CS7602 - MACHINE LEARNING ASSIGNMENT 1

SUBMITTED BY

JAYASREE LAKSHMI NARAYAN 2016103033

AKHILA G P 2016103503

DATE : 28-01-2019

CONTENTS

1. SINGLE PERCEPTRON
2. MULTI LAYER PERCEPTRON
3. LINEAR REGRESSION (WITH SINGLE VARIABLE)

DATASETS USED

1. PIMA INDIAN DIABETES DATASET (CLASSIFICATION)
2. AUTO-MPG DATASET (REGRESSION)

A DESCRIPTION ON THE DATASET UNDER STUDY

PIMA INDIAN DIABETES DATASET

DESCRIPTION AND BASIC IDEA

In [1]:
```
import pandas as pd
data = pd.read_csv('pima-id.csv',header=None)
```

In [2]:
```
print "DATASET SHAPE: ", data.shape, " AND ANY NULL VALUES PRESENT: ",data.isnull().values.any()
data.head(5)
```

```
DATASET SHAPE:  (768, 9)  AND ANY NULL VALUES PRESENT:  False
```

Out[2]:

	0	1	2	3	4	5	6	7	8
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

In [3]:
```
data.describe()
```

Out[3]:

	0	1	2	3	4	5	6	7	8
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885	0.348958
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

AUTOMPG DATASET

DESCRIPTION AND BASIC IDEA

In [10]:
```
import pandas as pd
data = pd.read_csv('auto-mpg.csv',header=None)
```

In [11]:
```
print "DATASET SHAPE: ", data.shape, " AND ANY NULL VALUES PRESENT: ",data.isnull().values.any()
data.head(5)
```

```
DATASET SHAPE:  (398, 9)  AND ANY NULL VALUES PRESENT:  False
```

Out[11]:

	0	1	2	3	4	5	6	7	8
0	18.0	8	307.0	130	3504	12.0	70	1	chevrolet chevelle malibu
1	15.0	8	350.0	165	3693	11.5	70	1	buick skylark 320
2	18.0	8	318.0	150	3436	11.0	70	1	plymouth satellite
3	16.0	8	304.0	150	3433	12.0	70	1	amc rebel sst
4	17.0	8	302.0	140	3449	10.5	70	1	ford torino

In [12]:
```
data.describe()
```

Out[12]:

	0	1	2	3	4	5	6	7
count	398.000000	398.000000	398.000000	398.000000	398.000000	398.000000	398.000000	398.000000
mean	23.514573	5.454774	193.425879	102.894472	2970.424623	15.568090	76.010050	1.572864
std	7.815984	1.701004	104.269838	40.269544	846.841774	2.757689	3.697627	0.802055
min	9.000000	3.000000	68.000000	0.000000	1613.000000	8.000000	70.000000	1.000000
25%	17.500000	4.000000	104.250000	75.000000	2223.750000	13.825000	73.000000	1.000000
50%	23.000000	4.000000	148.500000	92.000000	2803.500000	15.500000	76.000000	1.000000
75%	29.000000	8.000000	262.000000	125.000000	3608.000000	17.175000	79.000000	2.000000
max	46.600000	8.000000	455.000000	230.000000	5140.000000	24.800000	82.000000	3.000000

1. **PERCEPTRON**

The Perceptron Algorithm

- **Initialisation**
  - set all of the weights w_{ij} to small (positive and negative) random numbers
- **Training**
  - for T iterations or until all the outputs are correct:
    - for each input vector:
      - compute the activation of each neuron j using activation function g:

$$y_j = g\left(\sum_{i=0}^{m} w_{ij}x_i\right) = \begin{cases} 1 & \text{if } \sum_{i=0}^{m} w_{ij}x_i > 0 \\ 0 & \text{if } \sum_{i=0}^{m} w_{ij}x_i \leq 0 \end{cases} \tag{3.4}$$

      - update each of the weights individually using:

$$w_{ij} \leftarrow w_{ij} - \eta(y_j - t_j) \cdot x_i \tag{3.5}$$

- **Recall**
  - compute the activation of each neuron j using:

$$y_j = g\left(\sum_{i=0}^{m} w_{ij}x_i\right) = \begin{cases} 1 & \text{if } w_{ij}x_i > 0 \\ 0 & \text{if } w_{ij}x_i \leq 0 \end{cases} \tag{3.6}$$

The jupyter notebook with the code is uploaded in Github and the link for the document is https://github.com/Akhilagp/ML_Assignment.

PROCEDURE:

- The perceptron is based on activation and threshold concept.
- A neuron fires when the output of the activation function is above the threshold set.
- It has a single layer of neurons with random weights attached to it.
- PARAMETERS VARIED For Understanding
  1. Learning rate

2. Number of Iterations

INFERENCE:

- The perceptron does well on the training set of the pima dataset, when the number of iterations are higher for a particular learning rate.
- A nominal learning rate produces a good result on the preprocessed set.

OUTPUT:

Learning rate	Number of Iterations	Accuracy
0.01	100	0.6197916667
	500	0.7057291667
	1000	0.703125
	2000	0.6979166667
0.03	100	0.6276041667
	500	0.7083333333
	1000	0.703125
	2000	0.7083333333
0.1	100	0.6380208333
	500	0.6770833333
	1000	0.671875
	2000	0.6770833333
0.25	100	0.6432291667
	500	0.7213541667
	1000	0.6979166667
	2000	0.7083333333
0.3	100	0.7213541667
	500	0.7135416667
	1000	0.7083333333
	2000	0.671875

A learning rate of 0.25 and 500 iterations was the highest recorded accuracy for the particular run. By testing the algorithm, an accuracy of 78% was achieved.

2. **MULTI LAYER PERCEPTRON**

The Multi-layer Perceptron Algorithm

- **Initialisation**
  - initialise all weights to small (positive and negative) random values
- **Training**
  - repeat:
    - for each input vector:

      Forwards phase:

      · compute the activation of each neuron j in the hidden layer(s) using:

$$h_\zeta = \sum_{i=0}^{L} x_i v_{i\zeta} \tag{4.4}$$

$$a_\zeta = g(h_\zeta) = \frac{1}{1+\exp(-\beta h_\zeta)} \tag{4.5}$$

      · work through the network until you get to the output layer neurons, which have activations (although see also Section 4.2.3):

$$h_\kappa = \sum_{j} a_j w_{j\kappa} \tag{4.6}$$

$$y_\kappa = g(h_\kappa) = \frac{1}{1+\exp(-\beta h_\kappa)} \tag{4.7}$$

      Backwards phase:

      · compute the error at the output using:

$$\delta_o(\kappa) = (y_\kappa - t_\kappa)\, y_\kappa(1-y_\kappa) \tag{4.8}$$

      · compute the error in the hidden layer(s) using:

$$\delta_h(\zeta) = a_\zeta(1-a_\zeta)\sum_{k=1}^{N} w_\zeta \delta_o(k) \tag{4.9}$$

      · update the output layer weights using:

$$w_{\zeta\kappa} \leftarrow w_{\zeta\kappa} - \eta\delta_o(\kappa)a_\zeta^{\text{hidden}} \tag{4.10}$$

      · update the hidden layer weights using:

$$v_\iota \leftarrow v_\iota - \eta\delta_h(\kappa)x_\iota \tag{4.11}$$

    - (if using sequential updating) randomise the order of the input vectors so that you don't train in exactly the same order each iteration
  - until learning stops (see Section 4.3.3)
- **Recall**
  - use the Forwards phase in the training section above

The jupyter notebook with the code is uploaded in Github and the link for the document is https://github.com/Akhilagp/ML_Assignment.

PROCEDURE :

- Ten nodes were used in the hidden layer.
- Running a logistic function, on the training data, ouputs were obtained and tabulated.
- The dataset was split into training set (50%), validation set (20%) and test set (30%).
- PARAMETERS VARIED For a Deeper Insight
    1. Learning rate (eta)
    2. Number of Iterations

INFERENCE:

- Higher the learning rate, converging of the descent is not proper and the error seems to increase or stay stable.
- With lower learning rate(<0.1), accuracy is high and loss is minimized.
- Increasing the hidden nodes from 5 to 10 seem to increase the accuracy of the classifier.

OUTPUT:

To support the inferences made, the algorithm was run for different learning rates (0.001 < eta < 0.9) for different iterations (1000 < it < 9000) . The accuracy and loss for each variation is tabulated below

Learning rate	Number of Iterations	Accuracy	Error
0.001	1000	88.5416666667	18.6579579272
	2500	88.5416666667	17.8883626083
	5000	89.84375	16.8562355692
0.003	1000	90.625	16.2648364351
	2500	90.8854166667	15.0400030387
	5000	92.96875	13.2560325229
0.01	1000	94.2708333333	12.0870142879
	2500	95.0520833333	10.6999256481

	5000	95.3125	9.2779888677
0.03	1000	94.53125	10.5823039961
	2500	92.4479166667	12.8231271803
	5000	95.33	8.9968897062
0.1	1000	77.6041666667	39.5647531552
	2500	83.8541666667	29.0762649929
	5000	80.9895833333	29.8457619597
0.3	1000	74.21875	47.2384863936
	2500	68.78	59.9290039822
	5000	68.75	59.8750835422

The row corresponding to learning rate 0.03 and 5000 iteration shows minimum error and maximum accuracy. As the learning rate increases, the dataset gets over-fitted leading to a increasing value of error. The algorithm on test set produced an accuracy of 71-75%.

3. **LINEAR REGRESSION**

Linear regression is a **linear** approach to modeling the relationship between a dependent variable and one or more independent variables.

The Error in a linear regression is calculated as follows

The Code is uploaded in Github and the link is

https://github.com/Akhilagp/ML_Assignment.

$$\sum_{j=0}^{N}\left(t_j - \sum_{i=0}^{M}\beta_i x_{ij}\right)^2 . \qquad (3.21)$$

This can be written in matrix form as:

$$(\mathbf{t} - \mathbf{X}\beta)^T(\mathbf{t} - \mathbf{X}\beta), \qquad (3.22)$$

The weights can be adjusted by the following formula

$$\textbf{weights} = (X^T X)^{-1} X^T y$$

PROCEDURE:

- The Auto-mpg dataset is split into training(80%) and test sets(20%) and the regression is carried out on the input features.
- The features considered were
    1. Dependent variable: miles per gallon (mpg)
    2. Independent variables: cylinders, displacement and horsepower
- The data is normalized and split.
- The gradient and the intercept for the calculation of the decision boundary line is obtained from stats module

gradient, intercept, r_value, p_value, std_err = stats.linregress(xtrain,ytrain)

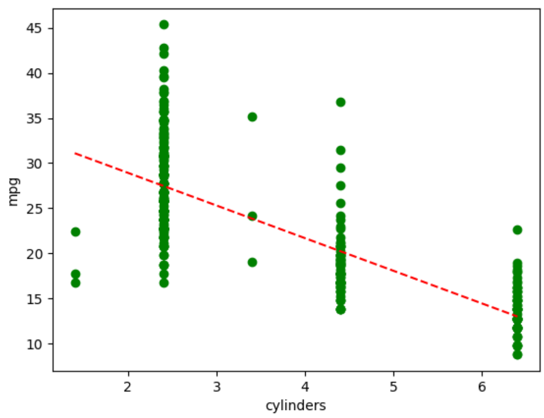
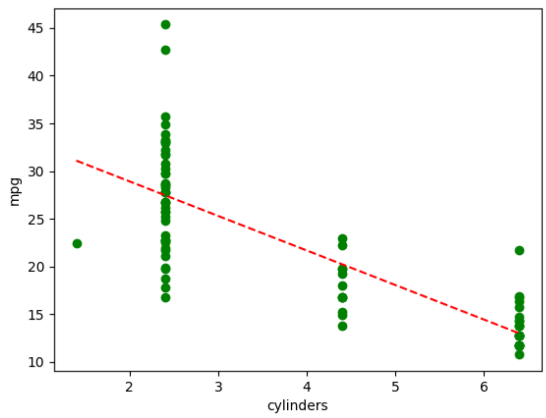
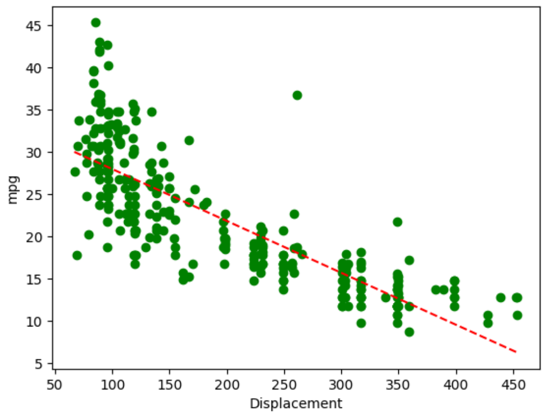
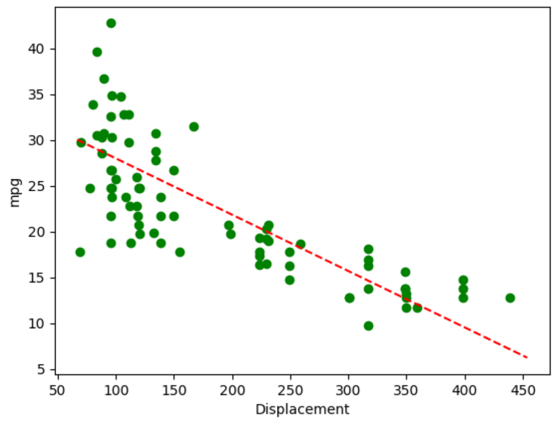
- The gradient turns out to be negative implying the negative co-relation between the variables taken.

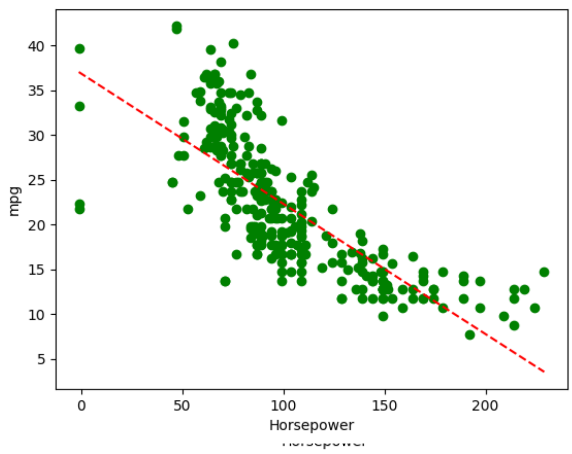
- PARAMETERS VARIED For Insight:

1. Split size of Training and testing
2. Independent variables taken for Linear Regression

INFERENCES:

The value of cost function/ error is computed and is found to be in powers of -26. The theta / weights matrix returned will be a column vector.

	
Training Cylinders vs mpg	**Testing** Cylinders vs mpg
	
Training Displacement vs mpg	**Testing** Displacement vs mpg

	
Training Horsepower vs mpg	**Testing** Horsepower vs mpg


```
lenovo@lenovo-Lenovo-ideapad-310-15IKB:~/Desktop/Akhila/ML_cs7602$ python auto.py
Training shape

X_train shape  (320, 8)  Y_train shape  (320, 1)
X_test shape  (78, 8)  Y_test shape  (78, 1)
The INITIAL WEIGHTS Chosen  [[ 1.43218770e-14]
 [ 1.00000000e+00]
 [ 3.05766697e-15]
 [-3.70601987e-15]
 [ 9.27534959e-17]
 [-2.44665399e-14]
 [-4.54254689e-14]
 [-7.99360578e-15]
 [-3.83937326e-12]]
Cost Function/Error  6.53414454149173e-24
lenovo@lenovo-Lenovo-ideapad-310-15IKB:~/Desktop/Akhila/ML_cs7602$
```