CS289a - HW1

John Semerdjian

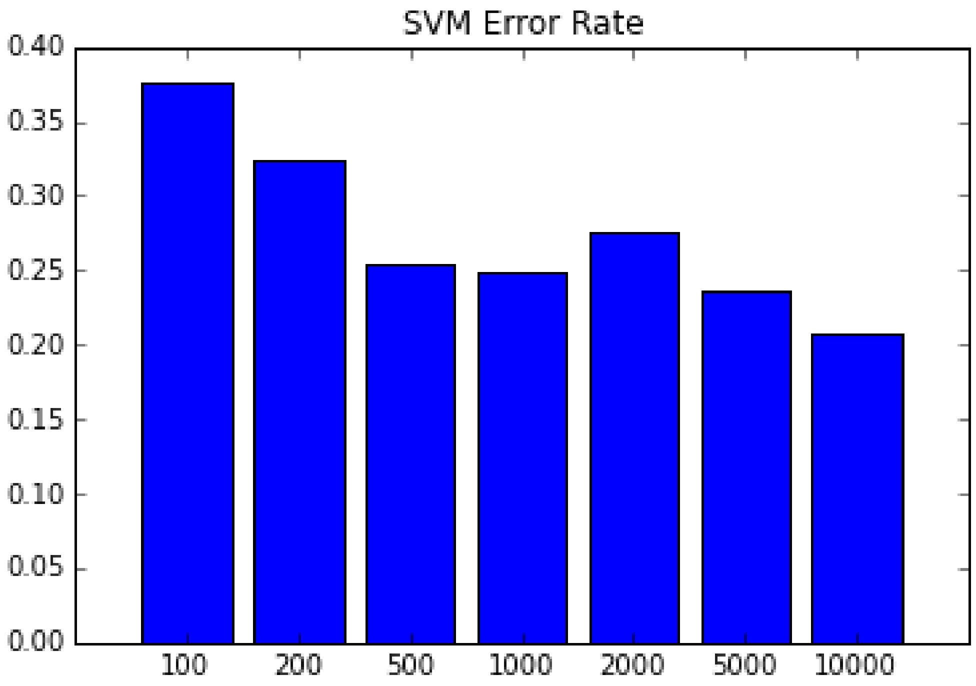
September 12, 2015

Problem 1 - Support Vector Machine

Train a linear SVM using raw pixels as features. Plot the error rate on a validation set versus the number of training examples that you used to train your classifier. The choices of the number of training examples should be 100, 200, 500, 1,000, 2,000, 5,000 and 10,000. Make sure you set aside 10,000 other training points as a validation set. You should expect accuracies between 70% and 90% at this stage.

```
In [1]: from IPython.display import Image
        Image(filename="./images/1_svm_error_rate.png")
```

```
Out[1]:
```



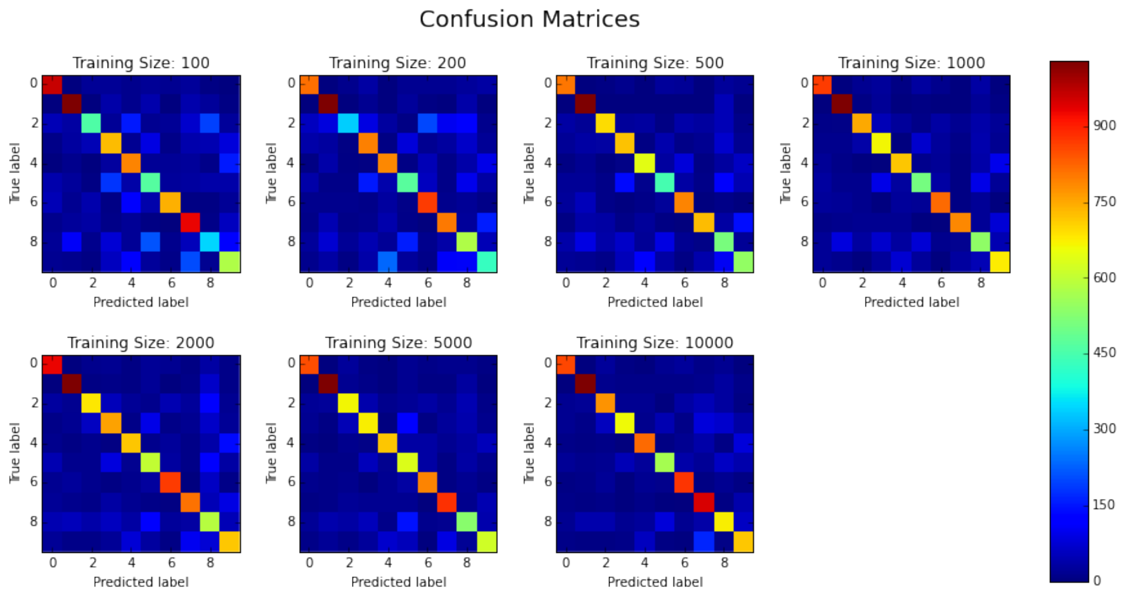
Problem 2 - Confusion Matrix

Create confusion matrices for your experiments in Problem 1. Color code your results and report them. You can use built-in implementations to generate confusion matrices. What insights can you get about the performance of your algorithm from looking at the confusion matrix?

The confusion matrices tell us the types of mistakes that are being committed in our multi-class classification problem. The correct values are highlighted diagonally, while the adjacent blue values mark the errors. For example, in training set size 100, several predicted 9 values were actually 4s. This type of information can help us develop new features that will do a better job distinguishing similar classes.

```
In [2]: Image(filename="./images/2_confusion_matrix.png")
```

```
Out[2]:
```



Problem 3 - Digit - Cross Validation

Explain why cross-validation helps:

Cross-validation helps by training multiple models on different subsets of data in order to help us pick models that will best generalize to new data. It prevents over-fitting and helps us find the best features and hyperparameters. Once we have selected the best features and hyperparameters from the cross validation process, we are able to then train a new model using all the training data without having to set aside a hold-out test set.

Find the optimal value of the parameter ‘C’ using 10-fold cross-validation on the training set with 10,000 examples. Train an SVM with this value of ‘C’ and report the validation error rate.

A **‘C’** value of 0.01 achieved a mean validation error rate of 15.6% with 10-fold cross-validation using LinearSVC. My Kaggle submission used a 3rd degree polynomial kernel SVC achieved a validation error rate of 2.64% with C=1000. The cross-validation process took a long time so the output has been omitted below. See appendix for full cross-validation results.

Problem 4 - Spam - Cross Validation

Train a linear SVM for your spam dataset. Please state your ‘C’ value, your accuracy using cross validation, and (briefly) what features you added, removed, or modified, if any. Adding features is optional.

My best performing **'C'** value was 100 using LinearSVC on 10-fold cross validation, which got an error rate of 19.06%. This same model achieved 23.6% error rate on Kaggle. I added features to count the number of `@`, `unsubscribe`, `sex`, `penis`, `pics`, `drugs`, `money`, `click`, `online`, `pills`, and `viagra`. These features are targeted more towards identifying spam. See appendix for full cross-validation results.

Appendix - Code

```
In [1]: import random
        import time
        import collections
        import glob
        import re
        import scipy.io
        import numpy as np
        from sklearn.svm import LinearSVC, SVC
        from sklearn import preprocessing
        from sklearn.metrics import accuracy_score, confusion_matrix
        import matplotlib.pyplot as plt
        %matplotlib inline

        import sys
        print sys.version
```

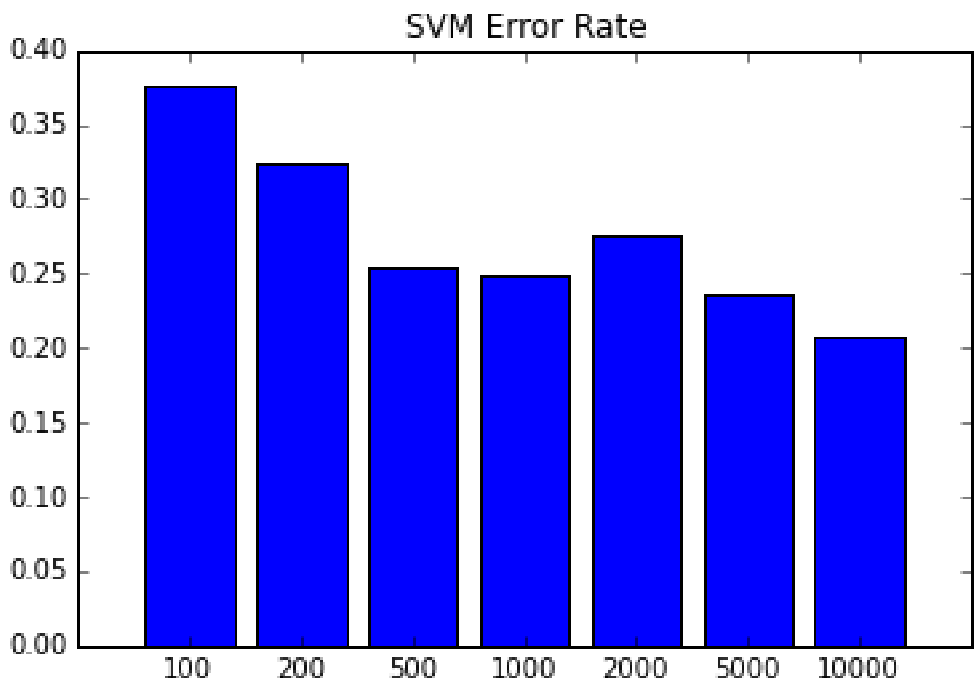
```
2.7.9 (default, Jan  7 2015, 11:49:12)
[GCC 4.2.1 Compatible Apple LLVM 6.0 (clang-600.0.56)]
```

Data Partitioning

```
In [2]: # Set seed
        random.seed(289)

        # Load data
        digit_test = scipy.io.loadmat('./data/digit-dataset/test.mat')
        digit_train = scipy.io.loadmat('./data/digit-dataset/train.mat')

        digit_n = len(digit_train['train_labels'])
        digit_y = digit_train['train_labels']
        digit_X = digit_train['train_images'].T.reshape(digit_n, 28 * 28)
        digit_test_X = digit_test['test_images'].T.reshape(len(digit_test['test_images'].T), 28 * 28)

        # Associate first element in each pixel array with its label
        digit_data = np.concatenate((digit_y, digit_X), axis=1)

In [3]: # Plot some digits
        def plot_digits(pixels, n, plot_n):
            plt.subplot(1, plot_n, 1)
            for x, y in enumerate(random.sample(xrange(n), plot_n)):
                plt.subplot(1, plot_n, x+1)
                plt.axis('off')
                plt.imshow(pixels[:, :, y], cmap='Greys')
            plt.show()

        plot_digits(digit_train['train_images'], n=digit_n, plot_n=5)
```

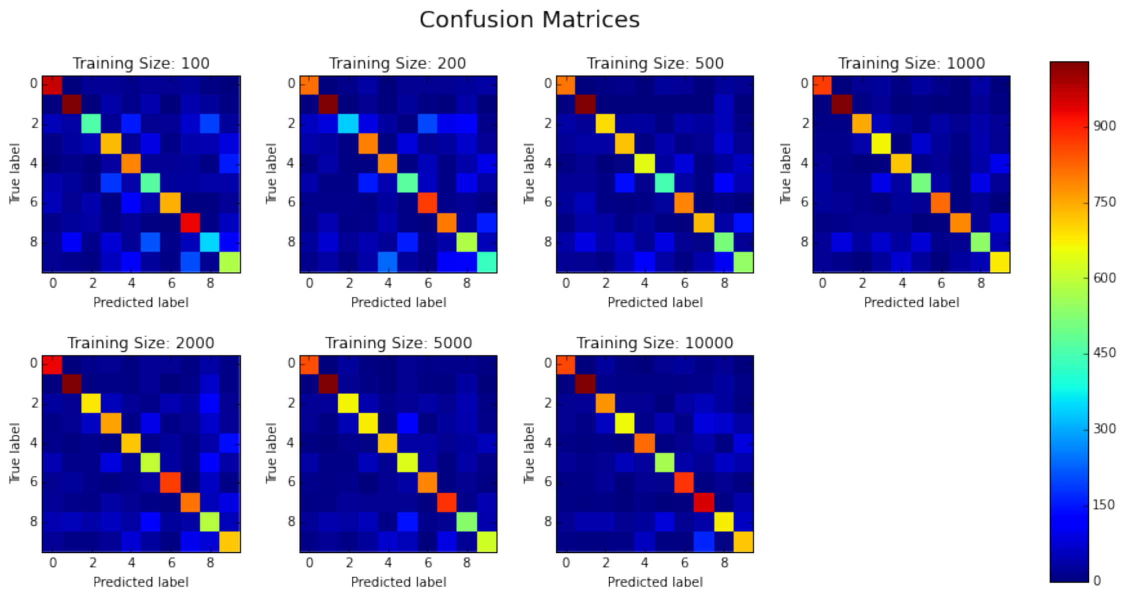
Problem 1 - Support Vector Machine

```
In [4]: def SVM_predict(data, x):
            # randomize and partition data
            np.random.shuffle(data)
            train_X = data[:x, 1:]
            train_y = data[:x, 0 ]
            test_X  = data[x:x+10000, 1:]
            test_y  = data[x:x+10000, 0 ]
            # fit SVM
            clf = LinearSVC().fit(train_X, train_y)
            predicted = clf.predict(test_X)
            return predicted, test_y

        def problem_1(data):
            train_size = [100, 200, 500, 1000, 2000, 5000, 10000]
            scores = []
            for x in train_size:
                predicted, test_y = SVM_predict(data, x)
                scores.append((x, 1 - accuracy_score(test_y, predicted)))
            return scores

        def problem_1_plot(scores):
            key = [x for x, y in scores]
            value = [y for x, y in scores]
            plt.bar(range(len(scores)), value, align='center')
            plt.xticks(range(len(scores)), key)
            plt.title('SVM Error Rate')
            plt.show()

        scores_1 = problem_1(digit_data)
        problem_1_plot(scores_1)
```



Problem 2 - Confusion Matrix

```
In [5]: def problem_2(data):
            train_size = [100, 200, 500, 1000, 2000, 5000, 10000]
            matrix_scores = []

            for x in train_size:
                predicted, test_y = SVM_predict(data, x)
                cm = confusion_matrix(test_y, predicted)
                matrix_scores.append((x, cm))

            return matrix_scores

        def problem_2_plot(matrix_scores):
            fig, axes = plt.subplots(nrows=2, ncols=4, figsize=(14, 7))
            axes[-1, -1].axis('off')
            fig.subplots_adjust(hspace=0.25, wspace=.3)

            for data, ax in zip(matrix_scores, axes.flat):
                im = ax.imshow(data[1], interpolation='none')
                ax.set_title('Training Size: {}'.format(data[0]))
                ax.set_xlabel('Predicted label')
                ax.set_ylabel('True label')

            # Make an axis for the colorbar on the right side
            cax = fig.add_axes([0.9, 0.1, 0.03, 0.8])
            fig.subplots_adjust(right=0.87)
            fig.colorbar(im, cax=cax)
            fig.suptitle("Confusion Matrices", fontsize=18)
            plt.show()

        problem_2_plot(problem_2(digit_data))
```



Problem 3 - Digit - Cross Validation

```
In [6]: def problem_3(data, n_samples, folds, C_hyperparam):
            print "Problem 3: Digit Classification Cross Validation"
            np.random.shuffle(data)
            scaler = preprocessing.StandardScaler()
            X = data[:n_samples, 1:]
            X = scaler.fit(X).transform(X)
            y = data[:n_samples, 0 ]
            fold_size = len(y)/folds

            for c in C_hyperparam:
                print "{:<7}{:<6}{:<8}{:<6}".format('C', 'Fold', 'Error', 'Time')
                kfold_errors = []
                for i, x in enumerate(range(0, n_samples, fold_size)):
                    start_time = time.time()
                    all_idx  = set(range(0, n_samples))
                    test_idx = set(range(x, x + fold_size))
                    train_idx = sorted(list(all_idx - test_idx))
                    test_idx  = sorted(list(test_idx))

                    train_fold_X, train_fold_y = X[train_idx], y[train_idx]
                    test_fold_X,  test_fold_y  = X[test_idx],  y[test_idx]

                    clf = LinearSVC(C=c).fit(train_fold_X, train_fold_y)
                    predicted = clf.predict(test_fold_X)
                    error = 1 - accuracy_score(test_fold_y, predicted)
                    kfold_errors.append(error)
                    end_time = time.time()
                    print "{:<7}{:<6}{:<8}{:<6}".\
                        format(c, i, error, round(end_time - start_time, 2))

                mean_error = np.mean(kfold_errors)
                print "Mean error:  {}\n".format(mean_error)


        problem_3(digit_data, n_samples=10000, folds=10, C_hyperparam=[.001, .01, .1, 1])
```

```
Problem 3: Digit Classification Cross Validation
C      Fold  Error   Time
0.001  0     0.161   12.32
0.001  1     0.157   11.15
0.001  2     0.16    14.9
0.001  3     0.186   14.12
0.001  4     0.161   16.71
0.001  5     0.15    11.79
0.001  6     0.173   13.99
0.001  7     0.166   15.41
0.001  8     0.172   16.73
0.001  9     0.162   13.6
Mean error:  0.1648

C      Fold  Error   Time
0.01   0     0.15    33.32
0.01   1     0.156   32.92
0.01   2     0.148   37.28
0.01   3     0.175   34.18
0.01   4     0.148   37.93
0.01   5     0.148   29.31
0.01   6     0.161   32.5
0.01   7     0.167   34.18
0.01   8     0.152   31.99
0.01   9     0.154   32.83
Mean error:  0.1559

C      Fold  Error   Time
0.1    0     0.166   37.75
0.1    1     0.167   37.96
0.1    2     0.169   38.97
0.1    3     0.196   38.38
0.1    4     0.175   39.72
0.1    5     0.164   39.1
0.1    6     0.185   39.39
0.1    7     0.19    38.59
0.1    8     0.174   39.43
0.1    9     0.167   39.31
Mean error:  0.1753

C      Fold  Error   Time
1      0     0.192   48.21
1      1     0.198   46.22
```

```
1      2     0.193   47.45
1      3     0.23    45.73
1      4     0.19    43.34
1      5     0.188   45.88
1      6     0.219   46.05
1      7     0.209   44.59
1      8     0.221   43.85
1      9     0.207   44.61
Mean error:  0.2047

/Users/jsemer/.virtualenvs/cs289/lib/python2.7/site-packages/sklearn/utils/validation.py:332: UserWarni
  "got %s" % (estimator, X.dtype))
```

```
In [7]: def digit_kaggle_submisison(train_data, test_X, n_samples, c):
            # combine test and training data for scaling
            scaler = preprocessing.StandardScaler()
            data = np.append(train_data[:, 1:], test_X, axis=0)
            X = scaler.fit(data).transform(data)

            # re-partition scaled training and test data
            train_X = X[:len(train_data), :]
            test_X  = X[len(train_data):, :]
            train_y = train_data[:, 0]

            clf = SVC(C=c, kernel='poly', degree=3).fit(train_X, train_y)
            predicted = clf.predict(test_X)
            return predicted

        # digit_kaggle_pred = digit_kaggle_submisison(train_data=digit_data, test_X=digit_test_X, n_sam
        # digit_save_data = zip(range(1, len(digit_test_X) + 1), digit_kaggle_pred)
        # np.savetxt("./predictions/digit_predictions_scaled_SVC_poly3.csv",
        #            digit_save_data, fmt="%i", delimiter=",", header="Id,Category", comments="")
```

Problem 4 - Spam - Cross Validation

```
In [8]: NUM_TRAINING_EXAMPLES = 5172
        NUM_TEST_EXAMPLES = 5857

        BASE_DIR = './data/spam-dataset/'
        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('&')

## my added features
def freq_and_feature(text, freq):
    return text.count('@')

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

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

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

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

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

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

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

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

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

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

# --------- 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):
    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):
    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 = collections.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)
```

```
In [9]: def problem_4(data, n_samples, folds, C_hyperparam):
            print "Problem 4: Spam Classification Cross Validation"
            np.random.shuffle(data)
            scaler = preprocessing.StandardScaler()
            X = data[:n_samples, 1:]
            X = scaler.fit(X).transform(X)
            y = data[:n_samples, 0 ]
            fold_size = len(y)/folds

            for c in C_hyperparam:
                print "{:<6}{:<6}{:<8}{:<6}".format('C', 'Fold', 'Error', 'Time')
                kfold_scores = []
                for i, x in enumerate(range(0, n_samples, fold_size)):
                    start_time = time.time()
                    all_idx  = set(range(0, n_samples))
                    test_idx = set(range(x, x + fold_size))
                    train_idx = sorted(list(all_idx - test_idx))
                    test_idx  = sorted(list(test_idx))

                    train_fold_X, train_fold_y = X[train_idx], y[train_idx]
                    test_fold_X,  test_fold_y  = X[test_idx],  y[test_idx]

                    clf = LinearSVC(C=c).fit(train_fold_X, train_fold_y)
                    predicted = clf.predict(test_fold_X)
                    score = 1-accuracy_score(test_fold_y, predicted)
                    kfold_scores.append(score)
                    end_time = time.time()
                    print "{:<6}{:<6}{:<8}{:<6}".\
                        format(c, i, round(np.mean(score),4), round(end_time - start_time, 2))

                mean_score = round(np.mean(kfold_scores), 4)
                mean_error = round(1 - mean_score, 4)
                print "Mean accuracy: {}\nMean error:    {}\n".format(mean_score, mean_error)


        spam_data = np.concatenate((np.array([[y] for y in Y]), np.array(X)), axis=1)
        problem_4(spam_data, n_samples=5000, folds=10, C_hyperparam=[.01, .1, 1, 10, 100])
```

```
Problem 4: Spam Classification Cross Validation
C     Fold  Error   Time
0.01  0     0.208   0.15
0.01  1     0.176   0.15
0.01  2     0.188   0.14
0.01  3     0.184   0.24
0.01  4     0.166   0.22
0.01  5     0.216   0.15
0.01  6     0.22    0.14
0.01  7     0.216   0.14
0.01  8     0.204   0.15
0.01  9     0.178   0.08
Mean accuracy: 0.1956
Mean error:    0.8044

C     Fold  Error   Time
0.1   0     0.204   0.46
0.1   1     0.176   0.42
0.1   2     0.186   0.44
0.1   3     0.184   0.5
0.1   4     0.172   0.48
0.1   5     0.222   0.42
0.1   6     0.216   0.49
0.1   7     0.212   0.5
0.1   8     0.206   0.42
0.1   9     0.178   0.42
Mean accuracy: 0.1956
Mean error:    0.8044

C     Fold  Error   Time
1     0     0.204   0.43
1     1     0.172   0.44
```

```
1     2     0.186   0.56
1     3     0.182   0.57
1     4     0.17    0.49
1     5     0.222   0.45
1     6     0.216   0.41
1     7     0.206   0.47
1     8     0.21    0.49
1     9     0.178   0.49
Mean accuracy: 0.1946
Mean error:    0.8054

C     Fold  Error   Time
10    0     0.208   0.52
10    1     0.17    0.42
10    2     0.19    0.48
10    3     0.186   0.46
10    4     0.172   0.49
10    5     0.22    0.46
10    6     0.21    0.45
10    7     0.202   0.46
10    8     0.204   0.48
10    9     0.172   0.45
Mean accuracy: 0.1934
Mean error:    0.8066

C     Fold  Error   Time
100   0     0.204   0.47
100   1     0.174   0.48
100   2     0.198   0.47
100   3     0.178   0.48
100   4     0.17    0.47
100   5     0.216   0.48
100   6     0.214   0.49
100   7     0.196   0.47
100   8     0.192   0.43
100   9     0.164   0.43
Mean accuracy: 0.1906
Mean error:    0.8094
```

```
In [10]: def spam_kaggle_submisison(train_data, test_X, n_samples, c):
             # combine test and training data for scaling
             scaler = preprocessing.StandardScaler()
             data = np.append(train_data[:, 1:], test_X, axis=0)
             X = scaler.fit(data).transform(data)

             # re-partition scaled training and test data
             train_X = X[:len(train_data), :]
             test_X  = X[len(train_data):, :]
             train_y = train_data[:, 0]

             clf = LinearSVC(C=c).fit(train_X, train_y)
             predicted = clf.predict(test_X)
             return predicted
```

```
# spam_test_X = np.array(test_design_matrix)
# spam_kaggle_pred = spam_kaggle_submisison(train_data=spam_data, test_X=spam_test_X, n_sample
# spam_save_data = zip(range(1, len(spam_test_X) + 1), spam_kaggle_pred)
# np.savetxt("./predictions/spam_predictions_scaled_linearSVC_c100.csv",
#            spam_save_data, fmt="%i", delimiter=",", header="Id,Category", comments="")
```