CS 189 –Spring 2016 — Homework 1 Solutions

Utsav Baral, SID 25694452

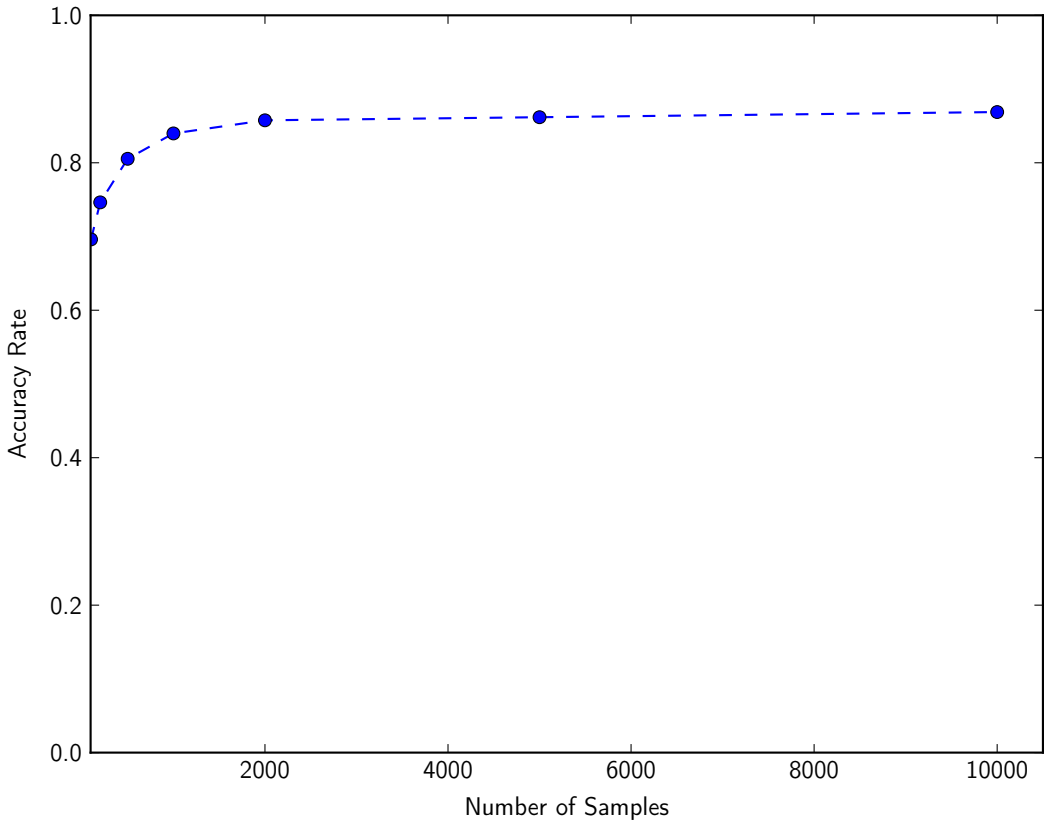
DUE: February 10th, 2016

Contents

Problem 1.

Train a linear SVM using raw pixels as features. Plot the error rate on a validation set versus thenumber of training examples that you used to train your classifier.

Here is the plot I generated by using a linear kernal svm from the scikit-learn library. The Number of Training samples is plotted against the Accuracy Rate. The number of samples, per trial are $[100, 200, 500, 1000, 2000, 5000, 10000]$, respectively. And the corresponding Accuracy Rates are $[0.6961, 0.7463, 0.8054, 0.8398, 0.8576, 0.8618, 0.8688]$:



Problem 2.

Create confusion matrices 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 gives us an easily visualizable way to look at the performance of the classifier.

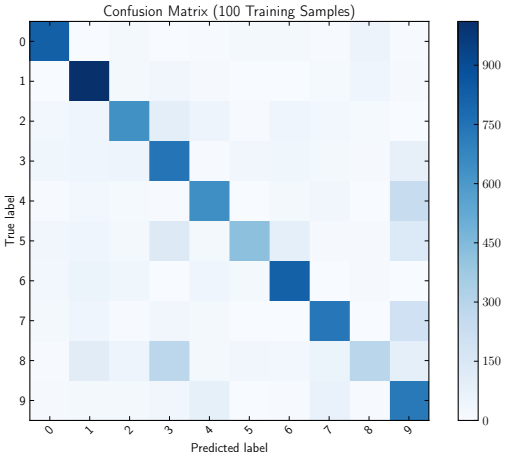
Along the diagonals we will see how many times the predicted value matched the actual value. The row number corresponds to what the actual digit was and the column number represents what digit it was predicted to be. We can also see which wrong thing it was classified most as. Since we are working with raw pixels as the features, it will probably missclassify similar looking shapes. Which is confirmed in the matrix by numbers like 4, which gets heavily misclassified as a 9; or 5 which often gets missclassified as 3, and 8 also gets frequently misclassified as a 3.

Additionally, looking at the same element but in the Transposed version of the Matrix, if two digits, say a and b, are similar we can see how many of a gets missclassified as b, or how many images of b get misclassified as a. Which would be usefull in fine tuning what features we may or may not want.

Here are the actual Confusion Matrices from the different trials, with the actual values as well as color coded for easier visualization:

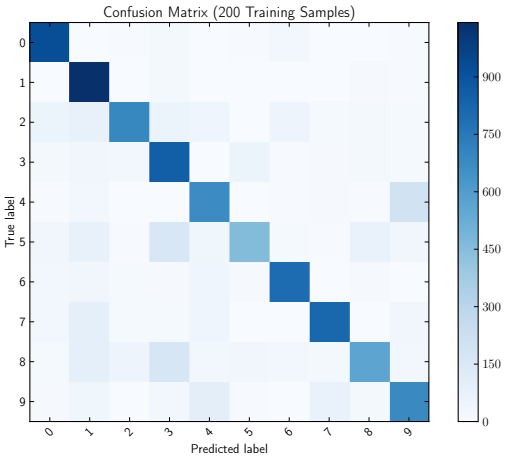
100 Training Samples:

$$\begin{bmatrix} 825 & 0 & 19 & 0 & 6 & 18 & 23 & 4 & 57 & 4 \\ 0 & 1011 & 21 & 33 & 9 & 1 & 2 & 15 & 46 & 10 \\ 28 & 42 & 633 & 101 & 48 & 6 & 42 & 25 & 14 & 2 \\ 36 & 40 & 51 & 744 & 6 & 33 & 38 & 18 & 9 & 81 \\ 7 & 31 & 14 & 4 & 643 & 0 & 17 & 34 & 3 & 238 \\ 35 & 41 & 18 & 127 & 23 & 419 & 89 & 9 & 10 & 135 \\ 28 & 52 & 37 & 2 & 42 & 21 & 820 & 0 & 11 & 0 \\ 17 & 42 & 5 & 35 & 22 & 0 & 1 & 732 & 0 & 202 \\ 5 & 112 & 50 & 284 & 17 & 33 & 31 & 53 & 289 & 87 \\ 9 & 22 & 21 & 33 & 83 & 1 & 4 & 69 & 5 & 725 \end{bmatrix}$$



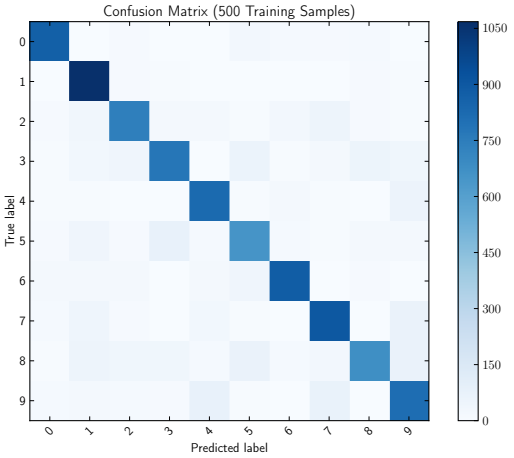
200 Training Samples:

$$\begin{bmatrix}
923 & 0 & 5 & 20 & 2 & 8 & 31 & 4 & 4 & 8 \\
1 & 1042 & 3 & 21 & 0 & 4 & 1 & 4 & 12 & 7 \\
53 & 74 & 688 & 56 & 44 & 4 & 51 & 14 & 24 & 13 \\
19 & 34 & 31 & 851 & 2 & 62 & 6 & 11 & 20 & 13 \\
5 & 31 & 1 & 1 & 673 & 2 & 5 & 9 & 7 & 198 \\
35 & 78 & 7 & 161 & 39 & 456 & 15 & 8 & 72 & 36 \\
32 & 33 & 14 & 9 & 45 & 11 & 800 & 0 & 9 & 2 \\
30 & 93 & 16 & 16 & 43 & 3 & 1 & 820 & 3 & 36 \\
13 & 91 & 50 & 165 & 25 & 34 & 27 & 19 & 563 & 30 \\
10 & 39 & 1 & 26 & 105 & 5 & 0 & 73 & 17 & 682
\end{bmatrix}$$



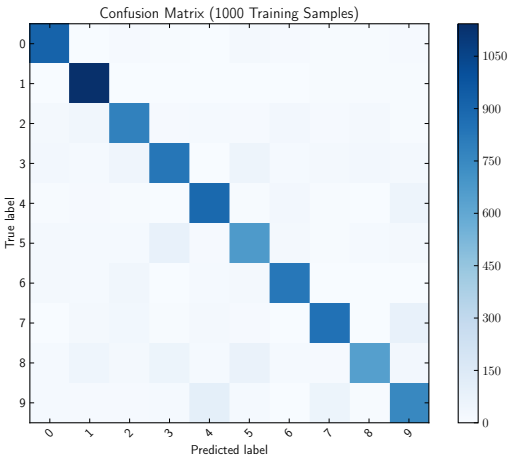
500 Training Samples:

$$\begin{bmatrix}
870 & 1 & 9 & 4 & 3 & 27 & 16 & 12 & 13 & 1 \\
0 & 1068 & 10 & 8 & 1 & 4 & 1 & 0 & 12 & 6 \\
12 & 34 & 740 & 20 & 19 & 6 & 26 & 51 & 11 & 7 \\
6 & 32 & 44 & 778 & 2 & 66 & 7 & 20 & 55 & 38 \\
6 & 7 & 4 & 3 & 827 & 6 & 24 & 5 & 4 & 61 \\
9 & 42 & 9 & 80 & 13 & 651 & 15 & 7 & 19 & 22 \\
17 & 20 & 18 & 0 & 18 & 45 & 882 & 0 & 9 & 0 \\
13 & 49 & 12 & 4 & 30 & 7 & 3 & 904 & 4 & 68 \\
7 & 52 & 41 & 39 & 15 & 68 & 15 & 27 & 676 & 67 \\
16 & 20 & 13 & 12 & 80 & 5 & 1 & 71 & 4 & 814
\end{bmatrix}$$



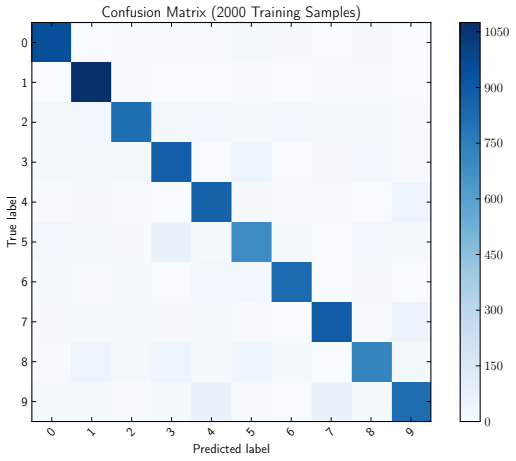
1000 Training Samples:

$$\begin{bmatrix}
916 & 1 & 10 & 5 & 1 & 21 & 11 & 7 & 7 & 10 \\
0 & 1143 & 2 & 1 & 1 & 4 & 1 & 5 & 6 & 5 \\
23 & 39 & 785 & 13 & 15 & 12 & 28 & 10 & 25 & 6 \\
28 & 17 & 46 & 834 & 1 & 56 & 14 & 21 & 29 & 23 \\
7 & 12 & 8 & 1 & 892 & 6 & 28 & 7 & 2 & 55 \\
23 & 17 & 16 & 85 & 10 & 670 & 20 & 6 & 16 & 21 \\
26 & 14 & 37 & 2 & 14 & 23 & 829 & 1 & 2 & 0 \\
2 & 25 & 34 & 5 & 18 & 10 & 0 & 856 & 2 & 85 \\
15 & 53 & 20 & 59 & 16 & 74 & 17 & 10 & 643 & 29 \\
17 & 10 & 12 & 14 & 103 & 14 & 2 & 59 & 7 & 757
\end{bmatrix}$$



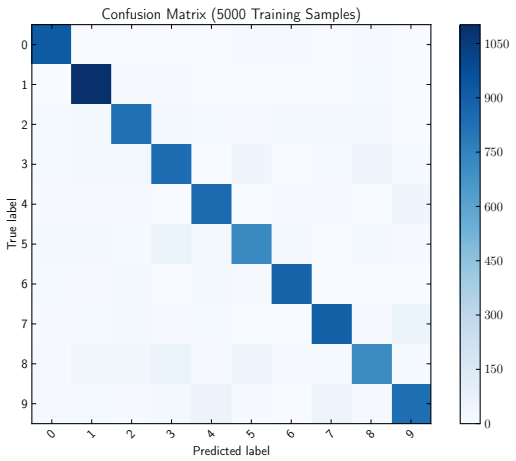
2000 Training Samples:

$$\begin{bmatrix}947 & 0 & 6 & 5 & 6 & 19 & 9 & 0 & 12 & 1\\ 0 & 1075 & 8 & 0 & 1 & 5 & 0 & 5 & 8 & 1\\ 19 & 27 & 818 & 18 & 30 & 16 & 34 & 13 & 25 & 7\\ 17 & 23 & 22 & 880 & 3 & 54 & 0 & 12 & 27 & 8\\ 7 & 9 & 7 & 1 & 861 & 11 & 8 & 5 & 2 & 52\\ 28 & 13 & 9 & 74 & 14 & 696 & 13 & 4 & 31 & 16\\ 13 & 5 & 16 & 1 & 29 & 32 & 834 & 0 & 10 & 0\\ 9 & 15 & 21 & 12 & 18 & 7 & 0 & 890 & 6 & 60\\ 7 & 52 & 21 & 53 & 14 & 47 & 14 & 3 & 719 & 21\\ 19 & 13 & 6 & 24 & 75 & 6 & 1 & 72 & 13 & 820\end{bmatrix}$$



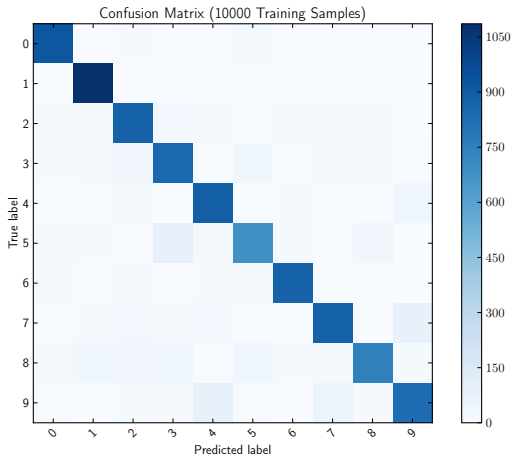
5000 Training Samples:

$$\begin{bmatrix}921 & 0 & 4 & 2 & 4 & 12 & 12 & 1 & 10 & 2\\ 2 & 1102 & 11 & 8 & 2 & 4 & 2 & 1 & 15 & 0\\ 17 & 18 & 830 & 26 & 17 & 5 & 29 & 23 & 28 & 3\\ 16 & 18 & 34 & 841 & 2 & 49 & 3 & 14 & 54 & 14\\ 7 & 6 & 11 & 0 & 851 & 1 & 11 & 11 & 3 & 46\\ 20 & 20 & 17 & 58 & 22 & 719 & 26 & 4 & 20 & 11\\ 13 & 9 & 24 & 2 & 19 & 10 & 885 & 0 & 1 & 0\\ 7 & 13 & 18 & 12 & 14 & 1 & 0 & 898 & 5 & 56\\ 9 & 38 & 35 & 71 & 12 & 45 & 12 & 15 & 718 & 17\\ 11 & 17 & 5 & 18 & 62 & 7 & 0 & 53 & 10 & 838\end{bmatrix}$$



10000 Training Samples:

$$\begin{bmatrix}924 & 0 & 10 & 4 & 6 & 24 & 3 & 1 & 4 & 1\\ 2 & 1086 & 3 & 1 & 2 & 3 & 2 & 1 & 8 & 2\\ 17 & 20 & 874 & 28 & 18 & 5 & 20 & 17 & 22 & 1\\ 15 & 13 & 37 & 849 & 4 & 40 & 5 & 24 & 20 & 8\\ 4 & 6 & 19 & 3 & 885 & 1 & 11 & 7 & 2 & 48\\ 23 & 13 & 8 & 87 & 20 & 684 & 9 & 6 & 38 & 2\\ 17 & 4 & 24 & 1 & 12 & 17 & 872 & 2 & 1 & 0\\ 2 & 9 & 27 & 9 & 27 & 2 & 2 & 879 & 2 & 83\\ 15 & 40 & 36 & 50 & 8 & 45 & 14 & 17 & 746 & 14\\ 5 & 4 & 14 & 9 & 84 & 8 & 0 & 58 & 12 & 82\end{bmatrix}$$



Problem 3.

Explain why cross-validation helps. Implement cross-validation and find the optimal value of the parameter C using 10-fold cross-validation on the training set with 10,000 examples. Train a linear SVM with this value of C. Please report your C value, the validation error rate, and your Kaggle score. If you used additional features, please (briefly) describe what features you added, removed, or modified.

Cross Validation is a powerful way to help prevent overfitting and make sure that the fit we are making will be good on real world practical applications. Additionally, at the tradeoff of time, we are getting use all the labeled data we have for validation purposes, by iterating through the k partitions and using the left over for training each time. Taking the average then gives us reliable information over the repeated testing, and on different data each time. It's a win win, except for the time consumption! Here is the output when printing out accuracy results for different c values in the code:

average accuracy for C = 1000.0 was 0.8689
average accuracy for C = 100.0 was 0.8689
average accuracy for C = 1.0was 0.8692
average accuracy for C = 0.1 was 0.8689
average accuracy for C = 0.01 was 0.8689
average accuracy for C = 0.001 was 0.8689
average accuracy for C = 0.0001 was 0.8693
average accuracy for C = 1e-05 was 0.8802
average accuracy for C = 1e-06 was 0.9018 <—Best from Trials.
average accuracy for C = 1e-08 was 0.8454
average accuracy for C = 1e-10 was 0.10869999999999999 <—lolol too soft!

Zooming in the 1e-6 range:
average accuracy for C = 2e-06 was 0.9025000000000001
average accuracy for C = 1e-06 was 0.9059000000000001 <—Best Again.
average accuracy for C = 5e-07 was 0.9047000000000001

Kaggle Score: 0.92960

Additional Features : I added a feature to detect the amount of loops that a given sample image has. For example a 9 typically has one loop, and a 8 typically has two loops. I added a feature for the amount of surface area and also dividing the images into quadrants.

Problem 4.

Use your cross-validation implementation from above to train a linear SVM for your spam dataset. Please report your C value, the validation error rate, and your Kaggle score. If you mod- ified the spam features, please (briefly) describe what features you added, removed, or modified.

Here are the C values, didn't add any more features than given, since I'm running out of time. I simply fixed up my cross-validation implementation to work on the spam data. It is clear from the data that we do not a soft classifier but rather something harder with higher c values.

average accuracy for C = 10 was 0.8022041763341067
average accuracy for C = 1 was 0.8020108275328691
average accuracy for C = 0.1 was 0.7964037122969838
average accuracy for C = 1e-06 was 0.7099767981438515

Kaggle Score: 0.73753

APPENDIX A: CODE LISTINGS

linearKernel_svm_plot.py

```
import scipy.io
import math
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm
import random

# montage_images.m, converted, function from Piazza Post
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


sampleNums = [100, 200, 500, 1000, 2000, 5000, 10000]
errRates = []

mat = scipy.io.loadmat("data/digit-dataset/train.mat")
train_images = mat["train_images"]
train_labels = mat["train_labels"]
numTrainImages = np.size(train_images, 2)
confusionMatrices = dict()

for numSamples in sampleNums:
    confusionMatrix = list()
    for i in range(10):
        row = list()
        for j in range(10):
            row.append(0)
        confusionMatrix.append(row)

    flattened_images = list()
    flattened_labels = list()
    indexSet = set([i for i in range(numTrainImages)])  # set of indices to sample from
    # choose samples we will use to train the classifier
    for _ in range(numSamples):
        i = random.sample(indexSet, 1)[0]
        indexSet.remove(i)
        flattened_images.append([item for sublist in train_images[:, :, i] for item in sublist])
        flattened_labels.append(train_labels[i][0])

    # use library to fit the actual samples to relevant classes
    clf = svm.SVC(kernel='linear')
    clf.fit(flattened_images, flattened_labels)

    # validation phase to calculate error
    numValidationImages = 10000
    correctGuess = 0
    for __ in range(numValidationImages):
        i = random.sample(indexSet, 1)[0]
        indexSet.remove(i)
        digPredicted = clf.predict([[item for sublist in train_images[:, :, i] for item in
            sublist]])[0]
        digActual = train_labels[i][0]
        confusionMatrix[digActual][digPredicted] += 1
        if digPredicted == digActual:
```

```
            correctGuess += 1

    errRate = correctGuess / numValidationImages
    errRates.append(errRate)

    confusionMatrices[numSamples] = confusionMatrix

print("x Coords = " + str(sampleNums))
print("y Coords = " + str(errRates))
print("\n\nconfusionMatrix : ")

for matr in confusionMatrices:
    print(str(matr) + ":")
    print(str(np.trace(confusionMatrices[matr])) + "<--TRACE")
    for row in confusionMatrices[matr]:
        print('[%s]' % (' '.join('%04s' % i for i in row)))

plt.plot(sampleNums, errRates, '--bo')
plt.ylabel('Accuracy Rate')
plt.xlabel('Number of Samples')
plt.ylim((0, 1))
if len(sampleNums) != 0:
    plt.xlim((sampleNums[0] - sampleNums[0] / 20, sampleNums[-1] + sampleNums[-1] / 20))
    plt.show()
```

plot_confusion_matrix.py

```
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt

mpl.rcParams['text.latex.unicode'] = True
mpl.rcParams['text.usetex'] = True
mpl.rcParams['pgf.texsystem'] = 'pdflatex'

# sublime text macros for the win.
confusionMat_100 = np.array([[825, 0, 19, 0, 6, 18, 23, 4, 57, 4],
                             [0, 1011, 21, 33, 9, 1, 2, 15, 46, 10],
                             [28, 42, 633, 101, 48, 6, 42, 25, 14, 2],
                             [36, 40, 51, 744, 6, 33, 38, 18, 9, 81],
                             [7, 31, 14, 4, 643, 0, 17, 34, 3, 238],
                             [35, 41, 18, 127, 23, 419, 89, 9, 10, 135],
                             [28, 52, 37, 2, 42, 21, 820, 0, 11, 0],
                             [17, 42, 5, 35, 22, 0, 1, 732, 0, 202],
                             [5, 112, 50, 284, 17, 33, 31, 53, 289, 87],
                             [9, 22, 21, 33, 83, 1, 4, 69, 5, 725]])

confusionMat_200 = np.array([[923, 0, 5, 20, 2, 8, 31, 4, 4, 8],
                             [1, 1042, 3, 21, 0, 4, 1, 4, 12, 7],
                             [53, 74, 688, 56, 44, 4, 51, 14, 24, 13],
                             [19, 34, 31, 851, 2, 62, 6, 11, 20, 13],
                             [5, 31, 1, 1, 673, 2, 5, 9, 7, 198],
                             [35, 78, 7, 161, 39, 456, 15, 8, 72, 36],
                             [32, 33, 14, 9, 45, 11, 800, 0, 9, 2],
                             [30, 93, 16, 16, 43, 3, 1, 820, 3, 36],
                             [13, 91, 50, 165, 25, 34, 27, 19, 563, 30],
                             [10, 39, 1, 26, 105, 5, 0, 73, 17, 682]])

confusionMat_500 = np.array([[870, 1, 9, 4, 3, 27, 16, 12, 13, 1],
                             [0, 1068, 10, 8, 1, 4, 1, 0, 12, 6],
                             [12, 34, 740, 20, 19, 6, 26, 51, 11, 7],
                             [6, 32, 44, 778, 2, 66, 7, 20, 55, 38],
                             [6, 7, 4, 3, 827, 6, 24, 5, 4, 61],
                             [9, 42, 9, 80, 13, 651, 15, 7, 19, 22],
                             [17, 20, 18, 0, 18, 45, 882, 0, 9, 0],
                             [13, 49, 12, 4, 30, 7, 3, 904, 4, 68],
                             [7, 52, 41, 39, 15, 68, 15, 27, 676, 67],
                             [16, 20, 13, 12, 80, 5, 1, 71, 4, 814]])

confusionMat_1000 = np.array([[916, 1, 10, 5, 1, 21, 11, 7, 7, 10],
                              [0, 1143, 2, 1, 1, 4, 1, 5, 6, 5],
                              [23, 39, 785, 13, 15, 12, 28, 10, 25, 6],
                              [28, 17, 46, 834, 1, 56, 14, 21, 29, 23],
                              [7, 12, 8, 1, 892, 6, 28, 7, 2, 55],
                              [23, 17, 16, 85, 10, 670, 20, 6, 16, 21],
                              [26, 14, 37, 2, 14, 23, 829, 1, 2, 0],
                              [2, 25, 34, 5, 18, 10, 0, 856, 2, 85],
                              [15, 53, 20, 59, 16, 74, 17, 10, 643, 29],
                              [17, 10, 12, 14, 103, 14, 2, 59, 7, 757]])

confusionMat_2000 = np.array([[947, 0, 6, 5, 6, 19, 9, 0, 12, 1],
                              [0, 1075, 8, 0, 1, 5, 0, 5, 8, 1],
                              [19, 27, 818, 18, 30, 16, 34, 13, 25, 7],
                              [17, 23, 22, 880, 3, 54, 0, 12, 27, 8],
                              [7, 9, 7, 1, 861, 11, 8, 5, 2, 52],
                              [28, 13, 9, 74, 14, 696, 13, 4, 31, 16],
                              [13, 5, 16, 1, 29, 32, 834, 0, 10, 0],
                              [9, 15, 21, 12, 18, 7, 0, 890, 6, 60],
                              [7, 52, 21, 53, 14, 47, 14, 3, 719, 21],
                              [19, 13, 6, 24, 75, 6, 1, 72, 13, 820]])
```

```
confusionMat_5000 = np.array([[921, 0, 4, 2, 4, 12, 12, 1, 10, 2],
                              [2, 1102, 11, 8, 2, 4, 2, 1, 15, 0],
                              [17, 18, 830, 26, 17, 5, 29, 23, 28, 3],
                              [16, 18, 34, 841, 2, 49, 3, 14, 54, 14],
                              [7, 6, 11, 0, 851, 1, 11, 11, 3, 46],
                              [20, 20, 17, 58, 22, 719, 26, 4, 20, 11],
                              [13, 9, 24, 2, 19, 10, 885, 0, 1, 0],
                              [7, 13, 18, 12, 14, 1, 0, 898, 5, 56],
                              [9, 38, 35, 71, 12, 45, 12, 15, 718, 17],
                              [11, 17, 5, 18, 62, 7, 0, 53, 10, 838]])

confusionMat_10000 = np.array([[924, 0, 10, 4, 6, 24, 3, 1, 4, 1],
                               [2, 1086, 3, 1, 2, 3, 2, 1, 8, 2],
                               [17, 20, 874, 28, 18, 5, 20, 17, 22, 1],
                               [15, 13, 37, 849, 4, 40, 5, 24, 20, 8],
                               [4, 6, 19, 3, 885, 1, 11, 7, 2, 48],
                               [23, 13, 8, 87, 20, 684, 9, 6, 38, 2],
                               [17, 4, 24, 1, 12, 17, 872, 2, 1, 0],
                               [2, 9, 27, 9, 27, 2, 2, 879, 2, 83],
                               [15, 40, 36, 50, 8, 45, 14, 17, 746, 14],
                               [5, 4, 14, 9, 84, 8, 0, 58, 12, 829]])


# This function was found online from the scikit-learn examples of confusion matrix website here
    is a link to the page:
# http://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html
def plot_confusion_matrix(cm, y, title='Confusion Matrix (10000 Training Samples)',
    cmap=plt.cm.Blues):
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(y))
    plt.xticks(tick_marks, y, rotation=45)
    plt.yticks(tick_marks, y)
    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

# plot_confusion_matrix(confusionMat_100, [0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
# plot_confusion_matrix(confusionMat_10000, [0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
# plot_confusion_matrix(confusionMat_500, [0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
# plot_confusion_matrix(confusionMat_1000, [0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
# plot_confusion_matrix(confusionMat_2000, [0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
# plot_confusion_matrix(confusionMat_5000, [0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
# plot_confusion_matrix(confusionMat_10000, [0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
# plt.savefig('LaTex Writeup/graphics/confusionMat_10000.pgf')
# plt.show()
```

ten_fold_crossValidation.py

```
import scipy.io
import numpy as np
from sklearn import svm
import random

# load in the data files
mat = scipy.io.loadmat("data/digit-dataset/train.mat")
train_images = mat["train_images"]
train_labels = mat["train_labels"]
numTrainImages = np.size(train_images, 2)

# we flatten the labels and images of 10,000 random images from our set of 60,000.
# These 10,000 images will be partitioned and used to perform the 10-cross validation later.
indexSet = set([i for i in range(numTrainImages)])

k = 10
partitionSize = int(10000 / k)
# each k integer value 0,1,2,...,9 will map to the kth partitioning of the 10000 images.
# Each partition will be a list of a list of images and a list of corresponding labels,
# for training/validation purposes.
partitionDict = dict()
for i in range(k):
    partitionDict[i] = [[], []]

# Here we select 10000 random images from the full set of 60,000 images we have.
# In the loop we will mod by k and send that image to the appropriate "bucket" partition that it
    belongs to.
lstOfImages = []
for _ in range(10000):
    partitionKey = _ % k
    i = random.sample(indexSet, 1)[0]
    indexSet.remove(i)
    partitionDict[partitionKey][0].append([item for sublist in train_images[:, :, i] for item in
        sublist])
    partitionDict[partitionKey][1].append(int(train_labels[i][0]))

# run k iterations and validate for kth partition and train on the rest.

C_Values = [2e-6, 1e-6, (.5) * (1e-6), 1e-7]
for c_val in C_Values:
    accuracyTotalSum = 0
    for kk in range(k):
        clf = svm.SVC(kernel='linear', C=c_val)
        flattened_images = []
        flattened_labels = []
        for j in range(k):
            if j != kk:
                flattened_images += partitionDict[j][0]
                flattened_labels += partitionDict[j][1]
        # train the classifier with all images except those in partition number kk
        flattened_images = np.array(flattened_images)
        flattened_labels = np.array(flattened_labels)
        # TRAIN!
        clf.fit(flattened_images, flattened_labels)
        # calculate the running total of accuracy's which will be used to calculate the average
        correctGuess = 0
        predictedLabels = clf.predict(partitionDict[kk][0])
        actualLabels = partitionDict[kk][1]
        for i in range(partitionSize):
            if predictedLabels[i] == actualLabels[i]:
                correctGuess += 1
        accuracyTotalSum += correctGuess / partitionSize

    averageAccuracy = accuracyTotalSum / k
```

```
    print("average accuracy for C = " + str(c_val) + " was " + str(averageAccuracy))
```

classifierWithExtraFeatures.py

```
import scipy.io
import numpy as np
from sklearn import svm
import matplotlib.pyplot as plt
import random


def plot_confusion_matrix(cm, y, title='Confusion Matrix (10000 Training Samples)',
    cmap=plt.cm.Blues):
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(y))
    plt.xticks(tick_marks, y, rotation=45)
    plt.yticks(tick_marks, y)
    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')


def custom_feature_extractor(imagetrue):
    featureslist = [item for sublist in imagetrue for item in sublist]
    h = np.size(imagetrue, 0)
    w = np.size(imagetrue, 1)
    featureslist += [np.average(imagetrue[h / 2:, w / 2:]), np.average(imagetrue[h / 2:, :w /
        2]),
                     np.average(imagetrue[:h / 2, w / 2:]), np.average(imagetrue[:h / 2, :w /
                         2])]
    topclosed = False
    prevrow = (False, False, [], [], [])
    numholes = 0
    edgecount = 0
    for row in np.floor(imagetrue / 35.0):
        for i in range(np.size(row) - 1):
            if row[i] != 0 and row[i + 1] == 0:
                edgecount += 1
            elif row[i] == 0 and row[i + 1] != 0:
                edgecount += 1

        nonzero_elems = np.nonzero(row)[0]
        if len(nonzero_elems) > 0:
            separation = np.ediff1d(nonzero_elems) - 1
            nonzerodiffs = np.nonzero(separation)[0]
            startnonzero = nonzero_elems[0]
            endnonzero = nonzero_elems[-1]
            if len(nonzerodiffs) > 0:  # has gaps
                gaps_in_row = []
                for block in nonzerodiffs:
                    gap_start_index = nonzero_elems[block]
                    gap_end_index = nonzero_elems[block] + separation[block] + 1
                    gaps_in_row.append((gap_start_index, gap_end_index))

                # do stuff here, that has gaps of 0pixels, figure out how to detect holes
                if prevrow[0]:
                    if prevrow[1]:  # prevrow has Gaps, this row has gaps
                        for prevGap in prevrow[2]:
                            leakfromtop = shortcircuit = True
                            for ii in range(len(gaps_in_row)):
                                for p in range(gaps_in_row[ii][0], gaps_in_row[ii][1] + 1):
                                    shortcircuit = shortcircuit and prevrow[4][p] != 0
                                    if not shortcircuit:
                                        break
                                if shortcircuit:
                                    topclosed = True
```

```
                                if prevGap[0] >= startnonzero and prevGap[1] <= endnonzero:
                                    leakfromtop = False
                                    break
                            if leakfromtop:
                                topclosed = False
                                leakfromtop = False
                        else:  # prevrow no gaps, this row has gaps
                            for gap in gaps_in_row:
                                if not prevrow[3][0] <= gap[0] + 1 or not prevrow[3][1] >= gap[1] -
                                    1:
                                    topclosed = False

                    else:  # zero pixels filled row
                        topclosed = False
                    prevrow = (True, True, gaps_in_row, (startnonzero, endnonzero), row)
                else:  # only stuff here has no zero gaps in it
                    if not prevrow[0]:  # prev row was all zeros and this row is all solid
                        topclosed = True
                    else:  # prev row not all zeros
                        if prevrow[1]:  # prevrow has gaps, this row solid
                            for gap in prevrow[2]:
                                if startnonzero - 1 <= gap[0] and gap[1] <= endnonzero + 1:
                                    if topclosed:
                                        numholes += 1
                        else:  # prevrow solid, this row solid
                            topclosed = True
                    prevrow = (True, False, [], (startnonzero, endnonzero), row)
            else:  # row is all zeros
                prevrow = (False, False, [], (), row)

    binary = [0, 0, 0]
    if numholes > 2:
        numholes = 2
    binary[numholes] = 1

    return featureslist + binary + [edgecount]


mat = scipy.io.loadmat("data/digit-dataset/train.mat")
testMat = scipy.io.loadmat("data/digit-dataset/test.mat")
train_images = np.array(mat["train_images"])
train_labels = np.array(mat["train_labels"])
test_images = np.array(testMat["test_images"])
test_images = np.transpose(test_images)

numTrainImages = np.size(train_images, 2)
print("numTrainSamples = " + str(numTrainImages))
numSamples = numTrainImages
flattened_images = []
flattened_labels = []
indexSet = set([i for i in range(numTrainImages)])  # set of indices to sample from
# choose samples we will use to train the classifier
for _ in range(numSamples):
    i = random.sample(indexSet, 1)[0]
    indexSet.remove(i)
    flattened_images.append(custom_feature_extractor(train_images[:, :, i]))
    flattened_labels.append(train_labels[i][0])

# use library to fit the actual samples to relevant classes using our best calculated C value
clf = svm.SVC(kernel='linear', C=1e-6)
clf.fit(np.array(flattened_images), np.array(flattened_labels))

imagesToPredict = []
for i in range(np.size(test_images, 2)):
    imagesToPredict.append(custom_feature_extractor(test_images[:, :, i]))

imagesToPredict = np.array(imagesToPredict)
```

```
predicted = clf.predict(imagesToPredict)
f = open("output.csv", 'w')
f.write("Id,Category\n")
for i in range(np.size(predicted)):
    f.write(str(i+1) + "," + str(predicted[i]) + "\n")
print("DONE!")
# Validation will remove for actual training, and train on all 60000 images!
# numValidationImages = 10000
# confusionMatrix = np.zeros((10, 10))
#
# correctGuess = 0
# imagesToPredict = [[], []]
# for __ in range(numValidationImages):
#     i = random.sample(indexSet, 1)[0]
#     indexSet.remove(i)
#     imagesToPredict[1].append(train_labels[i][0])
#     imagesToPredict[0].append(custom_feature_extractor(test_images[:, :, i]))
#
# predicted = clf.predict(np.array(imagesToPredict[0]))
# i = 0
# for digPredicted in predicted:
#     digActual = imagesToPredict[1][i]
#     confusionMatrix[digActual][digPredicted] += 1
#     if digPredicted == digActual:
#         correctGuess += 1
#     i += 1
#
# accreate = correctGuess / numValidationImages
# print("accuracy rate is " + str(accreate))

# plot_confusion_matrix(confusionMatrix, [0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
# plt.show()
```

SpamDataset.py

```
import scipy.io
import numpy as np
from sklearn import svm
import random

# load in the data files
mat = scipy.io.loadmat("data/spam-dataset/spam_data.mat")
training_data = mat["training_data"]
training_labels = mat["training_labels"]
testing_data = mat["test_data"]

y = training_labels[0]
X = training_data
k = 12
totalSamps = np.size(training_labels)
partitionSize = int(totalSamps / k)
partitionDict = dict()
for i in range(k):
    partitionDict[i] = [[], []]

for i in range(totalSamps):
    partitionKey = i % k
    partitionDict[partitionKey][0].append(training_data[i, :])
    partitionDict[partitionKey][1].append(training_labels[0][i])
# run k iterations and validate for kth partition and train on the rest.
C_Values = [10]
for c_val in C_Values:
    accuracyTotalSum = 0
    for kk in range(k):
        clf = svm.SVC(kernel='linear', C=c_val)
        flattened_images = []
        flattened_labels = []
        for j in range(k):
            if j != kk:
                flattened_images += partitionDict[j][0]
                flattened_labels += partitionDict[j][1]
        # train the classifier with all images except those in partition number kk
        flattened_images = np.array(flattened_images)
        flattened_labels = np.array(flattened_labels)
        # TRAIN!
        clf.fit(flattened_images, flattened_labels)
        # calculate the running total of accuracy's which will be used to calculate the average
        correctGuess = 0
        predictedLabels = clf.predict(partitionDict[kk][0])
        actualLabels = partitionDict[kk][1]
        for i in range(partitionSize):
            if predictedLabels[i] == actualLabels[i]:
                correctGuess += 1
        accuracyTotalSum += correctGuess / partitionSize

    averageAccuracy = accuracyTotalSum / k
    print("average accuracy for C = " + str(c_val) + " was " + str(averageAccuracy))
```