```python
# Important note: you do not have to modify this file for your homework.

import numpy as np
np.random.seed(123)


def train_and_predict_svm(train_matrix, train_labels, test_matrix, radius):
    """Train an SVM model and predict the resulting labels on a test set.

    Args:
        train_matrix: A numpy array containing the word counts for the train set
        train_labels: A numpy array containing the spam or not spam labels for the train set
        test_matrix: A numpy array containing the word counts for the test set
        radius: The RBF kernel radius to use for the SVM

    Return:
        The predicted labels for each message
    """
    model = svm_train(train_matrix, train_labels, radius)
    return svm_predict(model, test_matrix, radius)


def svm_train(matrix, category, radius):
    state = {}
    M, N = matrix.shape
    Y = 2 * category - 1
    matrix = 1. * (matrix > 0)
    squared = np.sum(matrix * matrix, axis=1)
    gram = matrix.dot(matrix.T)
    K = np.exp(-(squared.reshape((1, -1)) + squared.reshape((-1, 1)) - 2 * gram) / (2 * (radius ** 2)))

    alpha = np.zeros(M)
    alpha_avg = np.zeros(M)
    L = 1. / (64 * M)
    outer_loops = 10

    alpha_avg = 0
    ii = 0
    while ii < outer_loops * M:
        i = int(np.random.rand() * M)
        margin = Y[i] * np.dot(K[i, :], alpha)
        grad = M * L * K[:, i] * alpha[i]
        if margin < 1:
            grad -= Y[i] * K[:, i]
        alpha -= grad / np.sqrt(ii + 1)
        alpha_avg += alpha
        ii += 1

    alpha_avg /= (ii + 1) * M

    state['alpha'] = alpha
    state['alpha_avg'] = alpha_avg
    state['Xtrain'] = matrix
    state['Sqtrain'] = squared
    return state


def svm_predict(state, matrix, radius):
    M, N = matrix.shape

    Xtrain = state['Xtrain']
    Sqtrain = state['Sqtrain']
    matrix = 1. * (matrix > 0)
    squared = np.sum(matrix * matrix, axis=1)
    gram = matrix.dot(Xtrain.T)
    K = np.exp(-(squared.reshape((-1, 1)) + Sqtrain.reshape((1, -1)) - 2 * gram) / (2 * (radius ** 2)))
    alpha_avg = state['alpha_avg']
    preds = K.dot(alpha_avg)
    output = (1 + np.sign(preds)) // 2

    return output
```