Brenden Hein

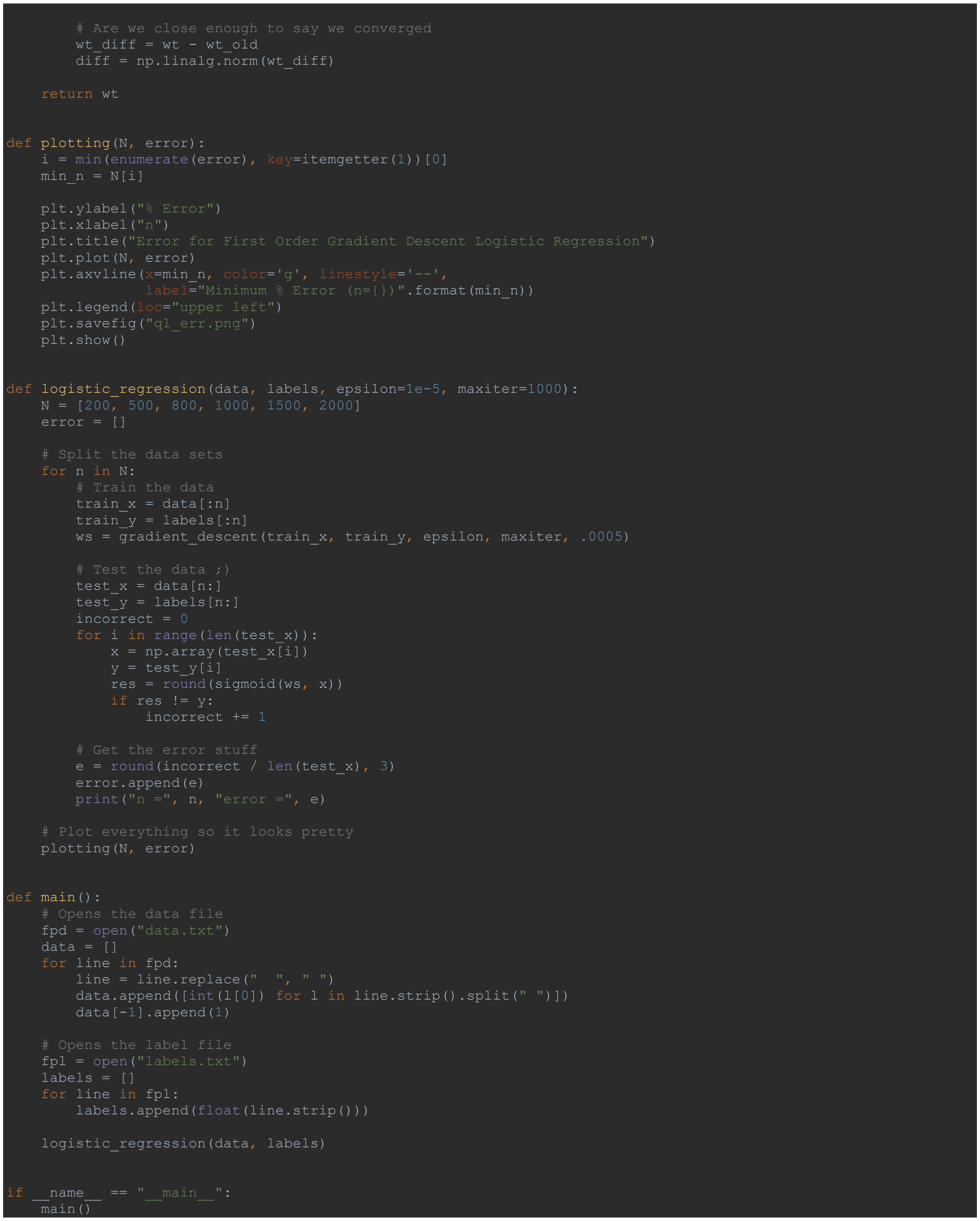
CSE 847

HW4

Logistic Regression #1

This experiment tested logistic regression on the spam email dataset using first order gradient descent to find the near-optimal weights vector for classification. With this weights vector, the program will be able to accurately classify sample emails as spam or not. I hypothesize that I will that using more datapoints as training samples will consistently lead to better results, up to a certain point. After that point, the error rate will start to go back up as their ends up being too few testing samples to evaluate with. I also hypothesize that the error rate will be around 5%.

```python
import matplotlib.pyplot as plt
from operator import itemgetter
import numpy as np
import math


def predict(x, w):
    """
    This function uses a sample and its weights to predict its values
    """
    x, w = np.array(x), np.array(w)
    return np.dot(x, w)


def sigmoid(x, w):
    """
    Calculates the output of the sigmoid function
    """
    s = predict(x, w)
    f = math.e ** (-1 * s)
    return 1 / (1 + f)


def error_gradient(w, data, labels):
    """
    This function calculates the error of the gradient with respect to w
    """
    error = 0
    for n in range(len(data)):
        xn, yn = data[n], labels[n]
        y_hat = sigmoid(xn, w)
        error += (y_hat - yn) * np.array(xn)
    return error


def gradient_descent(data, labels, epsilon, maxiter, n):
    """
    This function performs gradient descent to find the optimal vector of
    of weights to use for logistic regression
    """
    wt = len(data[0]) * [0]
    diff, i = 1, 0

    # Update the weights for each step of the first order gradient
    while diff > epsilon and i <= maxiter:
        wt_old = wt
        i += 1

        # Move the gradient down
        e_grad = error_gradient(wt, data, labels)
        wt = wt - n * e_grad
```



```python
        # Are we close enough to say we converged
        wt_diff = wt - wt_old
        diff = np.linalg.norm(wt_diff)

    return wt


def plotting(N, error):
    i = min(enumerate(error), key=itemgetter(1))[0]
    min_n = N[i]

    plt.ylabel("% Error")
    plt.xlabel("n")
    plt.title("Error for First Order Gradient Descent Logistic Regression")
    plt.plot(N, error)
    plt.axvline(x=min_n, color='g', linestyle='--',
                label="Minimum % Error (n={})".format(min_n))
    plt.legend(loc="upper left")
    plt.savefig("q1_err.png")
    plt.show()


def logistic_regression(data, labels, epsilon=1e-5, maxiter=1000):
    N = [200, 500, 800, 1000, 1500, 2000]
    error = []

    # Split the data sets
    for n in N:
        # Train the data
        train_x = data[:n]
        train_y = labels[:n]
        ws = gradient_descent(train_x, train_y, epsilon, maxiter, .0005)

        # Test the data ;)
        test_x = data[n:]
        test_y = labels[n:]
        incorrect = 0
        for i in range(len(test_x)):
            x = np.array(test_x[i])
            y = test_y[i]
            res = round(sigmoid(ws, x))
            if res != y:
                incorrect += 1

        # Get the error stuff
        e = round(incorrect / len(test_x), 3)
        error.append(e)
        print("n =", n, "error =", e)

    # Plot everything so it looks pretty
    plotting(N, error)


def main():
    # Opens the data file
    fpd = open("data.txt")
    data = []
    for line in fpd:
        line = line.replace("  ", " ")
        data.append([int(l[0]) for l in line.strip().split(" ")])
        data[-1].append(1)

    # Opens the label file
    fpl = open("labels.txt")
    labels = []
    for line in fpl:
        labels.append(float(line.strip()))

    logistic_regression(data, labels)


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

Fig. [1]. The program used to use first order gradient descent logistic regression on the spam email dataset

q1.py can be found here: https://github.com/Brenhein/CSE-847-Homework-4.git

Above is the code written to use logistic regression on the spam email dataset. Firstly, the program takes in the dataset file and brings the data into Python. After, the single dataset and labels are split on varying n of 200, 500, 800, 1000, 1500, 2000. The dataset is split into training samples and testing samples with n being the split point. Finally, the model is ready to be trained.

In order to train the model, the first method that must be employed is gradient descent. Passed to it are a convergence threshold and the number of iterations before stopping, along with the data and the step size hyperparameter. The optimal step size was estimated to be 0.0005. With this gradient descent is used to find the optimal weights. For each iteration, the error gradient is calculated using the error gradient function. This involves, for each data point, using the sigmoid function. After going through all the points, a weight vector is generated (which may not be optimal). As gradient descent converges, however, a close to optimal weights vector is discovered, which will be used to test the other samples when gradient descent completes.

Finally, we use the test samples to evaluate the accuracy of the model. For each sample, the dot product of the weights vector and the sample are generated. This is then passed to sigmoid function, which returns the resulting value. If the value is closer to 1, the email is classified as “spam”; if the value is closer to 0, it is classified as “not spam.” Once all the test samples have been evaluated, a final error rate for the specific n number of training samples is calculated. After all n values have been tested, a comparison plot is generated to find the optimal n.

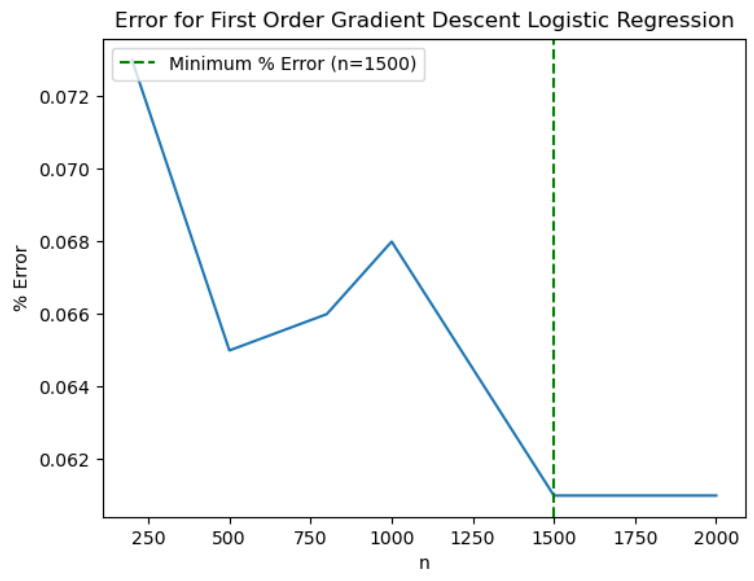


Fig. [2]. A chart of the different error rates for varying training dataset sizes

In figure 2 above, you can see how the error rate changes based on using different training dataset sizes. These dataset sizes involve using 200, 500, 800, 1000, 1500, 2000 as the training sizes and the rest of the data as the testing samples. Here, we can see a mostly downward trend as we increase the number of training samples used. This makes sense, as if we are training the model on fewer samples, it is not provided as much information to learn on, leading to less optimal weights for each of the features. Despite this, even only training on 200 datapoints still lead to fairly good results with an error rate of 7.3%. So, while it is not optimal, it shows that logistic regression can still generate good results in this case, even if there aren't enough training samples used.

Also seen on the figure above, there is a vertical dashed line which represents the optimal number of training samples to use. This is based solely on the error rate using a specific number of training samples. What's interesting is that the model actually yielded the same error rate for using training sample sizes of both 1500 and 2000.

In conclusion, it appears that the initial hypothesis on the trend of the error rate seems to be at least partially true. As n increases, the error rate has an overall trend downward. However, n was never increased enough to see if the error rate eventually increased, so the second part of that hypothesis may or may not be true. The other part of the initial hypothesis on the error rate seems to be fairly accurate. While it was initially thought that the optimal error rate would be around 5%, the optimal error rate turned out to be 6%. So, while they are different, they are still very close to each other, meaning the hypothesis was fairly true. Overall, it seems to be that logistic regression is a great and accurate classification model for this dataset.

Logistic Regression #2

In this model, sparse logistic regression is used which involves the l1 regularization parameter. Here, the varying parameter is the regularization parameter. The values of this will include 0, 0.01, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1, where each value will be used to train and test the logistic regression model. I hypothesize that, for the number of features selected, that as the regularization parameter increases, fewer and few features will be selected. I also hypothesize, that as the regularization parameter increases, the AUC curve will start to trend better, where the worst-case curve is at 45-degree angle and the best-case curve is at a 90-degree angle. Basically, this means that as the regularization parameter increases, each curve appears to be closer and closer to a 90-degree angle.

```
ad_data = load('C:\Users\brenh\Desktop\SS22\CSE 847\HW4\ad_data.mat');
features = load('C:\Users\brenh\Desktop\SS22\CSE 847\HW4\feature_name.mat');
train_x = ad_data.X_train;
train_y = ad_data.y_train;
test_x = ad_data.X_test;
test_y = ad_data.y_test;
features = features.FeatureNames_PET;

%% Set the options
opts.rFlag = 1; % range of par within [0, 1].
opts.tol = 1e-6; % optimization precision
opts.tFlag = 4; % termination options.
opts.maxIter = 5000; % maximum iterations.
```

```
%% Call the model for each
pars = [0, 0.01, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1];
features_s = [];
for i = 1:length(pars)

        %% Train the model for a given l1 regularization parameter
        par = pars(i);
        [w, c] = LogisticR(train_x, train_y, par, opts);

                %% Find the number of non-zero weights
                cnt = 0;
                for j = 1:length(w)
                if w(j) ~= 0
                        cnt = cnt + 1;
                end
        end

        features_s(end + 1) = cnt;

        %% Calculate the AUC metrix
        predictions = [];
        for j = 1:length(test_y)
                x = test_x(j, :);
                y = sign(dot(w, x));
                predictions(end + 1) = y;
        end
        [X, Y] = perfcurve(test_y, predictions, 1);

        %% Plot a subplot of the current AUC
        subplot(3, 4, i);
        plot(X, Y);
        head = strcat("λ=", string(par));
        title(head);
        xlabel("X");
        ylabel("Y");
end

%% Save the plot of all the subplots for AUC
name = 'C:\Users\brenh\Desktop\SS22\CSE 847\HW4\auc.jpg';
saveas(gcf, name, 'jpeg');

%% Plot the regularization parameters and number of features selected
figure;
plot(pars, features_s);
title("Regularization Parameter vs The Number of Selected Features")
xlabel("The Regularization Parameter")
ylabel("The Number of Features Selected")
saveas(gcf, ...
'C:\Users\brenh\Desktop\SS22\CSE 847\HW4\feature_cnt.jpg', ...
'jpeg');
```

Fig. [3]. The code used to calculate the sparse logistic regression model

q2.m can be found here: https://github.com/Brenhein/CSE-847-Homework-4.git

As seen in figure 3 above, sparse logistic regression is calculated. The implementation of the model was found from https://github.com/jiayuzhou/SLEP/tree/master/SLEP/functions/L1/L1R, where the logistic_train function was used for this experiment. The program starts by opening up the .mat files, where the "ad_data.mat" file contains the entire dataset, split between the training x and y and the testing x and y. The other file, "feature_name.mat," contains the feature names. Once all the data is brought in from the files, we can begin training the model.

Firstly, we iterate over the different regularization parameters, where for each one, we call LogisticR to with the training points to build the model. For each parameter, the LogisticR function will return two things: a weights vector and a bias. With these two datapoints, we will calculate two metrics: the number of features selected, i.e. the number of non-zero weights in the weight vector and the AUC curve.

Once these two metrics are gathered for all the regularization parameters, two plots will be generated for analysis. Firstly, a plot of subplots of the AUC curve with each value of the regularization parameter will be created to see the trend of an increasing regularization parameter along with their corresponding AUC curve. Secondly, a plot will be created to see the trend of the increasing regularization parameter along with the corresponding number of features selected for each parameter. Both of these will be used for analysis, which is described below.

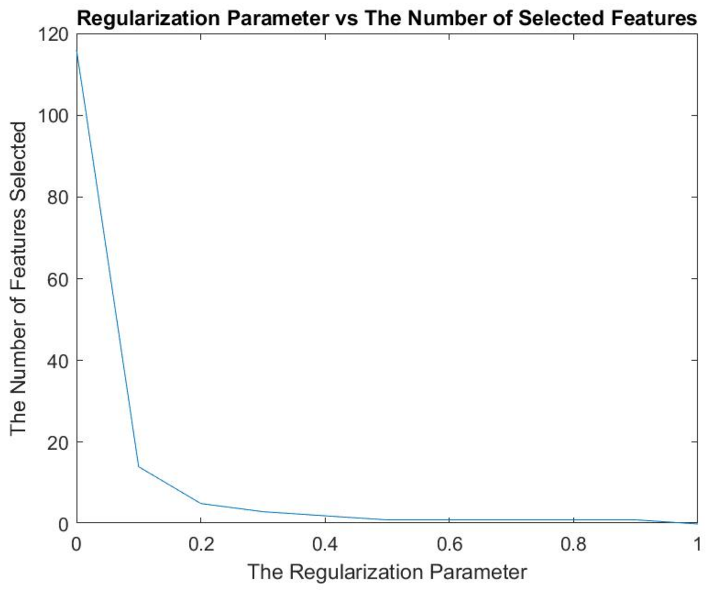


Fig. [4]. A comparison of the regularization parameter and the number of features selected

This chart demonstrates how, as you increase the regularization parameter, the corresponding number of features selected changes. Here, we can see a downward trend, where, as the regularization parameter increases, the corresponding number of features changes. What's unique about this graph is that, while the regularization parameter increases, the number of features selected actually decreases exponentially. This is due to the fact less important features are tossed out as the regularization parameter increases, aligning with the trend of the above chart.

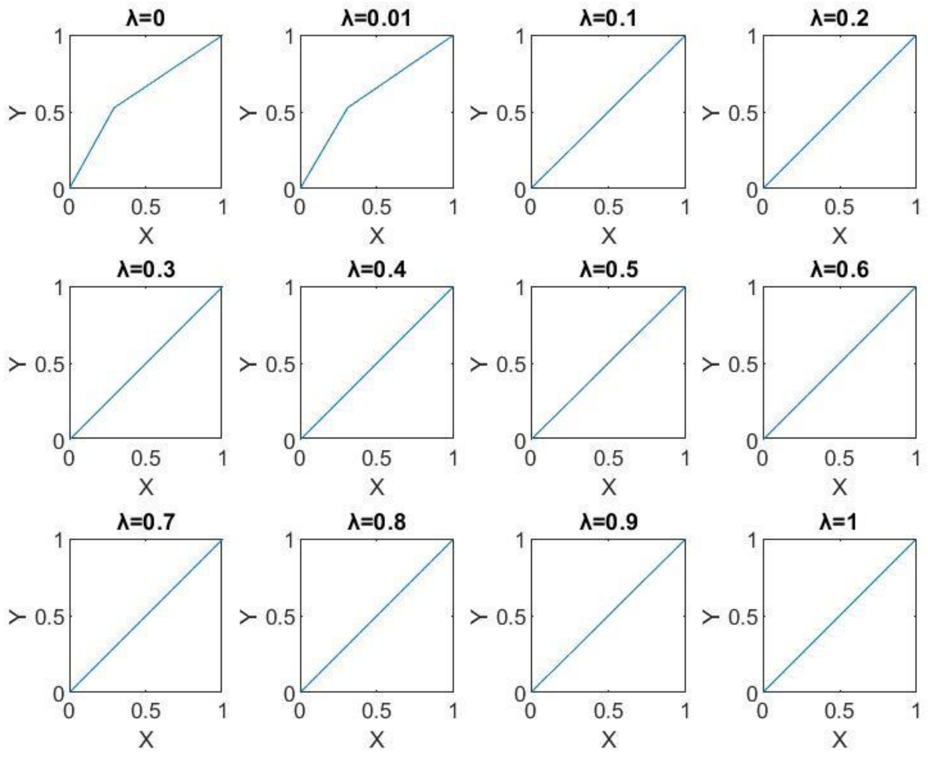


Fig. [5]. A comparison of the regularization parameter and the AUC curves

Here we can see how, as the regularization parameter increases, the AUC curves actually approaches worse curves. This means that the smaller the regularization parameter, the worse the AUC curve. With the smaller lambda (regularization parameters), while the AUC curve isn't great, it still has a more positive trend. However, the curve quickly converges to the worst-case scenario, making higher lambdas much less optimal and favorable to lower values of lambda.

In conclusion, there are some interesting trends associated with the increasing regularization parameter. In terms of the number of features selected, my hypothesis was somewhat correct. The idea of the decreasing number of features selected was right, but I thought the resulting trend would be linear; however, it appears to be exponential, meaning that a linear increment applied to the regularization parameter will result in an exponential decay of the number of features selected. On the other side, for the AUC curves, it seems that my hypothesis was incorrect. Unlike my idea that higher regularization parameters would lead to better AUC curves, the opposite is actually true. This may be due to the loss of data (features) in sparse logistic regression as the regularization parameter increases. Overall, there appears to be a trade off with the memory and time efficiency of using less features with the accuracy of the model.