```
'''
********** 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
import numpy as np

from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer

from tqdm import tqdm

NUM_TRAINING_EXAMPLES = 5172
NUM_TEST_EXAMPLES = 5857

BASE_DIR = 'data/input/'
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'])


```

```python
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'])


```

```python
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 ----------


```

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

# 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

    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, vectorizer=None):
    texts = []
    for filename in tqdm(filenames):
        with open(filename, 'r', encoding='utf-8', errors='ignore') as f:
            try:
                text = f.read()
            except Exception as e:
                continue
            text = text.replace('\r\n', ' ')  # Remove newline character
            texts.append(text)
```

```
240
241     if not vectorizer:
242         vectorizer = TfidfVectorizer(stop_words='english', max_features=2000)
243         vectorizer.fit(texts)
244
245     design_matrix = vectorizer.transform(texts)
246
247     # design_matrix = []
248     # for text in texts:
249     #     words = re.findall(r'\w+', text)
250     #     word_freq = defaultdict(int)  # Frequency of all words
251     #     for word in words:
252     #         word_freq[word] += 1
253
254     #     # Create a feature vector
255     #     feature_vector = vectorizer.transform(text)
256     #     design_matrix.append(feature_vector)
257
258     return design_matrix.todense(), vectorizer
259
260 # ************** Script starts here **************
261 # DO NOT MODIFY ANYTHING BELOW
262
263
264 spam_filenames = glob.glob(BASE_DIR + SPAM_DIR + '*.txt')
265 ham_filenames = glob.glob(BASE_DIR + HAM_DIR + '*.txt')
266 test_filenames = [BASE_DIR + TEST_DIR + str(x) + '.txt' for x in
    range(NUM_TEST_EXAMPLES)]
267 n_spam = len(spam_filenames)
268 n_ham = len(ham_filenames)
269 n_test = len(test_filenames)
270 all_filenames = spam_filenames + ham_filenames + test_filenames
271 all_design_matrix, vectorizer = generate_design_matrix(all_filenames)
272 # Important: the test_filenames must be in numerical order as that is the
273 # order we will be evaluating your classifier
274
275 X = all_design_matrix[:(n_spam + n_ham)]
276 Y = np.array([1] * n_spam + [0] * n_ham).reshape((-1, 1))
277 test_design_matrix = all_design_matrix[(n_spam + n_ham):]
278
279 print(f'X:{X.shape}')
280 print(f'X:{test_design_matrix.shape}')
281
282 file_dict = {}
283 file_dict['training_data'] = X
284 file_dict['training_labels'] = Y
285 file_dict['test_data'] = test_design_matrix
286 scipy.io.savemat(BASE_DIR + 'spam_data.mat', file_dict)
287
```