```python
import csv

import matplotlib.pyplot as plt
import numpy as np
import json


def add_intercept_fn(x):
    """Add intercept to matrix x.

    Args:
        x: 2D NumPy array.

    Returns:
        New matrix same as x with 1's in the 0th column.
    """
    new_x = np.zeros((x.shape[0], x.shape[1] + 1), dtype=x.dtype)
    new_x[:, 0] = 1
    new_x[:, 1:] = x

    return new_x

def load_csv(csv_path, label_col='y', add_intercept=False):
    """Load dataset from a CSV file.

    Args:
         csv_path: Path to CSV file containing dataset.
         label_col: Name of column to use as labels (should be 'y' or 'l').
         add_intercept: Add an intercept entry to x-values.

    Returns:
        xs: Numpy array of x-values (inputs).
        ys: Numpy array of y-values (labels).
    """

    # Load headers
    with open(csv_path, 'r', newline='') as csv_fh:
        headers = csv_fh.readline().strip().split(',')

    # Load features and labels
    x_cols = [i for i in range(len(headers)) if headers[i].startswith('x')]
    l_cols = [i for i in range(len(headers)) if headers[i] == label_col]
    inputs = np.loadtxt(csv_path, delimiter=',', skiprows=1, usecols=x_cols)
    labels = np.loadtxt(csv_path, delimiter=',', skiprows=1, usecols=l_cols)

    if inputs.ndim == 1:
        inputs = np.expand_dims(inputs, -1)

    if add_intercept:
        inputs = add_intercept_fn(inputs)

    return inputs, labels

def load_spam_dataset(tsv_path):
    """Load the spam dataset from a TSV file

    Args:
         csv_path: Path to TSV file containing dataset.

    Returns:
        messages: A list of string values containing the text of each message.
        labels: The binary labels (0 or 1) for each message. A 1 indicates spam.
    """

    messages = []
    labels = []

    with open(tsv_path, 'r', newline='', encoding='utf8') as tsv_file:
        reader = csv.reader(tsv_file, delimiter='\t')

        for label, message in reader:
            messages.append(message)
```

```
            labels.append(1 if label == 'spam' else 0)

    return messages, np.array(labels)

def plot(x, y, theta, save_path, correction=1.0):
    """Plot dataset and fitted logistic regression parameters.

    Args:
        x: Matrix of training examples, one per row.
        y: Vector of labels in {0, 1}.
        theta: Vector of parameters for logistic regression model.
        save_path: Path to save the plot.
        correction: Correction factor to apply (Problem 2(e) only).
    """
    # Plot dataset
    plt.figure()
    plt.plot(x[y == 1, -2], x[y == 1, -1], 'bx', linewidth=2)
    plt.plot(x[y == 0, -2], x[y == 0, -1], 'go', linewidth=2)

    # Plot decision boundary (found by solving for theta^T x = 0)
    x1 = np.arange(min(x[:, -2]), max(x[:, -2]), 0.01)
    x2 = -(theta[0] / theta[2] * correction + theta[1] / theta[2] * x1)
    plt.plot(x1, x2, c='red', linewidth=2)

    # Add labels and save to disk
    plt.xlabel('x1')
    plt.ylabel('x2')
    plt.savefig(save_path)


def plot_contour(predict_fn):
    """Plot a contour given the provided prediction function"""
    x, y = np.meshgrid(np.linspace(-10, 10, num=20), np.linspace(-10, 10, num=20))
    z = np.zeros(x.shape)

    for i in range(x.shape[0]):
        for j in range(y.shape[1]):
            z[i, j] = predict_fn([x[i, j], y[i, j]])

    plt.contourf(x, y, z, levels=[-float('inf'), 0, float('inf')], colors=['orange', 'cyan'])

def plot_points(x, y):
    """Plot some points where x are the coordinates and y is the label"""
    x_one = x[y == 0, :]
    x_two = x[y == 1, :]

    plt.scatter(x_one[:,0], x_one[:,1], marker='x', color='red')
    plt.scatter(x_two[:,0], x_two[:,1], marker='o', color='blue')

def write_json(filename, value):
    """Write the provided value as JSON to the given filename"""
    with open(filename, 'w') as f:
        json.dump(value, f)
```