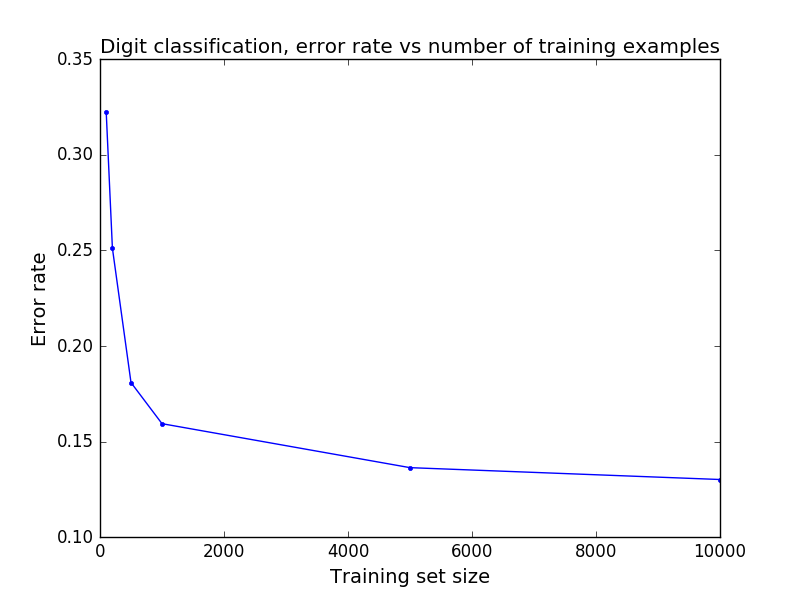
HW1 Write-up

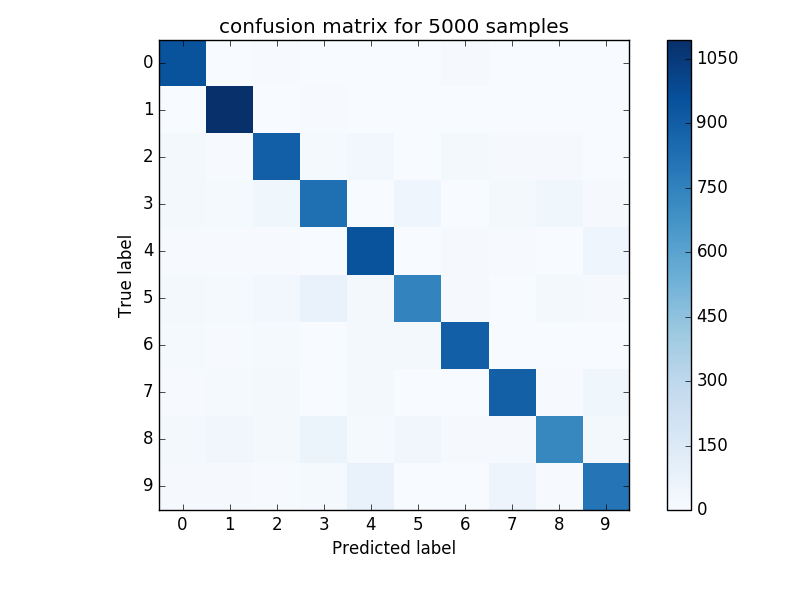
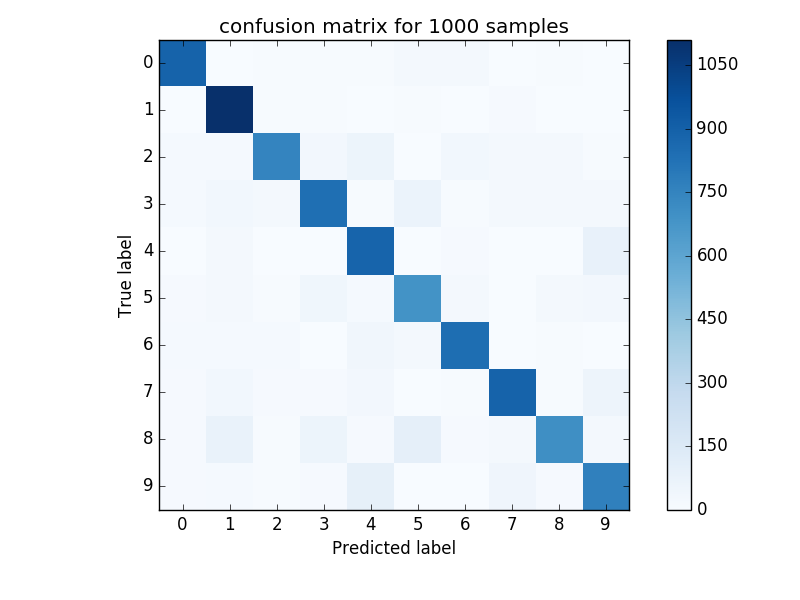
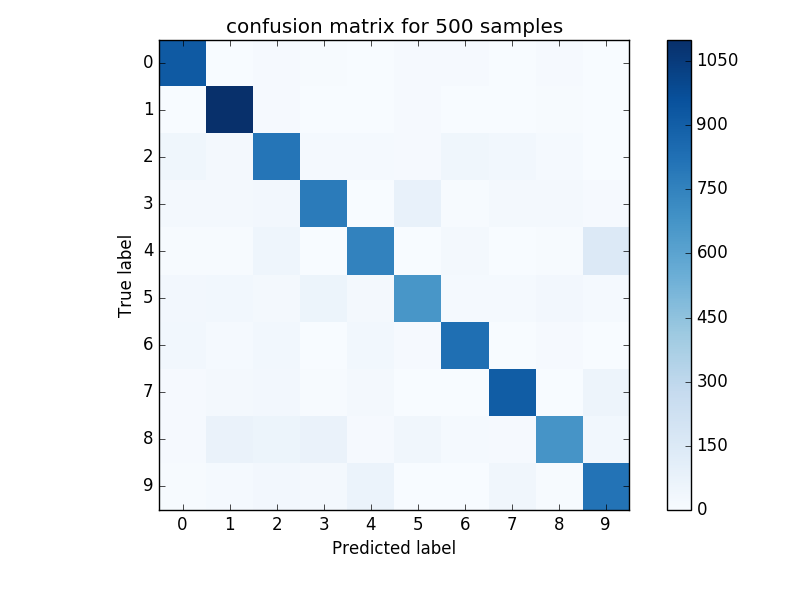
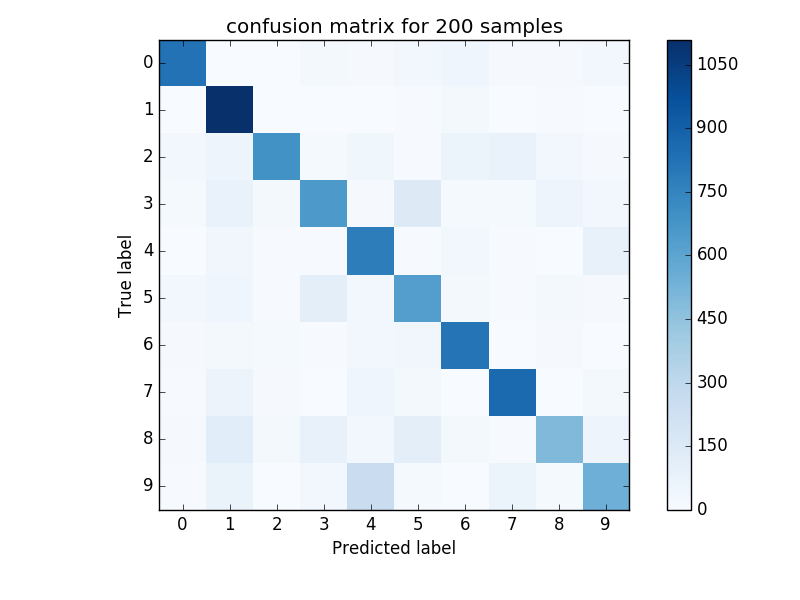
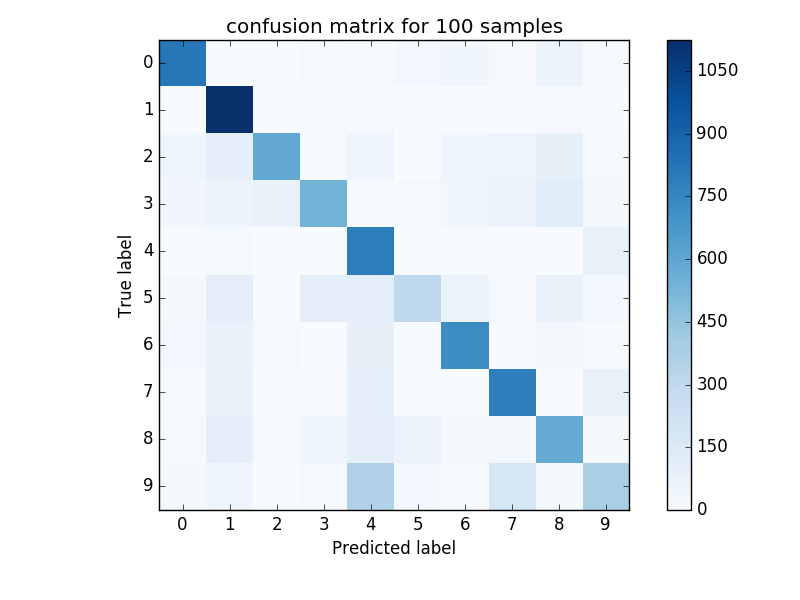
Jesse Li

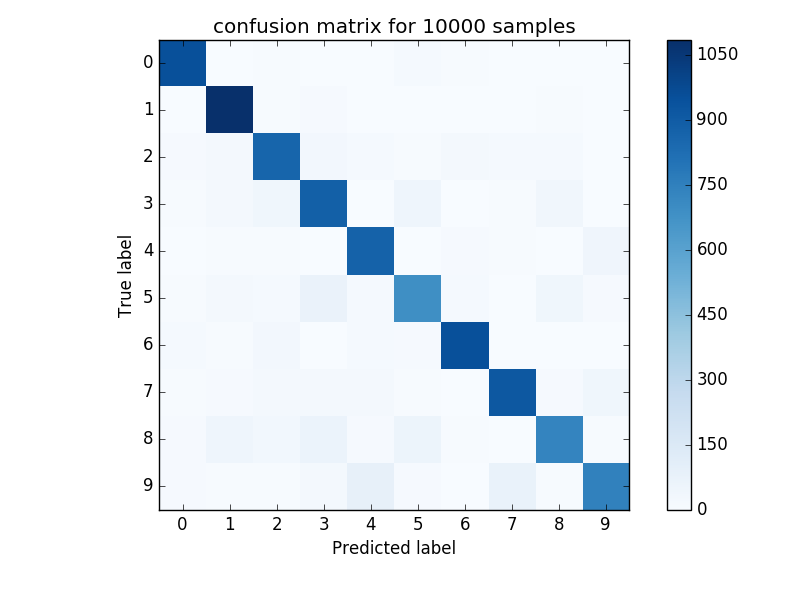
CS189





1. From the confusion matrices, we can see that there are more misclassified samples when the training size is relatively small. This is shown by the presence of more colored squares away from the diagonal for the matrices with small training set sizes. The non-diagonal squares in the confusion matrix also tell us which classes the algorithm is misclassifying and which classes the algorithm is classifying well.





1. Cross-validation is useful because it is one way to reduce overfitting. It improves the ability of the classifier to make new predictions for data it has not already seen. The k-fold cross-validation method helps to mitigate any biases that come from how the data was partitioned for training and validation. Every data point used in the training process gets to be part of the validation set once and part of the training set k – 1 times.

The optimal value for C was found to be 8e-7. This resulted in a validation set error of 9.82%. My Kaggle score is 0.90840.

1. The optimal C value was found to be 90, the validation error rate was 23.57%, and my Kaggle score is 0.76136.

Appendix: Code Used (hw1.py)

import math, random

import numpy as np

from sklearn.svm import SVC

from sklearn.svm import LinearSVC

import scipy.io

import matplotlib.pyplot as plt

from sklearn.metrics import confusion\_matrix

#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

digit\_train\_data = scipy.io.loadmat("data/digit-dataset/train.mat")

digit\_train\_images= digit\_train\_data["train\_images"]

digit\_train\_labels= digit\_train\_data["train\_labels"]

train\_vectors=[]

for i in range(np.shape(digit\_train\_images)[2]):

train\_vectors.append(digit\_train\_images[:,:,i].flatten())

digit\_train\_vectors= np.array(train\_vectors)

def pick\_examples(vectors,labels,N):

# returns a tuple with the chosen vectors, associated labels, and the indices for validation

indices = list(range(np.shape(vectors)[0]))

random.shuffle(indices)

indices = indices[:N+10000]

chosen\_vectors = []

chosen\_labels = []

for i in indices[:N]:

chosen\_vectors.append(vectors[i])

chosen\_labels.append(labels[i])

return np.array(chosen\_vectors), np.array(chosen\_labels), indices[N:]

def plot\_confusion\_matrix(cm, title, cmap=plt.cm.Blues):

global N

plt.imshow(cm, interpolation='nearest', cmap=cmap)

plt.title(title)

tick\_marks = np.arange(10)

plt.xticks(tick\_marks)

plt.yticks(tick\_marks)

plt.colorbar()

plt.tight\_layout()

plt.ylabel('True label')

plt.xlabel('Predicted label')

plt.gcf().subplots\_adjust(bottom=0.15)

#plt.savefig('q2\_'+ str(N) + '.png')

'''

#Uncomment this block to generate the plots for Q1 and Q2

trainsize = [100,200,500,1000,5000,10000]

print(trainsize)

errors = []

for N in trainsize:

vectors, labels, indices = pick\_examples(digit\_train\_vectors,digit\_train\_labels,N)

labels = np.ravel(labels)

classifier = SVC(kernel='linear')

classifier.fit(vectors, labels)

valid\_vectors = []

valid\_labels = []

for i in indices:

valid\_vectors.append(digit\_train\_vectors[i])

valid\_labels.append(digit\_train\_labels[i])

valid\_labels = np.ravel(np.array(valid\_labels))

predictions = classifier.predict(np.array(valid\_vectors))

err = benchmark(predictions, valid\_labels)[0]

errors.append(err)

print(err)

cm = confusion\_matrix(valid\_labels, predictions)

np.set\_printoptions(precision=2)

print('Confusion matrix, without normalization N = ' + str(N))

print(cm)

plot\_confusion\_matrix(cm, 'confusion matrix for ' + str(N) +' samples')

plt.figure()

plt.figure()

plt.plot(trainsize,errors,'-')

plt.title("Digit classification, error rate vs number of training examples",fontsize=14)

plt.xlabel("Training set size",fontsize=14)

plt.ylabel("Error rate",fontsize=14)

plt.show()

'''

digit\_test\_data = scipy.io.loadmat("data/digit-dataset/test.mat")

digit\_test\_images= digit\_test\_data["test\_images"].transpose()

test\_vectors=[]

for i in range(np.shape(digit\_test\_images)[2]):

test\_vectors.append(digit\_test\_images[:,:,i].flatten())

digit\_test\_vectors= np.array(test\_vectors)

def make\_classifer(training\_set, training\_labels, c):

classifier = SVC(kernel='linear',C = c)

classifier.fit(training\_set, training\_labels)

return classifier

def digit\_error(vectors, labels, indices, c):

chosen\_vectors = []

chosen\_labels = []

for i in range(10):

cut = indices[i\*1000:(i+1)\*1000]

cutVectors = []

cutLabels = []

for j in cut:

cutVectors.append(vectors[j])

cutLabels.append(labels[j])

chosen\_vectors.append(np.array(cutVectors))

chosen\_labels.append(np.array(cutLabels))

total\_accuracy = 0

for i in range(10):

validation\_vectors = chosen\_vectors[i]

validation\_labels = np.ravel(chosen\_labels[i])

training\_vectors = np.vstack(chosen\_vectors[:i] + chosen\_vectors[i+1:])

training\_labels = np.ravel(np.vstack(chosen\_labels[:i] + chosen\_labels[i+1:]))

classifier = make\_classifer(training\_vectors, training\_labels, c)

predictions = classifier.predict(validation\_vectors)

total\_accuracy += benchmark(predictions, validation\_labels)[0]

return total\_accuracy/10

'''

#Used to test for optimal C value for digit classifying, found to be C = 8e-7

c = 1e-7

indices = list(range(np.shape(training\_vectors)[0]))

random.shuffle(indices)

indices = indices[:10000]

while c < 1e-6:

print('c = ' + str(c))

print(error(digit\_train\_vectors, digit\_train\_labels, indices,c))

c = c + 0.5e-7

'''

'''

# Used to generate predictions for digit test set

vectors, labels, indices = pick\_examples(digit\_train\_vectors,digit\_train\_labels,8000)

labels = np.ravel(labels)

classifier = SVC(kernel='linear',C = 8e-7)

classifier.fit(vectors, labels)

valid\_vectors = []

valid\_labels = []

for i in indices:

valid\_vectors.append(digit\_train\_vectors[i])

valid\_labels.append(digit\_train\_labels[i])

valid\_labels = np.ravel(np.array(valid\_labels))

predictions = classifier.predict(digit\_test\_vectors)

numbers = (np.arange(10000) + 1)

predictions = np.vstack((numbers,predictions))

#print(benchmark(predictions, valid\_labels)[0])

np.savetxt("digits.csv", predictions.transpose(), delimiter=",",fmt = '%u')

#error(digit\_train\_vectors, digit\_train\_labels, indices,c)

'''

#----------------------------------------------------------------

# Spam section, used LinearSVC instead because of speed

spam\_data = scipy.io.loadmat("data/spam-dataset/spam\_data.mat")

spam\_train\_data= spam\_data["training\_data"]

spam\_train\_labels= np.ravel(spam\_data["training\_labels"])

spam\_test = spam\_data["test\_data"]

'''

# used to find optimal c value for spam classifier, found to be C = 90

def make\_classifer(training\_set, training\_labels, c):

classifier = LinearSVC(C = c)

classifier.fit(training\_set, training\_labels)

return classifier

def spam\_error(vectors, labels, indices, c):

chosen\_vectors = []

chosen\_labels = []

for i in range(12):

cut = indices[i\*431:(i+1)\*431]

cutVectors = []

cutLabels = []

for j in cut:

cutVectors.append(vectors[j])

cutLabels.append(labels[j])

chosen\_vectors.append(np.array(cutVectors))

chosen\_labels.append(np.array(cutLabels))

total\_accuracy = 0

for i in range(12):

validation\_vectors = chosen\_vectors[i]

validation\_labels = np.ravel(chosen\_labels[i])

training\_vectors = np.vstack(chosen\_vectors[:i] + chosen\_vectors[i+1:])

training\_labels = np.ravel(np.vstack(chosen\_labels[:i] + chosen\_labels[i+1:]))

classifier = make\_classifer(training\_vectors, training\_labels, c)

predictions = classifier.predict(validation\_vectors)

total\_accuracy += benchmark(predictions, validation\_labels)[0]

return total\_accuracy/10

minError = float('inf')

c = 80

indices = list(range(np.shape(spam\_train\_data)[0]))

random.shuffle(indices)

while c < 120:

print(c)

currErr = spam\_error(spam\_train\_data,spam\_train\_labels,indices,c)

print(currErr)

if currErr < minError:

minError = currErr

best\_c = c

c = c + 1

print('best c is:' + str(best\_c) + '. With an error of ' + str(minError))

'''

'''

#used to generate the predictions for the spam test data

classifier = LinearSVC(C = 90)

classifier.fit(spam\_train\_data, spam\_train\_labels)

predictions = classifier.predict(spam\_test)

numbers = (np.arange(np.shape(spam\_test)[0]) + 1)

predictions = np.vstack((numbers,predictions))

np.savetxt("spam.csv", predictions.transpose(), delimiter=",",fmt = '%u')

#error(digit\_train\_vectors, digit\_train\_labels, indices,c)

'''