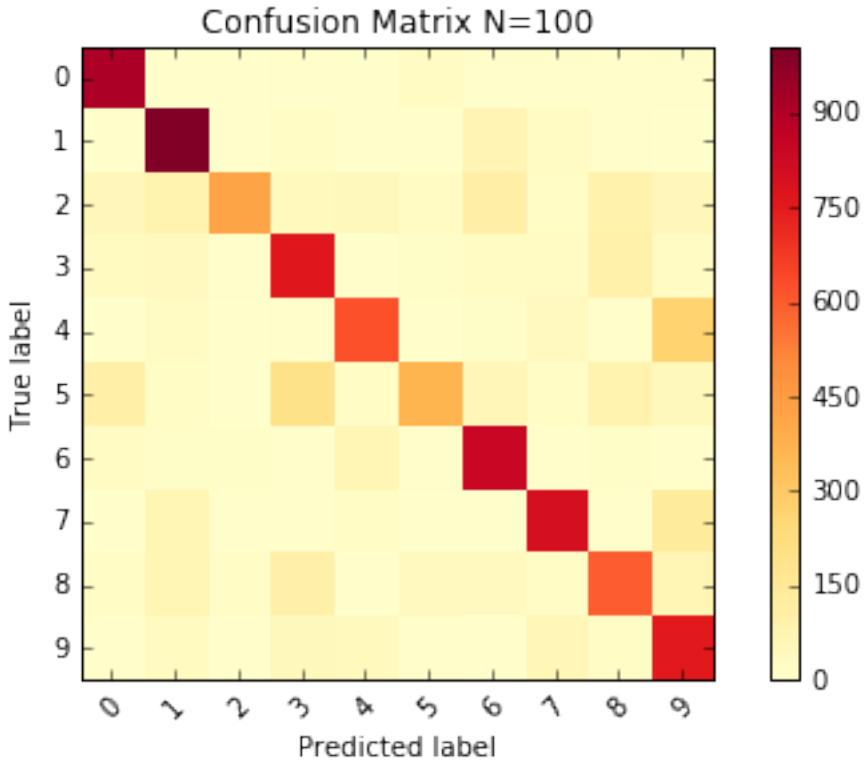
In [9]:
```
# Code written by Kunal Marwaha on Piazza
import math
#benchmark.m, converted
def benchmark(pred_labels, true_labels):
    errors = pred_labels != true_labels
    err_rate = sum(errors) / float(len(true_labels))
    indices = errors.nonzero()
    return err_rate, indices
#montage_images.m, converted
def montage_images(images):
    num_images=min(1000,np.size(images,2))
    numrows=math.floor(math.sqrt(num_images))
    numcols=math.ceil(num_images/numrows)
    img=np.zeros((numrows*28,numcols*28));
    for k in range(num_images):
        r = k % numrows
        c = k // numrows
        img[r*28:(r+1)*28,c*28:(c+1)*28]=images[:,:,k];
    return img
```

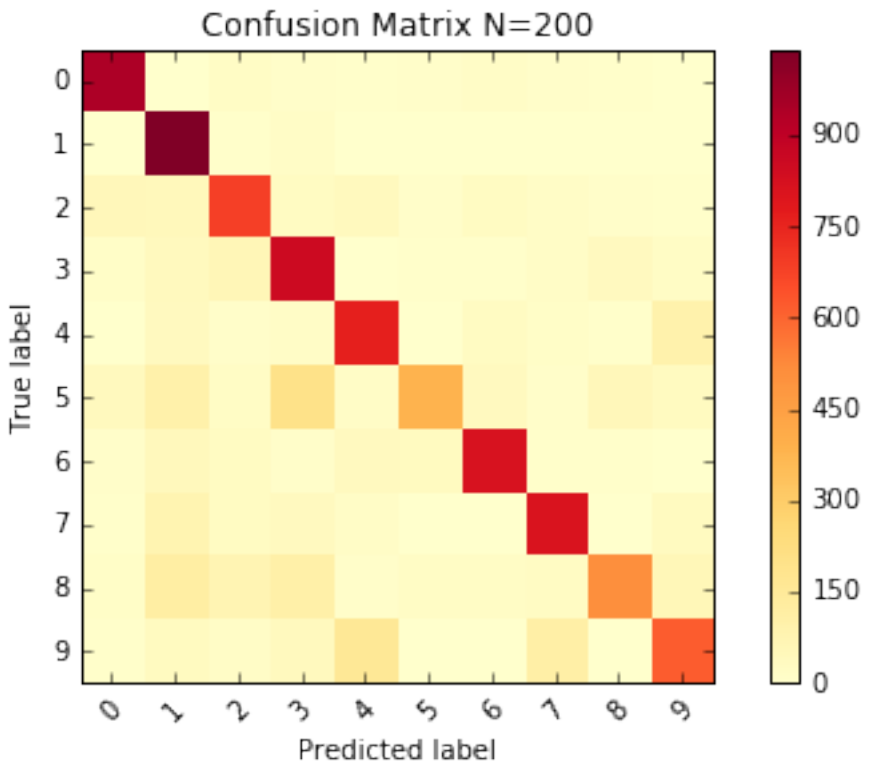
In [12]:
```
digit_data_test = scipy.io.loadmat("data/digit-dataset/test.mat")
digit_data_train = scipy.io.loadmat("data/digit-dataset/train.mat")
```

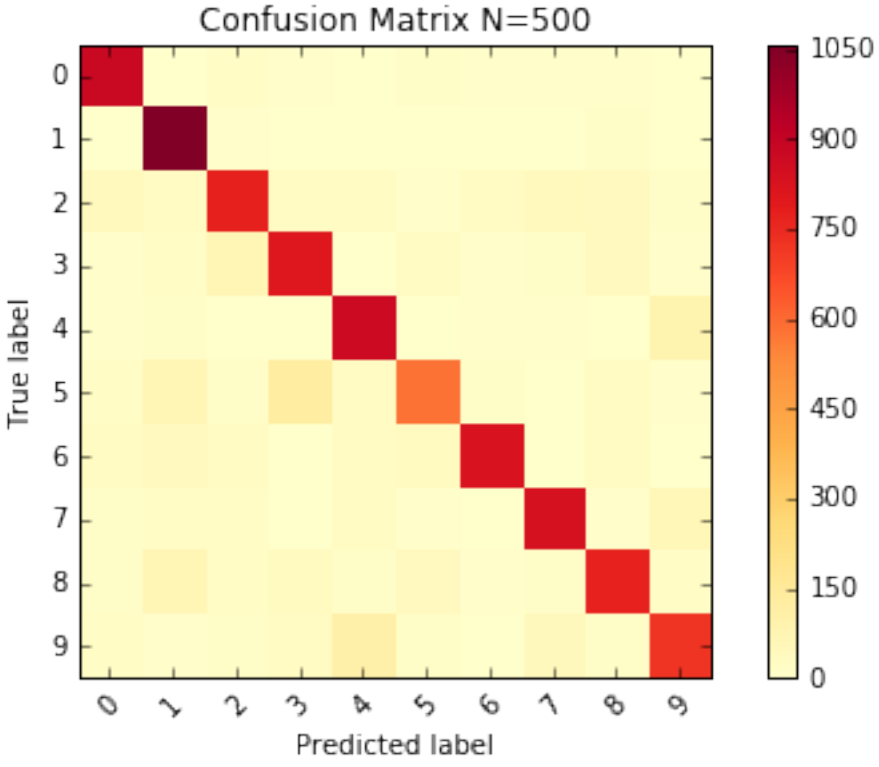
```
test_img= digit_data_test['test_images']
train_img= digit_data_train['train_images']
train_label= digit_data_train['train_labels']
def q1(N,DEBUG=False):
    #Flatten the 28x28 images into 784 pixel long vectors
    train_img_flat=[]
    for i in np.arange(shape(train_img)[2]):
        train_img_flat.append(train_img[:,:,i].flatten())
    train_img_flat= np.array(train_img_flat)
    #get a list of 10100 unique random numbers for indexing
#     N=1000
    num_verification = 10000
    s = set()
    while len(s) < N+num_verification:
        s.add(random.randint(60000))
    rand_idx=np.array(list(s))
    np.random.shuffle(rand_idx)
    if (DEBUG):print len(rand_idx)
    #Creating N number of Training set/Labels
    train_subset = []
    labels_subset = []
    for i in rand_idx[:N]:
        train_subset.append(train_img_flat[i])
        labels_subset.append(train_label[:,0][i])
    train_subset = np.array(train_subset)
    labels_subset = np.array(labels_subset)
    if (DEBUG) : print shape(train_subset)
    if (DEBUG) :print shape(labels_subset)
    #Creating 10000 verification subset
    verify_train_subset = []
    verify_labels_subset = []
    for i in rand_idx[N:num_verification+N]:
        verify_train_subset.append(train_img_flat[i])
        verify_labels_subset.append(train_label[:,0][i])
    verify_train_subset = np.array(verify_train_subset)
    verify_labels_subset = np.array(verify_labels_subset)
    if (DEBUG) : print shape(verify_train_subset)
    if (DEBUG) :print shape(verify_labels_subset)
    #Training SVM classifier
    clf = svm.SVC(kernel='linear')
    clf.fit(train_subset,labels_subset)
    result = clf.predict(verify_train_subset)
    if (DEBUG):
        plt.plot(verify_labels_subset,'o',label="true")
        plt.plot(result,'x',color="red",label="result")
        plt.legend()
        plt.xlim(0,100)
    error_rate = benchmark(result,verify_labels_subset)[0]
    wrong_labels = benchmark(result,verify_labels_subset)[1][0]
    print ("N={}".format(N))
    conf_mat = metrics.confusion_matrix(verify_labels_subset,result)
    print("Confusion matrix:\n%s" % conf_mat)
    plot_confusion_matrix(conf_mat, title="Confusion Matrix N={}".format(N))
    return error_rate
```

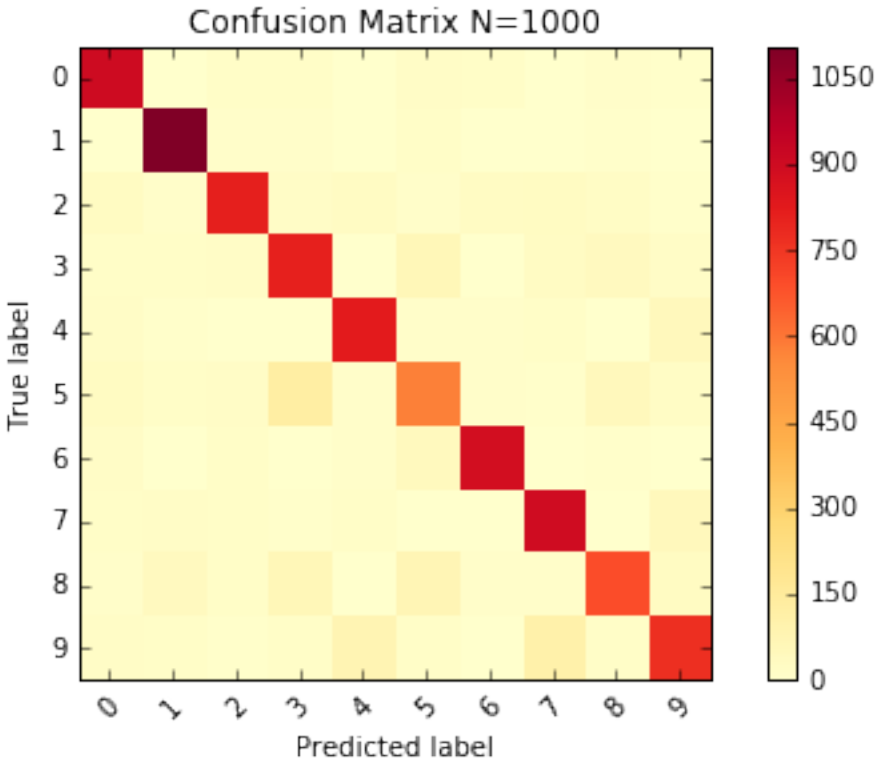
```
In [13]: err_lst = []
         trainset_size = [100, 200, 500, 1000, 2000, 5000]
         for i in trainset_size:
             err_lst.append(q1(i))

N=100
Confusion matrix:
[[ 909    0    3    5    2   30   11   10    4   10]
 [   0 1007    3   21    0    0   73   28    4    0]
 [  52   86  424   41   47   28  109   18   93   55]
 [  34   38    7  766    3   14   24   24   97   26]
 [   0   26    1    5  622    2   12   38    9  268]
 [ 106   17    1  194   19  366   60   20   85   51]
 [  28   13   12    2   64    6  843    6   12    6]
 [   2   69    4   10   16    1    2  803    7  139]
 [  21   70   20   99    3   36   38   21  598   67]
 [   8   32    1   50   41    4    1   62   16  759]]
N=200
Confusion matrix:
[[ 941    3   24   10    8   12   20   10    6    2]
 [   0 1041    7   17    1    1    1    1    4    1]
 [  55   52  687   31   44    9   31   20    9    6]
 [  13   43   61  854    4    7    7   17   37   23]
 [   0   37   10   17  767    3   31   15    7   97]
 [  44  100   23  200   18  383   38   12   55   33]
 [  12   45   30   12   39   35  818    5    5    0]
 [   6   79   29   40   20    1    0  813    0   36]
 [  16  119   76  108   10   21   23   26  511   57]
 [   7   35   17   44  159    1    4  110    2  619]]
N=500
Confusion matrix:
[[ 885    2   24    6    4   16    6    9    8    3]
 [   0 1059    9    4    0    1    2    2   16    1]
 [  42   32  777   25   33    5   26   42   38   13]
 [  10   21   72  808    0   28    6   15   41    5]
 [   8   14    2    0  870    1    9    6    2   84]
 [  22   71   13  126   25  584   14    4   30    6]
 [  29   41   26    4   33   36  828    4   30    0]
 [  14   22   21    4   28    6    0  837    9   61]
 [  13   70   18   37   15   39    5   13  772   17]
 [  17    8   16   33  104   14    3   49   13  724]]
N=1000
Confusion matrix:
[[ 905    0   16   15    1   20   15    4   12    5]
 [   0 1106    9    9    2   15    1    0    8    4]
 [  31   12  816   19   27   12   28   32   24    5]
 [  13   16   19  809    1   61    3   27   42   20]
 [  21    7    4    1  829   11   10   13    2   50]
 [  33   16   19  128   12  582   10    5   51   22]
 [  20    3   16    0   12   44  889    0    8    0]
 [  13   18   13    7   15    0    4  901    1   48]
 [  12   43   14   63    3   73   11   11  694   32]
 [  19   13    5   13   82   14    1  108   14  773]]
N=2000
```

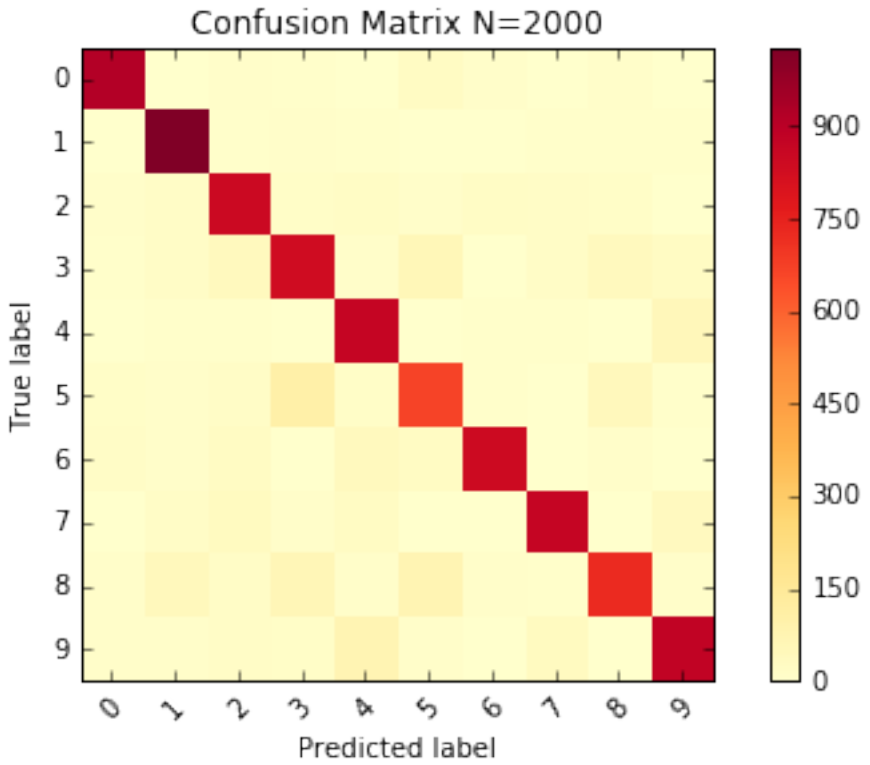
```
Confusion matrix:
[[ 917    1   10    6    4   28   10    0   10    1]
 [   1 1029    8   11    5    3    1    5    5    6]
 [   9   19  852   16   21   10   23   18   15    4]
 [   6   17   43  837   10   57    4   19   41   25]
 [   2    5    7    2  872    4    7    5    2   53]
 [  16   12   19  108   16  669    8    3   48    7]
 [  17   12   25    1   41   27  845    0   12    0]
 [   4   19   36   10   26    2    0  872    4   37]
 [  12   45   18   62   11   73   10    6  729   12]
 [  10   10   19   13   76    9    1   35    4  883]]
N=5000
Confusion matrix:
[[ 954    2    6    3    2    5    5    0    5    3]
 [   0 1092    4    2    1    1    2    3   11    5]
 [  12   32  872   16   19    9   13   12   20    4]
 [   7   23   33  898    3   53    3   14   33    7]
 [   8    6   11    1  899    4   12   10    1   33]
 [  14   14   15   59   13  721   19    1   20    7]
 [   8   10   18    2   20   22  832    0    4    0]
 [   7    9   19    6   20    3    1  910    1   50]
 [   9   37   35   55    4   44    4    6  815   19]
 [   7   12    6   16   73    3    0   50    5  801]]
```

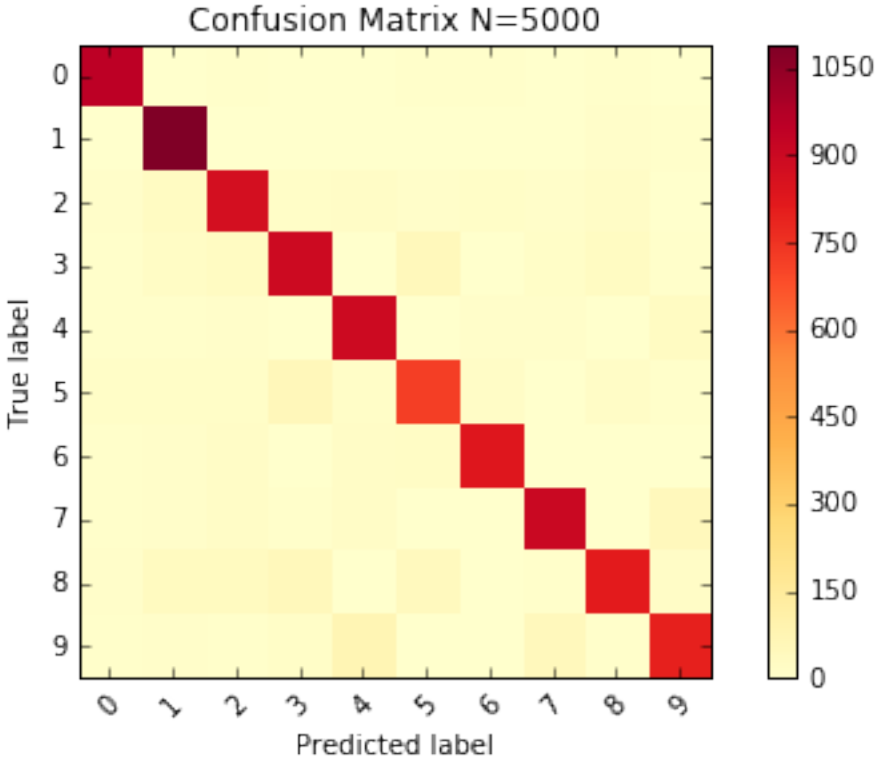



Confusion Matrix N=200
True label
Predicted label
0
1
2
3
4
5
6
7
8
9
900
750
600
450
300
150
0


Confusion Matrix N=500
True label
Predicted label
0
1
2
3
4
5
6
7
8
9
1050
900
750
600
450
300
150
0

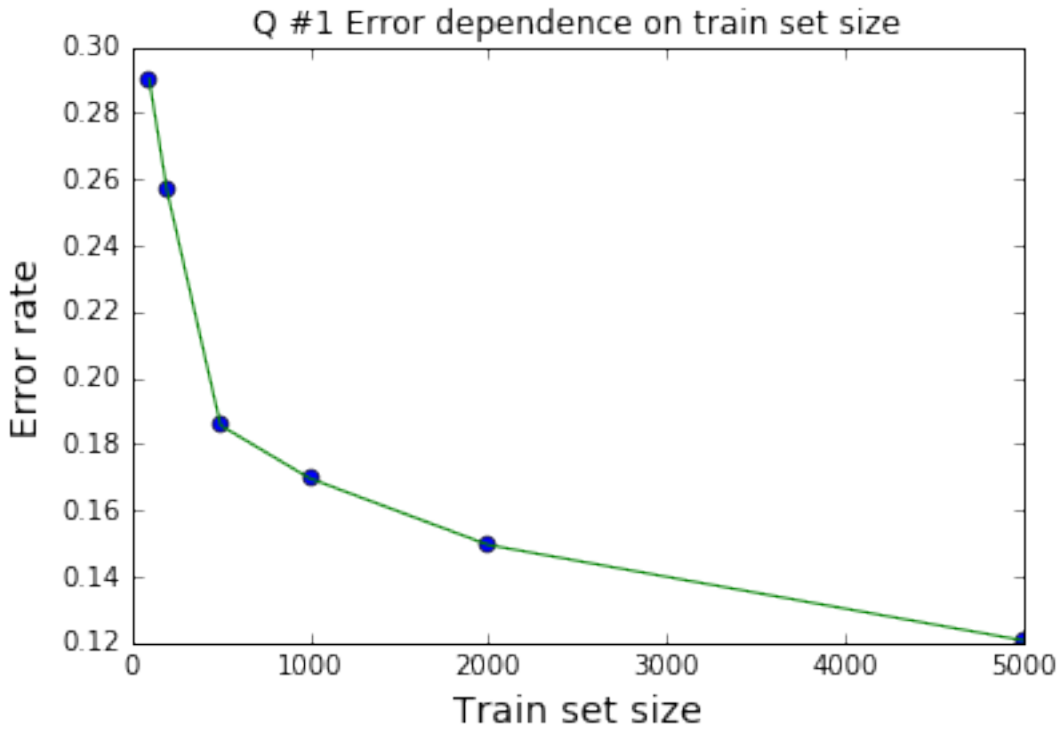

Confusion Matrix N=1000
True label
Predicted label
0
1
2
3
4
5
6
7
8
9
1050
900
750
600
450
300
150
0


Confusion Matrix N=2000
True label
Predicted label
0
1
2
3
4
5
6
7
8
9
900
750
600
450
300
150
0


Confusion Matrix N=5000
True label
Predicted label
0
1
2
3
4
5
6
7
8
9
1050
900
750
600
450
300
150
0

```
In [14]: plt.plot(trainset_size,np.array(err_lst),'o')
         plt.plot(trainset_size,np.array(err_lst),'-')
         plt.xlabel("Train set size",fontsize=14)
         plt.ylabel("Error rate",fontsize=14)
         plt.title("Q #1 Error dependence on train set size")
```

```
Out[14]: <matplotlib.text.Text at 0x7f35c5e53cd0>
```



2 Problem 2.

Create confusion matrices2 for each experiment in Problem 1. Color code and report your results. You may use built-in implementations to generate confusion matrices. What insights can you get about the performance of your algorithm from looking at the confusion matrix? The confusion matrix is a 10x10 matrix since we have 10 features (numbers 0˜9). We see very strong central diagonals because those indicate the number of datapoints that have their predicted classiciation the same as the same as the actual label from the verification dataset, this indicates that are classifier is doing a good job. As the sample size increases, we see that the non-diagonal elements have a lower and lower value (more yellower in my colormap), this is because the non-diagonal elements indicate that the labels and predicted labels don't correspond. We find that there are less misclassifications as the training set increases. Confusion Matrix for each training test size is plotted above.

```
In [10]: from sklearn import metrics
         def plot_confusion_matrix(conf_mat, title='Confusion matrix'):
             plt.figure()
```

```
plt.imshow(conf_mat, interpolation='nearest', cmap= plt.cm.YlOrRd)
plt.title(title)
plt.colorbar()
tick_marks = np.arange(10)
plt.xticks(tick_marks, tick_marks, rotation=45)
plt.yticks(tick_marks, tick_marks)
plt.tight_layout()
plt.ylabel('True label')
plt.xlabel('Predicted label')
```