ASSIGNMENT

SUBMITTED BY

AHMED SAQUIF ALAM ANOUGH
ID: 14201001
CSE427 SECTION 1
FALL 2018

SUBMITTED TO

PROFESSOR MAHBUB ALAM MAJUMDAR
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
BUILDING-5, FLOOR-4TH, ROOM: UB50402

BRAC University

Department of Computer Science and Engineering

CSE427 - Machine Learning

Summer, 2018

Problem Set - 02

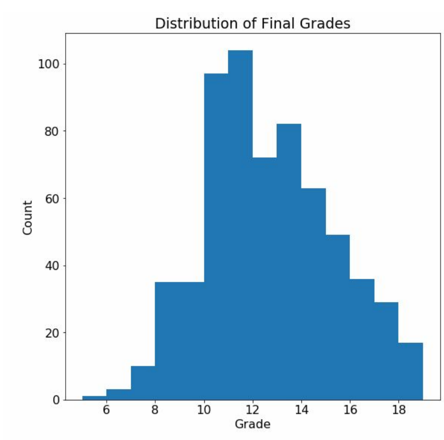
Bayesian Linear Regression.

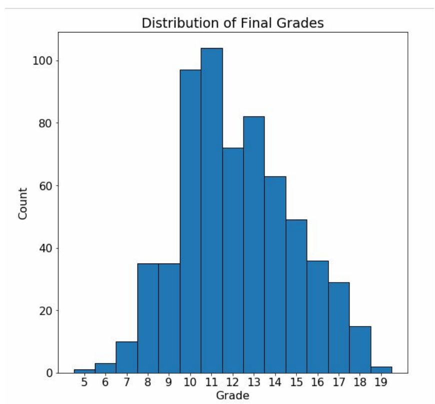
Question 01: What is the *dimension* of the *student* − *mat* dataset?(Hint: try to use your knowledge from the previous problem set)

The dimension of this dataset is 649 rows × 33 columns.

	school	sex	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob	...	famrel	freetime	goout	Dalc	Walc	health	absences	G1	G2	Grade
0	GP	F	18	U	GT3	A	4	4	at_home	teacher	...	4	3	4	1	1	3	4	0	11	11
1	GP	F	17	U	GT3	T	1	1	at_home	other	...	5	3	3	1	1	3	2	9	11	11
2	GP	F	15	U	LE3	T	1	1	at_home	other	...	4	3	2	2	3	3	6	12	13	12
3	GP	F	15	U	GT3	T	4	2	health	services	...	3	2	2	1	1	5	0	14	14	14
4	GP	F	16	U	GT3	T	3	3	other	other	...	4	3	2	1	2	5	0	11	13	13

Question 02: Are the grades normally distributed? If yes, at which point?



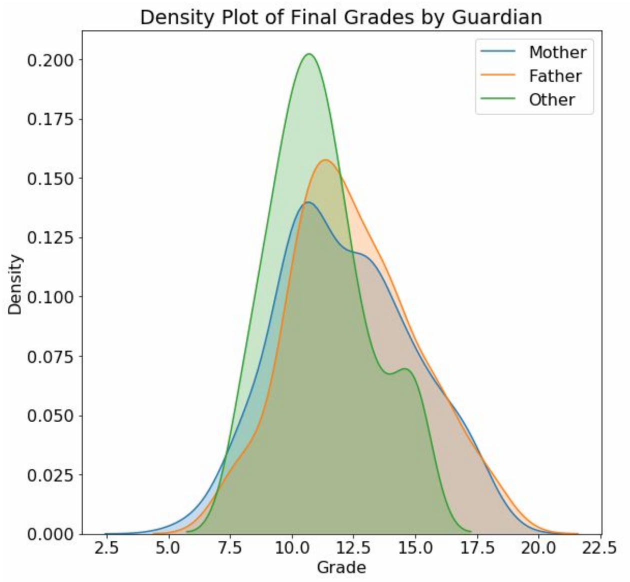


As we can see in the above image, the grades are largely normally distributed. But there may be a skew since the normalization is not in the middle. In Grade 12, there is an unlikely fall in the value.

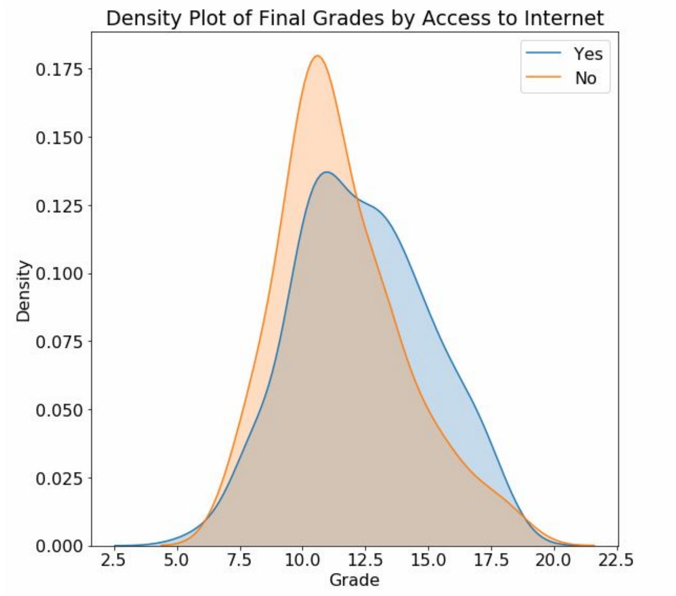
<u>Question 03:</u> Write a similar block of codes to plot the distribution of the grades by,

1. Guardian
2. Internet Access
3. Schools

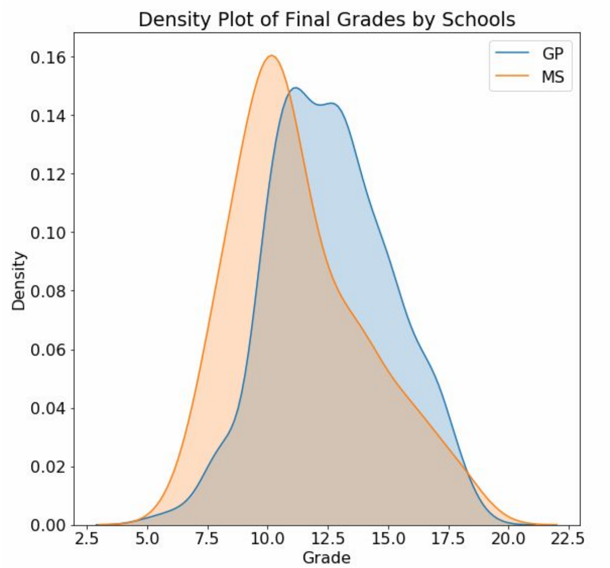
```
# Grade distribution by Guardian
sns.kdeplot (df.ix[df['guardian'] == 'mother', 'Grade'], label = 'Mother',shade = True)
sns.kdeplot (df.ix[df['guardian'] == 'father', 'Grade'], label = 'Father',shade = True)
sns.kdeplot (df.ix[df['guardian'] == 'other', 'Grade'], label = 'Other',shade = True)
plt.xlabel ('Grade');
plt.ylabel ('Density');
plt.title ('Density Plot of Final Grades by Guardian');
```




```
# Grade distribution by Internet Access
sns.kdeplot (df.ix[df['internet'] == 'yes', 'Grade'], label = 'Yes',shade = True)
sns.kdeplot (df.ix[df['internet'] == 'no', 'Grade'], label = 'No',shade = True)
plt.xlabel ('Grade');
plt.ylabel ('Density');
plt.title ('Density Plot of Final Grades by Access');
```



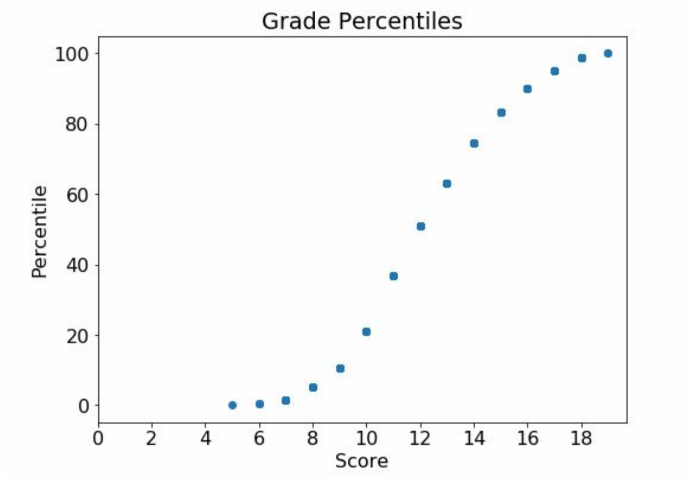

```
# Grade distribution by Schools
sns.kdeplot (df.ix[df['school'] == 'GP', 'Grade'], label = 'GP',shade = True)
sns.kdeplot (df.ix[df['school'] == 'MS', 'Grade'], label = 'MS',shade = True)
plt.xlabel ('Grade');
plt.ylabel ('Density');
plt.title ('Density Plot of Final Grades by Schools');
```



Question 03: Do the other categorical variables have any impact on the grades of the students? Answer from the graphs that you have plotted.

As it appears, students who do not have Fathers/Mothers tend to do much worse (for instance, for the same grade 10) than students who do have internet. On the other hand, students who do not have internet have a slightly higher grade. And Students from GP School do slightly better (Grade 12.5 upwards).

Question 04: What is the 50^{th} percentile score? Answer from the percentile graph.



This graph suggests that the 50th Percentile Score should be approximated at 100 percentiles.

Question 04: Write down the list of the correlated variables and underline the values those have most impact on the grades (You can ignore the variables mentioned before).

```
# Find correlations and sort
df.corr ()['Grade'].sort_values ()
```

```
failures      -0.384569
absences      -0.204230
Dalc          -0.196891
Walc          -0.178839
traveltime    -0.129654
goout         -0.111228
freetime      -0.105206
health        -0.096461
age           -0.042505
famrel         0.072888
Fedu           0.204392
studytime      0.249855
Medu           0.278690
G1             0.874777
G2             0.942691
percentile     0.985253
percentile     0.985253
Grade          1.000000
Name: Grade, dtype: float64
```

Here, when we find the correlations and sort them, we can see that after the mentioned ones, percentile has a large impact (even though it does not seem relevant). But, Medu, studytime, and Fedu as the next largest ones.

Question 05: What are the dimensions of the training and the test datasets?

The training set contains 25% of the actual data, and the test dataset contains 75% of the actual data. With 7 columns.

```
X_train, X_test, y_train, y_test = format_data (df)
y_train.head ()
```

```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_la
.ix is deprecated. Please use
.loc for label based indexing or
.iloc for positional indexing

See the documentation here:
http://pandas.pydata.org/pandas-docs/stable/indexing.ht
```

```
619    13
323    10
201    16
410    17
426    11
Name: Grade, dtype: int64
```

```
X_train, X_test, y_train, y_test = format_data (df)
y_test.head ()
```

```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_laun
.ix is deprecated. Please use
.loc for label based indexing or
.iloc for positional indexing

See the documentation here:
http://pandas.pydata.org/pandas-docs/stable/indexing.html
```

```
331    12
249    12
392    15
145    10
500     7
Name: Grade, dtype: int64
```

```
X_train, X_test, y_train, y_test = format_data (df)
X_test.head ()
```

```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.p
.ix is deprecated. Please use
.loc for label based indexing or
.iloc for positional indexing

See the documentation here:
http://pandas.pydata.org/pandas-docs/stable/indexing.html#ix-ir
```

	Grade	percentile	failures	higher_yes	Medu	studytime	Fedu
331	12	50.789889	0	1	1	2	1
249	12	50.789889	0	1	3	3	2
392	15	83.254344	0	1	3	2	2
145	10	21.011058	0	1	3	1	3
500	7	1.500790	1	0	1	1	2

```
X_train, X_test, y_train, y_test = format_data (df)
X_train.head ()
```

```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.p
.ix is deprecated. Please use
.loc for label based indexing or
.iloc for positional indexing

See the documentation here:
http://pandas.pydata.org/pandas-docs/stable/indexing.html#ix-ir
```

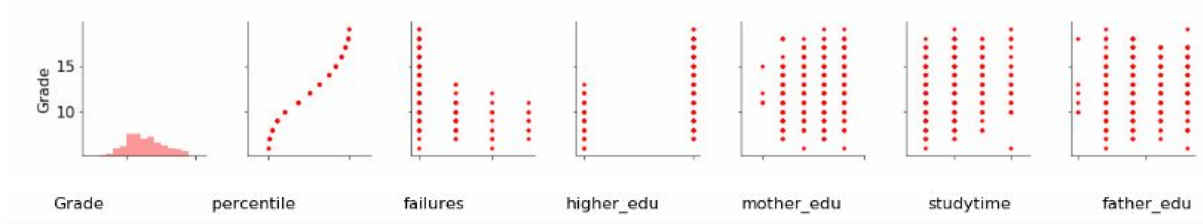
	Grade	percentile	failures	higher_yes	Medu	studytime	Fedu
619	13	62.954186	0	1	3	2	3
323	10	21.011058	0	1	4	1	3
201	16	89.968404	0	1	2	2	1
410	17	95.102686	0	1	2	2	2
426	11	36.887836	0	1	3	1	3

Bonus Question 01: Check if the names of the variables have been changed or not. If yes, then write the names of the changed variables.

Medu was renamed to mother_edu
Fedu was renamed to father_edu

Bonus Question 02: Which variable has the greatest correlation with the final grade in terms of absolute magnitude.

As we can see below, mother_edu seems to have the highest relation. But it appears that father_edu and study_time also have a large magnitude of relation.



ASSIGNMENT

BRAC
UNIVERSITY

Inspiring Excellence

SUBMITTED BY

AHMED SAQUIF ALAM ANOUGH
ID: 14201001
CSE427 SECTION 1
FALL 2018

SUBMITTED TO

PROFESSOR MAHBUB ALAM MAJUMDAR
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
BUILDING-5, FLOOR-4TH, ROOM: UB50402

BRAC University

Department of Computer Science and Engineering

CSE427 - Machine Learning

Summer, 2018

Problem Set - 03

Bayesian Linear Regression.

Part - 02: Use of Machine Learning Algorithms.

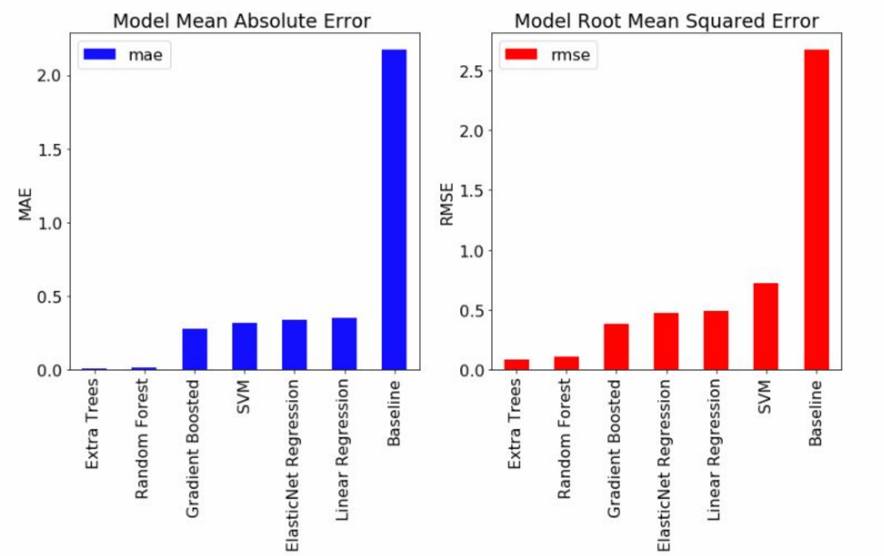
Question 01: Write down the values of the MAE and RMSE.

```
# Display the naive baseline metrics
mb_mae,mb_rmse = evaluate_predictions (median_preds,true)
print (mb_mae)
print (mb_rmse)
```

```
2.1761006289308176
2.6776503357897044
```

Please note that the given code had the error “Format Specifier missing precision”. So I removed it to just print the values directly.

Question 02: Which of these six standard algorithms works best for our dataset?



From the above, we can ascertain that the Extra Trees algorithm has the least Mean Absolute Error and the Root Mean Squared Error. It is thus the best idea to use this one.

Bonus Question 01: By seeing the comparison plot, can you say whether we can use machine learning for our problem or not?

We know that MAE measures the average magnitude of the errors in a set of predictions, without considering their direction. On the other hand, RMSE is a quadratic scoring rule that also measures the average magnitude of the error. Largely, we cannot make a judgement of whether this is an ML problem or not just based on these values. The data set could be small. The hypothesis could be bad. At the end of the day, the fact that a lot of the algorithms have a similar error means that this is mathematically a solid problem, but whether this data is adequate for it to be regarded as an ML problem is actually absolutely a larger question that cannot be said just off this graphs alone.

Question 03: Write down the comparative result matrix.

	mae	rmse
Linear Regression	0.354316	0.488203
ElasticNet Regression	0.340612	0.471816
Random Forest	0.0124528	0.111649
Extra Trees	0.00955975	0.0818343
SVM	0.321774	0.722182
Gradient Boosted	0.279416	0.38178
Baseline	2.1761	2.67765

Question 04: According to the outputs of the above code, which algorithm shows the best result

The results we got are:

The Linear Regression is 83.72% better than the baseline.
The ElasticNet Regression is 84.35% better than the baseline.
The Random Forest is 99.43% better than the baseline.
The Extra Trees is 99.56% better than the baseline.
The SVM is 85.21% better than the baseline.
The Gradient Boosted is 87.16% better than the baseline.

Thus, we can say that the Extra Trees is performing considerably better, with Random Forest closely trailing that.

Bonus Question 02: Write down the formula for the OLS Linear Regression.

In the case of a model with p explanatory variables, the OLS regression model writes:

$$Y = \beta 0 + \Sigma j=1..p\ \beta jXj + \varepsilon$$

where Y is the dependent variable, β0, is the intercept of the model, X j corresponds to the jthexplanatory variable of the model (j= 1 to p), and e is the random error with expectation 0 and variance σ^2.

In the case where there are n observations, the estimation of the predicted value of the dependent variable Y for the ith observation is given by:

$$yi = \beta 0 + \Sigma j=1..p\ \beta jXij$$

The OLS method corresponds to minimizing the sum of square differences between the observed and predicted values. This minimization leads to the following estimators of the parameters of the model:

$$[\beta = (X'DX)-1\ X'\ Dy\ \sigma^2 = 1/(W - p^*)\ \Sigma i=1..n\ wi(yi - yi)]$$

where β is the vector of the estimators of the βiparameters, X is the matrix of the explanatory variables preceded by a vector of 1s, y is the vector of the n observed values of the dependent variable, p* is the number of explanatory variables to which we add 1 if the intercept is not fixed, wi is the weight of the ith observation, and W is the sum of the wi weights, and D is a matrix with the wi weights on its diagonal.

The vector of the predicted values can be written as follows:

$$y = X\ (X'\ DX)-1\ X'Dy$$

In our code(provided), this formula is written down as “'Grade = %0.2f +' % lr.intercept_”

Question 05: Run the above code. You will get a message that says something about the probability that you are trying to calculate. Write down the message.

```
PatsyError: Error evaluating factor: NameError: name 'percentile' is not defined
    Grade ~ percentile  + failures + higher_edu + mother_edu + studytime + father_edu
            ^^^^^^^^^^
```

Bonus Question 03: Which of the density plot looks more normal to you, the test plot or the new observation plot?

The new density plot looks like expected values.

Question 07: If you have sketched all the line plots successfully from Question - 06 you will have some similar sketches. Now, write down the names of the variables that you think have the most and the least effect.(Hint: try to use the sketches to answer)

Largest Effect: mother_edu
Least Effect: studytime

ASSIGNMENT

BRAC
UNIVERSITY
Inspiring Excellence

SUBMITTED BY

AHMED SAQUIF ALAM ANOUGH
ID: 14201001
CSE427 SECTION 1
FALL 2018

SUBMITTED TO

PROFESSOR MAHBUB ALAM MAJUMDAR
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
BUILDING-5, FLOOR-4TH, ROOM: UB50402

BRAC University

Department of Computer Science and Engineering

CSE427 - Machine Learning

Problem Set - 4.1

Artificial Neural Network

1. Write down the predicted output matrix of every 50 iterations. *(Hint: you will need to change the code to get output for every 50 iterations.)*

2. List down the losses for every 50 iterations.

Doing both at the same time. Pasted below is the code change, and below it is the matrix - written down in three columns in order to save space.

```
NN = NeuralNetwork(X,y)
for i in range(1500): # trains the NN 1,000 times
    if i % 50 ==0:
        print ("for iteration # " + str(i) + "\n")
        print ("Input : \n" + str(X))
        print ("Actual Output: \n" + str(y))
        print ("Predicted Output: \n" + str(NN.feedforward ()))
        print ("Loss: \n" + str(np.mean(np.square(y - NN.feedforward ())))) #mean sum squared loss
        print ("\n")
NN.train(X, y)
```

for iteration # 0

Input :
[[0. 0. 1.]
 [0. 1. 1.]
 [1. 0. 1.]
 [1. 1. 1.]]
Actual Output:
[[0.]
 [1.]
 [1.]
 [0.]]
Predicted Output:
[[0.79460132]
 [0.84045415]
 [0.83030651]
 [0.86190036]]
Loss:
0.3571285618667454

for iteration # 50

Input :
[[0. 0. 1.]
 [0. 1. 1.]
 [1. 0. 1.]
 [1. 1. 1.]]
Actual Output:
[[0.]
 [1.]
 [1.]
 [0.]]
Predicted Output:
[[0.79460132]
 [0.84045415]
 [0.83030651]
 [0.86190036]]
Loss:

0.3571285618667454

for iteration # 100

Input :
[[0. 0. 1.]
 [0. 1. 1.]
 [1. 0. 1.]
 [1. 1. 1.]]
Actual Output:
[[0.]
 [1.]
 [1.]
 [0.]]
Predicted Output:
[[0.79460132]
 [0.84045415]
 [0.83030651]
 [0.86190036]]
Loss:
0.3571285618667454

for iteration # 150

Input :
[[0. 0. 1.]
 [0. 1. 1.]
 [1. 0. 1.]
 [1. 1. 1.]]
Actual Output:
[[0.]
 [1.]
 [1.]
 [0.]]
Predicted Output:
[[0.79460132]
 [0.84045415]
 [0.83030651]
 [0.86190036]]
Loss:
0.3571285618667454

for iteration # 200

Input :
[[0. 0. 1.]
 [0. 1. 1.]
 [1. 0. 1.]
 [1. 1. 1.]]
Actual Output:
[[0.]
 [1.]
 [1.]
 [0.]]
Predicted Output:
[[0.79460132]
 [0.84045415]
 [0.83030651]
 [0.86190036]]
Loss:
0.3571285618667454

for iteration # 250

Input :
[[0. 0. 1.]
 [0. 1. 1.]
 [1. 0. 1.]
 [1. 1. 1.]]
Actual Output:
[[0.]
 [1.]
 [1.]
 [0.]]
Predicted Output:
[[0.79460132]
 [0.84045415]
 [0.83030651]
 [0.86190036]]
Loss:
0.3571285618667454

for iteration # 300

Input :
[[0. 0. 1.]
 [0. 1. 1.]
 [1. 0. 1.]
 [1. 1. 1.]]
Actual Output:
[[0.]
 [1.]
 [1.]
 [0.]]
Predicted Output:
[[0.79460132]
 [0.84045415]
 [0.83030651]
 [0.86190036]]
Loss:
0.3571285618667454

for iteration # 350

Input :
[[0. 0. 1.]
 [0. 1. 1.]
 [1. 0. 1.]
 [1. 1. 1.]]
Actual Output:
[[0.]
 [1.]
 [1.]
 [0.]]
Predicted Output:
[[0.79460132]
 [0.84045415]
 [0.83030651]
 [0.86190036]]
Loss:
0.3571285618667454

for iteration # 400

Input :
[[0. 0. 1.]
 [0. 1. 1.]
 [1. 0. 1.]
 [1. 1. 1.]]
Actual Output:
[[0.]
 [1.]
 [1.]
 [0.]]
Predicted Output:
[[0.79460132]
 [0.84045415]

[0.83030651]
 [0.86190036]]
Loss:
0.3571285618667454

for iteration # 450

Input :
[[0. 0. 1.]
 [0. 1. 1.]
 [1. 0. 1.]
 [1. 1. 1.]]
Actual Output:
[[0.]
 [1.]
 [1.]
 [0.]]
Predicted Output:
[[0.79460132]
 [0.84045415]
 [0.83030651]
 [0.86190036]]
Loss:
0.3571285618667454

for iteration # 500

Input :
[[0. 0. 1.]
 [0. 1. 1.]
 [1. 0. 1.]
 [1. 1. 1.]]
Actual Output:
[[0.]
 [1.]
 [1.]
 [0.]]
Predicted Output:
[[0.79460132]
 [0.84045415]
 [0.83030651]
 [0.86190036]]
Loss:
0.3571285618667454

for iteration # 550

Input :
[[0. 0. 1.]
 [0. 1. 1.]
 [1. 0. 1.]
 [1. 1. 1.]]
Actual Output:
[[0.]
 [1.]
 [1.]
 [0.]]
Predicted Output:
[[0.79460132]
 [0.84045415]
 [0.83030651]
 [0.86190036]]
Loss:
0.3571285618667454

for iteration # 600

Input :
[[0. 0. 1.]
 [0. 1. 1.]
 [1. 0. 1.]
 [1. 1. 1.]]
Actual Output:
[[0.]
 [1.]
 [1.]
 [0.]]
Predicted Output:
[[0.79460132]
 [0.84045415]
 [0.83030651]
 [0.86190036]]
Loss:
0.3571285618667454

for iteration # 650

Input :
[[0. 0. 1.]
 [0. 1. 1.]
 [1. 0. 1.]
 [1. 1. 1.]]
Actual Output:
[[0.]
 [1.]
 [1.]
 [0.]]
Predicted Output:
[[0.79460132]
 [0.84045415]
 [0.83030651]
 [0.86190036]]
Loss:
0.3571285618667454

for iteration # 700

Input :
[[0. 0. 1.]
 [0. 1. 1.]
 [1. 0. 1.]
 [1. 1. 1.]]
Actual Output:
[[0.]
 [1.]
 [1.]
 [0.]]
Predicted Output:
[[0.79460132]
 [0.84045415]
 [0.83030651]
 [0.86190036]]
Loss:
0.3571285618667454

for iteration # 750

Input :
[[0. 0. 1.]
 [0. 1. 1.]
 [1. 0. 1.]
 [1. 1. 1.]]
Actual Output:
[[0.]
 [1.]
 [1.]
 [0.]]

Predicted Output:
[[0.79460132]
 [0.84045415]
 [0.83030651]
 [0.86190036]]
Loss:
0.3571285618667454

for iteration # 800

Input :
[[0. 0. 1.]
 [0. 1. 1.]
 [1. 0. 1.]
 [1. 1. 1.]]
Actual Output:
[[0.]
 [1.]
 [1.]
 [0.]]
Predicted Output:
[[0.79460132]
 [0.84045415]
 [0.83030651]
 [0.86190036]]
Loss:
0.3571285618667454

for iteration # 850

Input :
[[0. 0. 1.]
 [0. 1. 1.]
 [1. 0. 1.]
 [1. 1. 1.]]
Actual Output:
[[0.]
 [1.]
 [1.]
 [0.]]
Predicted Output:
[[0.79460132]
 [0.84045415]
 [0.83030651]
 [0.86190036]]
Loss:
0.3571285618667454

for iteration # 900

Input :
[[0. 0. 1.]
 [0. 1. 1.]
 [1. 0. 1.]
 [1. 1. 1.]]
Actual Output:
[[0.]
 [1.]
 [1.]
 [0.]]
Predicted Output:
[[0.79460132]
 [0.84045415]
 [0.83030651]
 [0.86190036]]
Loss:
0.3571285618667454

for iteration # 950

Input :
[[0. 0. 1.]
 [0. 1. 1.]
 [1. 0. 1.]
 [1. 1. 1.]]
Actual Output:
[[0.]
 [1.]
 [1.]
 [0.]]
Predicted Output:
[[0.79460132]
 [0.84045415]
 [0.83030651]
 [0.86190036]]
Loss:
0.3571285618667454

for iteration # 1000

Input :
[[0. 0. 1.]
 [0. 1. 1.]
 [1. 0. 1.]
 [1. 1. 1.]]
Actual Output:
[[0.]
 [1.]
 [1.]
 [0.]]
Predicted Output:
[[0.79460132]
 [0.84045415]
 [0.83030651]
 [0.86190036]]
Loss:
0.3571285618667454

for iteration # 1050

Input :
[[0. 0. 1.]
 [0. 1. 1.]
 [1. 0. 1.]
 [1. 1. 1.]]
Actual Output:
[[0.]
 [1.]
 [1.]
 [0.]]
Predicted Output:
[[0.79460132]
 [0.84045415]
 [0.83030651]
 [0.86190036]]
Loss:
0.3571285618667454

for iteration # 1100

Input :
[[0. 0. 1.]
 [0. 1. 1.]
 [1. 0. 1.]
 [1. 1. 1.]]
Actual Output:
[[0.]

 [1.]
 [1.]
 [0.]]
Predicted Output:
[[0.79460132]
 [0.84045415]
 [0.83030651]
 [0.86190036]]
Loss:
0.3571285618667454

for iteration # 1150

Input :
[[0. 0. 1.]
 [0. 1. 1.]
 [1. 0. 1.]
 [1. 1. 1.]]
Actual Output:
[[0.]
 [1.]
 [1.]
 [0.]]
Predicted Output:
[[0.79460132]
 [0.84045415]
 [0.83030651]
 [0.86190036]]
Loss:
0.3571285618667454

for iteration # 1200

Input :
[[0. 0. 1.]
 [0. 1. 1.]
 [1. 0. 1.]
 [1. 1. 1.]]
Actual Output:
[[0.]
 [1.]
 [1.]
 [0.]]
Predicted Output:
[[0.79460132]
 [0.84045415]
 [0.83030651]
 [0.86190036]]
Loss:
0.3571285618667454

for iteration # 1250

Input :
[[0. 0. 1.]
 [0. 1. 1.]
 [1. 0. 1.]
 [1. 1. 1.]]
Actual Output:
[[0.]
 [1.]
 [1.]
 [0.]]
Predicted Output:
[[0.79460132]
 [0.84045415]
 [0.83030651]
 [0.86190036]]
Loss:
0.3571285618667454

for iteration # 1300

Input :
[[0. 0. 1.]
 [0. 1. 1.]
 [1. 0. 1.]
 [1. 1. 1.]]
Actual Output:
[[0.]
 [1.]
 [1.]
 [0.]]
Predicted Output:
[[0.79460132]
 [0.84045415]
 [0.83030651]
 [0.86190036]]
Loss:
0.3571285618667454

for iteration # 1350

Input :
[[0. 0. 1.]
 [0. 1. 1.]
 [1. 0. 1.]
 [1. 1. 1.]]
Actual Output:
[[0.]
 [1.]
 [1.]
 [0.]]
Predicted Output:
[[0.79460132]
 [0.84045415]
 [0.83030651]
 [0.86190036]]
Loss:
0.3571285618667454

for iteration # 1400

Input :
[[0. 0. 1.]
 [0. 1. 1.]
 [1. 0. 1.]
 [1. 1. 1.]]
Actual Output:
[[0.]
 [1.]
 [1.]
 [0.]]
Predicted Output:
[[0.79460132]
 [0.84045415]
 [0.83030651]
 [0.86190036]]
Loss:
0.3571285618667454

for iteration # 1450

Input :
[[0. 0. 1.]
 [0. 1. 1.]
 [1. 0. 1.]

[1. 1. 1.]]
Actual Output:
[[0.]
[1.]
[1.]

[0.]]
Predicted Output:
[[0.79460132]
[0.84045415]
[0.83030651]

[0.86190036]]
Loss:
0.3571285618667454

DOWNLOAD DATASET FROM PS 2 ER QUESTION

CODE:

https://drive.google.com/file/d/1flNJr-KMKQ9_ebZUJvLcrANbp6tHZFf_/view?usp=sharing

https://drive.google.com/file/d/1CxDJLpddrcwjwUfq1eRJHbdKg9adKmzP/view?usp=sharing