Logistic Regression Classifier Implementation

Abstract: This report is about the implementation of Logistic Regression Classifier from scratch. IRIS dataset is used for training and testing the classifiers. There are 3 sections in this report: the first one describes the theoretical/mathematical details which are translated to the implementation, the second section describes the real implementation and the last one presents the accuracy results and other observations.

Section 1: Theory behind Implementation:

Given a feature vector X = (a1, a2 , ... , ad)^T (d= 4 acc. to the given dataset) , which is the mathematical representation of the object to be classified, Logistic Regression Classifier outputs a class label y. This implementation works for a 2-class classification problem only.

We assume the form of the classifier function F(Y) = 1/(1 + exp(- theta^{T}Y)) where theta is a parameter vector = theta = (theta_0,theta_1,theta_2,….,theta_d)^T and Y is the augmented feature vector of X that is Y = (1, a1,a2,…..,ad)^T when X = (a1, a2 , ... , ad)^T.

This is actually:

Sigmoid Activation applied to the perceptron's linear discriminant function's output.

h(g(Y)) = 1/(1 + exp(- theta_vec^{T}Y)

[because sigmoid function: h(z) = 1/(1 + exp(-z)) and perceptron output: g(Y) = theta_vec^{T}Y]

Now after assuming this form of the classification function, we aim to find the parameter vector.

We represent one of the class labels by 1 and the other one by 0.

Since the sigmoid function's output lies in range :(0,1), we treat the h(g(Y)) (= F(Y)) as probability values i.e. F_{theta_vec}(Y) gives P(class = 1|Y; theta_vec) for a parameter vector.

Then we find the parameter vector such that the likelihood of observing the given random sample of the feature vectors is maximised.

L(theta_vec) = $\prod_{i=1}^{n} F(Y^i)^{class(i)}(1 - F(Y^i))^{1 - class(i)}$ (Likelihood function of theta_vec)

l(theta_vec) = $\sum^{n}_{i=1}$ (class$^{(i)}$ log(F(Y^i)) + (1- class$^{(i)}$) log(1 - F(Y^i))) (log likelihood)

Now we can take our loss/cost function as:

J(theta_vec) = -l(theta_vec) (Negative of log likelihood)

We find a parameter vector theta_vec such that the loss/cost function is minimized.

Iterative method - Gradient descent is used to find such theta_vec. We can use various stopping criteria in gradient descent with different batch_sizes, learning rates etc. More details about it in the next section.

nabla(J(theta_vec)) = $- \sum^{n}_{i=1} (class^{(i)} - F(Y^i))*Y^i$ (Considering whole batch)

We move in -ve direction of nabla(J(theta_vec)).

theta_vec_new = theta_vec + eta * $\sum^{n}_{i=1} (class^{(i)} - F(Y^i))*Y^i$ (e.g: A full batch gradient descent)

Once the parameter vector is found, we can use F(Y) whose output lies between 0 and 1. Next, we assume a threshold say (=0.5) :

F(Y) < 0.5 →output class corresponding to 0

F(Y) > 0.5 → output corresponding to 1

F(Y) = 0.5 → decide arbitrarily

Section 2: Implementation Details:

Now to find a parameter vector such that the loss function stated above is minimized, gradient descent algorithm is used.

Stopping Criteria 1):

Here we consider:

mini-batch of size = m

learning rate = eta

epsilon = E

In each iteration we pick up m training examples randomly(random sampling from the training set) and apply the gradient descent step considering this mini-batch. We continue iterating till Euclidean norm of the update vector in the gradient descent step is >= epsilon.

Stopping Criteria 2):

Here consider:

mini-batch of size = m

learning rate = eta

epochs_limit = E

In each iteration we pick up m training examples in an ordered manner and apply the gradient descent step considering this mini-batch. One epoch basically means that we have iterated through the whole training set once(by taking a number of mini-batches) . We continue iterating till we cross the epochs limit.

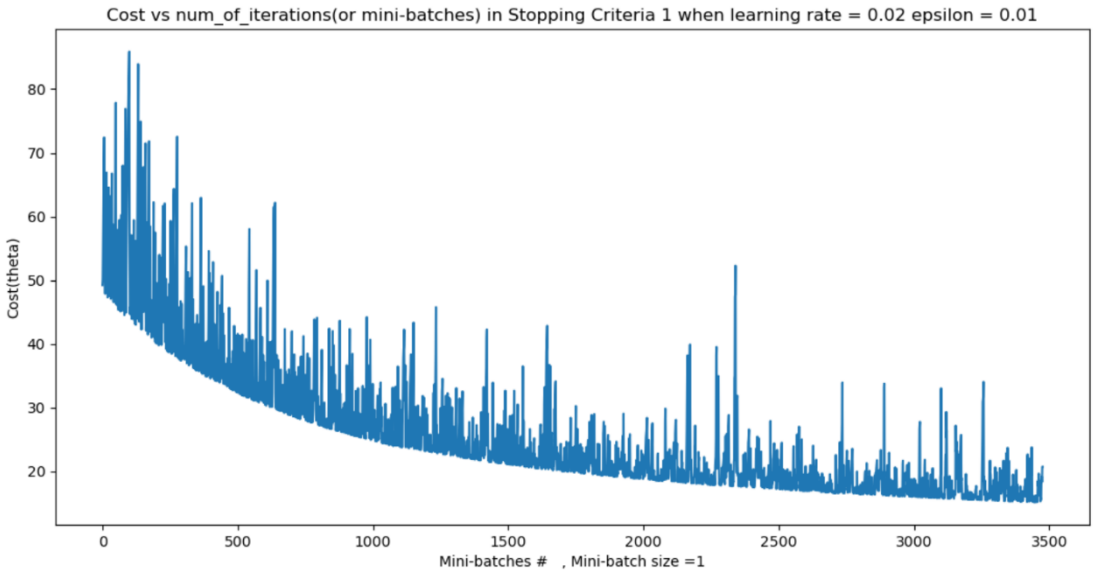
All implementation is done from scratch. Class LogisticRegressionClassifier contains various methods for reading training and testing data, learning the parameter vector from the training set and computing the accuracy.

Section 3: Observations and Results:

In case of stopping criteria 1):

When learning rate = 0.02, batchsize = 1 and epsilon =0.01 :

The graph of cost function value vs the number of iterations is(one iteration is over 1 mini-batch) :



Accuracy of the classifier over the test set

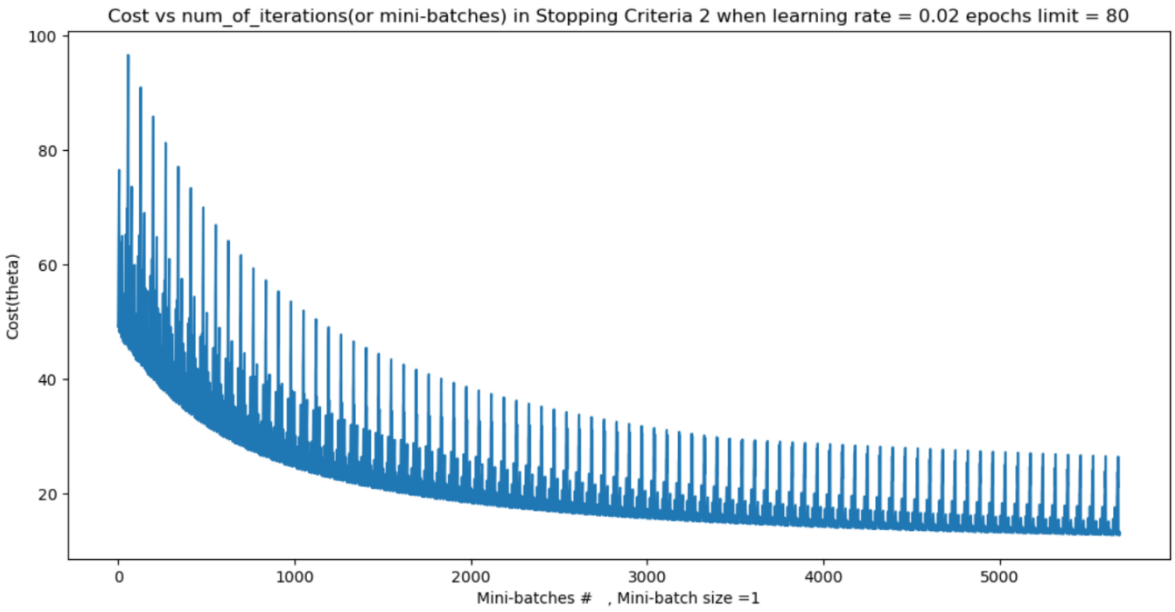
= 100* # of correct decisions/total # of test set examples

→ **lies in range 75% to 97%.** (The variation is because stochastic gradient descent is used.)
Note: I calculated this range just by running it several times. This result may not be fully correct due to random nature of the algorithm.

In case of stopping criteria 2):

When learning rate = 0.02, batchsize = 1 and epochs_limit = 80 :

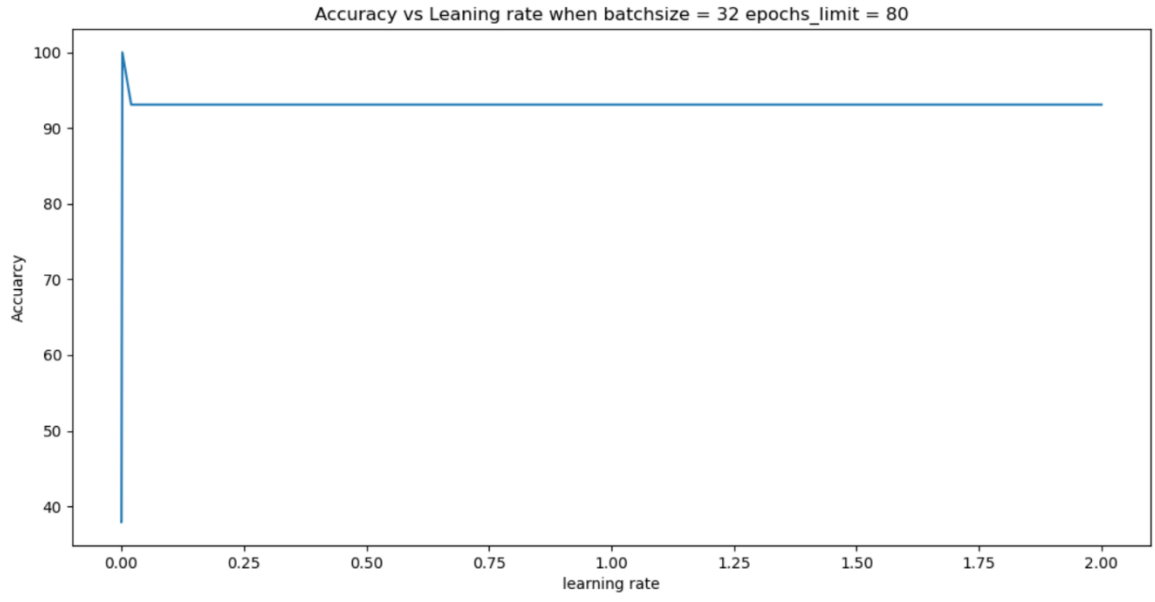
The graph of cost function value vs the number of iterations is(one iteration is over 1 mini-batch) :



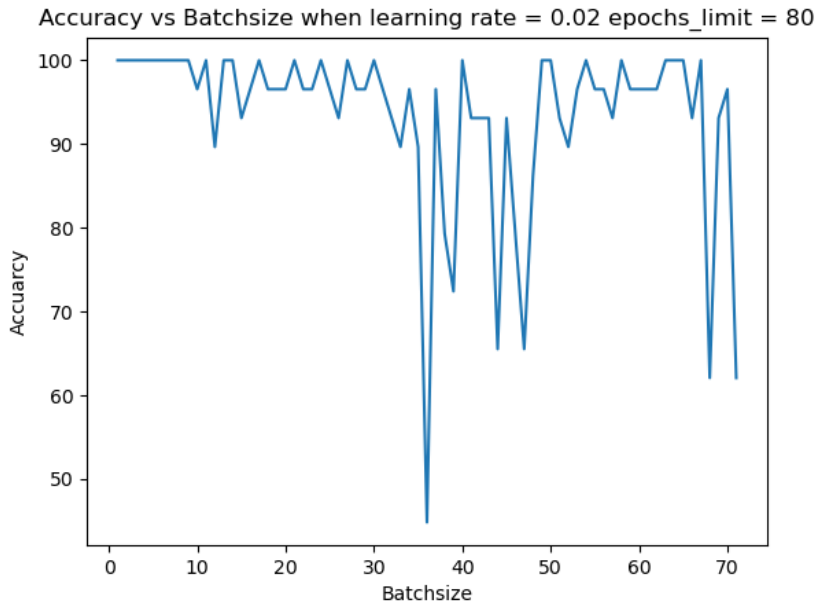
In this case, the accuracy over test set is 100%.

Other observations:

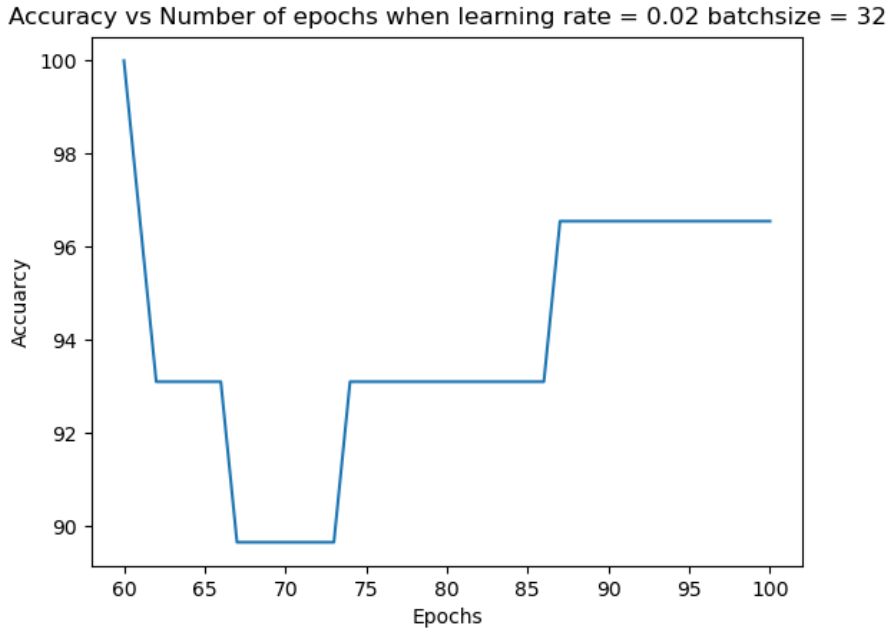
1) Variation in accuracy over test set with the learning rate when batch size = 32, the epochs_limit = 80.



2) Variation in accuracy over test set with the batchsize when learning rate = 0.02 and epochs_limt = 80



3) Variation in accuracy over test set with the epoch limit when learning rate = 0.02 and batch size = 32.



_____

_____