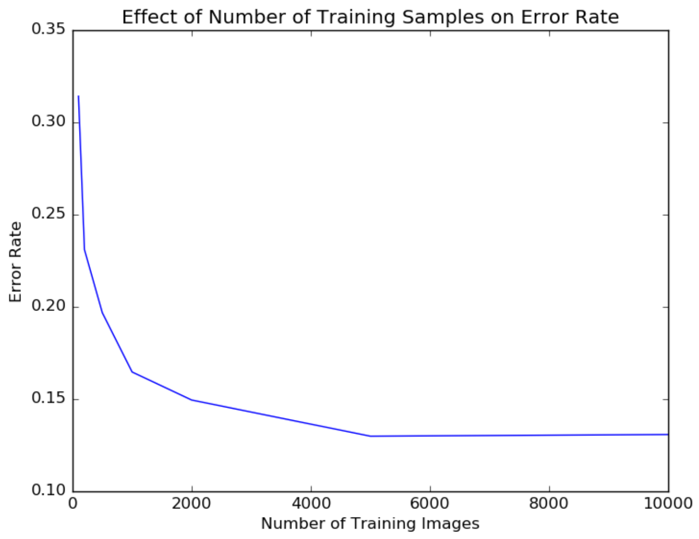
CS 189: Homework 1

Michael Stephen Chen
Kaggle Acct: michaelstchen
SID: 23567341

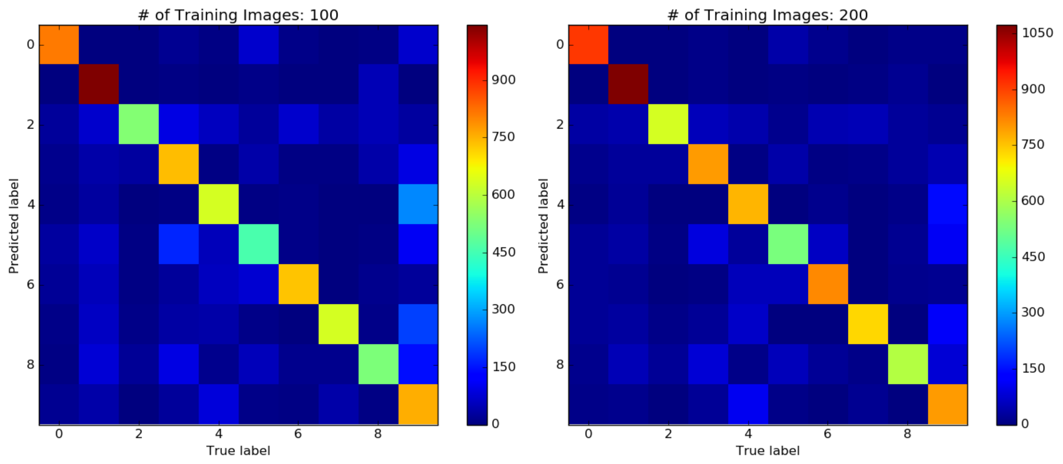
February 10, 2016

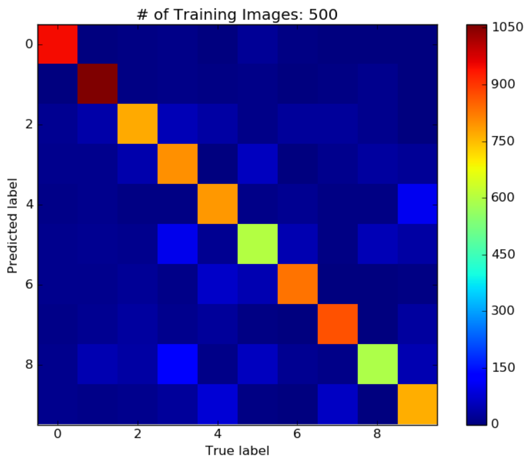
Problem 1

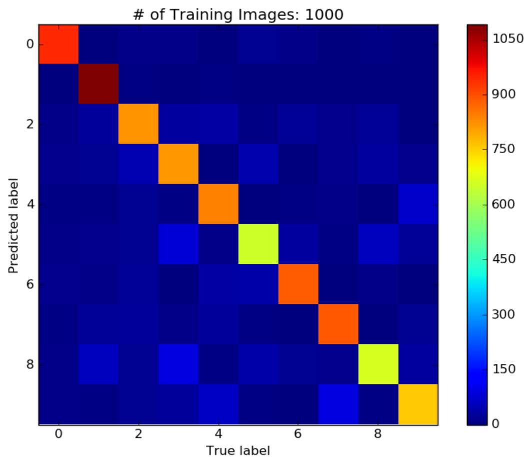
$NumImgs$	$ErrorRate$
100	0.6858
200	0.7690
500	0.8032
1000	0.8353
2000	0.8505
5000	0.8701
10000	0.8692

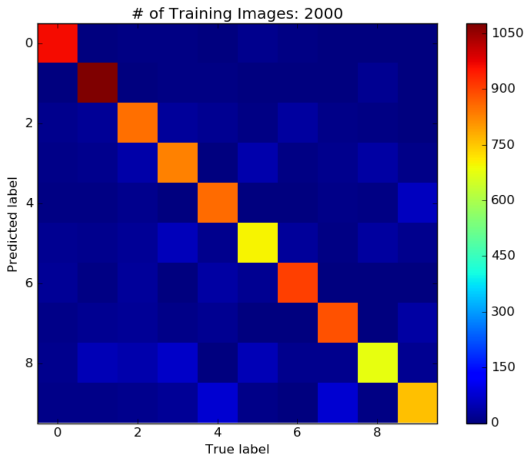


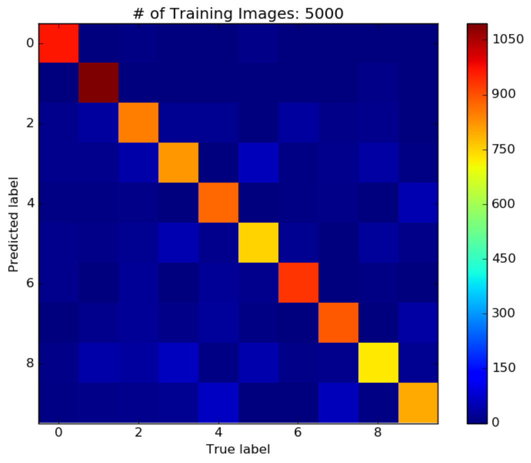
Problem 2

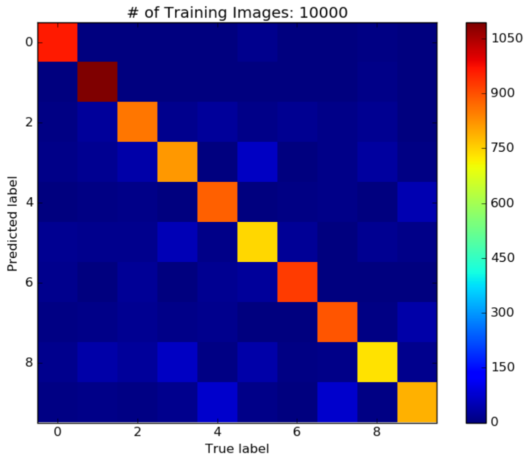












From the confusion matrices we see that as we increase the number of training images the diagonals become ”redder”, indicating that our prediction accuracy is increasing, while the off-diagonals get ”bluer,” indicating that our error rates are going down. Both of these indicate that the accuracy of our classification method generally increases with the number of training samples.

We also see that from 2000 images onward, the change in coloring is just about unnoticeable. This is in accord with our plot from Problem 1 above that shows the error rate decreasing with an almost aymptotic character. So our naive classification method can only get us so far no matter how many training samples we use.

Problem 3

- Cross-validation produces a more stable result with lower variablity because we are taking an average of k trials. It also help to reduce overfitting because we don't use just one set of validation and training sets.
- I ran various C values during my experimental runs but I failed to record those. But I used those runs to narrow down the optimal C value. Note that I am reporting the accuracy and the error rate is just $1 - accuracy$. My optimal C value was $1e - 06$ with a accuracy of 0.9059.

C	*Accuracy*
$1e - 07$	0.8962
1e-06	**0.9059**
$1e - 05$	0.8807

- My best Kaggle score was **0.91380**.
- I did not modify any features.

Problem 4

- I ran 10 different C values. One thing to note is that my validation accuracies are considerably higher than my Kaggle score which may be indicative of overfitting. Also one interesting thing is that using LinearSVC instead SVC jumped my accuracy from 0.7399 to 0.79024. Again I am reporting accuracies (so $1 - errorRate$). My optimal C value was 1 with a validation accuracy of 0.94023.

C	*Accuracy*
$1e - 05$	0.82640
0.0001	0.83897
0.001	0.85444
0.01	0.87940
0.1	0.93404
1	**0.94023**
10	0.92708
100	0.92766
1000	0.84681
10000	0.87447

- My best Kaggle score was **0.79024**.
- I added a couple more word-count features for words that I commonly saw when I was reading over the spam files (e.g. 'viagra', 'microsoft', 'www', 'sexual', etc.). Please see 'featurize.py' for more.

Appendix A: prob1.py

```
import numpy as np
import math
import csv
import matplotlib.pyplot as plt
from scipy.io import loadmat
from sklearn import metrics, svm

#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

train_set = loadmat(file_name="train.mat")
train_data = np.transpose(train_set["train_images"])
train_labels = train_set["train_labels"].reshape(1,-1)[0]

numImages = train_data.shape[0]
height = train_data.shape[1]
width = train_data.shape[2]

train_data_flat = train_data.reshape(numImages, -1)

random_state = np.random.get_state()
np.random.shuffle(train_data)
np.random.set_state(random_state)
np.random.shuffle(train_labels)

train_nums = [100, 200, 500, 1000, 2000, 5000, 10000]
valid_errs = [0, 0, 0, 0, 0, 0, 0]
pred_labels_array = np.zeros((len(train_nums), 10000));
for i in range(0,len(train_nums)):
    print(train_nums[i])
    clf = svm.SVC(kernel='linear')
    clf.fit(train_data_flat[10000:10000+train_nums[i]], \
            train_labels[10000:10000+train_nums[i]])
    pred_labels = clf.predict(train_data_flat[:10000])
    err_rate, indices = benchmark(pred_labels, train_labels[:10000])
    valid_errs[i] = 1-err_rate
    pred_labels_array[i,:] = np.array(pred_labels)

with open('prob1_validation.csv', 'wb') as f:
    writer = csv.writer(f)
    writer.writerow(['# of Images'] + ['% Correct'])
    for i in range(0, len(train_nums)):
        writer.writerow([train_nums[i]] + [valid_errs[i]])



plt.plot(train_nums, [1 - x for x in valid_errs])
plt.title('Effect of Number of Training Samples on Error Rate')
plt.tight_layout()
plt.xlabel('Number of Training Images')
plt.ylabel('Error Rate')
plt.savefig('err_plot')
plt.close()
```

Appendix B: prob2.py

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn import metrics

try:
    train_nums
    pred_labels_array
except NameError:
    execfile('prob1.py')

def write_cm_image(cm, num_im):
    plt.imshow(cm, interpolation='nearest')
    plt.title('# of Training Images: ' + str(num_im))
    plt.tight_layout()
    plt.colorbar()
    plt.xlabel('True label')
    plt.ylabel('Predicted label')
    plt.savefig(str(num_im), bbox_inches='tight')
    plt.close()

for i in range(0, len(train_nums)):
    cm = metrics.confusion_matrix(train_labels[:10000], pred_labels_array[i,:])
    write_cm_image(cm, train_nums[i])
```

Appendix C: prob3.py

```
import numpy as np
import math
import csv
import matplotlib.pyplot as plt
from scipy.io import loadmat
from sklearn import metrics, svm

#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

train_set = loadmat(file_name="train.mat")
test_set = loadmat(file_name="test.mat")

train_data = np.transpose(train_set["train_images"])
train_labels = train_set["train_labels"].reshape(1,-1)[0]
test_data = test_set["test_images"]

num_train = train_data.shape[0]
num_test = test_data.shape[0]

train_data_flat = train_data.reshape(num_train, -1)
test_data_flat = test_data.reshape(num_test, -1)

random_state = np.random.get_state()
np.random.shuffle(train_data)
np.random.set_state(random_state)
np.random.shuffle(train_labels)

c_list = [0.0000001, 0.000001, 0.00001]
xc_err = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
for c in range(0,len(c_list)):
    print("Current C index: " + str(c))
    for k in range(0,10000,1000):
        print("k: " + str(k))
        cv_train = np.vstack((train_data_flat[:k, :], \
                              train_data_flat[k+1000:10000, :]))
        cv_train_labels = np.append(train_labels[:k],\
                                    train_labels[k+1000:10000])
        cv_valid = train_data_flat[k:k+1000, :]
        cv_valid_labels = train_labels[k:k+1000]

        clf = svm.SVC(C=c_list[c], kernel='linear')
        clf.fit(cv_train, cv_train_labels)

        pred_labels = clf.predict(cv_valid)
        err_rate, indices = benchmark(pred_labels, cv_valid_labels)
        c_err[c] = c_err[c] + (1-err_rate)
    print("\n")
    c_err[c] = c_err[c] / 10

with open('prob3_crossvalid.csv', 'wb') as f:
    writer = csv.writer(f)
    writer.writerow(['C Value'] + ['% Valid'])
    for i in range(0, len(c_list)):
        writer.writerow([c_list[i]] + [c_err[i]])

c_optimal = c_list[c_err.index(max(c_err))]

clf = svm.SVC(C=c_optimal, kernel='linear')
clf.fit(train_data_flat[:10000], train_labels[:10000])

pred_test_labels = clf.predict(test_data_flat)

with open('test_labels.csv', 'wb') as f:
    writer = csv.writer(f)
    writer.writerow(['Id'] + ['Category'])
    for i in range(0, len(pred_test_labels)):
        writer.writerow([i+1] + [pred_test_labels[i]])
```

Appendix D: prob4.py

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

#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

execfile('featurize.py')
train_data = np.array(file_dict['training_data'])
train_labels = np.array(file_dict['training_labels'])
test_data = np.array(file_dict['test_data'])

random_state = np.random.get_state()
np.random.shuffle(train_data)
np.random.set_state(random_state)
np.random.shuffle(train_labels)

num_msg = len(train_data)
c_list = [0.00001, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000]
c_err = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
for c in range(0, len(c_list)):
    print("Current C index: " + str(c))
    for k in range(0, num_msg, num_msg/10):
        print("k: " + str(k))
        cv_train = np.vstack((train_data[:k], \
                              train_data[k+num_msg/10:]))
        cv_train_labels = np.append(train_labels[:k],\
                                    train_labels[k+num_msg/10:])
        cv_valid = train_data[k:k+num_msg/10]
        cv_valid_labels = train_labels[k:k+num_msg/10]

        clf = svm.LinearSVC(C=c_list[c])
        clf.fit(cv_train, cv_train_labels)

        pred_labels = clf.predict(cv_valid)
        err_rate, indices = benchmark(pred_labels, cv_valid_labels)
        c_err[c] = c_err[c] + (1-err_rate)
    print("\n")
    c_err[c] = c_err[c] / 10


with open('prob4_crossvalid.csv', 'wb') as f:
    writer = csv.writer(f)
    writer.writerow(['C Value'] + ['% Valid'])
    for i in range(0, len(c_list)):
        writer.writerow([c_list[i]] + [c_err[i]])

c_optimal = c_list[c_err.index(max(c_err))]

clf = svm.LinearSVC(C=c_optimal)
clf.fit(train_data, train_labels)

pred_labels = clf.predict(test_data)


with open('test_labels.csv', 'wb') as f:
    writer = csv.writer(f)
    writer.writerow(['Id'] + ['Category'])
    for i in range(0, len(pred_labels)):
        writer.writerow([i+1] + [pred_labels[i]])
```

Appendix E: featurize.py

```
'''
**************** PLEASE READ ***************

Script that reads in spam and ham messages and converts each training example
into a feature vector

Code intended for UC Berkeley course CS 189/289A: Machine Learning

Requirements:
-scipy ('pip install scipy')

To add your own features, create a function that takes in the raw text and
word frequency dictionary and outputs a int or float. Then add your feature
in the function 'def generate_feature_vector'

The output of your file will be a .mat file. The data will be accessible using
the following keys:
    -'training_data'
    -'training_labels'
    -'test_data'

Please direct any bugs to kevintee@berkeley.edu
'''

from collections import defaultdict
import glob
import re
import scipy.io

NUM_TRAINING_EXAMPLES = 5172
NUM_TEST_EXAMPLES = 5857

BASE_DIR = './'
SPAM_DIR = 'spam/'
HAM_DIR = 'ham/'
TEST_DIR = 'test/'

# ************* Features *************

# Features that look for certain words
def freq_pain_feature(text, freq):
    return float(freq['pain'])

def freq_private_feature(text, freq):
    return float(freq['private'])

def freq_bank_feature(text, freq):
    return float(freq['bank'])

def freq_money_feature(text, freq):
    return float(freq['money'])
```

```
def freq_drug_feature(text, freq):
    return float(freq['drug'])

def freq_spam_feature(text, freq):
    return float(freq['spam'])

def freq_prescription_feature(text, freq):
    return float(freq['prescription'])

def freq_creative_feature(text, freq):
    return float(freq['creative'])

def freq_height_feature(text, freq):
    return float(freq['height'])

def freq_featured_feature(text, freq):
    return float(freq['featured'])

def freq_differ_feature(text, freq):
    return float(freq['differ'])

def freq_width_feature(text, freq):
    return float(freq['width'])

def freq_other_feature(text, freq):
    return float(freq['other'])

def freq_energy_feature(text, freq):
    return float(freq['energy'])

def freq_business_feature(text, freq):
    return float(freq['business'])

def freq_message_feature(text, freq):
    return float(freq['message'])

def freq_volumes_feature(text, freq):
    return float(freq['volumes'])

def freq_revision_feature(text, freq):
    return float(freq['revision'])

def freq_path_feature(text, freq):
    return float(freq['path'])

def freq_meter_feature(text, freq):
    return float(freq['meter'])

def freq_memo_feature(text, freq):
    return float(freq['memo'])

def freq_planning_feature(text, freq):
    return float(freq['planning'])
```

```
def freq_pleased_feature(text, freq):
    return float(freq['pleased'])

def freq_record_feature(text, freq):
    return float(freq['record'])

def freq_out_feature(text, freq):
    return float(freq['out'])

# Features that look for certain characters
def freq_semicolon_feature(text, freq):
    return text.count(';')

def freq_dollar_feature(text, freq):
    return text.count('$')

def freq_sharp_feature(text, freq):
    return text.count('#')

def freq_exclamation_feature(text, freq):
    return text.count('!')

def freq_para_feature(text, freq):
    return text.count('(')

def freq_bracket_feature(text, freq):
    return text.count('[')

def freq_and_feature(text, freq):
    return text.count('&')

# ---------- Add your own feature methods -----------

def freq_buy_feature(text, freq):
    return text.count('buy')

def freq_cash_feature(text, freq):
    return text.count('cash')

def freq_freetrial_feature(text, freq):
    return text.count('free trial')

def freq_microsoft_feature(text, freq):
    return text.count('microsoft')

def freq_medication_feature(text, freq):
    return text.count('medication')

def freq_viagra_feature(text, freq):
    return text.count('viagra')

def freq_cialis_feature(text, freq):
    return text.count('cialis')
```

```
def freq_sex_feature(text, freq):
    return text.count('sex')

def freq_sexual_feature(text, freq):
    return text.count('sexual')

def freq_penis_feature(text, freq):
    return text.count('penis')

def freq_hyphen_feature(text, freq):
    return text.count('-')

def freq_star_feature(text, freq):
    return text.count('*')

def freq_percent_feature(text, freq):
    return text.count('%')

def freq_www_feature(text, freq):
    return text.count('www')

def freq_100_feature(text, freq):
    return text.count('100')

# Generates a feature vector
def generate_feature_vector(text, freq):
    feature = []
    feature.append(freq_pain_feature(text, freq))
    feature.append(freq_private_feature(text, freq))
    feature.append(freq_bank_feature(text, freq))
    feature.append(freq_money_feature(text, freq))
    feature.append(freq_drug_feature(text, freq))
    feature.append(freq_spam_feature(text, freq))
    feature.append(freq_prescription_feature(text, freq))
    feature.append(freq_creative_feature(text, freq))
    feature.append(freq_height_feature(text, freq))
    feature.append(freq_featured_feature(text, freq))
    feature.append(freq_differ_feature(text, freq))
    feature.append(freq_width_feature(text, freq))
    feature.append(freq_other_feature(text, freq))
    feature.append(freq_energy_feature(text, freq))
    feature.append(freq_business_feature(text, freq))
    feature.append(freq_message_feature(text, freq))
    feature.append(freq_volumes_feature(text, freq))
    feature.append(freq_revision_feature(text, freq))
    feature.append(freq_path_feature(text, freq))
    feature.append(freq_meter_feature(text, freq))
    feature.append(freq_memo_feature(text, freq))
    feature.append(freq_planning_feature(text, freq))
    feature.append(freq_pleased_feature(text, freq))
    feature.append(freq_record_feature(text, freq))
    feature.append(freq_out_feature(text, freq))
    feature.append(freq_semicolon_feature(text, freq))
    feature.append(freq_dollar_feature(text, freq))
    feature.append(freq_sharp_feature(text, freq))
    feature.append(freq_exclamation_feature(text, freq))
    feature.append(freq_para_feature(text, freq))
    feature.append(freq_bracket_feature(text, freq))
    feature.append(freq_and_feature(text, freq))

    # --------- Add your own features here ---------
    # Make sure type is int or float

    feature.append(freq_buy_feature(text, freq))
    feature.append(freq_cash_feature(text, freq))
    feature.append(freq_freetrial_feature(text, freq))
    feature.append(freq_microsoft_feature(text, freq))
    feature.append(freq_medication_feature(text, freq))
    feature.append(freq_viagra_feature(text, freq))
    feature.append(freq_cialis_feature(text, freq))
    feature.append(freq_sex_feature(text, freq))
    feature.append(freq_sexual_feature(text, freq))
    feature.append(freq_penis_feature(text, freq))
    feature.append(freq_hyphen_feature(text, freq))
    feature.append(freq_star_feature(text, freq))
    feature.append(freq_percent_feature(text, freq))
    feature.append(freq_www_feature(text, freq))
    feature.append(freq_100_feature(text, freq))

    return feature

# This method generates a design matrix with a list of filenames
# Each file is a single training example
def generate_design_matrix(filenames):
    design_matrix = []
    for filename in filenames:
        with open(filename) as f:
            text = f.read() # Read in text from file
            text = text.replace('\r\n', ' ') # Remove newline character
            words = re.findall(r'\w+', text)
            word_freq = defaultdict(int) # Frequency of all words
            for word in words:
                word_freq[word] += 1

            # Create a feature vector
            feature_vector = generate_feature_vector(text, word_freq)
            design_matrix.append(feature_vector)
    return design_matrix

# ************** Script starts here **************
# DO NOT MODIFY ANYTHING BELOW

spam_filenames = glob.glob(BASE_DIR + SPAM_DIR + '*.txt')
spam_design_matrix = generate_design_matrix(spam_filenames)
ham_filenames = glob.glob(BASE_DIR + HAM_DIR + '*.txt')
ham_design_matrix = generate_design_matrix(ham_filenames)
# Important: the test_filenames must be in numerical order as that is the
# order we will be evaluating your classifier
```

```
 test_filenames = [BASE_DIR + TEST_DIR + str(x) + '.txt' for x in range(NUM_TEST_EXAMPLES)]
 test_design_matrix = generate_design_matrix(test_filenames)

X = spam_design_matrix + ham_design_matrix
Y = [1]*len(spam_design_matrix) + [0]*len(ham_design_matrix)

 file_dict = {}
 file_dict['training_data'] = X
 file_dict['training_labels'] = Y
 file_dict['test_data'] = test_design_matrix
 scipy.io.savemat('spam_data.mat', file_dict)
```