ML Library

WOC 6.0

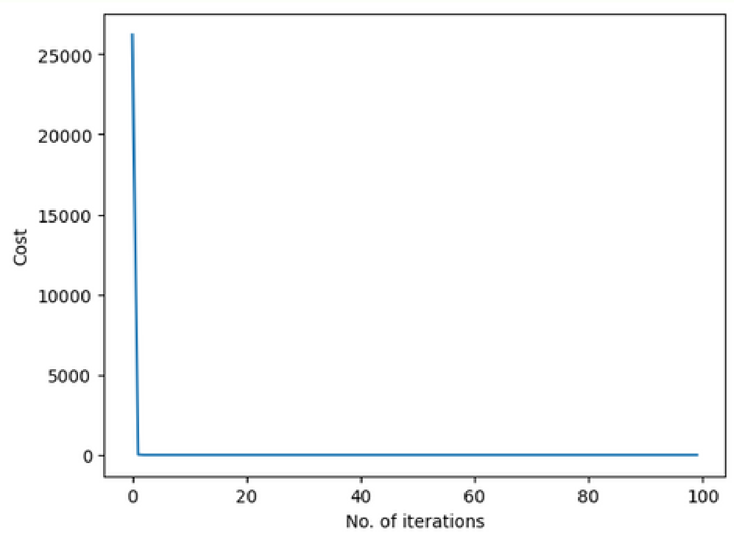
Prepared by: Harshvardhan Saini
(23JE0398)

Linear Regression

I shuffled the given dataset and split it into train, cross validation and test sets in a 0.8 : 0.1 : 0.1 proportion in order to check for overfitting (high variance problem) using the cross validation set and having a neutral check on performance using the test set; carried out shuffling in order to avoid biases in a set. Then, checked whether the dataset is distributed normally or not, in order to check for outliers. I used Z-score normalization to normalize each feature so as to ensure smooth descent in gradient.

Then, i tried different values for the hyperparameter learning rate and ran gradient descent for different number of iterations, and obtained great results for learning rate=1, the cost (Mean Squared Error) reached the global minima: 0.005078585402287006 within 20 iterations.

Cost vs Iterations:



Initial Cost: 65683803.199869424

Cost for cross validation dataset:
0.00490850113873416

Cost for test dataset:
0.004976562444356282

Used R^2 score as an evaluation metric to check for testing the performance of the model.

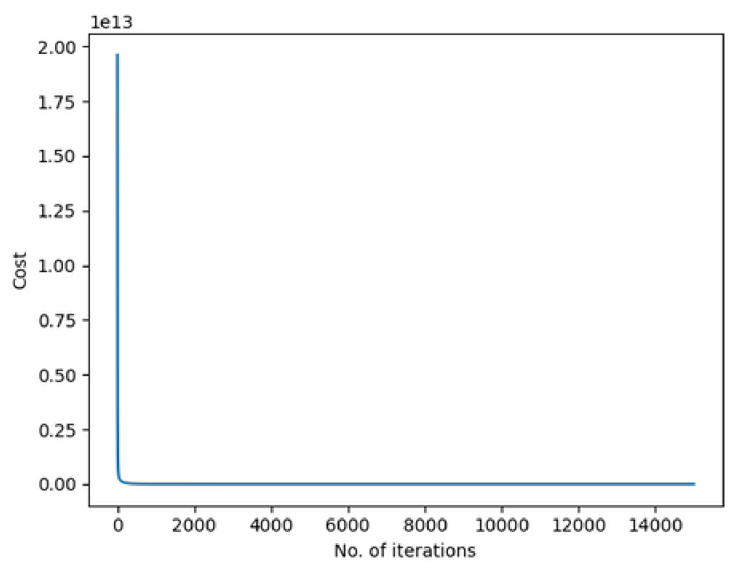
R^2 Score for training dataset: 0.999999999922728
R^2 Score for cross validation dataset: 0.999999999924968
R^2 Score for test dataset: 0.9999999999221306

Polynomial Regression

For polynomial regression, the first decision to make was the degree of the polynomial to be used. I considered the powers of the initial three features as (u, v, w) and used list comprehension to solve for u + v + w = i where i= (1,2,...,n) , n being the degree of polynomial taken as input. So, in this way, I engineered new features using the given features. Then , I shuffled the dataset, did a 0.8 : 0.1 : 0.1 split, and Z-score normalized the features.

Keeping in mind the increased complexity of the dataset, I implemented L2 regularization. Then, I ran gradient descent for different values of degree n, learning rate and regularization parameter (lambda), finally i found optimal results for n=6, learning rate of 0.25 and 15000 iterations. I also tried different values for lambda and observed the cost on cross validation set and realized that the model wasn't suffering high variance issue, so i decided to set lambda to 0. The cost (Mean Squared Error) saturated out to 1.607e-16.

Cost vs Iterations:



Initial Cost: 7.230e+13

Cost for cross validation dataset:
1.0080838554572531e-16

Cost for test dataset:
1.0732292957743687e-16

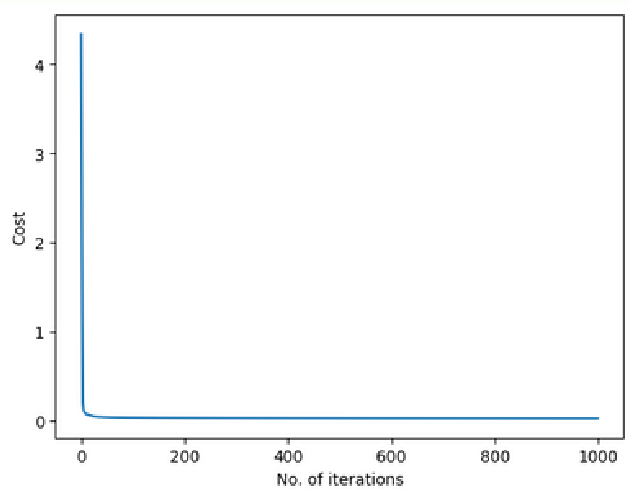
Used R^2 score as an evaluation metric to check for testing the performance of the model.

R^2 Score for training dataset: 1.0
R^2 Score for cross validation dataset: 1.0
R^2 Score for test dataset: 1.0

Logistic Regression

I did a 0.8 : 0.1 : 0.1 (train : cross validation : test) split for Logistic Regression and then scaled the data (pixel values) to between 0 and 1 by dividing them by 255 (maximum value for a pixel). Then, I one hot encoded the labels in each set and stored them in different variables.
Further, since the dataset had multiple classes (10) , the optimal choice would have been to use the softmax function, but since the task was to build a model for 'logistic regression' , I felt it implied using the sigmoid function, thus I decided to go ahead with the 'One vs rest' approach. I considered trying the 'One vs One' approach but eventually decided that training 10C2 models would take a lot of time, so dropped that idea.
So, I vectorized the 10 models for the One vs rest approach by taking the dimensions of the weight as (number of features, number of classes).
I implemented L2 regularization but found that high variance was not an issue in the model so set the lambda to 0.
I ran gradient descent on different values of learning rate and got the following results:
A cost (Binary Cross Entropy) of around 0.02416 with a learning rate of 5 and 5000 iterations

Cost vs Iterations:



Initial Cost: 0.6931471805599451

Accuracy for training dataset: 0.9745833333333334

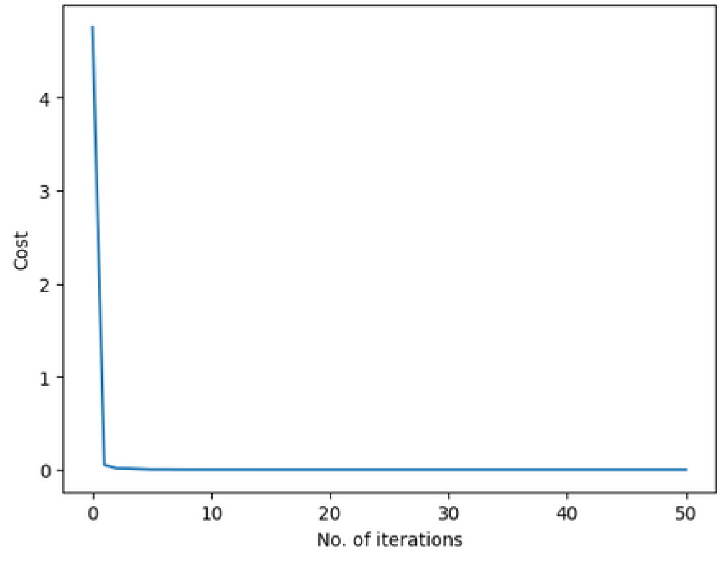
Accuracy for training dataset: 0.9716666666666667

Neural Network

I shuffled the dataset and did a 0.8:0.1:0.1 (train: cross validation: test) split and then scaled the pixel values to between 0 and 1 by dividing the dataset by 255 (maximum value for a pixel), one hot encoded the labels and stored them separately. Then I initialized the weights with the 'He initialization' , to avoid the issue of vanishing and exploding gradients. I saved the parameters in a dictionary for easy access ahead.
So, I used a model with three hidden ReLU activation layers having 512,256 and 128 neurons and used the softmax function in the output layer having 10 neurons corresponding to 10 classes.

Then I ran gradient descent but it turned out to be pretty slow, so I decided to implement mini batch gradient descent which was much faster and even better in terms of adding variance with each mini batch. I also implemented L2 regularization but later figured out that there was no high variance issue in the model so I set the lambda to 0. I used a mini batch size of 64 examples.
The cost (Sparse Categorical Cross Entropy) came around 1.6e-05 after 50 epochs, with a learning rate of 0.25.

Cost vs epochs:



Initial Cost: 4.757739175084508

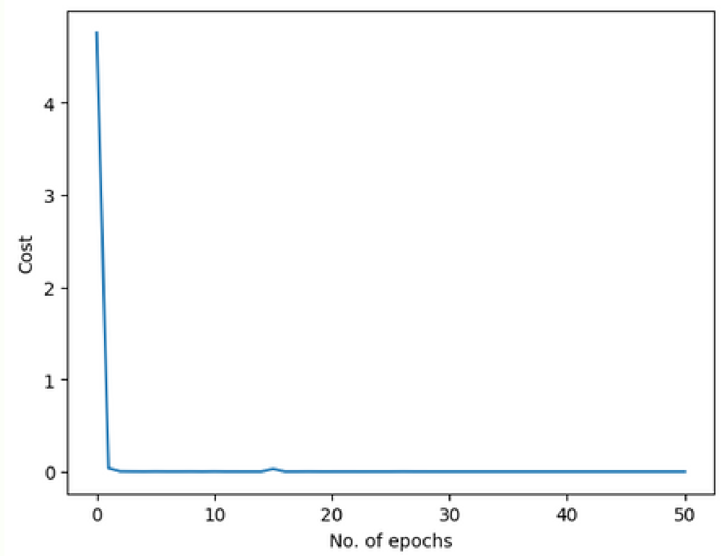
Accuracies:
Accuracy for training dataset:
1.0
Accuracy for cross validation dataset:
0.9886666666666667
Accuracy for test dataset:
0.989

Then, out of curiosity, I tried optimizers like gradient descent with momentum, Adam (Adaptive moment estimation); which helped in improving the accuracies.

Adam:

Got optimal results with learning rate of 0.001 and 50 epochs, beta1 of 0.9 and beta2=0.999 with a cost of around 4.6e-09.

Cost vs epochs:



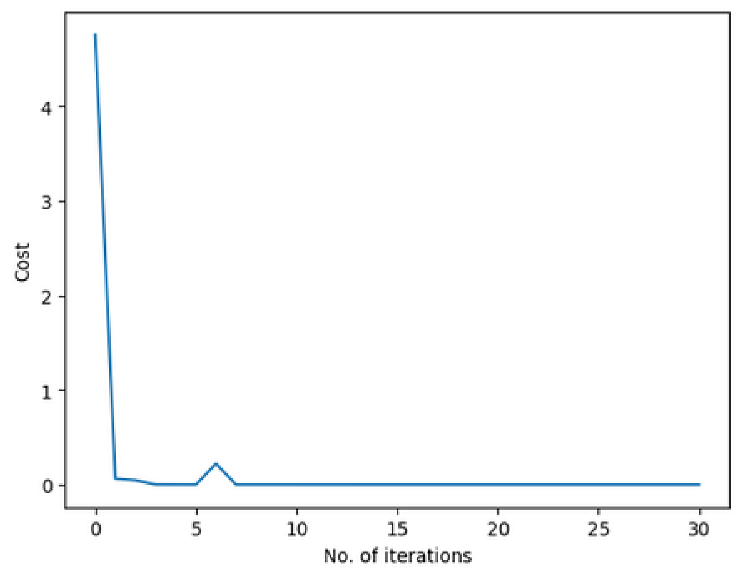
Initial Cost: 4.757739175084508

Accuracies:
Accuracy for training dataset:
1.0
Accuracy for cross validation dataset:
0.9916666666666667
Accuracy for test dataset:
0.988

Gradient Descent with momentum:

Tried different values of learning and epochs and got the following results-
A cost of around 5.86e-06 with a learning rate of 0.5, beta of 0.9 and 30 epochs

Cost vs epochs:

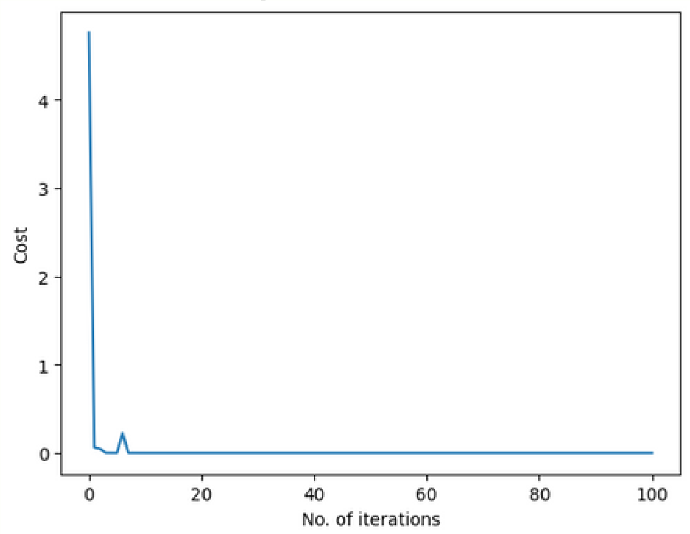


Initial Cost: 4.757739175084508

Accuracies:
Accuracy for training dataset:
1.0
Accuracy for cross validation dataset:
0.992
Accuracy for test dataset:
0.989

A cost of around 1.9e-06 with a learning rate of 0.5, beta of 0.9 and 100 epochs.

Cost vs epochs:



Initial Cost: 4.757739175084508

Accuracies:
Accuracy for training dataset:
1.0
Accuracy for cross validation dataset:
0.9923333333333333
Accuracy for test dataset:
0.9893333333333333

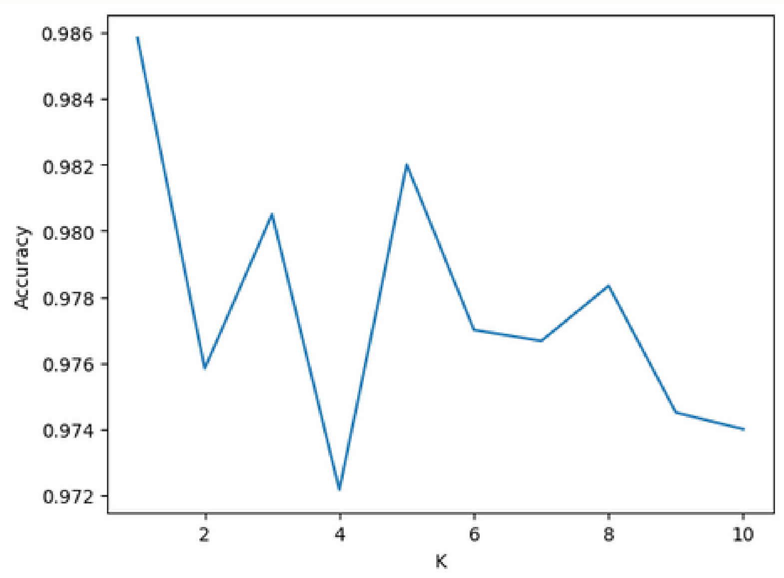
While using Adam and momentum, the cost doesn't converge to the global minima but rather revolves around it, thus I tried using decay for reducing the learning rate when the cost nears the minima, so that it converges. But I figured out that the results were better or almost similar without the decay, so i set the decay rate to 0.

It is to be noted that the Adam optimizer gives much lesser cost but a lower accuracy than momentum.

KNN

I shuffled the data and split it into the train set and the test set in a 0.8:0.2 ratio, then I scaled the pixel values to between 0 and 1 by dividing the data by 255 (maximum value of a pixel).
First I tried the 'one loop method', in which calculating the distance of a test data point from each point in the train data was vectorized but had to run a loop for all the data points in test set. This method was taking a lot of time, about 5-10 minutes.
Then, I searched and studied the 'no loop method' which vectorizes the process of calculating the distances of all data points in test set from the train set.
After calculating the distances, I used np.argsort to get the indices of K shortest distances for all test data points, and stored the labels corresponding to these nearest indices. Then I calculated and assigned the most common label in the K labels for each test data point.

Accuracy vs K:



So, the best accuracy is achieved for the value of 1 for the hyperparameter K ,and it is observed that each successive maxima is achieved at odd values of K.

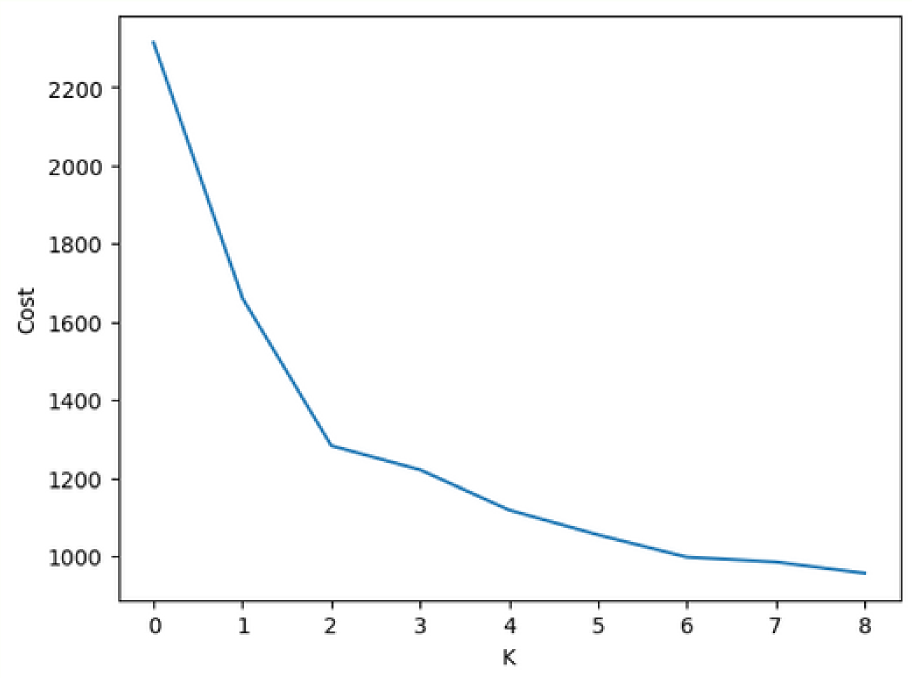
Accuracy at K=1: 0.9858333333333333

K-Means

I first Z-score normalized the dataset. Then I randomly initialized the clusters by choosing any K data points from the dataset.

Then I iterated over the steps of finding the closest centroids for each data point and then updating each centroid with the mean of the data points assigned to it.

Cost vs K:



Using the elbow method, it can be identified that K=2 and 6 are forming elbows, but K=6 should be the optimal choice considering that it is more practical.