Logistic Regression

Introduction:

In this project we implement the "Logistic Regression Function" for binary class classification and examine the hyperparameter effects on learning along with measuring the performance on relatively simple datasets with 2 features as well as higher dimensional feature sets (>2 features) as well. We re-formulate the logistic regression cost function derived in the earlier section where the label space is {0, 1} to a new space with {-1, 1} for ease of implementation. The datasets that have been used to train and evaluate our models are as follows:

1.) Synthetic binary class dataset with 2 features
2.) Digits dataset given under sklearn's library (digit:0 and 1 have been used).
3.) Wisconsin breast cancer dataset from sklearn.
4.) Wine dataset from sklearn

The evaluation metric in this project is accuracy, checked on both training and testing dataset for analysis (bias/variance). Finally, we discuss approaches that can be used to extending this binary classifier to multi-class classification problems.

Approach:

As the first step of this project, we start normalizing the synthetic dataset and then proceed to implementing the logistic regression algorithm with SGD (different from GD i.e., we update for each instance individually). We assume one of the class-labels to be represented by the value **1** and the other by **-1**.

Now, the formulation for our algorithm is based on the following equations which have been mentioned earlier in the document as well (i.e., when we have y ϵ {-1, 1} the modified equation looks like the below equation):

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} \log\left(1 + e^{-y^{(i)}\theta^T x^{(i)}}\right) = -\frac{1}{m}\sum_{i=1}^{m} \log\left(h_\theta\left(y^{(i)}x^{(i)}\right)\right)$$

where the hypothesis is the **sigmoid function**:

$$h_\theta\left(y^{(i)}x^{(i)}\right) = \frac{1}{1 + e^{-y^{(i)}\theta^T x^{(i)}}}$$

Now, for **SGD update**, we have the following equation for every instance $\boldsymbol{i}$ in the dataset and every feature index $\boldsymbol{j}$ in our total number of features:

$$\theta_j := \theta_j - \alpha\frac{1}{m}h_\theta\left(-y^{(i)}x^{(i)}\right)\left(-y^{(i)}x_j^{(i)}\right)$$

Additionally, we perform experimentation to find the base performance on the above datasets, unveil the effects of learning rate on the accuracies (given the number of fixed iterations), plot the cost function behavior during training, etc.

Evaluation:

The first dataset used here is a simple 2D distribution of two distinct classes. The dataset consists of 99 instances in all and is split in the ratio of 5:1 to build the training and the testing set.

Now, the cost function of SGD was evaluated after every new instance is taken for the gradient update. This means that for ***n*** instances and ***d*** iterations, we have ***nd*** number of values added to the history of costs. Below, is the instance plot, the SGD cost on the left, and the GD cost plot on the right:

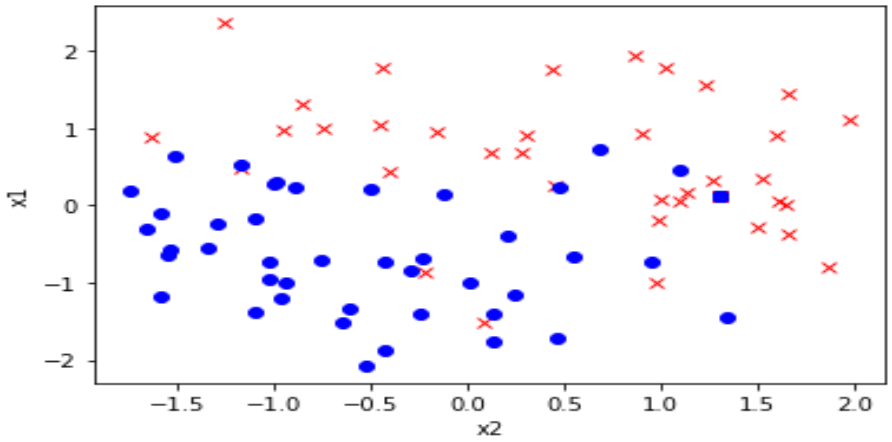

Figure 1: Plot of the 2D data-points

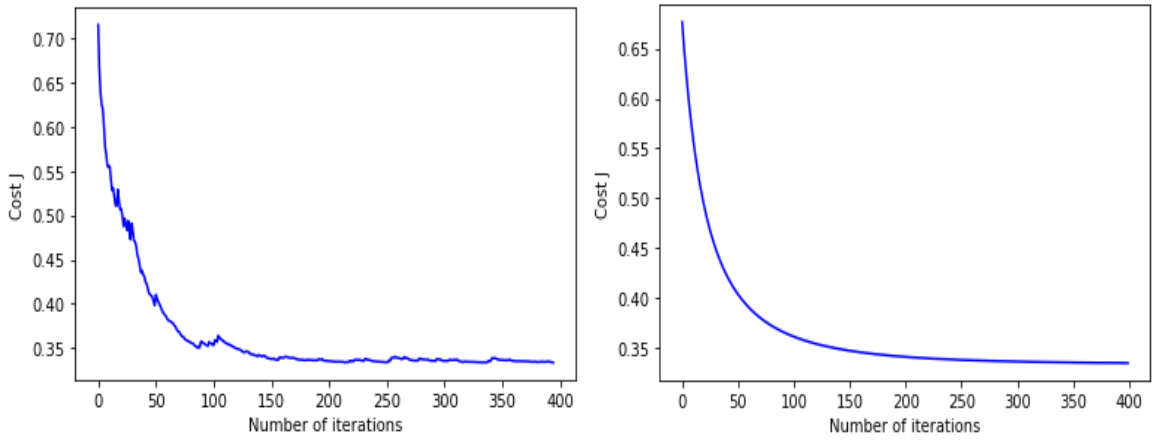

Figure 2:SGD (left) and GD cost function plots

If we compare these updates with say GD method, then we'll see that the cost function plot for GD is much smoother. Finally, we compute the coefficients of the line equation that separates the two datasets and compare with scikit-learn's Logistic regression model in the table below on the right and the final plot on the left.

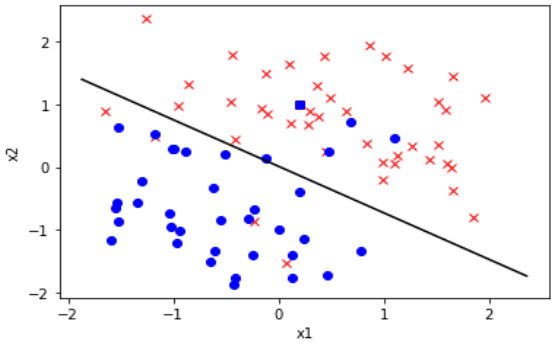

Figure 3: Decision boundary for our implementation

Synthetic data set	X_0
Train	88.6%
Test	90%

The learning rate is the hyperparameter that controls our algorithm's portion of update, i.e., for a higher rate we take larger updates and vice-versa. We add the accuracy table on the right and the graph of alpha experiment, wherein the natural log of 6 alpha values [10^{-3}, 10^{-2}, 10^{-1}, 10^{0}, 10^{1}, 10^{2}] in increasing order of magnitude is given on the left:

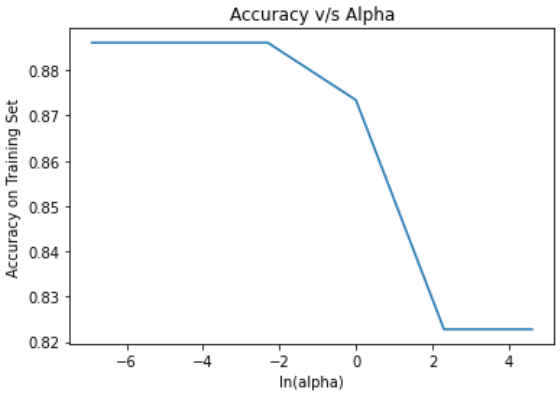

Figure 4: Alpha Experiment

Method Name	X_0	X_1	X_2
Scikit-learn	-0.023	1.40	1.91
Our	-0.023	1.37	1.85

Finally, we train our model for 3 standard datasets with the following results shown in the table below:

Data set	**No. of Features**	**No. of instances**	**Train Accuracy**	**Test Accuracy**
Digit	64	360	100%	100%
Cancer	30	512	83.7%	85.9%
Wine	13	178	98.7%	99%

Conclusion & Discussion:

In this project we saw how the accuracy of a model depends on the update method, hyperparameters, and the dataset complexity as well. The key takeaways are as follows:

- Keeping the learning rate low improves upon the final accuracy, however, it'll be much slower (smaller steps) given the time.
- SGD improves much more in one sweep through the dataset than GD (mainly due to the updates being performed in every instance). Additionally, GD turns out to be faster when the dataset size is small (as we can make use of numpy broadcasting to accelerate training.)
- Upon changing the label values, one can use simpler equations for logistic regression (in our case, using -1, +1 instead of 1, 0).
- In most cases, normalizing our dataset may yield better results as the model doesn't have to learn the differences due to variance/ mean of the dataset.

How to extend this to multi-class classification problems?

The approach that I'd take in this case it to **k** binary classifiers for **k** classes. We set the positive examples to be the main class (50%) and we set the negative class to be a uniform distribution from the rest of the classes. Then we train the model $\mathbf{M_k}$**.** Once, all the **k** models have been trained, for any new instance **X** we check the prediction confidence for all the models and choose the label to be **k*** such that it has the highest prediction confidence for **X** to belong to its positive label.