```python
import collections

import numpy as np

import util
import svm


def get_words(message):
    """Get the normalized list of words from a message string.

    This function should split a message into words, normalize them, and return
    the resulting list. For splitting, you should split on spaces. For normalization,
    you should convert everything to lowercase.

    Args:
        message: A string containing an SMS message

    Returns:
       The list of normalized words from the message.
    """

    # *** START CODE HERE ***
    return [word.lower() for word in message.split(' ')]
    # *** END CODE HERE ***


def create_dictionary(messages):
    """Create a dictionary mapping words to integer indices.

    This function should create a dictionary of word to indices using the provided
    training messages. Use get_words to process each message.

    Rare words are often not useful for modeling. Please only add words to the dictionary
    if they occur in at least five messages.

    Args:
        messages: A list of strings containing SMS messages

    Returns:
        A python dict mapping words to integers.
    """

    # *** START CODE HERE ***
    word_counts = collections.defaultdict(int)

    for message in messages:
        for word in set(get_words(message)):
            word_counts[word] += 1

    resulting_dictionary = {}

    for word, count in word_counts.items():
        if count >= 5:
            next_index = len(resulting_dictionary)
            resulting_dictionary[word] = next_index

    print("RESULTING DICTIONARY: ", resulting_dictionary)
    return resulting_dictionary
    # *** END CODE HERE ***


def transform_text(messages, word_dictionary):
    """Transform a list of text messages into a numpy array for further processing.

    This function should create a numpy array that contains the number of times each word
    of the vocabulary appears in each message.
    Each row in the resulting array should correspond to each message
    and each column should correspond to a word of the vocabulary.

    Use the provided word dictionary to map words to column indices. Ignore words that
    are not present in the dictionary. Use get_words to get the words for a message.
```

```
    Args:
        messages: A list of strings where each string is an SMS message.
        word_dictionary: A python dict mapping words to integers.

    Returns:
        A numpy array marking the words present in each message.
        Where the component (i,j) is the number of occurrences of the
        j-th vocabulary word in the i-th message.
    """
    # *** START CODE HERE ***
    result = np.zeros((len(messages), len(word_dictionary)))

    for i, message in enumerate(messages):
        for word in get_words(message):
            if word in word_dictionary:
                result[i, word_dictionary[word]] += 1

    return result
    # *** END CODE HERE ***


def fit_naive_bayes_model(matrix, labels):
    """Fit a naive bayes model.

    This function should fit a Naive Bayes model given a training matrix and labels.

    The function should return the state of that model.

    Feel free to use whatever datatype you wish for the state of the model.

    Args:
        matrix: A numpy array containing word counts for the training data
        labels: The binary (0 or 1) labels for that training data

    Returns: The trained model
    """

    # *** START CODE HERE ***
    model = {}

    phi = 1. * sum(labels) / len(labels)
    model['logphi_0'] = np.log(1.-phi)
    model['logphi_1'] = np.log(phi)
    theta_0 = (matrix[labels == 0]).sum(axis=0) + 1
    theta_1 = (matrix[labels == 1]).sum(axis=0) + 1
    theta_0 /= theta_0.sum()
    theta_1 /= theta_1.sum()
    model['logtheta_0'] = np.log(theta_0)
    model['logtheta_1'] = np.log(theta_1)

    return model
    # *** END CODE HERE ***


def predict_from_naive_bayes_model(model, matrix):
    """Use a Naive Bayes model to compute predictions for a target matrix.

    This function should be able to predict on the models that fit_naive_bayes_model
    outputs.

    Args:
        model: A trained model from fit_naive_bayes_model
        matrix: A numpy array containing word counts

    Returns: A numpy array containg the predictions from the model
    """
    # *** START CODE HERE ***
    output = np.zeros(matrix.shape[0])

    logphi_0 = model['logphi_0']
    logphi_1 = model['logphi_1']
```

```
    logtheta_0 = model['logtheta_0']
    logtheta_1 = model['logtheta_1']
    logprobs_0 = (matrix * logtheta_0).sum(axis=1) + logphi_0
    logprobs_1 = (matrix * logtheta_1).sum(axis=1) + logphi_1

    output = (logprobs_1 > logprobs_0).astype(int)
    return output
    # *** END CODE HERE ***


def get_top_five_naive_bayes_words(model, dictionary):
    """Compute the top five words that are most indicative of the spam (i.e positive) class.

    Ues the metric given in part-c as a measure of how indicative a word is.
    Return the words in sorted form, with the most indicative word first.

    Args:
        model: The Naive Bayes model returned from fit_naive_bayes_model
        dictionary: A mapping of word to integer ids

    Returns: A list of the top five most indicative words in sorted order with the most indicative first
    """
    # *** START CODE HERE ***
    ids = np.argsort(model['logtheta_0'] - model['logtheta_1'])[:5]

    reverse_dictionary = {i: word for word, i in dictionary.items()}

    return [reverse_dictionary[i] for i in ids]
    # *** END CODE HERE ***


def compute_best_svm_radius(train_matrix, train_labels, val_matrix, val_labels, radius_to_consider):
    """Compute the optimal SVM radius using the provided training and evaluation datasets.

    You should only consider radius values within the radius_to_consider list.
    You should use accuracy as a metric for comparing the different radius values.

    Args:
        train_matrix: The word counts for the training data
        train_labels: The spma or not spam labels for the training data
        val_matrix: The word counts for the validation data
        val_labels: The spam or not spam labels for the validation data
        radius_to_consider: The radius values to consider

    Returns:
        The best radius which maximizes SVM accuracy.
    """
    # *** START CODE HERE ***
    best = None

    for radius in radius_to_consider:
        svm_predictions = svm.train_and_predict_svm(train_matrix, train_labels, val_matrix, radius)
        svm_accuracy = np.mean(svm_predictions == val_labels)

        if best is None:
            best = (svm_accuracy, radius)
        else:
            best = max(best, (svm_accuracy, radius))

    return best[1]
    # *** END CODE HERE ***


def main():
    train_messages, train_labels = util.load_spam_dataset('spam_train.tsv')
    val_messages, val_labels = util.load_spam_dataset('spam_val.tsv')
    test_messages, test_labels = util.load_spam_dataset('spam_test.tsv')

    dictionary = create_dictionary(train_messages)

    print('Size of dictionary: ', len(dictionary))
```

```
    util.write_json('spam_dictionary', dictionary)

    train_matrix = transform_text(train_messages, dictionary)

    np.savetxt('spam_sample_train_matrix', train_matrix[:100,:])

    val_matrix = transform_text(val_messages, dictionary)
    test_matrix = transform_text(test_messages, dictionary)

    naive_bayes_model = fit_naive_bayes_model(train_matrix, train_labels)

    naive_bayes_predictions = predict_from_naive_bayes_model(naive_bayes_model, test_matrix)

    np.savetxt('spam_naive_bayes_predictions', naive_bayes_predictions)

    naive_bayes_accuracy = np.mean(naive_bayes_predictions == test_labels)

    print('Naive Bayes had an accuracy of {} on the testing set'.format(naive_bayes_accuracy))

    top_5_words = get_top_five_naive_bayes_words(naive_bayes_model, dictionary)

    print('The top 5 indicative words for Naive Bayes are: ', top_5_words)

    util.write_json('spam_top_indicative_words', top_5_words)

    optimal_radius = compute_best_svm_radius(train_matrix, train_labels, val_matrix, val_labels, [0.01, 0.1, 1, 10])

    util.write_json('spam_optimal_radius', optimal_radius)

    print('The optimal SVM radius was {}'.format(optimal_radius))

    svm_predictions = svm.train_and_predict_svm(train_matrix, train_labels, test_matrix, optimal_radius)

    svm_accuracy = np.mean(svm_predictions == test_labels)

    print('The SVM model had an accuracy of {} on the testing set'.format(svm_accuracy, optimal_radius))


if __name__ == "__main__":
    main()
```