**WINTER OF CODE**

**ML BOOTCAMP**

REPORT

By PRANAV GUPTA

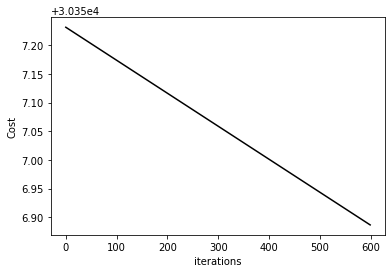
IIT (ISM) DHANBAD

1. **LINEAR REGRESSION**

* Initially used iterative statements to verify logic of code. On proper verification with dataset of a small size, proceeded to vectorization. Vectorization was chosen because it minimizes code, maximizes efficiency and leads to better implementation time.
* All features were normalized due to high variation in original values causing difficulties in computation and training.
* **Root Mean Square Error (RMSE)** is used to calculate cost.
* Various values of **learning rate (alpha)** were considered and decision regarding best value was taken by analyzing value of cost function over multiple iterations of gradient descent.

Learning rate=10-9

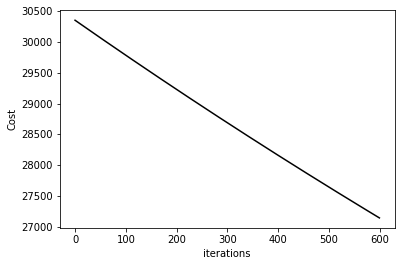
For this value, it was observed that cost function had negligible variation over large iterations like 450 and 500. The graph plotted between cost and iterations gives a straight line for iterations up to 600. Therefore, cost may decrease over large number of iterations but, on account of inefficiency, this rate value was discarded.



Alpha=10-9

Learning rate=10-5

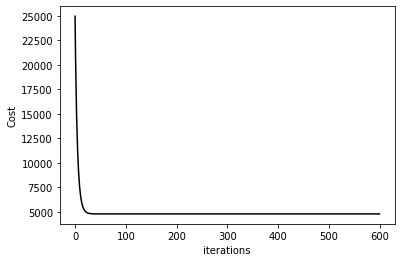
Graph still shows a straight line. So, it can be inferred that cost can still reach its minimum value for larger alpha within same number of iterations.



Alpha=10-5

Learning rate=10-2

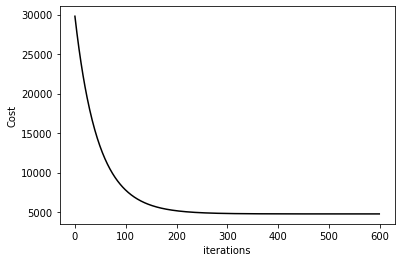
Here, on analyzing the graph, it is observed that cost reaches its saturation value initially, leading to the conclusion that alpha is larger than optimally required.



Alpha= 10-2

Learning rate=10-3

The cost decreases evenly at this value and reaches a minimum in over 400 iterations, as verified by the graph shown below.



Alpha=10-3

Seeing as this learning rate provides observable decrease in cost over considerably few iterations, alpha is chosen as 10-3.

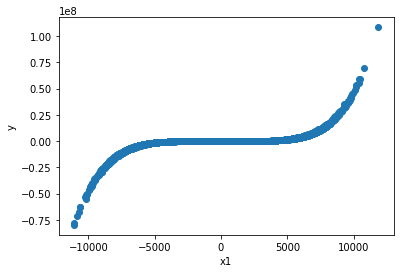
* **R2 score=0.9647361092141585.**

For training set (40000 entries) and testing set (10000 entries).

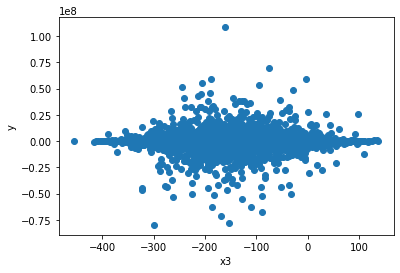
1. **POLYNOMIAL REGRESSION**

* Starting with iterative, code was checked for small dataset and then vectorized.
* Looking at the plot of labels and data structures, one feature at a time, a curve similar to odd exponential function xa , where a is any odd number, is seen. Resulting in the conclusion that the **polynomial function must be of odd degree**.

Below are graphs of labels and features taken one at a time.



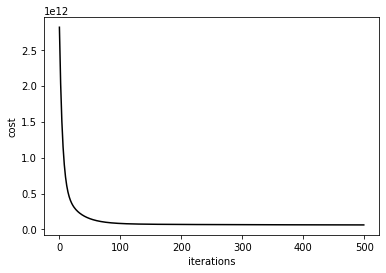




* On taking features as 1, x, x2 and x3 cost obtained was much larger than features of degree 5. Hence, **features were taken as 1, x, x2, x3, x4 and x5**.
* All features taken were normalized for ease of training.
* **Root Mean Square Error (RMSE)** is used for cost calculation.
* Like Linear regression, **learning rate** is chosen through cost function analysis.

Learning rate=4x10-3

A suitable decrease in cost function is seen with saturation after observable iterations, as shown in the graph below.



Alpha=4x10-3

* Various values of **feature scaling constant (lambda)** were employed for computation due to many features and probable overfitting.

For evaluating the model for a certain value of lambda, the training set was divided into two parts- one for training the model and the other for verifying the labels generated. Using different values of lambda, final cost for training subset and testing subset were calculated and considered.

Lambda=0

Cost for training subset is small, however, cost for testing subset is large. Thus, it can be inferred that the model may fit adequately to the training data, it fails in case of generalization.

Lambda=0.1

Negligible change in cost is seen in comparison to the previous case. Consequently, a similar problem persists.

Lambda=1000

The cost of training subset is adequately low with relatively small testing cost as well, leading to the conclusion that the model suitably generalizes the results.

* **R2 score=0.9884516246846113.**

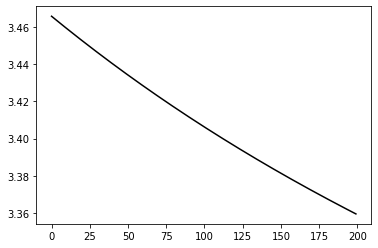
For training subset (35000 entries) and testing subset (15000 entries).

1. **LOGISTIC REGRESSION**

* Initially building one vs one model and using that as one vs rest by adding one-hot encoded labels for each class.
* Label vector was modified for **one vs rest** approach by appending columns of one hot encoded labels for each class.
* Features normalized for ease of training.
* The hypothesis is passed in the **Sigmoid** function.
* **Log Loss or Binary Cross Entropy** is used as cost function.
* The **learning rate** was decided not only by cost analysis but also by observing precision over training set (20000 entries) and testing set(10000 entries), since the model tackles a classification problem.

1. Learning rate=0.00005

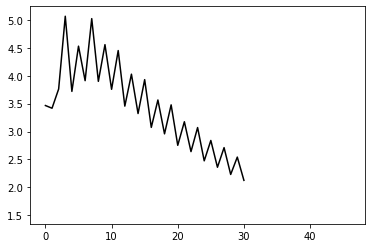
As seen in the graph cost curve provided below, the cost decreases but does not reach its minimum, so alpha must be smaller than optimum value. It is also seen that this alpha results in only 65.51% and 65.65% training and testing precisions respectively. Thus, it is discarded.



Alpha=0.00005

1. Learning rate=0.09

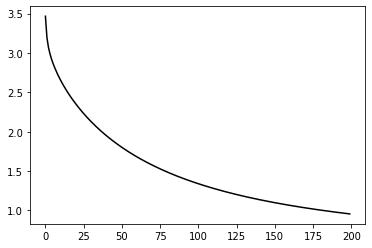
Seeing the cost vs iteration graph, it can be inferred that this value is larger than required. The oscillation of cost accounts for this value being discarded.



Alpha=0.09

1. Learning rate=0.05

In this case, the curve shows an observable and significant decrease in cost, adequate for the model. The precision of training set and testing set is relatively high at 81.73% and 81.14%.



Alpha= 0.04

So, through the above analysis, it can be concluded that the suitable Learning rate is alpha=0.05.

* As illustrated by the above results, the precision of the model on both sets is adequately high and overfitting does not seem probable. So, **the regularization constant (lambda) is taken as 0**.
* **Precision** of model=

**81.73%** on training set (20000 entries)

**81.14%** on testing set (10000 entries)

1. **K NEAREST NEIGHBOURS**

* For calculation of Euclidean Distance of new data point, the terms were broken to squares a2, b2 and 2\*a\*b for efficient computation.
* After the distances between new data point and training data were calculated, NumPy function argsort() was used and the retained index of distances were passed to labels so that the classes of nearest training data can be obtained.
* Choice of number of **nearest neighbours(K)**

For choosing K, precision of each value is compared. Precision is calculated for testing set of 5000 entries.

1. K=100

The model learned slowly in comparison and reached a precision of 80.3%.

1. K=50

The model yielded a precision of 81.38%.

1. K=10

Model’s precision was 83.4% on the testing set.

1. K=5

Precision on testing set was 84.5%.

Considering the results for various value of K, it is clear that K=5 is most efficient.

* **Precision** of the model

84.5% on testing set.

1. **N-LAYERED NEURAL NETWORK**

* Initially, code was written for finite number of layers and then generalized for ‘n’ layers which must be given as input along with number of neurons in each layer.
* All hidden layers use the **Rectified Linear Unit or ReLU** as the activation function so as to add non-linear characteristics to the model’s predictions.
* The output layer implements **SoftMax function** for 10 classes (as observed through NumPy unique() by passing labels). Implementation of numerically stable SoftMax is done such that no overflow may occur in exponent function.
* The final cost is taken as **Cross Entropy** function.
* Forward propagation is done, and outputs of all hidden layers are stored in lists.
* Gradients were calculated using backpropagation and used in gradient descent algorithm.
* A Neural Network containing **3 hidden layers and an output layer** is chosen to facilitate predictions while keeping the model safe from overfitting.
* The **Initialization of weights** of all the ReLU layers was done using the He or Kaiming Initialization, which factors in the non-linearity of ReLU and prevents the weights from being exponentially magnified or reduced.
* The **architecture** of the network was decided by analyzing precision when all other hyperparameters (learning rate and epochs) are same. It is noted that since output layer is SoftMax for 10 classes, the last hidden layer (layer 3 in this case) should have 10 neurons so that it can be operated on by SoftMax Function.

For precision analysis, training set contains 25000 entries and testing set contains 5000 entries.

1. 128 neurons🡪64 neurons🡪10 neurons

For constant parameters, this network resulted in 68.28% precision on training set and 68.36% precision on testing set.

1. 250 neurons🡪50 neurons🡪10 neurons

For the same parameters, this network was able to yield 69.4% on training and 69.24% on testing set.

1. 300 neurons🡪75 neurons🡪10 neurons

Precision of 73.65% on training and 74% on testing set was achieved.

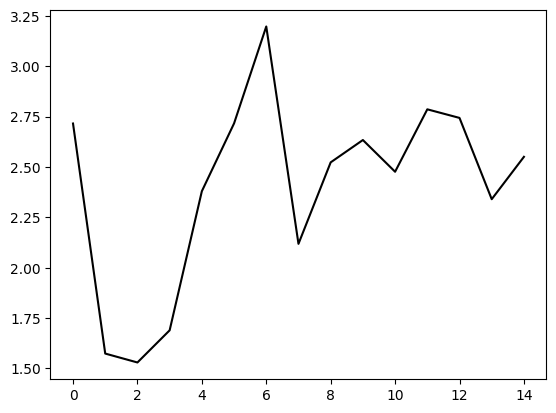
1. 600 neurons🡪300 neurons🡪10 neurons

Precision of 74% on training set and 74.75% on testing set was achieved.

From the above comparison, it can be observed that the most efficient model is the fourth one, being the most precise.

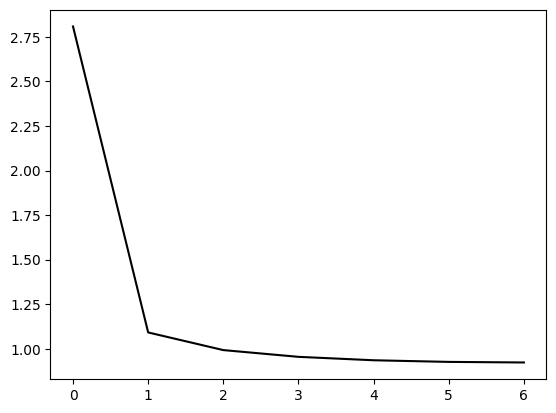
* After making architectural choices, **the Learning rate** was decided through cost and precision analysis.

1. Alpha=0.00005



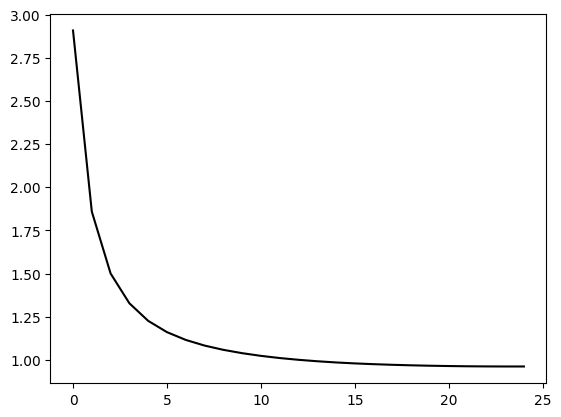
Unstable oscillations shown in the curve make it clear that value of alpha is larger than optimum.

1. Alpha=0.000005



The abrupt drop of cost shows that the learning is larger than suitable value. This yields a precision of 71.7% on the training set and 70% on the testing set.

1. Alpha=0.000001

****

As seen in the curve, cost decreases observably and to an adequately low value. This value of alpha gives a relatively high precision of 75.29% on training set and 75.42% on testing set.

Therefore, from all the above values of learning rate, the third one was chosen due to high precision and proper implementation of gradient descent.

* As seen in the above values of hyperparameters, the precision on training as well as testing set remains high and closely valued. So, no probable overfitting is seen and thus no regularization is employed.
* **Precision** of model-

**75.5%** on training set

**75%** on testing set.