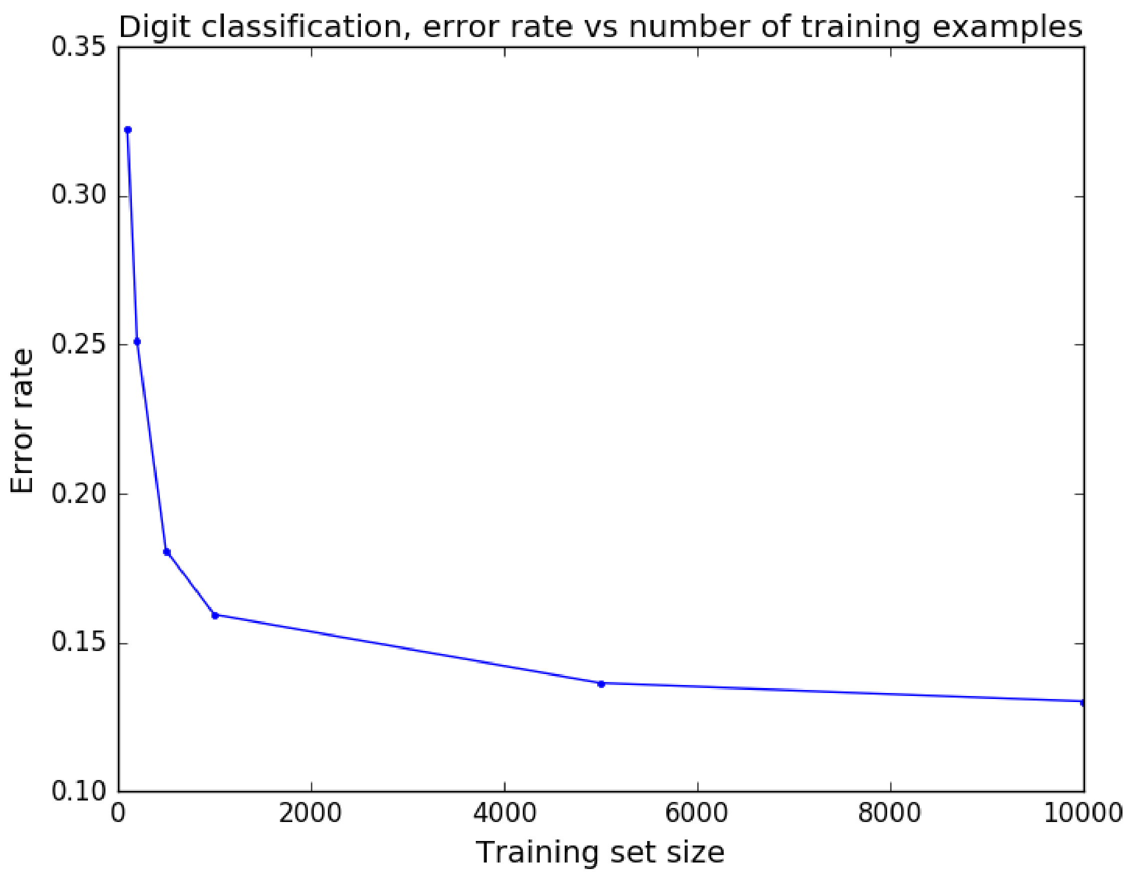
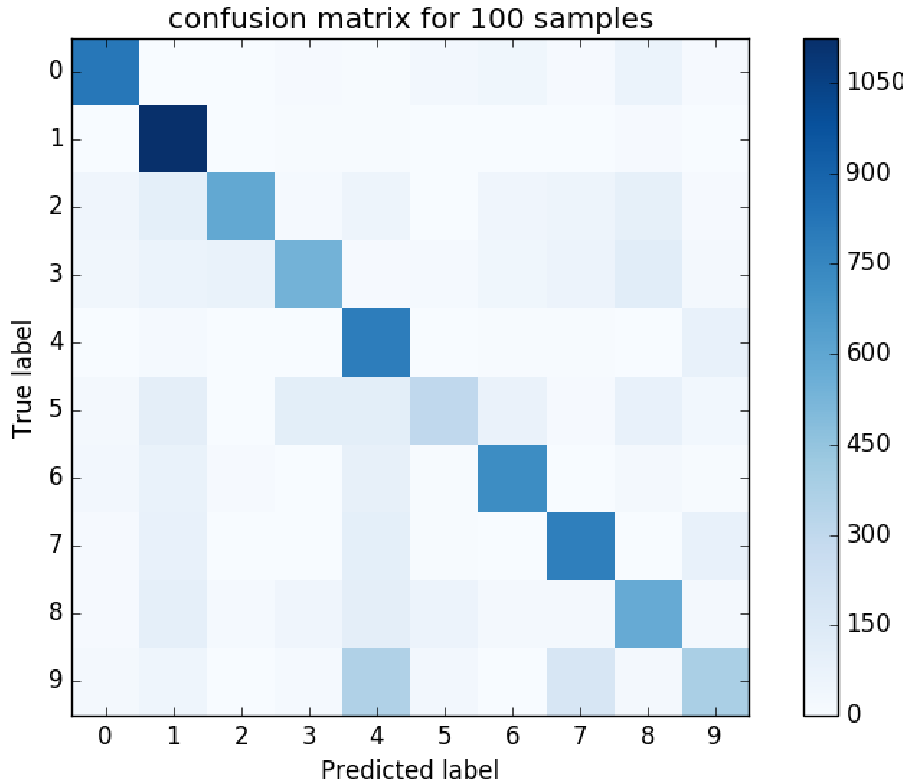
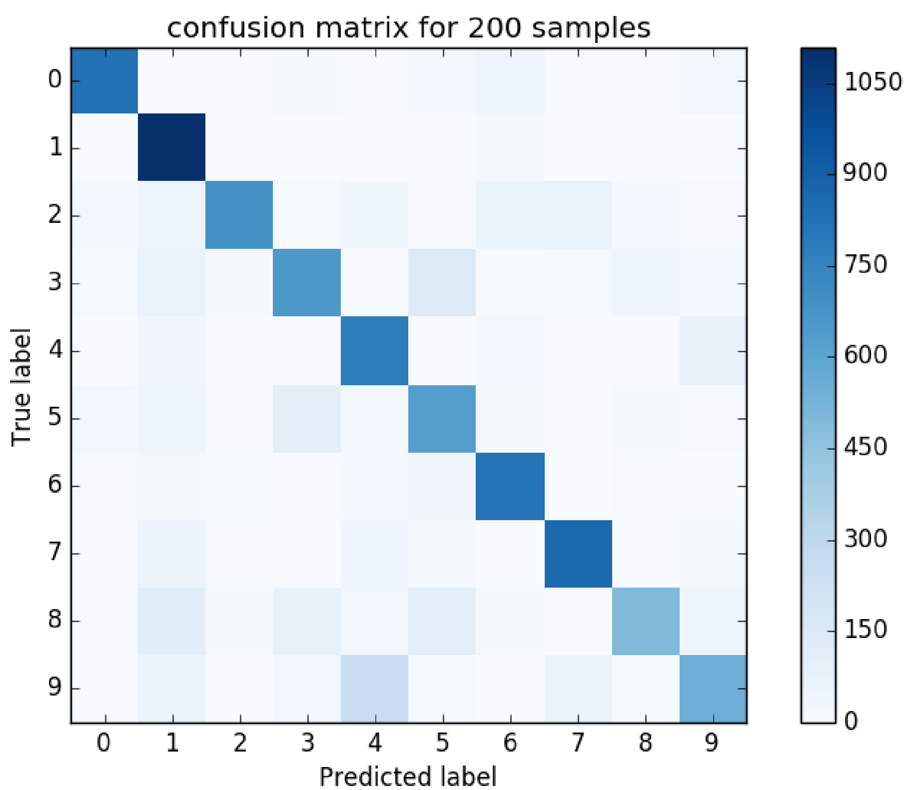
HW1 Write-up
Jesse Li
CS189

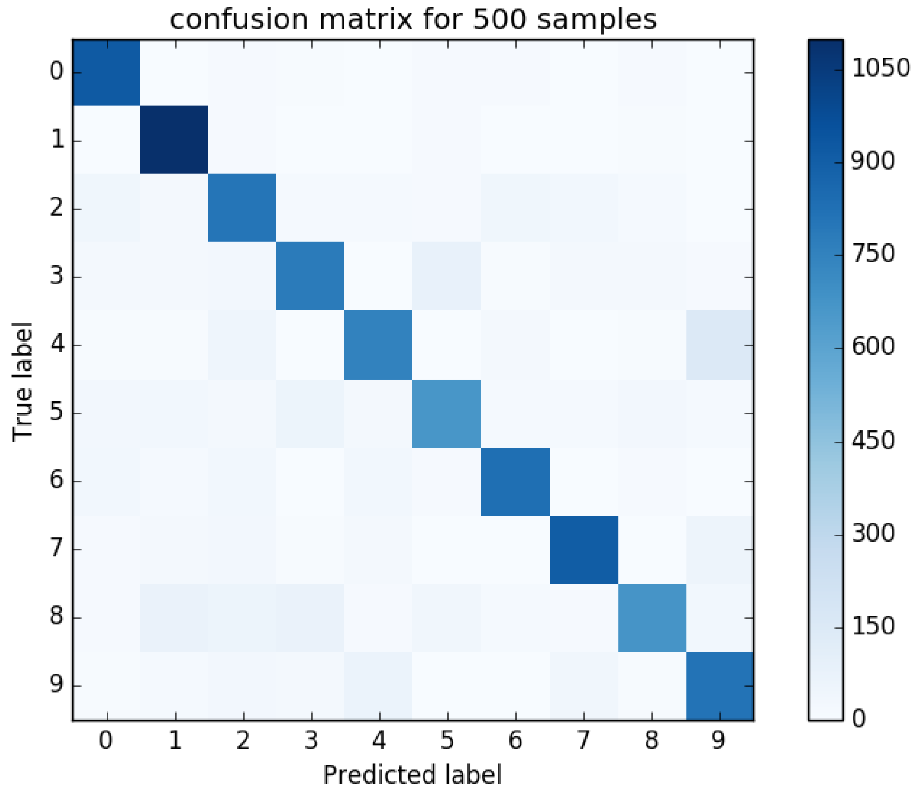
1)

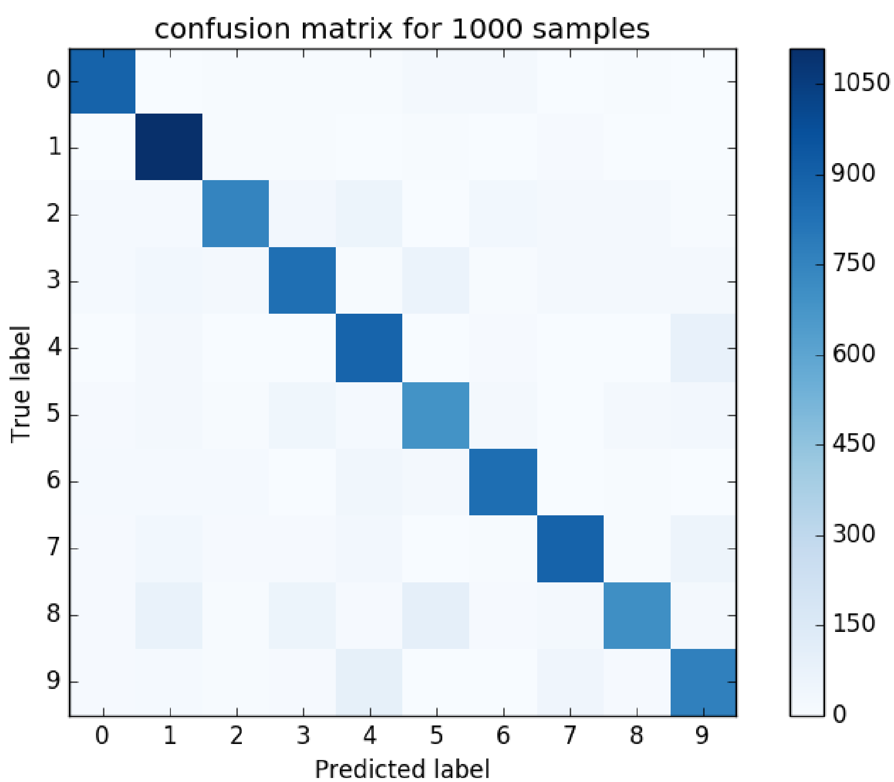


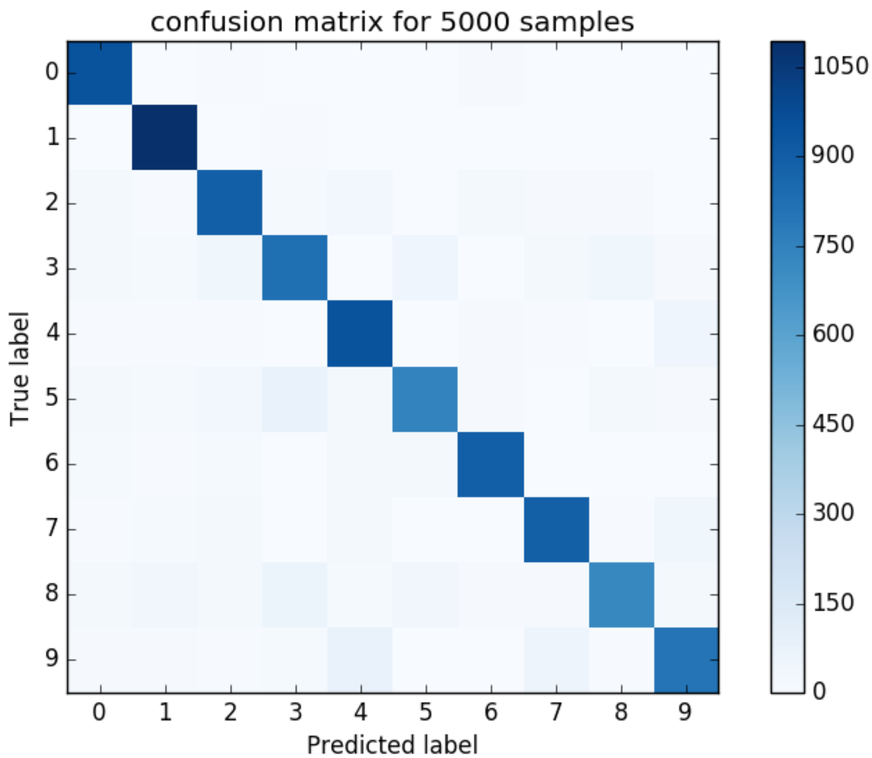
2) From the confusion matrices, we can see that there are more misclassified samples when the training size is relatively small. This is shown by the presence of more colored squares away from the diagonal for the matrices with small training set sizes. The non-diagonal squares in the confusion matrix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

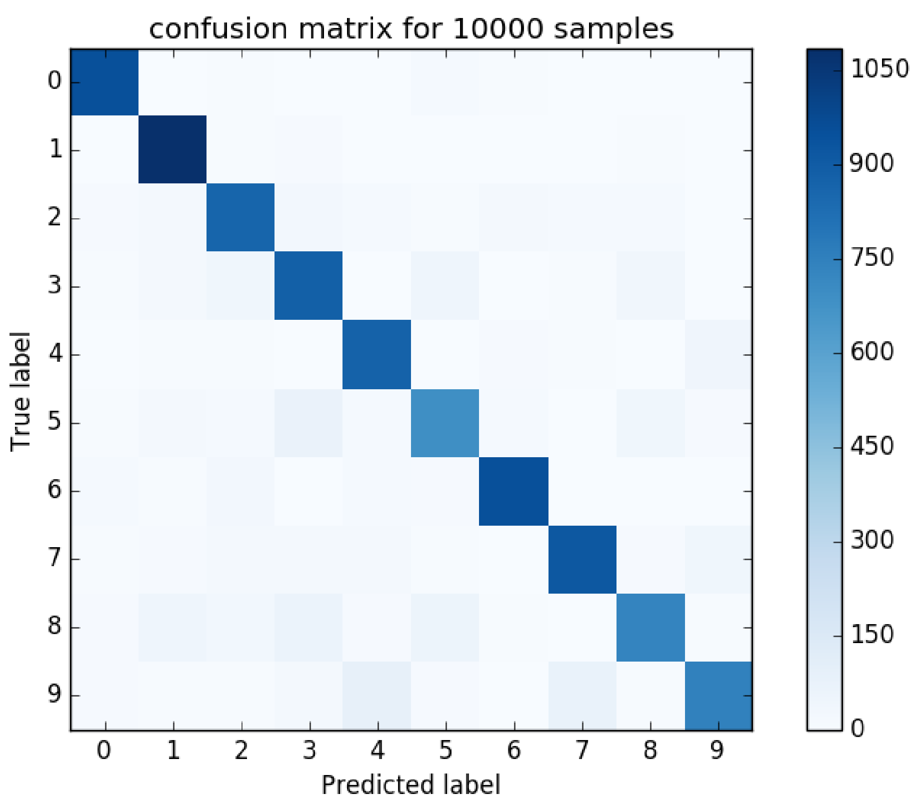

confusion matrix for 100 samples
True label
Predicted label
0
1
2
3
4
5
6
7
8
9
1050
900
750
600
450
300
150
0


confusion matrix for 200 samples
True label
Predicted label
0
1
2
3
4
5
6
7
8
9
1050
900
750
600
450
300
150
0


confusion matrix for 500 samples
True label
Predicted label
0
1
2
3
4
5
6
7
8
9
1050
900
750
600
450
300
150
0


confusion matrix for 1000 samples
True label
Predicted label
0
1
2
3
4
5
6
7
8
9
1050
900
750
600
450
300
150
0


confusion matrix for 5000 samples
True label
Predicted label
0
1
2
3
4
5
6
7
8
9
1050
900
750
600
450
300
150
0


confusion matrix for 10000 samples
True label
Predicted label
0
1
2
3
4
5
6
7
8
9
1050
900
750
600
450
300
150
0

3) 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.

4) 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)
'''
```