# Report

Name – Suyash Rajput
Admission No – 22JE1003
Email – 22je@iitism.ac.in

---

Linear Regression and Polynomial Regression

To implement linear regression, I have used the following procedure:-

1. calculated the cost by using Root mean Squared error to train the data.

   j -> 1/2*m∑( y_pred – y_actual)^2

   where, j = cost

   y_pred = np.dot(x,w) + b

   x = data for train

   w = weight

   b = bias

   y_actual = label provided

2. Then moved to calculate gradient of the cost with respect to the weight and bias individually.

   dj/dw = gradient of cost w.r.t weight

   dj/db = gradient of cost w.r.t bias

   dj/dw = 1/m(np.dot(x,w) + b - y_actual)*x

   dj/db = 1/m(np.dot(x,w) + b - y_actual)

3. To minimise the cost,used the gradient descent.

   w = w - alpha * (dj/dw)

   b = b - alpha*(dj/db)

where alpha is a learning rate

4. Apart from this also implemented some function like:-
   Zscore for normalising the data
   R2score for calculating the efficiency of the training process
5. Details of all function, implemented for linear and polynomial regression
   - Zscore for normalising the data
   - R2_score for calculating the efficiency of the training process

- Prediction ,this is used to calculate predicted value
- Cost_gradient for estimating the cost and gradient
- Descent for minimising the cost

- Data_to_polynomial ,this is only used for polynomial regression to convert the data in polynomial form.

6. For linear regression
   -alpha used for given data = 0.5
   -no of iteration = 20
   -min cost achieved for normalised data = 4769.7687
   -R2_score = 0.84
   - time – less than a second
   -before initialization of data weight and bias were taken equal to zero.
7. For polynomial regression
   -alpha used for given data = 0.1
   -no of iteration = 300000
   -min cost achieved for normalised data = 0.1259847
   -R2_score = 0.9999999999999623
   -degree of polynomial is taken upto 5.
   -time = taking around 1 hour for 300000 iterations
   -before initialization of data weight and bias were taken equal to zero.

**Logistic regression**

->Functions implemented:

- zscore for normalising the data.
- multiclass_to_binary for changing labels from multiclass to binary.
- Sigmoid for activation.
- Cost_gradient - for estimating cost and gradient.
- Gradient_descent - for miniming the cost.
- Predict - for predicting the label of new data.
- Accuracy - for calculation of accuracy of data by using y_train. and y_pred( it is predicted on training data).

->for logistic regression:-

- data is normalised before training
- alpha(learning rate) = 0.00375
- iteration = 100
- time = 5 – 10 min
- accuracy obtained = 72% (can be increased if using more Iteration)

->formula used in implementation of algorithm:-

- f_wb = 1/(1 + exp(-z))
- cost(j) = -(y*log(f_wb) + (1 - y)*log(1- f_wb))/m (cost calculation)
- dj/dw = -1/m(f_wb – y)*x (gradient of cost w.r.t weight)
- dj/db = -1/m(f_wb – y) (gradient of cost w.r.t bias)
- w = w – alpha*dj/dw (for updating weight)
- b = b – alpha*dj/db (for updating bias)
- accuracy = mean of (y_pred == y_train)*100 (accuracy calculation)

**N- LAYER NEURAL NETWORK**

> Functions implemented:
- Relu – for activation.
- Derivative_relu – derivative of the relu.
- Zscore - for normalising.
- Sigmoid – for activation of output layer.
- Multi_to_binary for changing labels from multiclass to binary
- Initialiaze – for giving initial values to weight and bias
- Forward_propagation – for calculation in forward direction layer by layer, this function also uses for calculating output of test data.
- Cost – for computing cost
- Backward_propagation – this function give gradient of cost w.r.t weight and bias in each layer
- Upadate_parameters – this function is used to update value of weight and bias
- model – overall execution of data takes place here
- accuracy – provides accuracy at each iteration
- predict - for predicting the label of new data.
- Acc – for measuring accuracy of predicted label after training process

->For N-LAYER neural network:-
- data is normalised before training
- alpha (learning_rate) – 20
- iteration – 1000
- time – less than 10 min
- no of the hidden layer used – 2 hidden layer
- least cost obtained – 0.67(on 1000 iteration)
- accuracy obtained on training data – 0.81(can be improved if no of iteration increases)

->formula used in implementation of :-
- relu = maximum(x,0) (for activation)
- derivative of relu = {1,if x otherwise zero}

- $A1$ = activation( x*w + b),A2 = activation(A1*w + b),….

  …… AL = activation($A_{L-1}$ *W + B)

-j(cost) = (1./m)*∑ (-Y*np.log(AL)-(1- Y)*np.log(1 - AL)

- dzL = (AL - Y)
- dwL = (1/m)*dot product(dzL,$A_{L-1}$.T)
- dbL = (1/m) *np.sum(dzL,1)
- $dz_n$ = dot product($W_{n+1}$.T,$dz_{n+1}$)*$A_n$
- $dw_n$ = (1/m)* dot product ($dz_n$,$A_{n-1}$.T)
- dbn = (1/m)*np.sum($dz_n$,1)
- $w_n$ = $w_n$ – alpha*$dw_n$
- $b_n$ = $b_n$ – alpha*$db_n$

---

**K -NEAREST NEIGHBOUR**

**->**Functions implemented :-

-> KNN_fullyvectorised = this function is fully vectorised, first it calculate the distance of all testing data point from training data point and select first K-nearest neighbour and then calculate which label is repeating maximum time.

> KNN = in this function FOR loop is used to calculate the distance of all testing data point from training data point, after that same process is followed.

->For KNN:-

- k = 100
- no of x_train data points taken = 30000
- no of x_test data points taken = 100
- time = less than 30 sec
- accuracy obtained = 84