NEURAL NETWORK WITH BACKWARD PROPAGATION

File name: Readme.pdf
Created by: Maj Amit Pathania
Roll No: 163054001

----

1. **Files included**: train_net.py, test_net.py, train_standard.py

2. **Feature engineering**:

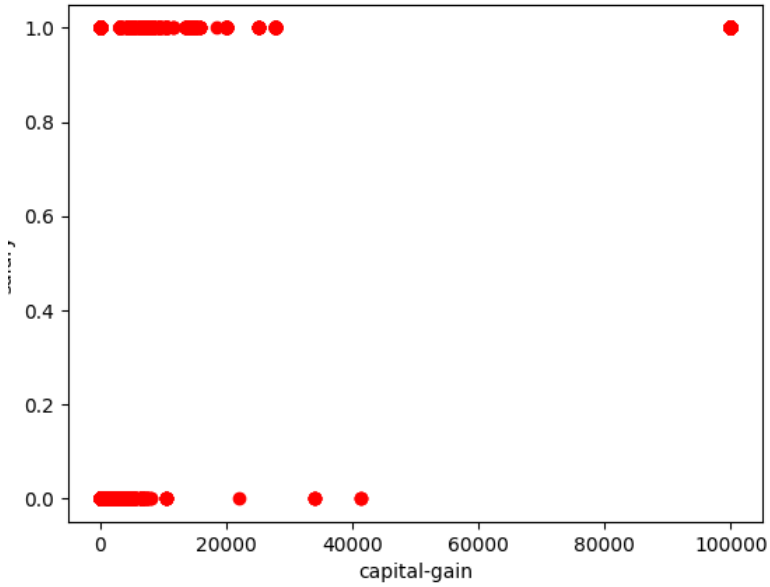
(a) Given training data had many continuous value and categorical value features. Since we can't fit categorical variables into our mathematical formulation, hot encoding was used. Dummy coding/**One hot encoding** was used method for converting a categorical input variable into continuous variable. One hot encoding was used for 'workclass', 'education', 'marital-status', 'occupation', 'relationship' ,'race', 'sex' and 'native-country' features. For every level present, one dummy variable was created thus increasing total number of feature columns.

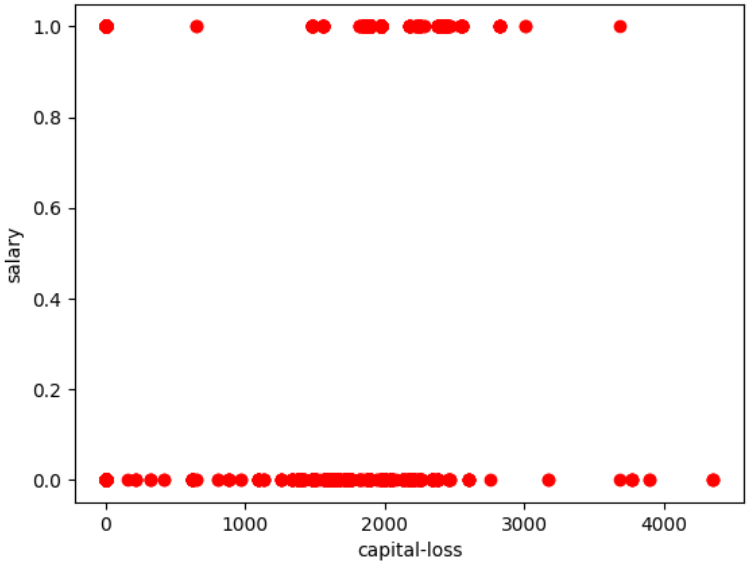
(b) Since 'education-num' and 'education' column were related.ie education-num contained numerical values for categorical class 'education'. Hence, there exists dependence between these features. Each educational qualification has been given some education-num as a numerical code.

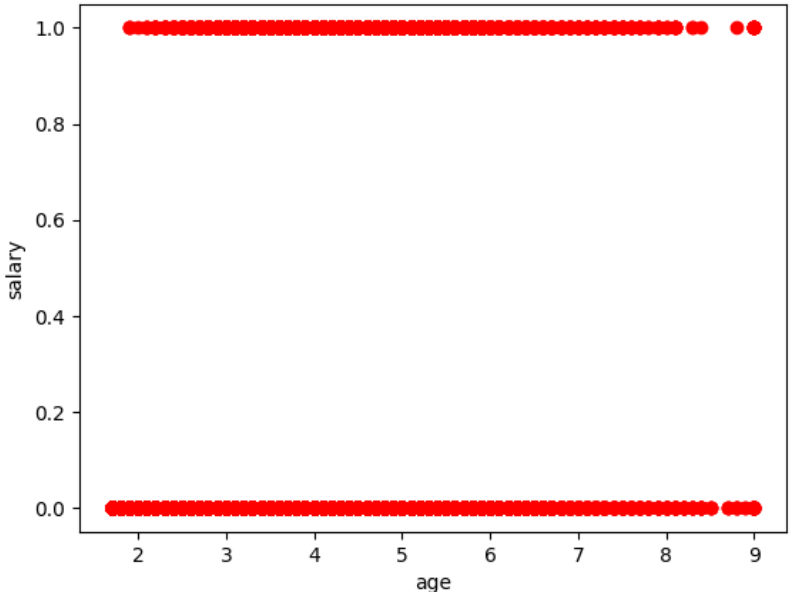
(c) In order to reduce wide range of values **features 'fnlwgt', 'capital-gain' and 'capital-loss', the log was taken** for these.Eg: total_data["capital-gain"] = np.log (1 + total_data ["capital-gain"]).If taken normalisation using maximum and minimum values instead of log for capital-loss and gain, there was decrease in accuracy.

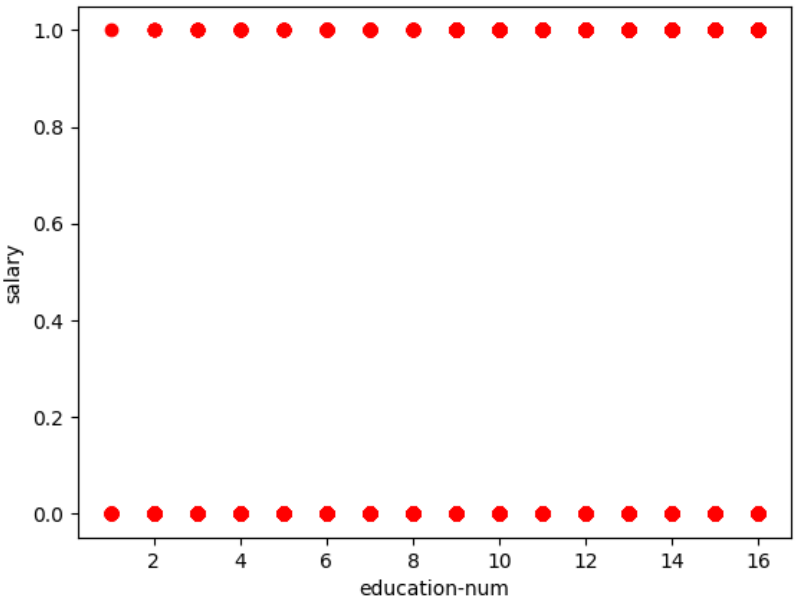
(d) Similary , to **feature "age" was scaled using factor of 10.**. Eg Age=Age/10.This reduced the range of this feature vector between 1-9 (at max).

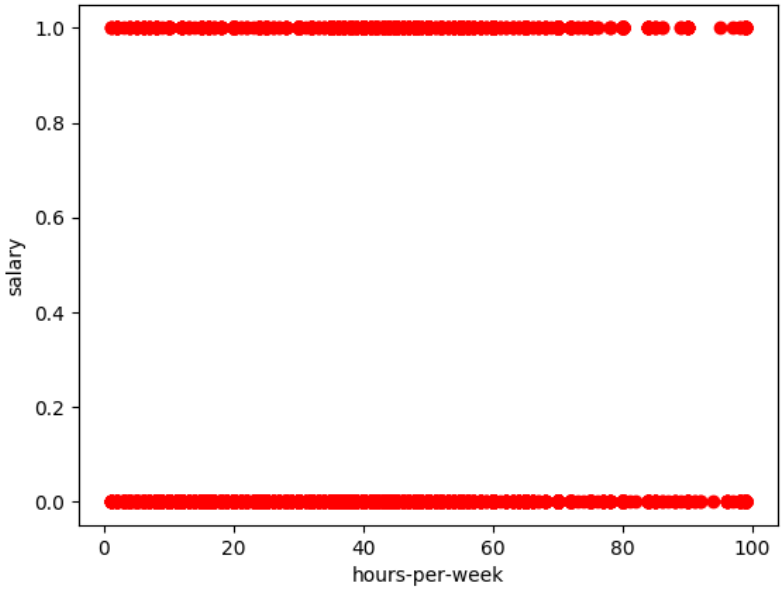
(e) Graphs were plotted between output and features.


1.0
0.8
0.6
0.4
0.2
0.0
0
20000
40000
60000
80000
100000
capital-gain


1.0
0.8
0.6
0.4
0.2
0.0
salary
0
1000
2000
3000
4000
capital-loss


1.0
0.8
0.6
0.4
0.2
0.0
salary
2
3
4
5
6
7
8
9
age

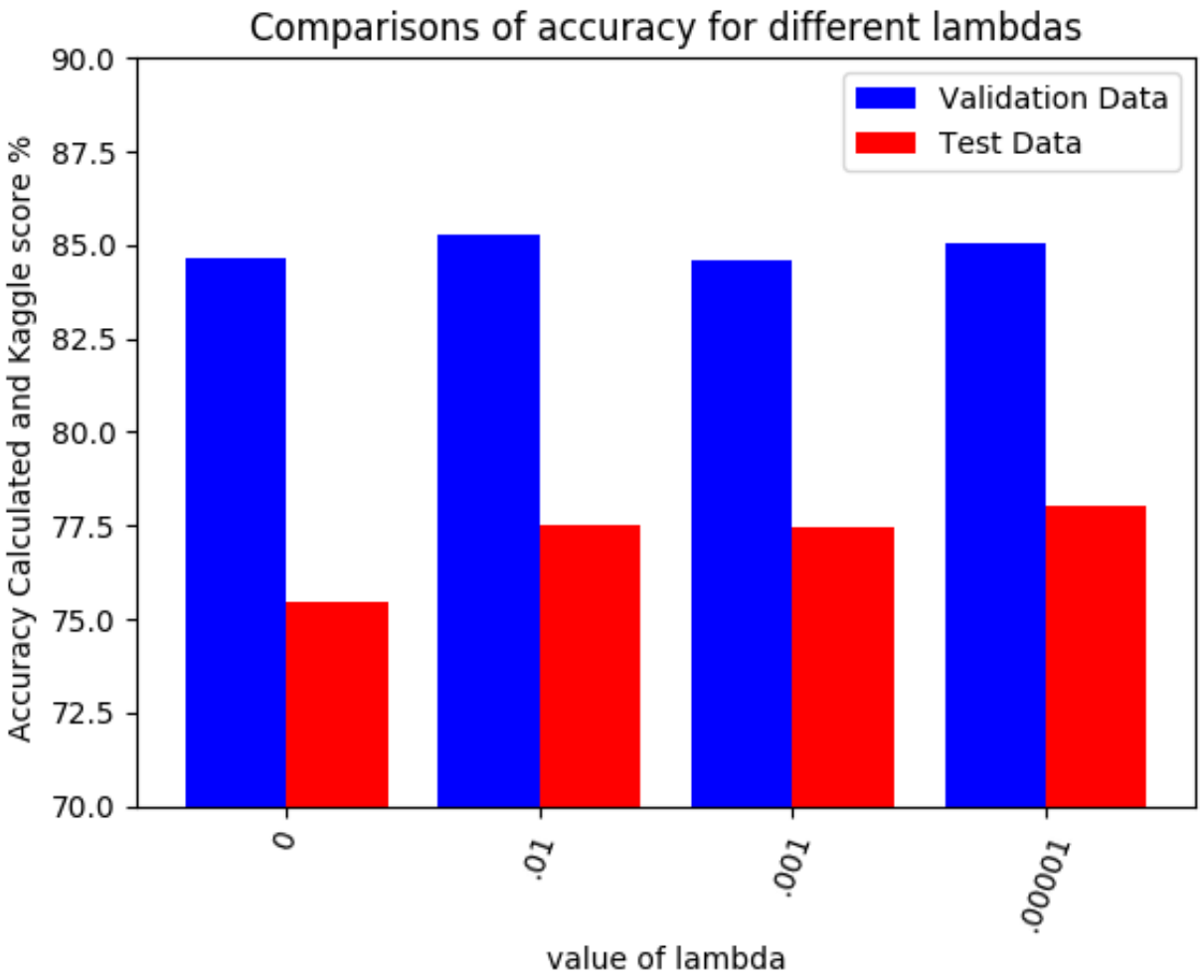




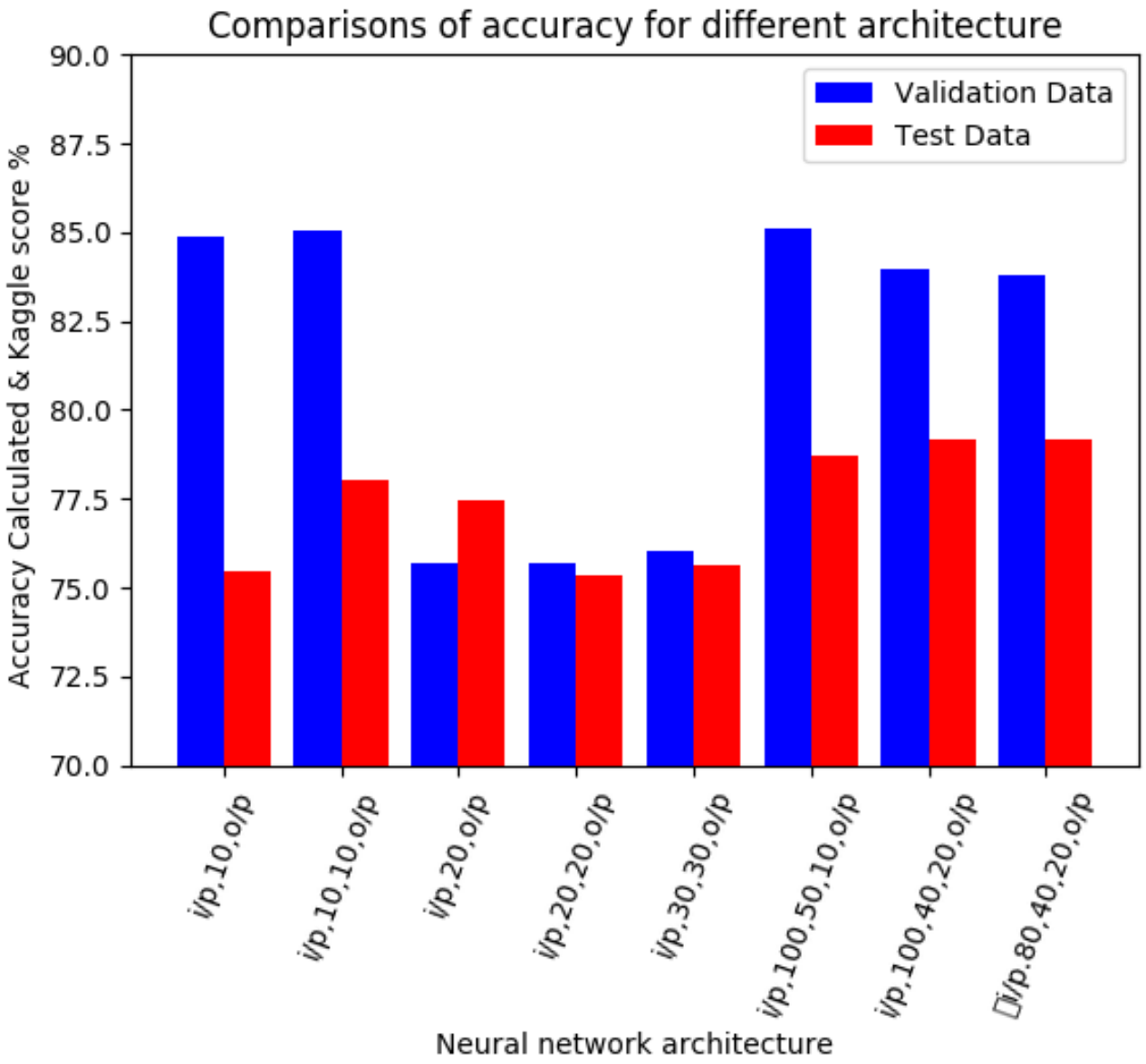
3. **<u>Implementation details</u>**:
   Activation function used : Sigmoid
   Input layer= number of features after hot encoding ie 106
   Output layer =1
   Neural Network Architecture: Input layer, 80,40, 20, output layer
   Learning rate-.02
   Lambda- .00001
   Num_iterations=500

4. **Input**: Input data was divided into training set and validation set. Also, since using complete training set in forward propagation was giving overflow error. One option was to reduce the learning rate and other option was to divide the input into batches and then use batches to update weights. Hence, neural network was implemented batch wise instead of complete data. In each iteration, input was divided in batch of 500 training samples to calculate error. This error was then back propagated to update the weights.

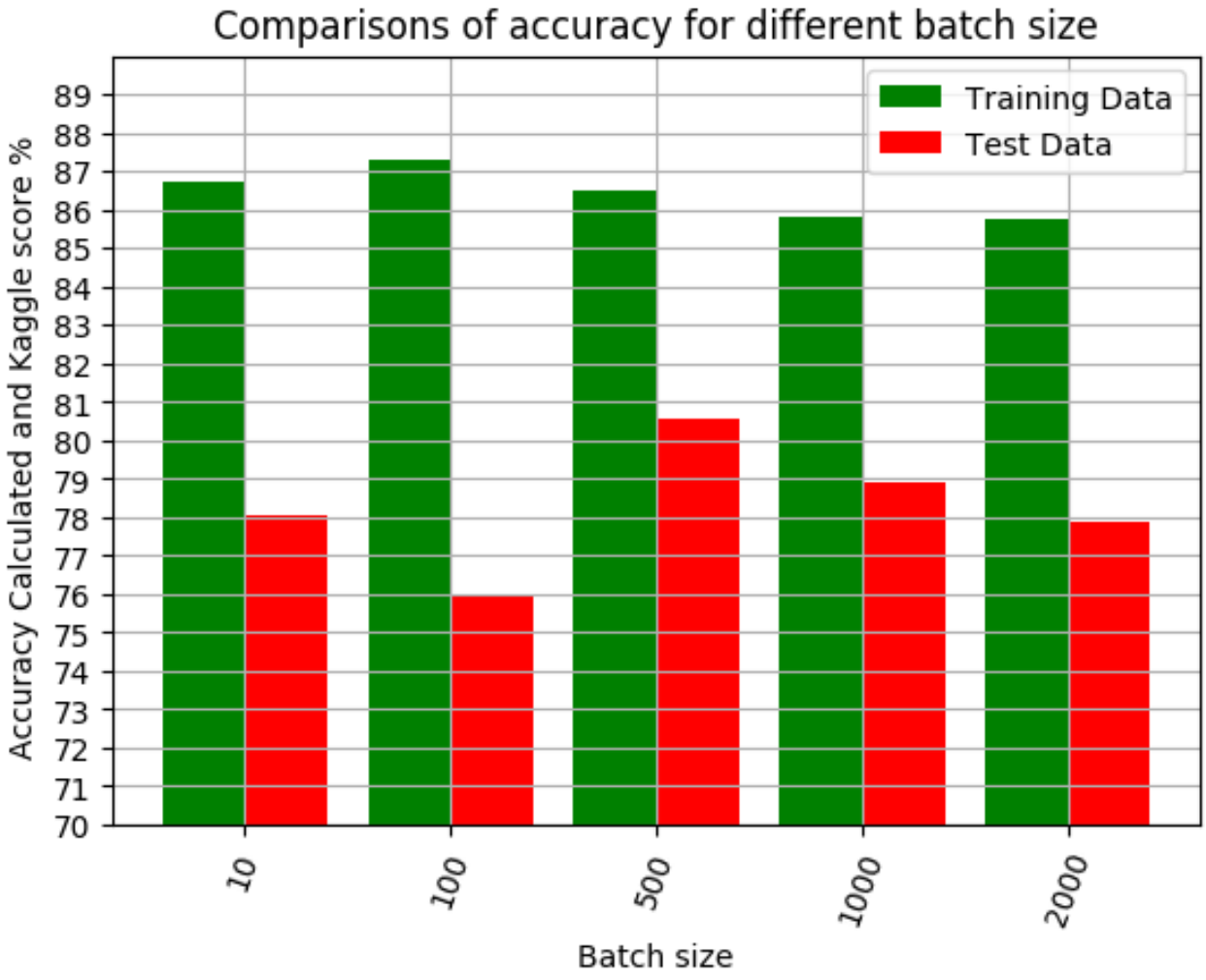
5. **Selection of lambda**: Different values of lambda were tried to find optimal solution. For higher values of lambda (more than .01),there is very less difference between error in k and k+1 iteration step ie very slow convergence rate towards optimal solution. Further, for the higher values of lambda, the accuracy in validation and test data was less. The value of lambda helps us to control overfitting. Increasing lambda results in less overfitting but also greater bias as seen in results.



6. **Selection of architecture**: Different neural network architectures were tried on validation and test data and accuracy was calculated based on validation y for validation set and kaggle score for test data. Adding more layers helped to increase accuracy on test data but it was taking more time and more iterations to converge to optimal solution.



7. **Selection of batch_size**: To select the batch_size for each iterative step, different batch size of input data was tried keeping number of iterations=500 and learning rate =.02 and architecture as input layer,80,40,20,output layer. With less batch_size of 10 and 100, we were getting better accuracy with test data but the time taken to complete was too long (appx 15-16 mins). As we increased the batch size to 500, the decrease in test data accuracy was very less as compared to time taken to complete all iterations which reduced to 2-3 minutes. Further, as batch size was kept 1000 and more at learning rate of .02, the error increased as algorithm was not converging with all output values turning zeros hence, learning rate was reduced to .002 and we saw validation accuracy increased but test data accuracy decreased. The reason for such behaviour can be as the program may be needing more number of steps for finding optimal solution with reduced learning rate.



8. **Working of code**:

(a) **Feature manipulation**:

(i) The continuous numeric valued features were scaled using log function

```
def normalize_data(total_data):
    result = total_data.copy()
    #feature “age” was scaled using factor of 10.. Eg Age=Age/10.
    #This reduced the range of this feature vector between 1-9 (at max).
    result["age"] = ((total_data["age"]) / (10))
    #to reduce wide range of values features ‘fnlwgt’, ‘capital-gain’,'hours-per-
    week" and ‘capital-loss’, the log was taken
    result["fnlwgt"]=np.log(1+total_data["fnlwgt"])
    result["capital-gain"]=np.log(1+total_data["capital-gain"])
    result["capital-loss"]=np.log(1+total_data["capital-loss"])
    result["hours-per-week"]=(np.log(1+total_data["hours-per-week"]) )
    return(result)
```

(ii) Categorical non numeric features were converted to numeric data using one hot encoding. A duplicate variable(new column) which represents one level of a categorical variable was created. Presence of a level is represent by 1 and

absence is represented by 0. For every level present, one dummy variable will be created.

```
def hot_encoding(total_data):
        #define column as category type with given only categories
        total_data["workclass"] = total_data["workclass"].astype('category',categories=[" Federal-gov"," Local-gov"," Never-worked"," Private"," Self-emp-inc"," Self-emp-not-inc"," State-gov"," Without-pay"])
        total_data["marital-status"] =total_data["marital-status"].astype('category',categories=[" Divorced"," Married-AF-spouse"," Married-civ-spouse"," Married-spouse-absent"," Never-married"," Separated"," Widowed"])
        total_data["occupation"] =total_data["occupation"].astype('category',categories=[" Adm-clerical"," Armed-Forces"," Craft-repair"," Exec-managerial"," Farming-fishing"," Handlers-cleaners"," Machine-op-inspct"," Other-service"," Priv-house-serv"," Prof-specialty"," Protective-serv"," Sales"," Tech-support"," Transport-moving"])
        total_data["relationship"] =total_data["relationship"].astype('category',categories=[" Wife"," Own-child"," Husband"," Not-in-family"," Other-relative"," Unmarried"])
        total_data["race"] =total_data["race"].astype('category',categories=[" Amer-Indian-Eskimo"," Asian-Pac-Islander"," Black"," Other"," White"])
        total_data["sex"] =total_data["sex"].astype('category',categories=[" Female"," Male"])
        total_data["native-country"] =total_data["native-country"].astype('category',categories=[" United-States"," Cambodia"," England"," Puerto-Rico"," Canada"," Germany"," Outlying-US(Guam-USVI-etc)"," India"," Japan"," Greece"," South"," China"," Cuba"," Iran"," Honduras"," Philippines"," Italy"," Poland"," Jamaica"," Vietnam"," Mexico"," Portugal"," Ireland"," France"," Dominican-Republic"," Laos"," Ecuador"," Taiwan"," Haiti"," Columbia"," Hungary"," Guatemala"," Nicaragua"," Scotland"," Thailand"," Yugoslavia"," El-Salvador"," Trinadad&Tobago"," Peru"," Hong"," Holand-Netherlands"])
        total_data["education"] =total_data["education"].astype('category',categories=[" Bachelors"," Some-college"," 11th"," HS-grad"," Prof-school"," Assoc-acdm"," Assoc-voc"," 9th"," 7th-8th"," 12th"," Masters"," 1st-4th"," 10th"," Doctorate"," 5th-6th"," Preschool"])

        cats=[" United-States"," Cambodia"," England"," Puerto-Rico"," Canada"," Germany"," Outlying-US(Guam-USVI-etc)"," India"," Japan"," Greece"," South"," China"," Cuba"," Iran"," Honduras"," Philippines"," Italy"," Poland"," Jamaica"," Vietnam"," Mexico"," Portugal"," Ireland"," France"," Dominican-Republic"," Laos"," Ecuador"," Taiwan"," Haiti"," Columbia"," Hungary"," Guatemala"," Nicaragua"," Scotland"," Thailand"," Yugoslavia"," El-Salvador"," Trinadad&Tobago"," Peru"," Hong"," Holand-Netherlands"]

        bitencode_workclass=pd.get_dummies(total_data.ix[0:,2],prefix='workclass')
        bitencode_education=pd.get_dummies(total_data.ix[0:, 4],prefix='education')
        bitencode_maritalstatus=pd.get_dummies(total_data.ix[0:,6],prefix='maritalstatus')
        bitencode_occupation=pd.get_dummies(total_data.ix[0:, 7],prefix='occupation')
        bitencode_relationship=pd.get_dummies(total_data.ix[0:, 8],prefix='relationship')
        bitencode_race=pd.get_dummies(total_data.ix[0:, 9],prefix='race')
        bitencode_sex=pd.get_dummies(total_data.ix[0:, 10],prefix='', prefix_sep='')
        bitencode_nativecountry=pd.get_dummies(total_data.ix[0:, 14],prefix='', prefix_sep='')
        bitencode_nativecountry = bitencode_nativecountry.reindex(columns=cats, fill_value=0)
        #remove columns containing categorical non-numeric attributes.
        total_data.drop(['id','workclass','education','marital-status','occupation','relationship','race','sex','native-country'],axis=1,inplace=True)
        X_Full = total_data.ix[0:, 0:6]
        #add hot encoded values to the data
        X_Full=pd.concat((X_Full,bitencode_workclass,bitencode_education,bitencode_maritalstatus,bitencode_occupation,bitencode_relationship,bitencode_race,bitencode_sex,bitencode_nativecountry),axis=1)
        return(X_Full)
```

(b) **Defining neural network architecture and parameters**

```
neural_net=[input_layer,80,40,20,output_layer]
max_iter=500 #max number of iterations
learning_rate=.02  #learning rate
lamda=0.00001  #value of lambda for regularizer
```

```
conv_condn=0.00000001  #condition for convergence
batch_size=500   #input data batch size for training the net
```

(c) **Initialise weights of neural net and add bias**

```
wt_matrix=[]
for i in range(1,L-1):
        wt_matrix.append(2*np.random.random((neural_net[i-1]+1,neural_net[i]+1))-
        1)
        wt_matrix.append(2*np.random.random((neural_net[i]+1,neural_net[i+1]))-1)
```

(d) **Training the neural network**.

(i) Feed forward propagation

```
for l in range(1,L):
#input to current layer=output of previous layer
        l_input=l_output[l-1]
#output of current layer=sigmoid(input*weight)
        l_output[l]=(activationfn(np.dot(l_input,wt_matrix[l-1])))
```

(ii) Calculate error for final output layer (Lth layer)

```
l_error[L-1]= Y_train[start:end] - l_output[L-1]
```

(iii) Calculating delta_error for final output layer (Lth layer)

```
delta_error[L-1]=l_error[L-1]*activationfnderivative(l_output[L-1])
```

(iv) Backward propagation from l=L-1 to 2

```
for l in range(L-2, 0 ,-1):
#error of l-1 layer=derivative of error of l layer * weight
        l_error[l] = delta_error[l+1].dot(wt_matrix[l].T)
#calculate derivate of error for current layer
        delta_error[l] = l_error[l]*activationfnderivative(l_output[l])
```

(v) Update weights of the matrix

```
for l in range(0,L- 1):
#update for (l-1)th layer = output of (l-1)th layer* delta_error of (l)th layer
        update=l_output[l].T.dot(delta_error[l+1]) + lamda*wt_matrix[l]/(2*total_m)
        wt_matrix[l] = wt_matrix[l] + learning_rate*update
```

(e) **Reading test data and predicting Y value using weights.**

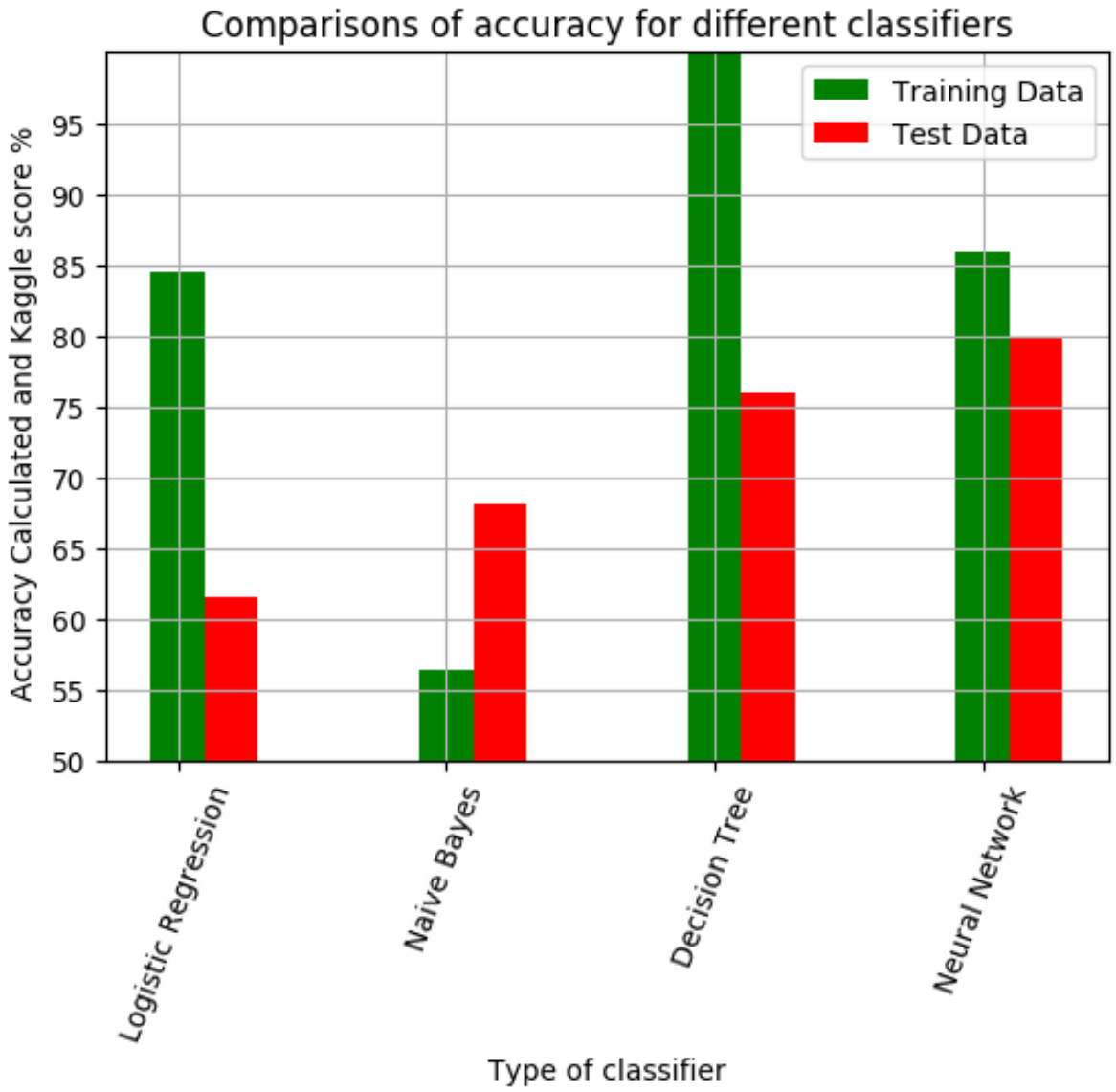
```
for l in range(1,L):
#input to current layer=output of previous layer
        l_input=l_output[l-1]
#output of current layer=sigmoid(input*weight)
        l_output[l]=(activationfn(np.dot(l_input,wt_matrix[l-1])))
```

(f) **Thresholding predicted Y to get 1 or 0.**

```
for i in range(0,len(output)):
if output[i] >= 0.5:
        output[i]=1.0
else:
        output[i]=0.0
```

9. **Comparison with other classifiers**: Different classifiers were tried for comparison. Each classifier was used and it's accuracy was calculated on training dataset. Also, model predicted was classifier was tested on test dataset and accuracy was calculated based on Kaggle score. It was found that model predicted by Decision tree gave almost 99.99% accuracy on training data set but accuracy reduced when that model was applied on validation and test data. However, model predicted by neural network gave almost same accuracy on training and validation dataset with very less decrease in accuracy on test data as shown on kaggle. Detailed comparison between accuracy achieved from different classifiers is as follows:

	Logistic Regression	**Naïve Bayes**	**Decision Tree**	**Neural Network**
Training set accuracy %	84.5633	56.3366	99.99333333	85.91
Test data set accuracy % (kaggle score)	61.56	68.161	75.907	80.022


Comparisons of accuracy for different classifiers
Training Data
Test Data
Accuracy Calculated and Kaggle score %
95
90
85
80
75
70
65
60
55
50
Logistic Regression
Naive Bayes
Decision Tree
Neural Network
Type of classifier