CS189/CS289A – Spring 2017 — Homework 1

Yaoyang Zhang, SID 3032114788

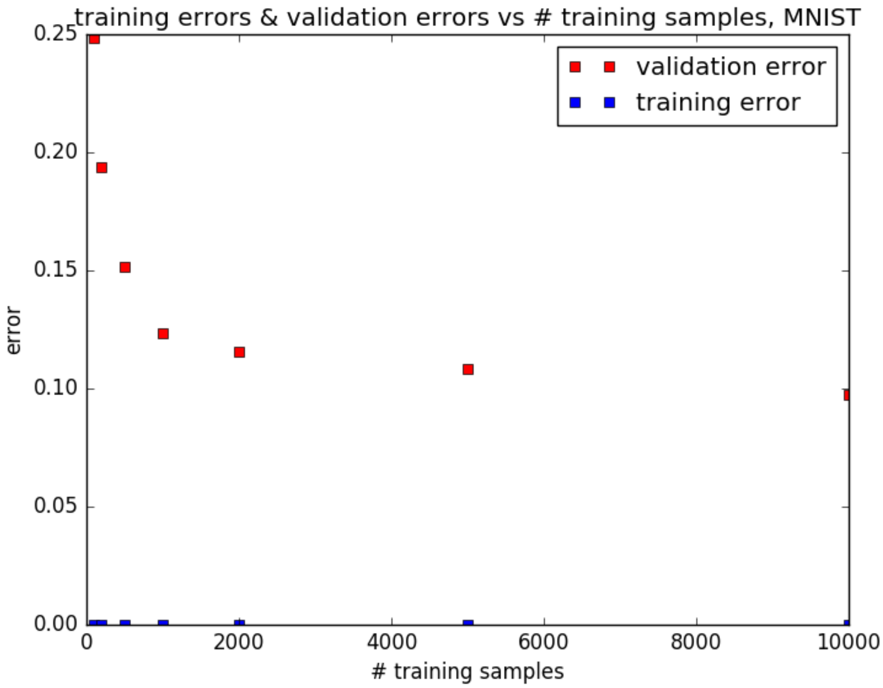
Problem 1

Please refer to code ”p1.py”.

Problem 2

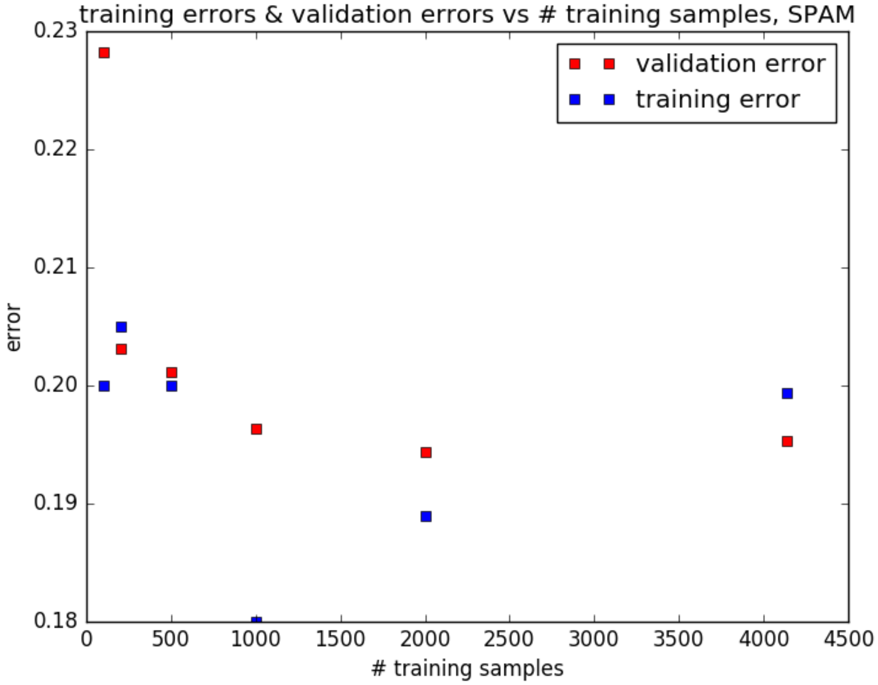
(a) MNIST

The plot of training error and validation error against different number of training samples is shown in the following figure



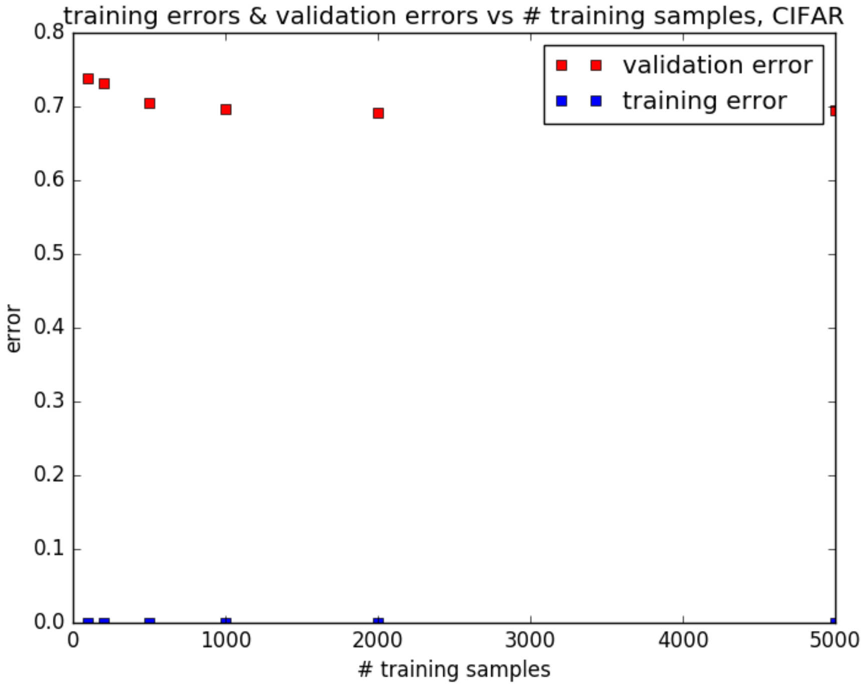
(b) SPAM

The plot of training error and validation error against different number of training samples is shown in the following figure



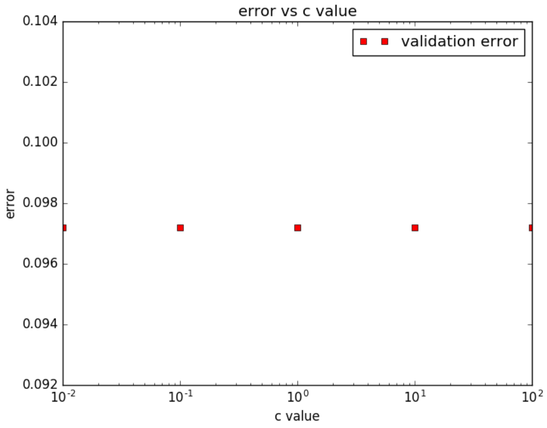
(c) CIFAR

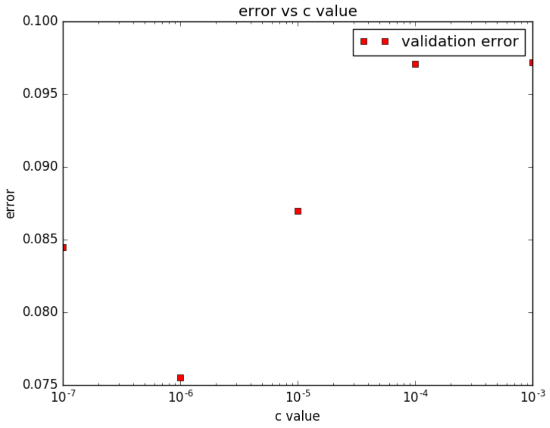
The plot of training error and validation error against different number of training samples is shown in the following figure



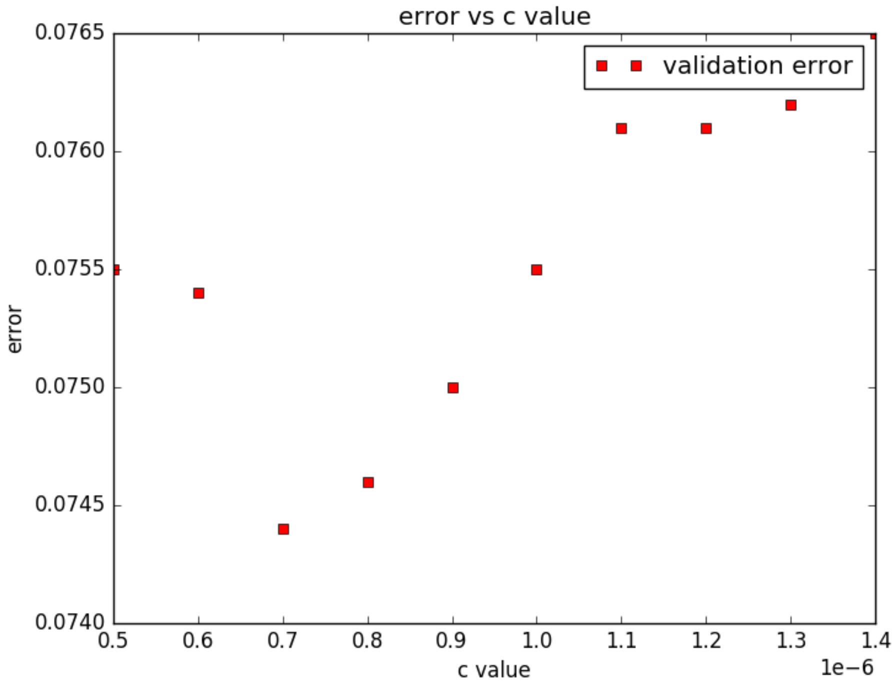
Problem 3

I first tried different C values in different orders of magnitude, and there is a slight change of validation error rate around the order of 10^{-6}.





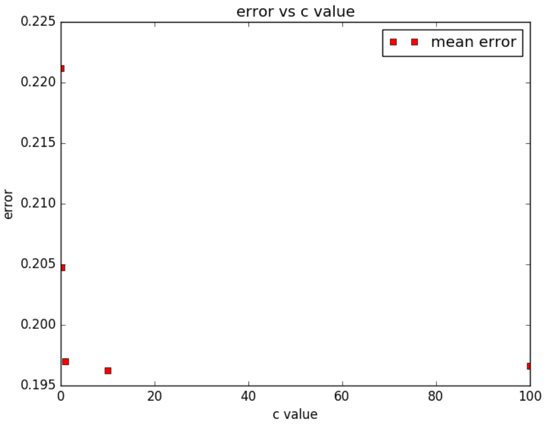
Then I tuned the C value around 10^{-6}.

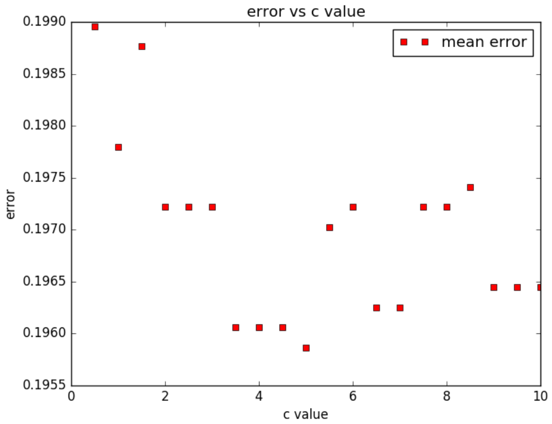


The best C value is around 7×10^{-7}

Problem 4

I first tried different C values in different orders of magnitude and I found that the C value stays almost the same (or increases a little bit) after it exceeds 10. So then I tuned C values in the range between 0.5 and 10, and the corresponding error rates are shown as follows. The error rates are calculated based on the mean values of the 5 cross-validation error rates.





C values	error rate
0.5	0.19895
1	0.19780
1.5	0.19780
2	0.19877
2.5	0.19721
3	0.19721
3.5	0.19721
4	0.19606
4.5	0.19606
5	0.19606
5.5	0.19587
6	0.19703
6.5	0.19721
7	0.19625
7.5	0.19625
8	0.19721
8.5	0.19721
9	0.19741
9.5	0.19644
10	0.19644

The best C value is around 5.5.

Problem 5

Kaggle username: YaoyangZhang

(a) MNIST

I tried different kernels (linear, rbf, polynomial, sigmoid) and found linear kernel has the best performance. Also I trained the model with 20000 samples and use a C value of 10^{-7}. This gives me a score of 0.94160 in the Kaggle leaderboard.

(b) SPAM

I added about 150 additional features to the data. I went through all the SPAM and HAM files to find the 200 most frequently used words in both categories respectively. I merged them together to generate about 150 distinct features. Then I trained a linear SVM with a C value of 10. This gives me a score of 0.88115 in the Kaggle leaderboard.

Appendix: python code

Problem 1

```
import numpy as np
import scipy.io as sio

# mnist data
data = sio.loadmat('hw01_data/mnist/train.mat')
data_mnist = np.array(data['trainX'])
# shuffle
np.random.shuffle(data_mnist)
# validation data
valid_data_mnist = data_mnist[:10000]
valid_label_mnist = valid_data_mnist[:,-1]
valid_data_mnist = valid_data_mnist[:,:-1]
# training data
train_data_mnist = data_mnist[10000:]
train_label_mnist = train_data_mnist[:,-1]
train_data_mnist = train_data_mnist[:,:-1]
# test data
test_data = sio.loadmat('hw01_data/mnist/test.mat')
test_data_mnist = test_data['testX']
# save data
np.savetxt('mnist_train_data.txt', train_data_mnist, fmt='%d')
np.savetxt('mnist_train_label.txt', train_label_mnist, fmt='%d')
np.savetxt('mnist_valid_data.txt', valid_data_mnist, fmt='%d')
np.savetxt('mnist_valid_label.txt', valid_label_mnist, fmt='%d')
np.savetxt('mnist_test_data.txt', test_data_mnist, fmt='%d')


# spam data

data = sio.loadmat('hw01_data/spam/spam_data.mat')
train_data_spam = np.array(data['training_data'])
train_label_spam = np.array(data['training_labels'])
data_spam = np.concatenate((train_data_spam, train_label_spam.T), axis=1)

np.random.shuffle(data_spam)
valid_data_spam = data_spam[:1034]
valid_label_spam = valid_data_spam[:,-1]
valid_data_spam = valid_data_spam[:,:-1]

train_data_spam = data_spam[1034:]
train_label_spam = train_data_spam[:,-1]
train_data_spam = train_data_spam[:,:-1]

test_data_spam = np.array(data['test_data'])
```

```
np.savetxt('spam_train_data.txt', train_data_spam, fmt='%d')
np.savetxt('spam_train_label.txt', train_label_spam, fmt='%d')
np.savetxt('spam_valid_data.txt', valid_data_spam, fmt='%d')
np.savetxt('spam_valid_label.txt', valid_label_spam, fmt='%d')
np.savetxt('spam_test_data.txt', test_data_spam, fmt='%d')




# cifar data
data = sio.loadmat('hw01_data/cifar/train.mat')
data_cifar = np.array(data['trainX'])
np.random.shuffle(data_cifar)
valid_data_cifar = data_cifar[:5000]
valid_label_cifar = valid_data_cifar[:, -1]
valid_data_cifar = valid_data_cifar[:, :-1]

train_data_cifar = data_cifar[5000:]
train_label_cifar = train_data_cifar[:, -1]
train_data_cifar = train_data_cifar[:, :-1]

test_data = sio.loadmat('hw01_data/cifar/test.mat')
test_data_cifar = test_data['testX']
np.savetxt('cifar_train_data.txt', train_data_cifar, fmt='%d')
np.savetxt('cifar_train_label.txt', train_label_cifar, fmt='%d')
np.savetxt('cifar_valid_data.txt', valid_data_cifar, fmt='%d')
np.savetxt('cifar_valid_label.txt', valid_label_cifar, fmt='%d')
np.savetxt('cifar_test_data.txt', test_data_cifar, fmt='%d')
```

Problem 2

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


# minst
train_X = np.loadtxt('mnist_train_data.txt', dtype=int)
train_Y = np.loadtxt('mnist_train_label.txt', dtype=int)
valid_X = np.loadtxt('mnist_valid_data.txt', dtype=int)
valid_Y = np.loadtxt('mnist_valid_label.txt', dtype=int)

print(train_X.shape, train_Y.shape)

num = [100, 200, 500, 1000, 2000, 5000, 10000]
err_valid = []
err_train = []
```

```
for i in range(7):
    clf = SVC(C=1, kernel='linear')
    clf.fit(train_X[:num[i],:], train_Y[:num[i]])
    predict_train_Y = clf.predict(train_X[:num[i],:])
    predict_Y = clf.predict(valid_X)
    err_valid.append(1 - accuracy_score(valid_Y, predict_Y))
    err_train.append(1 - accuracy_score(train_Y[:num[i]], predict_train_Y))
    print(i)
plt.plot(num, err_valid, 'rs', label='validation error' )
plt.plot(num, err_train, 'bs', label='training error')
plt.legend()
plt.xlabel('# training samples')
plt.ylabel('error')
plt.title('training errors & validation errors vs # training samples, MNIST')
plt.savefig('p2-MNIST.png')
plt.show()

# spam

train_X = np.loadtxt('spam_train_data.txt', dtype=int)
train_Y = np.loadtxt('spam_train_label.txt', dtype=int)
valid_X = np.loadtxt('spam_valid_data.txt', dtype=int)
valid_Y = np.loadtxt('spam_valid_label.txt', dtype=int)

train_size = train_X.shape[0]

print(train_size)

num = [100, 200, 500, 1000, 2000, train_size]
err_valid = []
err_train = []
for i in range(6):
    clf = SVC(C=1, kernel='linear')
    clf.fit(train_X[:num[i],:], train_Y[:num[i]])
    predict_train_Y = clf.predict(train_X[:num[i],:])
    predict_Y = clf.predict(valid_X)
    err_valid.append(1 - accuracy_score(valid_Y, predict_Y))
    err_train.append(1 - accuracy_score(train_Y[:num[i]], predict_train_Y))
    print(i)
plt.plot(num, err_valid, 'rs', label='validation error' )
plt.plot(num, err_train, 'bs', label='training error')
plt.legend()
plt.xlabel('# training samples')
plt.ylabel('error')
plt.title('training errors & validation errors vs # training samples, SPAM')
plt.savefig('p2-SPAM.png')
plt.show()
```

```
# cifar
#
train_X = np.loadtxt('cifar_train_data.txt', dtype=int)
train_Y = np.loadtxt('cifar_train_label.txt', dtype=int)
valid_X = np.loadtxt('cifar_valid_data.txt', dtype=int)
valid_Y = np.loadtxt('cifar_valid_label.txt', dtype=int)

train_size = train_X.shape[0]

print(train_size)

num = [100, 200, 500, 1000, 2000, 5000]
err_valid = []
err_train = []
for i in range(6):
    clf = SVC(C=1, kernel='linear')
    clf.fit(train_X[:num[i],:], train_Y[:num[i]])
    predict_train_Y = clf.predict(train_X[:num[i],:])
    predict_Y = clf.predict(valid_X)
    err_valid.append(1 - accuracy_score(valid_Y, predict_Y))
    err_train.append(1 - accuracy_score(train_Y[:num[i]], predict_train_Y))
    print(i)
plt.plot(num, err_valid, 'rs', label='validation error' )
plt.plot(num, err_train, 'bs', label='training error')
plt.legend()
plt.xlabel('# training samples')
plt.ylabel('error')
plt.title('training errors & validation errors vs # training samples, CIFAR')
plt.savefig('p2-CIFAR.png')
plt.show()
```

Problem 3

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


train_X = np.loadtxt('mnist_train_data.txt', dtype=int)
train_Y = np.loadtxt('mnist_train_label.txt', dtype=int)
valid_X = np.loadtxt('mnist_valid_data.txt', dtype=int)
valid_Y = np.loadtxt('mnist_valid_label.txt', dtype=int)
err_valid = []
num = []
print('start...')
for i in range(10):
    c = (i+5)*10 ** (-7)
    clf = SVC(C=c, kernel='linear')
    clf.fit(train_X[:10000, :], train_Y[:10000])
    predict_Y = clf.predict(valid_X)
    err_valid.append(1 - accuracy_score(valid_Y, predict_Y))
    num.append(c)
    print(err_valid[i])

plt.plot(num, err_valid, 'rs', label='validation error' )
plt.ticklabel_format(style='sci', axis='x', scilimits=(0,0))
plt.legend()
plt.xlabel('c value')
plt.ylabel('error')
plt.title('error vs c value')
plt.savefig('p3_3.png')
```

Problem 4

```
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_score
import numpy as np
import matplotlib.pyplot as plt
import scipy.io as sio


data = sio.loadmat('hw01_data/spam/spam_data.mat')
train_data_spam = np.array(data['training_data'])
train_label_spam = np.array(data['training_labels'])
data_spam = np.concatenate((train_data_spam, train_label_spam.T), axis=1)
np.random.shuffle(data_spam)

train_data_spam = data_spam[:,:-1]
train_label_spam = data_spam[:,-1]

num = []
scores = []
for i in range(20):
    c = (i+1)*0.5
    c = num[i]
    clf = SVC(C=c, kernel='linear')
    score = cross_val_score(clf, train_data_spam, train_label_spam, cv=5, scorin
    scores.append(1-np.mean(score))
    print(i)
    num.append(c)
print(scores)
plt.plot(num, scores, 'rs', label='mean error' )
plt.legend()
plt.xlabel('c value')
plt.ylabel('error')
plt.title('error vs c value')
plt.savefig('p4_1.png')
plt.show()
```

Problem 5

```
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_score
import numpy as np
import matplotlib.pyplot as plt
import scipy.io as sio
import csv
from sklearn.ensemble import BaggingClassifier, AdaBoostClassifier

def toCSV(array, filename):
    length = array.shape[0]
    with open(filename, 'w') as f:
        writer = csv.writer(f)
        writer.writerow(['Id','Category'])
        for i in range(length):
            writer.writerow([i,int(array[i])])


# MNIST

data = sio.loadmat('hw01_data/mnist/train.mat')
data_mnist = np.array(data['trainX'])
print(data_mnist.shape)
np.random.shuffle(data_mnist)
data_mnist = data_mnist[:20000,:]
train_data_mnist = data_mnist[:,:-1]
train_label_mnist = data_mnist[:,-1]

test_data = sio.loadmat('hw01_data/mnist/test.mat')
test_data_mnist = test_data['testX']
kernel = ['linear', 'poly', 'rbf', 'sigmoid']
scores = []
print('start...')
for i in range(1):
    clf = SVC(C = 0.000001, kernel='linear')
    score = cross_val_score(clf, train_data_mnist, train_label_mnist, cv=5, scor
    scores.append(score)
    print(np.mean(score))
    clf.fit(train_data_mnist, train_label_mnist)
    predict_Y = clf.predict(test_data_mnist)
    toCSV(predict_Y, 'mnist_'+kernel[i]+'_result.csv')
print(scores)

# SPAM
```

```
data = sio.loadmat('hw01_data/spam/spam_data_2.mat')
train_data_spam = np.array(data['training_data'])
train_label_spam = np.array(data['training_labels'])
test_data_spam = np.array(data['test_data'])
data_spam = np.concatenate((train_data_spam, train_label_spam.T), axis=1)
np.random.shuffle(data_spam)
print(data_spam.shape)
train_data_spam = data_spam[:,:-1]
train_label_spam = data_spam[:,-1]
kernel = ['linear', 'rbf', 'poly', 'sigmoid']
scores = []
print('start...')
for i in range(1):
    clf = SVC(C=10, kernel=kernel[i])
    score = cross_val_score(clf, train_data_spam, train_label_spam, cv=5, scorin
    clf.fit(train_data_spam,train_label_spam)
    predict_Y = clf.predict(test_data_spam)
    toCSV(predict_Y, kernel[i]+'_result.csv')
    scores.append(score)
    print(np.mean(score))
print(scores)
```

Featurize

```
'''
**************** 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 example_feature(text, freq):
    return int('example' in text)

# Generates a feature vector
def generate_feature_vector(text, freq):
    words1 = []
    words2 = []
    with open('ham_dict.txt','rt') as f:
        for line in f:
            line = line.strip()
            words1.append(line)
    with open('spam_dict.txt','rt') as f:
        for line in f:
            line = line.strip()
            words2.append(line)
    words = set(words1) − set(words2)

    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))

    for word in words:
        feature.append(freq[word])
    # ———————— Add your own features here ————————
    # Make sure type is int or float

    return feature
# generate the most frequently used words in a document
def generate_most_freq(filenames, name):
    word_freq = defaultdict(int)
    res = []
    for filename in filenames:
        with open(filename, ”r”, encoding='utf-8', errors='ignore') as f:
            text = f.read() # Read in text from file
            text = text.replace('\r\n', ' ') # Remove newline character
            words = re.findall(r'\w+', text)
             # Frequency of all words
            for word in words:
                word_freq[word] += 1
    for w in sorted(word_freq, key=word_freq.get, reverse=True):
        res.append(w)
    with open(name+'_dict.txt','w') as f:
        for i in range(200):
            f.write(res[i]+'\n')

# 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, "r", encoding='utf-8', errors='ignore') 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')
ham_filenames = glob.glob(BASE_DIR + HAM_DIR + '*.txt')
generate_most_freq(spam_filenames,'spam')
generate_most_freq(ham_filenames,'ham')

spam_design_matrix = generate_design_matrix(spam_filenames)
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_
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_2.mat', file_dict)
```