EC3066D - ARTIFICIAL INTELLIGENCE: THEORY AND PRACTICE

Implementation of Algorithms

Artificial Intelligence, Machine Learning and Deep Learning

Submitted by

Roll No	Names of Students
Narendiran S	M190412EC
M.R.K Siva Naga Sai	M190413EC
Rahul Kumar	M190616EE
K. Sai Kiran	M190296EE

Course Faculty: Ms. Lyla B. Das/Mr. Sreeram M.

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING
NATIONAL INSTITUTE OF TECHNOLOGY, CALICUT-673601
APRIL 2020

Contents

1 IMPLEMENTATION OF MACHINE LEARNING ALGORITHMS

1.1 GRADIENT DESCENT

1.1.1 Aim

To Implement Gradient Descent with different values of the Learning Parameter and show with help of a Simulated Figure, Convergence rates change.

1.1.2 Description

Optimization is a big part of machine learning. Almost every machine learning algorithm has an optimization algorithm at it's core.

Gradient descent is an optimization algorithm used to find the values of parameters (coefficients) of a function (f) that minimizes a cost function (cost). It is best used when the parameters cannot be calculated analytically (e.g. using linear algebra) and must be searched for by an optimization algorithm.

The procedure starts off with initial values for the coefficient or coefficients for the function. These could be 0.0 or a small random value.

$$\text{Co-efficient} = 0.0$$

The cost of the coefficients is evaluated by plugging them into the function and calculating the cost.

$$\text{cost} = \text{f(coefficient) or cost} = \text{evaluate(f(coefficient))}$$

The derivative of the cost is calculated. The derivative is a concept from calculus and refers to the slope of the function at a given point. We need to know the slope so that we know the direction (sign) to move the coefficient values in order to get a lower cost on the next iteration.

$$\text{delta} = \text{derivative(cost)}$$

Now that we know from the derivative which direction is downhill, we can now update the coefficient values. A learning rate parameter (alpha) must be specified that controls how much the coefficients can change on each update.

$$\text{coefficient} = \text{coefficient} - (\text{alpha} * \text{delta})$$

This process is repeated until the cost of the coefficients (cost) is 0.0 or close enough to zero to be good enough.

You can see how simple gradient descent is. It does require you to know the gradient of your cost function or the function you are optimizing, but besides that, it's very straightforward

1.1.3 Dataset used

The dataset used consist of 100 data points and two features. The two features are exam scores in two different subject. The output variable is either 0 or 1 representing Not Getting Admitted or Getting Admitted.

1.1.4 Methodology To Implement This Algorithm

Gradient descent is a generic optimization algorithm used in many machine learning algorithms. It iteratively tweaks the parameters of the model in order to minimize the cost function. The steps of gradient descent is outlined below.

1. We first initialize the model parameters with some random values. This is also called as random initialization.

2. Now we need to measure how the c st function changes with change in it's parameters. Therefore we compute the partial derivatives of the cost function w.r.t to the parameters $\theta_0, \theta_1, ..., \theta_n$.

$$\frac{\partial J(\theta)}{\partial \theta_0} = \frac{1}{m}\sum_{i=1}^{m}(h(x^i) - y^i)$$

$$\frac{\partial J(\theta)}{\partial \theta_1} = \frac{1}{m}\sum_{i=1}^{m}(h(x^i) - y^i)x_1^i$$

   Similarly, the partial derivative of the cost function w.r.t to any parameter can be denoted by

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m}\sum_{i=1}^{m}(h(x^i) - y^i)x_j^i$$

   We can compute the partial derivatives for all parameters at once using

$$\begin{bmatrix} \frac{\partial J(\theta)}{\partial \theta_0} \\ \frac{\partial J(\theta)}{\partial \theta_1} \\ \vdots \\ \frac{\partial J(\theta)}{\partial \theta_n} \end{bmatrix} = \frac{1}{m}x^T(h(x) - y)$$

   where h(x) is

$$h(x) = \theta_0 + \theta_1 x_1 + ... + \theta_n x_n$$

3. After computing the derivative we update the parameters as given below

$$\theta_0 = \theta_0 - \frac{\alpha}{m}\sum_{i=1}^{m}(h(x^i) - y^i)$$

$$\theta_1 = \theta_1 - \frac{\alpha}{m}\sum_{i=1}^{m}(h(x^i) - y^i)x_1^i$$

   where α is the learning parameter.

We can update all the parameters at once using,

$$\begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix} - \alpha \begin{bmatrix} \frac{\partial J(\theta)}{\partial \theta_0} \\ \frac{\partial J(\theta)}{\partial \theta_1} \\ \vdots \\ \frac{\partial J(\theta)}{\partial \theta_n} \end{bmatrix}$$

We repeat the steps 2,3 until the cost function converges to the minimum value. If the value of $\alpha$is too small, the cost function takes larger time to converge. If α is too large, gradient descent may overshoot the minimum and may finally fail to converge.

To demonstrate the gradient descent algorithm, we initialize the model parameters with 0. The equation becomes Y = 0. Gradient descent algorithm now tries to update the value of the parameters so that we arrive at the best fit line.

1.1.5 Python code Implementation

```python
#!/usr/bin/env python
import numpy as np
import matplotlib.pyplot as plt

def Scale(x):
    return (x-np.min(x))/(np.max(x) -  np.min(x))
def sigmoid(Z):
    return 1.0 / (1 + np.exp(-Z))
def hyp(X_,theta):
    R=np.dot(X_,theta)
    return sigmoid(R)
def cost(X_,Y_,theta):
    m=Y_.shape[0]
    p1 = np.multiply(Y_,np.log(hyp(X_,theta)))
    p2 = np.multiply(1-Y_,np.log(1- hyp(X_,theta)))
    return (-1.0/m) * np.sum( p1 + p2 )
def update_theta(X_,Y_,theta,alpha):
    m=Y_.shape[0]
    return((alpha/m)*np.dot(X_.T,(hyp(X_,theta) - Y_)))
def gradient_descent(X_,Y_,alpha = 0.01, num_iter = 100):
    theta = np.zeros((X_.shape[1],1))
    cost_ =[]
    for i in range(num_iter):
        theta = theta - update_theta(X_,Y_,theta,alpha)
        cost_.append(cost(X_,Y_,theta))
    return theta,cost_

DataSet = np.genfromtxt('../../../Datasets/exam_vs_adm.csv', \
delimiter=',')
X= DataSet[:,:2]
Y = DataSet[:,2]
X=Scale(X)
Y=Y.reshape(-1,1)
X = np.hstack((np.ones((X.shape[0],1)),X))

fig,a =  plt.subplots(4,2,figsize=(15,15))
alphas = np.asarray([0.1,1.5,2.5,5,10,15,20,50]).reshape(4,2)

for i in range(4):
    for j in range(2):
        theta_final, cost_ = gradient_descent(X,Y,alphas[i,j],100)
        a[i][j].plot(cost_)
        a[i][j].set_title('alpha = '+str(alphas[i,j]))
        a[i][j].xlabel = 'No. of iterations'
plt.show()
```

1.1.6 Results

The Convergence rate change for different learning rate(α) values are shown below.

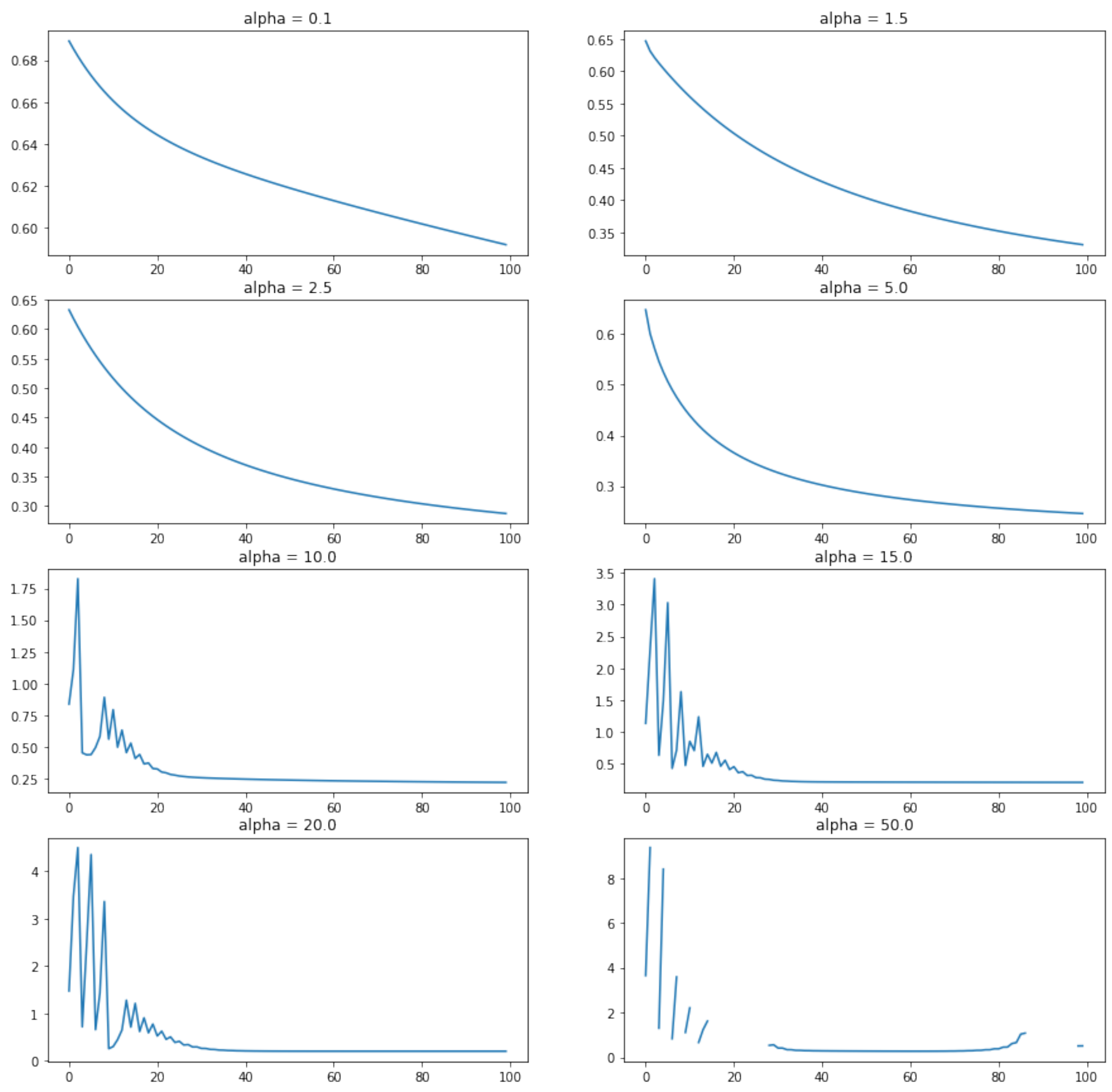


Figure 1: Testing data output

1.2 LINEAR REGRESSION

1. With One Parameter
2. With more than One Parameter

1.2.1 Aim

To Implement Linear Regression Model with One Parameter and with more than one parameter.

1.2.2 Description

Optimization is a big part of machine learning. Almost every machine learning algorithm has an optimization. Similarly we have for Linear regression.

Linear Regression is usually the first machine learning algorithm. It is a simple model but it lays the foundation for other machine learning algorithms. It is a very powerful technique and can be used to understand the factors that influence profitability. It can be used to forecast sales in the coming months by analysing the sales data for previous months. It can also be used to gain various insights about customer behaviour. By the end of the blog we will build a model which looks like the below picture i.e, determine a line which best fits the data.

The objective of a linear regression model is to find a relationship between one or more features(independent variables) and a continuous target variable(dependent variable). When there is only feature it is called Uni-variate Linear Regression and if there are multiple features, it is called Multiple Linear Regression.

The linear regression model can be represented by the following equation.

$$Y = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + ... + \theta_n x_n$$

- Y is the predicted value
- θ_0 is the bias term
- $\theta_1, \theta_2, ..., \theta_n$ are the model parameters
- $x_1, x_2, ..., x_n$ are the feature values

The above hypothesis can also be represented by

$$Y = \theta^T x$$

- θ is the model's parameter vector including the bias term θ_0
- x is the feature vector with $x_0 = 1$

1.2.3 Dataset used

1. With One Parameter

   The dataset used consist of 48 data points and a single features. The feature is Production in thousands of pairs of Hosiery items. The output variable is the cost for production of these items.

2. With more than One Parameter

   The dataset used consist of 44 data points and two features. The features are fish weight and scale size. The output variable is the length of the fish.

1.2.4 Methodology To Implement This Algorithm

Training of the model here means to find the parameters so that the model best fits the data. By following the procedure, we can find the best fit line for our Linear regression model.

The line for which the error between the predicted values and the observed values is minimum is called the best fit line or the regression line. These errors are also called as residuals. The residuals can be visualized by the vertical lines from the observed data value to the regression line.

To define and measure the error of our model we define the cost function as the sum of the squares of the residuals. The cost function is denoted by

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h(x^i) - y^i)^2$$

where the hypothesis function h(x) is denoted by

$$h(x) = \theta_0 + \theta_1 x_1 + \ldots + \theta_n x_n$$

and m is the total number of training examples in our data-set.

1.2.5 Python code Implementation

1. With One Parameter

```
#!/usr/bin/env python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

Dfile = '../../../Datasets/millcost.csv'
dataset = np.genfromtxt(Dfile, delimiter = ',')
X = dataset[:,1].reshape(-1,1)
Y = dataset[:,2].reshape(-1,1)

plt.plot(X,Y,'g*')
plt.xlabel('Production (Thousands of dozens of pairs)')
plt.ylabel('Cost($1000s)')

lr = LinearRegression().fit(X,Y)
print('Coefficient and constant: ', lr.coef_[0], lr.intercept_)
Ynew = lr.coef_[0] * X + lr.intercept_
plt.plot(X,Y,'g*')
plt.plot(X,Ynew,'r')
plt.xlabel('Production (Thousands of dozens of pairs)')
plt.ylabel('Cost($1000s)')

print('Mean squared Error: ', mean_squared_error(Y,Ynew))
```

2. With more than One Parameter

```python
#!/usr/bin/env python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

def Scale(x):
    return (x-np.min(x))/(np.max(x) -  np.min(x))
def hyp(X_,theta):
    R=np.dot(X_,theta)
    return R
def cost(X_,Y_,theta):
    m=Y_.shape[0]
    return (1/(2.0*m))*np.sum((hyp(X_,theta) - Y_)**2)
def update_theta(X_,Y_,theta,alpha):
    m=Y_.shape[0]
    return((alpha/m)*np.dot(X_.T,(hyp(X_,theta) - Y_)))
def gradient_descent(X_,Y_,alpha = 0.01, num_iter = 100):
    theta = np.zeros((X_.shape[1],1))
    print('Inital Cost : ',cost(X_,Y_,theta))
    cost_ =[]
    for i in range(num_iter):
        theta = theta - update_theta(X_,Y_,theta,alpha)
        cost_.append(cost(X_,Y_,theta))
    print('Final Cost : ',cost(X_,Y_,theta))
    return theta,cost_

DataSet = np.genfromtxt('../../../Datasets/fish_length.csv', \
delimiter=',')
X= DataSet[:,1:3]
Y = DataSet[:,3]
Y= Scale(Y)
Y=Y.reshape(-1,1)
X = Scale(X)
X = np.hstack((np.ones((X.shape[0],1)),X))
plt.plot(X[:,1],Y,'g*')
plt.show()
plt.plot(X[:,2],Y,'r.')
plt.show()

theta_final, cost_ = gradient_descent(X,Y,0.5,100000)
print('Final Coefficient and intercept: ',\
theta_final[1:],theta_final[0][0])
plt.plot(cost_)
plt.show()

lr = LinearRegression()
lr.fit(X[:,1:], Y)
print('Final Coefficient and intercept: ',lr.coef_,lr.intercept_)
```

1.2.6 Results

Thus a Linear Regression Model with One Parameter and with more than one parameter has been implemented. The output is visible for Linear Regression with one parameter.

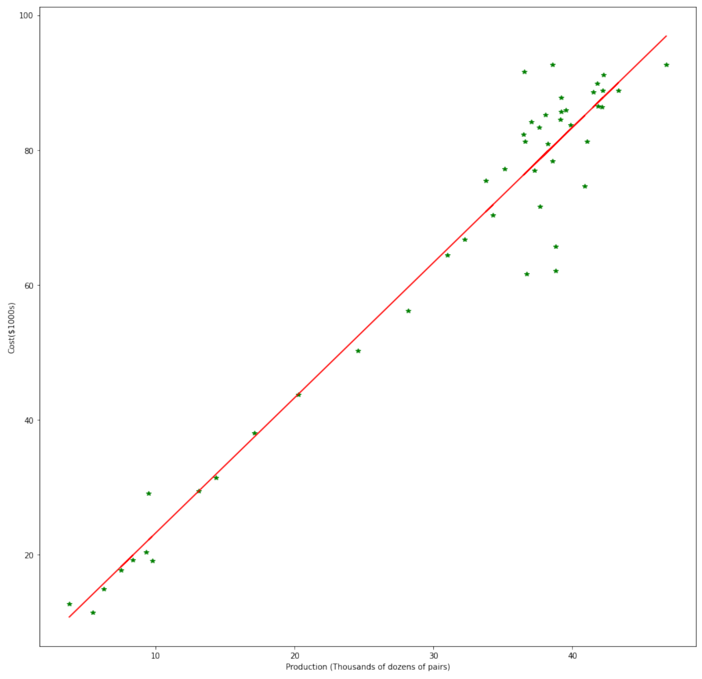


Figure 2: Linear Regression with One parameter

1.3 LOGISTIC REGRESSION

1.3.1 Aim

To Implement Logistic Regression Algorithm.

1.3.2 Description

Logistic regression is a fundamental classification technique. It belongs to the group of linear classifiers and is somewhat similar to polynomial and linear regression.

Logistic regression is fast and relatively uncomplicated, and it's convenient for you to interpret the results. Although it's essentially a method for binary classification, it can also be applied to multi-class problems. There are 2 kinds of Logistic Regression. They are:

- Single-variate logistic regression is the most straightforward case of logistic regression. There is only one independent variable (or feature), which is X = X.
- Multi-variate logistic regression has more than one input variable.

1.3.3 Dataset used

The dataset used consist of 100 data points and two features. The two features are exam scores in two different subject. The output variable is either 0 or 1 representing Not Getting Admitted or Getting Admitted.

1.3.4 Methodology To Implement This Algorithm

There are several general steps we taken when we're preparing this classification model. They are:

1. Import packages, functions, and classes
2. Get data to work with and, if appropriate, transform it
3. Create a classification model and train (or fit) it with your existing data
4. Evaluate your model to see if its performance is satisfactory

1.3.5 Python code Implementation

```python
#!/usr/bin/env python
import numpy as np
import matplotlib.pyplot as plt

def Scale(x):
    return (x-np.min(x))/(np.max(x) -  np.min(x))
def sigmoid(Z):
    return 1.0 / (1 + np.exp(-Z))
def hyp(X_,theta):
    R=np.dot(X_,theta)
    return sigmoid(R)
def cost(X_,Y_,theta):
    m=Y_.shape[0]
    p1 = np.multiply(Y_,np.log(hyp(X_,theta)))
    p2 = np.multiply(1-Y_,np.log(1- hyp(X_,theta)))
    return (-1.0/m) * np.sum(p1 + p2)
def update_theta(X_,Y_,theta,alpha):
    m=Y_.shape[0]
    return((alpha/m)*np.dot(X_.T,(hyp(X_,theta) - Y_)))
def gradient_descent(X_,Y_,alpha = 0.01, num_iter = 100):
    theta = np.zeros((X_.shape[1],1))
    print('Inital Cost : ',cost(X_,Y_,theta))
    cost_ =[]
    for i in range(num_iter):
        theta = theta - update_theta(X_,Y_,theta,alpha)
        cost_.append(cost(X_,Y_,theta))
    print('Final Cost : ',cost(X_,Y_,theta))
    return theta,cost_
def accuracy(X_,Y_,theta):
    m=Y_.shape[0]
    return (100.0 * np.sum((hyp(X_,theta)>0.5) == Y_)) / m

DataSet = np.genfromtxt('../../../Datasets/exam_vs_adm.csv', \
delimiter=',')
X= Scale(DataSet[:,:2])
Y = DataSet[:,2].reshape(-1,1)
xpos=X[(Y==1)[:,0]]
xneg=X[(Y==0)[:,0]]
plt.plot(xpos[:,0],xpos[:,1],'g*')
plt.plot(xneg[:,0],xneg[:,1],'b.')
plt.show()
X = np.hstack((np.ones((X.shape[0],1)),X))
theta_final, cost_ = gradient_descent(X,Y,0.01,8500)
print('Final Coefficient and intercept: ',\
theta_final[1:],theta_final[0][0])
print('Accuracy: ',accuracy(X,Y,theta_final),'%')
plt.plot(cost_)
plt.show()
```

1.3.6 Results

Thus a Logistic Regression Model has been implemented. The output is visible for Logistic Regression.

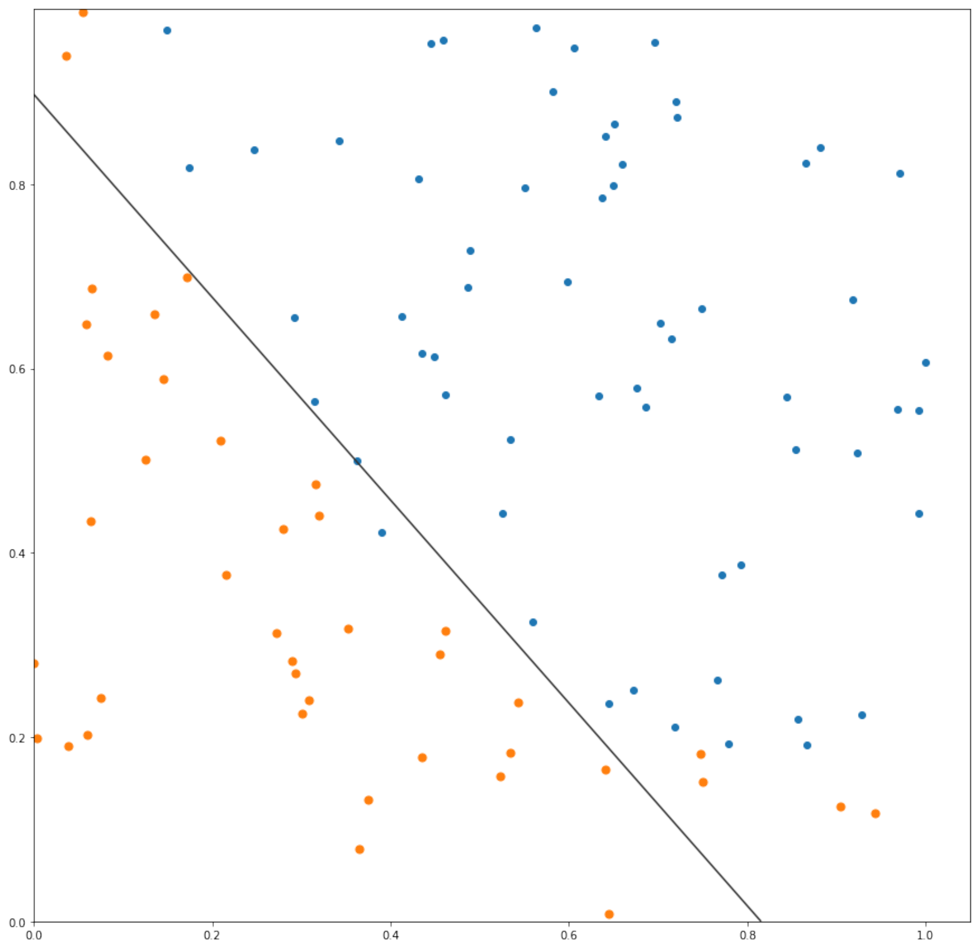


Figure 3: Logistic Regression separating data points

1.4 DECISION TREES

1.4.1 Aim

To Implement Decision trees Algorithm.

1.4.2 Description

Decision tree is one of the predictive modelling approaches used in statistics, data mining and machine learning.

A decision tree is a flowchart-like structure in which each internal node represents a test on a feature (e.g. whether a coin flip comes up heads or tails) , each leaf node represents a class label (decision taken after computing all features) and branches represent conjunctions of features that lead to those class labels. The paths from root to leaf represent classification rules Decision trees are constructed via an algorithmic approach that identifies ways to split a data set based on different conditions. It is one of the most widely used and practical methods for supervised learning. Decision Trees are a non-parametric supervised learning method used for both classification and regression tasks.

Tree models where the target variable can take a discrete set of values are called classification trees. Decision trees where the target variable can take continuous values (typically real numbers) are called regression trees. Classification And Regression Tree (CART) is general term for this.

1.4.3 Dataset used

The dataset used consist of 748 data points and four features. The features are recency of blood donation, frequency of blood donation, quantity in cc and months when blood was donated . The output variable is either 0 or 1 representing Will not donate blood or Will donate blood.

1.4.4 Methodology To Implement This Algorithm

- Get list of rows (dataset) which are taken into consideration for making decision tree (recursively at each nodes).
- Calculate uncertainty of our dataset or Gini impurity or how much our data is mixed up etc.
- Generate list of all question which needs to be asked at that node.
- Partition rows into True rows and False rows based on each question asked.
- Calculate information gain based on gini impurity and partition of data from previous step.
- Update highest information gain based on each question asked.
- Update best question based on information gain (higher information gain).
- Divide the node on best question. Repeat again from step 1 again until we get pure node (leaf nodes).

1.4.5 Python code Implementation

```
#!/usr/bin/env python
import numpy as np
from sklearn import tree

np.random.seed(seed=4)

dataLoc = '../../../Datasets/transfusion.csv'
DataSet = np.genfromtxt(dataLoc , delimiter=',',skip_header=1)
X= DataSet[:,1:4]
```

```
Y = DataSet[:,4]
Y=Y.reshape(-1,1)
percent = 85
split = int((percent/100.0) * X.shape[0])
rand_indx = np.random.choice(range(X.shape[0]), X.shape[0], \
replace=False)
Xtrain,Ytrain = X[rand_indx[:split],:],Y[rand_indx[:split],:]
Xtest,Ytest = X[rand_indx[split:],:],Y[rand_indx[split:],:]

DST = tree.DecisionTreeClassifier(criterion='entropy')
fitted_DST =DST.fit(Xtrain,Ytrain)


print('Training accuracy : ',fitted_DST.score(Xtrain,Ytrain)*100.0)
print('Testing accuracy : ',fitted_DST.score(Xtest,Ytest)*100.0)



print(DST.feature_importances_)
```

1.4.6 Results

Thus Decision Tree has been implemented. The Decision tree is visible below.

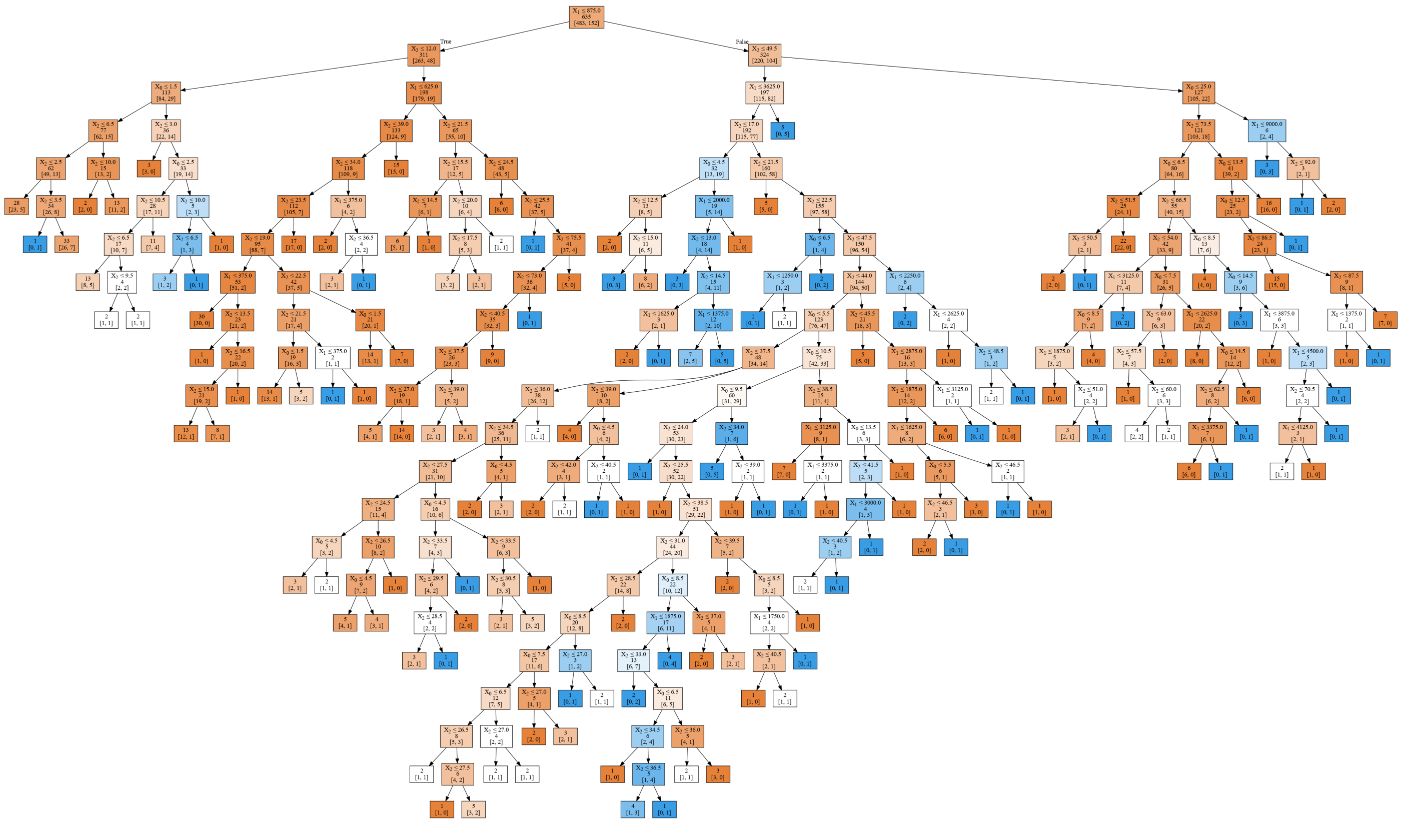


Figure 4: Decision Tree

1.5 NAIVE BAYES ALGORITHM

1.5.1 Aim

To Implement Naive Bayes Algorithm.

1.5.2 Description

It is a classification technique based on Bayes' Theorem with an assumption of independence among predictors. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature.

Naive Bayes model is easy to build and particularly useful for very large data sets. Along with simplicity, Naive Bayes is known to outperform even highly sophisticated classification methods. Bayes theorem provides a way of calculating posterior probability P(c—x) from P(c), P(x) and P(x—c). Look at the equation below:

$$\underbrace{P(c\,|\,x)}_{\text{Posterior Probability}} = \frac{\overbrace{P(x\,|\,c)}^{\text{Likelihood}}\,\overbrace{P(c)}^{\text{Class Prior Probability}}}{\underbrace{P(x)}_{\text{Predictor Prior Probability}}}$$

$$P(c\,|\,X) = P(x_1\,|\,c)\times P(x_2\,|\,c)\times\cdots\times P(x_n\,|\,c)\times P(c)$$

- P(c|x) is the posterior probability of class (c,target) given predictor (x,attributes).
- P(c) is the prior probability of class.
- P(x|c) is the likelihood which is the probability of predictor given class.
- P(x) is the prior probability of predictor.

  DATASET USED:

1.5.3 Dataset used

The dataset used consist of 748 data points and four features. The features are recency of blood donation, frequency of blood donation, quantity in cc and months when blood was donated . The output variable is either 0 or 1 representing Will not donate blood or Will donate blood.

1.5.4 Methodology To Implement This Algorithm

First we will develop each piece of the algorithm in this section, then we will tie all of the elements together into a working implementation applied to a real dataset in the next section.

This Naive Bayes tutorial is broken down into 5 parts:
Step 1: Separate By Class.
Step 2: Summarize Dataset.
Step 3: Summarize Data By Class.
Step 4: Gaussian Probability Density Function.
Step 5: Class Probabilities.

1.5.5 Python code Implementation

```
#!/usr/bin/env python
import numpy as np
from sklearn.naive_bayes import GaussianNB

def Scale(x):
    return (x-np.min(x))/(np.max(x) -  np.min(x))
datasetLoc = '../../../Datasets/transfusion.csv'
DataSet = np.genfromtxt(datasetLoc, delimiter=',',skip_header=1)
X= DataSet[:,1:4]
Y = DataSet[:,4]
Y=Y.reshape(-1,1)
X = np.hstack((np.ones((X.shape[0],1)),X))
percent = 85
split = int((percent/100.0) * X.shape[0])
rand_indx = np.random.choice(range(X.shape[0]), X.shape[0], \
replace=False)
Xtrain,Ytrain = X[rand_indx[:split],:],Y[rand_indx[:split],:]
Xtest,Ytest = X[rand_indx[split:],:],Y[rand_indx[split:],:]
NaivB = GaussianNB()
fitted_NaivB =NaivB.fit(Xtrain,Ytrain.ravel())
print('Training accuracy : ',fitted_NaivB.score(Xtrain,Ytrain)*100.0)
print('Testing accuracy : ',fitted_NaivB.score(Xtest,Ytest)*100.0)
```

1.5.6 Results

Thus the Naive Bayes Algorithm has been implemented

1.6 RANDOM FOREST

1.6.1 Aim

To Implement Random Forest Algorithm.

1.6.2 Description

Random forest is a supervised learning algorithm which is used for both classification as well as regression. But however, it is mainly used for classification problems. As we know that a forest is made up of trees and more trees means more robust forest. Similarly, random forest algorithm creates decision trees on data samples and then gets the prediction from each of them and finally selects the best solution by means of voting. It is an ensemble method which is better than a single decision tree because it reduces the over-fitting by averaging the result.

1.6.3 Dataset used

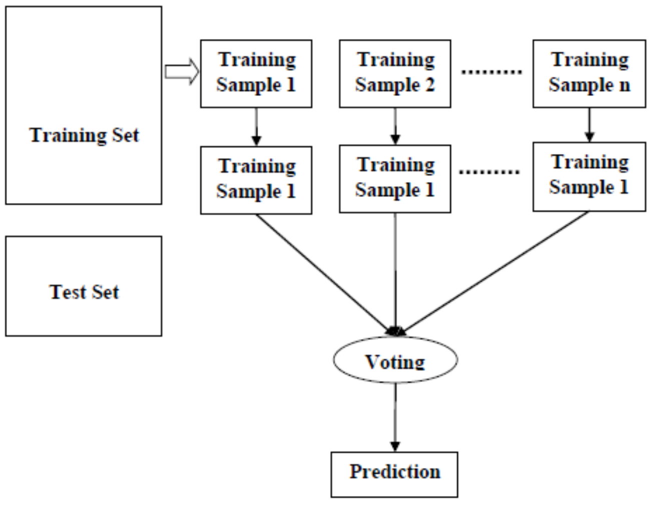
The dataset used consist of 748 data points and four features. The features are recency of blood donation, frequency of blood donation, quantity in cc and months when blood was donated . The output variable is either 0 or 1 representing Will not donate blood or Will donate blood.

1.6.4 Methodology To Implement This Algorithm

We can understand the working of Random Forest algorithm with the help of following steps:

1. First, start with the selection of random samples from a given dataset.
2. Next, this algorithm will construct a decision tree for every sample. Then it will get the prediction result from every decision tree.
3. In this step, voting will be performed for every predicted result.
4. At last, select the most voted prediction result as the final prediction result.

The following diagram will illustrate its working:



1.6.5 Python code Implementation

```
#!/usr/bin/env python
import numpy as np
from sklearn.ensemble import RandomForestClassifier
np.random.seed(seed=41)
```

```
def Scale(x):
    return (x-np.min(x))/(np.max(x) -  np.min(x))

datasetLoc = '../../../Datasets/transfusion.csv'
DataSet = np.genfromtxt(datasetLoc, delimiter=',',skip_header=1)
X= DataSet[:,1:4]
Y = DataSet[:,4]
Y=Y.reshape(-1,1)
percent = 85
split = int((percent/100.0) * X.shape[0])
rand_indx = np.random.choice(range(X.shape[0]), X.shape[0],\
 replace=False)

Xtrain,Ytrain = X[rand_indx[:split],:],Y[rand_indx[:split],:]
Xtest,Ytest = X[rand_indx[split:],:],Y[rand_indx[split:],:]

RForest=RandomForestClassifier(n_estimators=200,max_depth=4,max_features='log2')
fitted_RForest =RForest.fit(Xtrain,Ytrain.ravel())

print('Training accuracy : ',\
fitted_RForest.score(Xtrain,Ytrain)*100.0)
print('Testing accuracy : ',\
fitted_RForest.score(Xtest,Ytest)*100.0)

print(fitted_RForest.feature_importances_)
```

1.6.6 Results

Thus the Random Forest Algorithm has been implemented

1.7 ADABOOST ALGORITHM

1.7.1 Aim

To Implement ADABOOST Algorithm.

1.7.2 Description

AdaBoost is best used to boost the performance of decision trees on binary classification problems. It was originally called AdaBoost.M1 by the authors of the technique Freund and Schapire. More recently it may be referred to as discrete AdaBoost because it is used for classification rather than regression.

AdaBoost can be used to boost the performance of any machine learning algorithm. It is best used with weak learners. These are models that achieve accuracy just above random chance on a classification problem. The most suited and therefore most common algorithm used with AdaBoost are decision trees with one level. Because these trees are so short and only contain one decision for classification, they are often called decision stumps.

Each instance in the training dataset is weighted. The initial weight is set to:

$$\text{weight(xi)} = 1/\text{n}$$

Where xi is the ith training instance and n is the number of training instances.

1.7.3 Dataset used

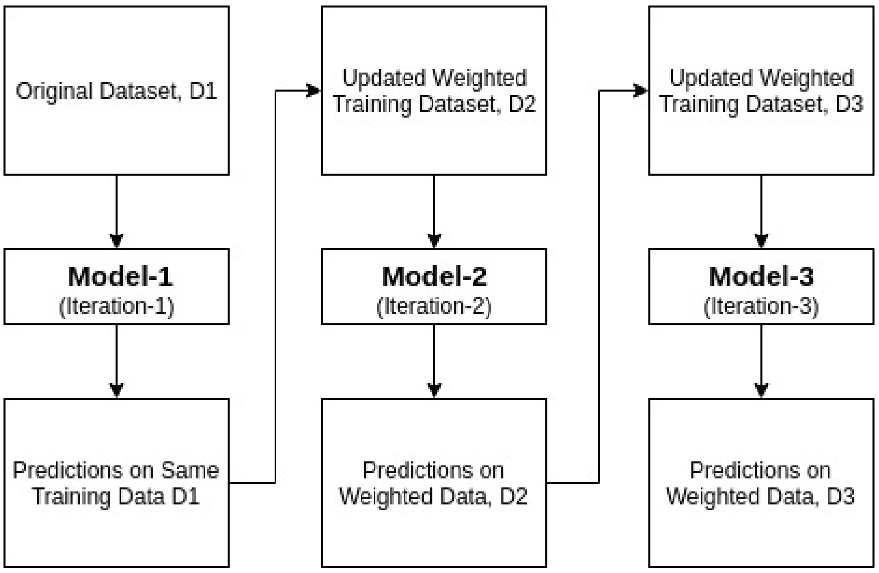
The dataset used consist of 748 data points and four features. The features are recency of blood donation, frequency of blood donation, quantity in cc and months when blood was donated . The output variable is either 0 or 1 representing Will not donate blood or Will donate blood.

1.7.4 Methodology To Implement This Algorithm

We can understand the working of AdaBoost algorithm with the help of following steps:

1. Initially, Adaboost selects a training subset randomly.
2. It iteratively trains the AdaBoost machine learning model by selecting the training set based on the accurate prediction of the last training.
3. It assigns the higher weight to wrong classified observations so that in the next iteration these observations will get the high probability for classification.
4. Also, It assigns the weight to the trained classifier in each iteration according to the accuracy of the classifier. The more accurate classifier will get high weight.
5. This process iterate until the complete training data fits without any error or until reached to the specified maximum number of estimators.
6. To classify, perform a "vote" across all of the learning algorithms you built.

The following diagram will illustrate its working:


Original Dataset, D1
Updated Weighted Training Dataset, D2
Updated Weighted Training Dataset, D3
Model-1 (Iteration-1)
Model-2 (Iteration-2)
Model-3 (Iteration-3)
Predictions on Same Training Data D1
Predictions on Weighted Data, D2
Predictions on Weighted Data, D3

1.7.5 Python code Implementation

```
#!/usr/bin/env python
import numpy as np
from sklearn.ensemble import AdaBoostClassifier

def Scale(x):
    return (x-np.min(x))/(np.max(x) -  np.min(x))
datasetLoc = 'Datasets/transfusion.csv'
DataSet = np.genfromtxt(datasetLoc, delimiter=',',skip_header=1)
X= DataSet[:,1:4]
Y = DataSet[:,4]
Y=Y.reshape(-1,1)
percent = 85
split = int((percent/100.0) * X.shape[0])
rand_indx = np.random.choice(range(X.shape[0]), X.shape[0], replace=False)
Xtrain,Ytrain = X[rand_indx[:split],:],Y[rand_indx[:split],:]
Xtest,Ytest = X[rand_indx[split:],:],Y[rand_indx[split:],:]

Ada_C = AdaBoostClassifier(n_estimators=250)
fitted_Ada_C =Ada_C.fit(Xtrain,Ytrain.ravel())

print('Training accuracy : ',fitted_Ada_C.score(Xtrain,Ytrain)*100.0)
print('Testing accuracy : ',fitted_Ada_C.score(Xtest,Ytest)*100.0)
```

1.7.6 Results

Thus the Adaboost Algorithm has been implemented

1.8 SUPPORT VECTOR MACHINES

1.8.1 Aim

To Implement Support Vector Machines Algorithm- with Linearly separable and Non- Linearly Separable cases.

1.8.2 Description

SVM is an exciting algorithm and the concepts are relatively simple. The classifier separates data points using a hyperplane with the largest amount of margin. That's why an SVM classifier is also known as a discriminative classifier. SVM finds an optimal hyperplane which helps in classifying new data points.

In general, Support Vector Machines is considered to be a classification approach, it but can be employed in both types of classification and regression problems. It can easily handle multiple continuous and categorical variables. SVM constructs a hyperplane in multidimensional space to separate different classes. SVM generates optimal hyperplane in an iterative manner, which is used to minimize an error. The core idea of SVM is to find a maximum marginal hyperplane(MMH) that best divides the dataset into classes.

Support vectors are the data points, which are closest to the hyperplane. These points will define the separating line better by calculating margins. These points are more relevant to the construction of the classifier.

A hyperplane is a decision plane which separates between a set of objects having different class memberships.

A margin is a gap between the two lines on the closest class points. This is calculated as the perpendicular distance from the line to support vectors or closest points. If the margin is larger in between the classes, then it is considered a good margin, a smaller margin is a bad margin.

1.8.3 Dataset used

1. Linearly Separable

   The dataset used consist of 100 data points and two features. The two features are exam scores in two different subject. The output variable is either 0 or 1 representing Not Getting Admitted or Getting Admitted.

2. Non-Linearly Separable

   The dataset used consist of 118 data points and two features. The two features are housing area and length. The output variable is either 0 or 1 representing Not sold and House being sold.

1.8.4 Methodology To Implement This Algorithm

SVM searches for the maximum marginal hyperplane in the following steps:

1. Generate hyperplanes which segregates the classes in the best way. Left-hand side figure showing three hyperplanes black, blue and orange. Here, the blue and orange have higher classification error, but the black is separating the two classes correctly.

2. Select the right hyperplane with the maximum segregation from the either nearest data points as shown in the right-hand side figure.

Dealing with non-linear and inseparable planes:

In some situations like Non-linear and inseparable planes, we are unable to use, linear Separable models, then SVM uses a kernel trick to transform the input space to a higher dimensional space as shown on the right. The data points are plotted on the x-axis and z-axis (Z is the squared sum of both x and y: z=x2=y2). Now you can easily segregate these points using linear separation.

1.8.5 Python code Implementation

1. Linearly Separable

```
#!/usr/bin/env python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.svm import LinearSVC

datasetLoc = '../../../Datasets/exam_vs_adm.csv'
DataSet = np.genfromtxt(datasetLoc, delimiter=',')
X,Y=DataSet[:,:-1],DataSet[:,-1].reshape(-1,1)
xpos=X[(Y==1)[:,0]]
xneg=X[(Y==0)[:,0]]

plt.plot(xpos[:,0],xpos[:,1],'g*')
plt.plot(xneg[:,0],xneg[:,1],'b.')
plt.show()

X=np.hstack((np.ones((X.shape[0],1)),X))
Y=Y.ravel()

clf = LinearSVC(penalty='l2',random_state=0, tol=1e-5,dual=False)
clf.fit(X, Y)
print(clf.coef_)

print('accuracy : ',100.0*np.sum(clf.predict(X) == Y)/Y.shape[0])
```

2. Non-Linearly Separable

```
#!/usr/bin/env python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.svm import LinearSVC

datasetLoc = '../../../Datasets/nsvm.csv'

DataSet = np.genfromtxt(datasetLoc, delimiter=',')
X,Y=DataSet[:,:-1],DataSet[:,-1].reshape(-1,1)
print(X.shape,Y.shape)
xpos=X[(Y==1)[:,0]]
xneg=X[(Y==0)[:,0]]

plt.plot(xpos[:,0],xpos[:,1],'g*')
plt.plot(xneg[:,0],xneg[:,1],'b.')
plt.show()
X=np.hstack((np.ones((X.shape[0],1)),X))
Y=Y.ravel()
clf = LinearSVC(penalty='l2',random_state=0, tol=1e-5,dual=False)
clf.fit(X, Y)

print('accuracy : ',100.0*np.sum(clf.predict(X) == Y)/Y.shape[0])
```

1.8.6 Results

Thus the Support Vector Machines Algorithm- with Linearly separable and Non- Linearly Separable cases has been implemented

1.9 K-MEANS CLUSTERING

1.9.1 Aim

To Implement K-Means Clustering Algorithm.

1.9.2 Description

K-means algorithm explores for a pre-planned number of clusters in an unlabelled multidimensional dataset, it concludes this via an easy interpretation of how an optimized cluster can be expressed.

Primarily the concept would be in two steps:

- The cluster center is the arithmetic mean (AM) of all the data points associated with the cluster.
- Each point is adjoint to its cluster center in comparison to other cluster centers. These two interpretations are the foundation of the k-means clustering model.

You can take the center as a data point that outlines the means of the cluster, also it might not possibly be a member of the dataset.

1.9.3 Dataset used

The dataset used consist of 3100 data points and two features. The features are length and width of fishes.

1.9.4 Methodology To Implement This Algorithm

1. K-centers are modeled randomly in accordance with the present value of K.
2. K-means assigns each data point in the dataset to the adjacent center and attempts to curtail Euclidean distance between data points. Data points are assumed to be present in the peculiar cluster as if it is nearby to center to that cluster than any other cluster center.
3. After that, k-means determines the center by accounting the mean of all data points referred to that cluster center.It reduces the complete variance of the intra-clusters with respect to the prior step.Here, the "means" defines the average of data points and identifies a new center in the method of k-means clustering. will get the high probability for classification.
4. The algorithm gets repeated among the steps 2 and 3 till some paradigm will be achieved such as the sum of distances in between data points and their respective centers are diminished, an appropriate number of iterations is attained, no variation in the value of cluster center or no change in the cluster due to data points.

1.9.5 Python code Implementation

```
#!/usr/bin/env python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

datasetLoc = '../../../Datasets/kcl2.csv'
DataSet = np.genfromtxt(datasetLoc, delimiter=',')
ataSet[:,0:2]
Y=DataSet[:,1]
X=X.reshape(-1,2)
plt.plot(X[:,0],X[:,1],'r*')
plt.show()
```

```
kmeans = KMeans(n_clusters=35, random_state=0).fit(X)
ans = kmeans.cluster_centers_
plt.plot(X[:,0],X[:,1],'r.')
plt.plot(ans[:,0],ans[:,1],'bo')
plt.show()
```

1.9.6 Results

Thus the K-MEANS CLUSTERING has been implemented and clustering can be seen below.

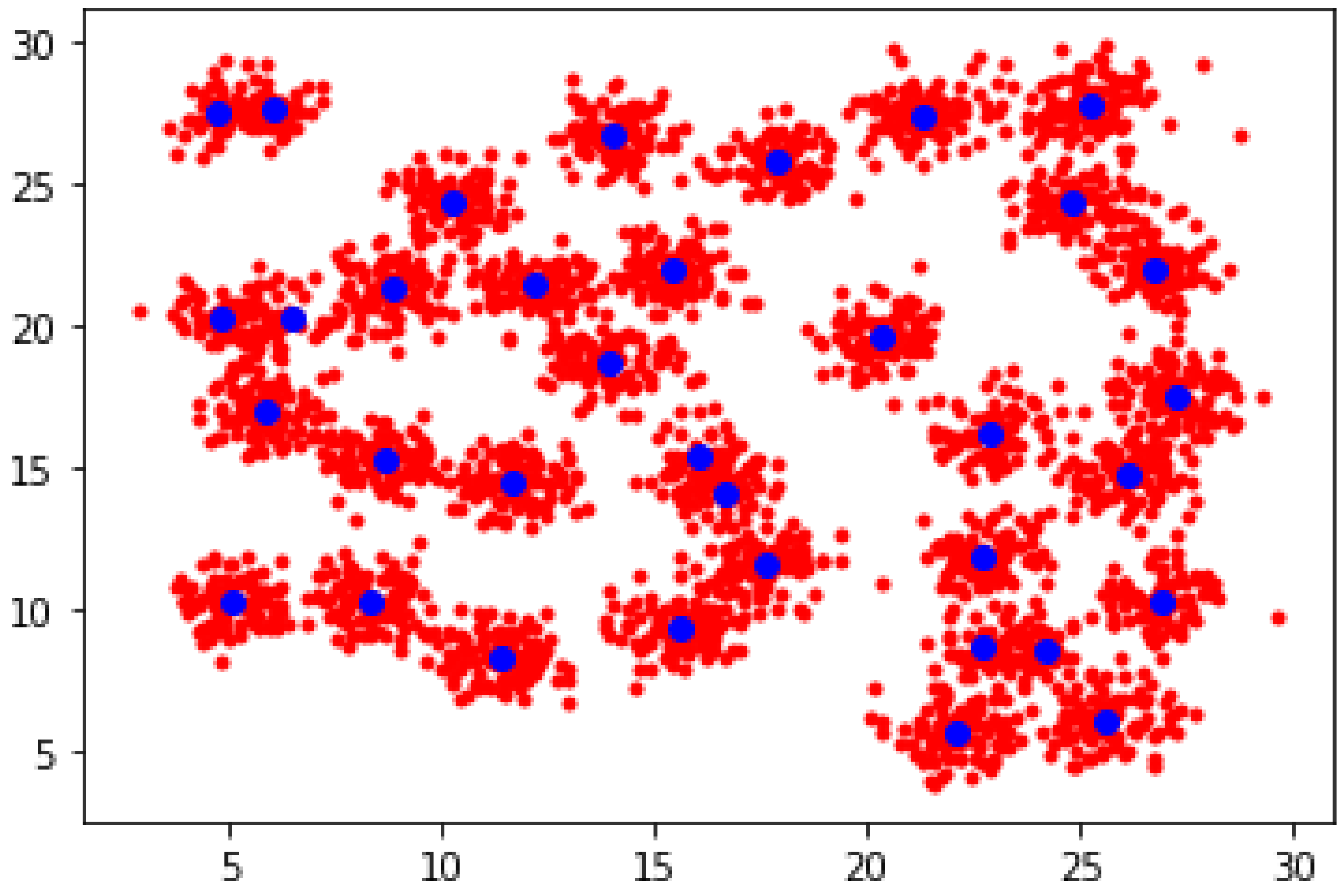


Figure 5: K-means Clustering with 35 clusters

2 Implementation of Deeplearning Alogorithms using Tensorflow

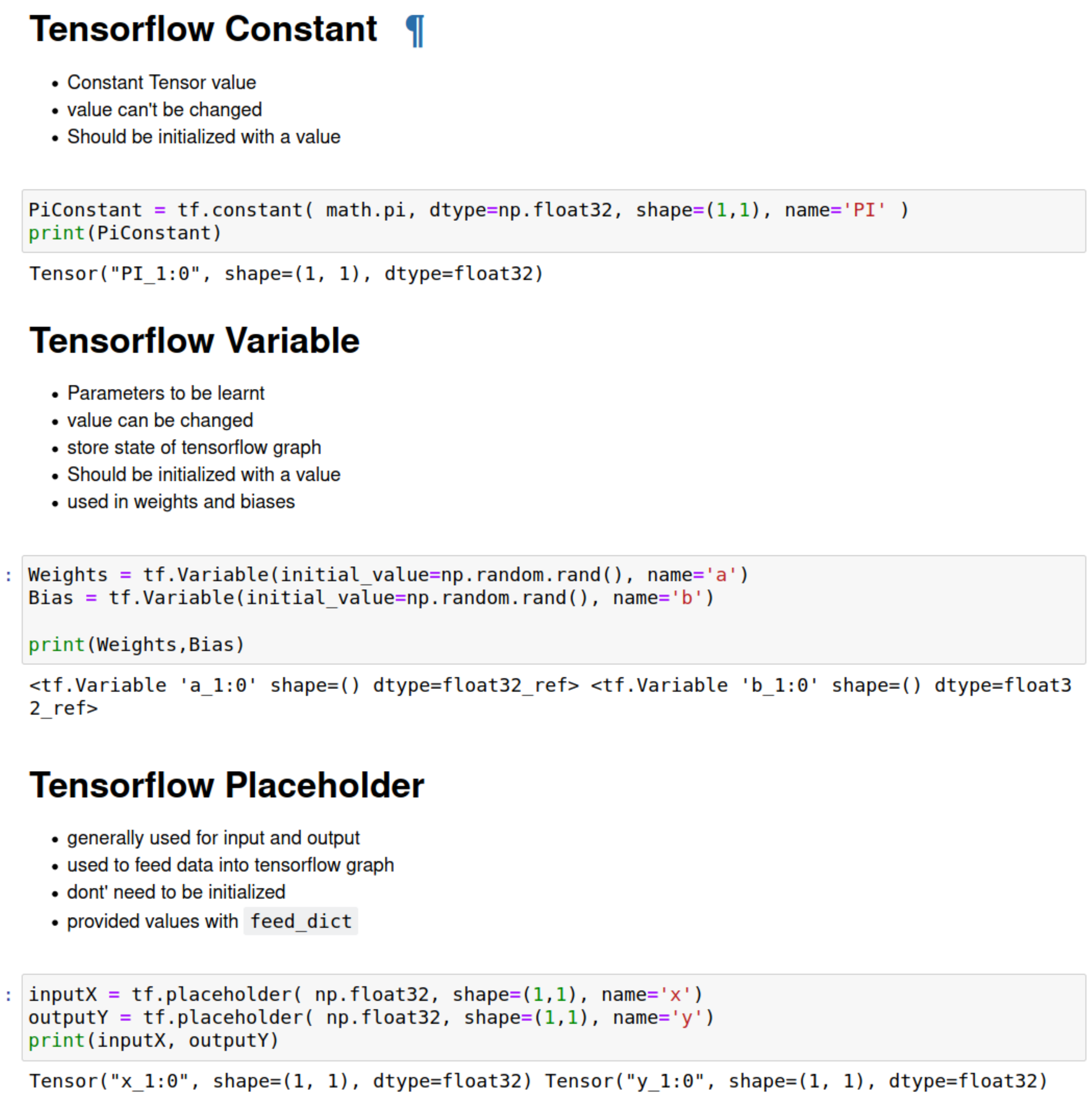
2.1 AIM

To write short programs in Tensor flow which uses the following objects:

1. Tensorflow Constants
2. Tensorflow Variables
3. Tensorflow Placeholders

To write and test the back propagation algorithm using the steps of the algorithm using Tensor Flow.

2.2 PYTHON CODE and OUTPUT



Tensorflow Constant ¶

- Constant Tensor value
- value can't be changed
- Should be initialized with a value

```
PiConstant = tf.constant( math.pi, dtype=np.float32, shape=(1,1), name='PI' )
print(PiConstant)
```

```
Tensor("PI_1:0", shape=(1, 1), dtype=float32)
```

Tensorflow Variable

- Parameters to be learnt
- value can be changed
- store state of tensorflow graph
- Should be initialized with a value
- used in weights and biases

```
Weights = tf.Variable(initial_value=np.random.rand(), name='a')
Bias = tf.Variable(initial_value=np.random.rand(), name='b')

print(Weights,Bias)
```

```
<tf.Variable 'a_1:0' shape=() dtype=float32_ref> <tf.Variable 'b_1:0' shape=() dtype=float3
2_ref>
```

Tensorflow Placeholder

- generally used for input and output
- used to feed data into tensorflow graph
- dont' need to be initialized
- provided values with `feed_dict`

```
inputX = tf.placeholder( np.float32, shape=(1,1), name='x')
outputY = tf.placeholder( np.float32, shape=(1,1), name='y')
print(inputX, outputY)
```

```
Tensor("x_1:0", shape=(1, 1), dtype=float32) Tensor("y_1:0", shape=(1, 1), dtype=float32)
```

2.3 BACKPROPAGATION

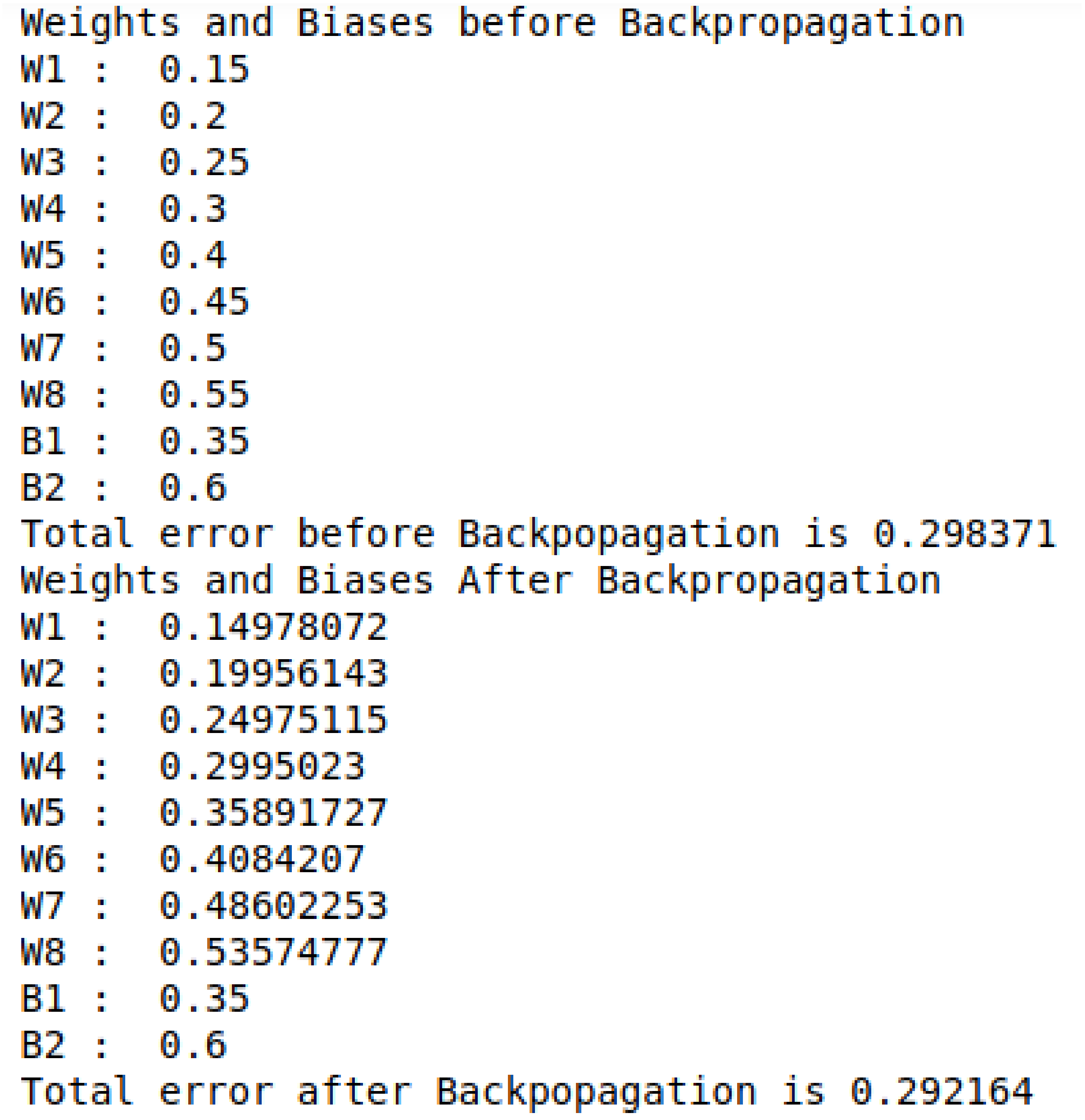
```
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()
import math
import numpy as np

eta = 0.5

input1 = tf.placeholder(np.float32, (1, 1), name='i1')
input2 = tf.placeholder(np.float32, (1, 1), name='i2')
output1 = tf.placeholder(np.float32, (1, 1), name='o1')
output2 = tf.placeholder(np.float32, (1, 1), name='o2')


w1 =  tf.Variable(0.15, name='w1')
w2 =  tf.Variable(0.20, name='w2')
w3 =  tf.Variable(0.25, name='w3')
w4 =  tf.Variable(0.30, name='w4')
b1 =  tf.Variable(0.35, name='b1')

w5 =  tf.Variable(0.40, name='w5')
w6 =  tf.Variable(0.45, name='w6')
w7 =  tf.Variable(0.50, name='w7')
w8 =  tf.Variable(0.55, name='w8')
b2 =  tf.Variable(0.60, name='b2')


# ## FeedForwared


h1net = tf.multiply(w1, input1) + tf.multiply(w2, input2) + b1
h2net = tf.multiply(w3, input1) + tf.multiply(w4, input2) + b1
h1out = tf.nn.sigmoid(h1net)
h2out = tf.nn.sigmoid(h2net)

o1net = tf.multiply(w5, h1out) + tf.multiply(w6, h2out) + b2
o2net =  tf.multiply(w7, h1out) + tf.multiply(w8, h2out) + b2
o1out = tf.nn.sigmoid(o1net)
o2out = tf.nn.sigmoid(o2net)


Eo1 = 0.5*(output1 - o1out)**2
Eo2 = 0.5*(output2 - o2out)**2
Etotal = Eo1 + Eo2


# ## BackPropagation


del_Etotal_o1out = - (output1 - o1out)
del_o1out_o1net = o1out * (1 - o1out)
del_o1net_w5 = h1out
```

```
del_Etotal_w5 = del_Etotal_o1out * del_o1out_o1net * del_o1net_w5

del_o1net_w6 = h2out
del_Etotal_w6 = del_Etotal_o1out * del_o1out_o1net * del_o1net_w6

del_Etotal_o2out = - (output2 - o2out)
del_o2out_o2net = o2out * (1 - o2out)
del_o2net_w7 = h1out
del_Etotal_w7 = del_Etotal_o2out * del_Etotal_o2out * del_o2net_w7

del_o2net_w8 = h2out
del_Etotal_w8 = del_Etotal_o2out * del_Etotal_o2out * del_o2net_w8



del_o1net_h1out = w5
del_Eo1_o1out = - (output1 - o1out)
del_Eo1_o1net = del_Eo1_o1out * del_o1out_o1net
del_Eo1_h1out = del_Eo1_o1net * del_o1net_h1out

del_o2net_h1out = w7
del_Eo2_o2out = - (output2 - o2out)
del_Eo2_o2net = del_Eo2_o2out * del_o2out_o2net
del_Eo2_h1out = del_Eo2_o2net * del_o2net_h1out

del_Etotal_h1out = del_Eo1_h1out + del_Eo2_h1out

del_h1net_w1 = input1
del_h1out_h1net = h1out * (1 - h1out)
del_Etotal_w1 = del_Etotal_h1out * del_h1out_h1net * del_h1net_w1

del_h1net_w2 = input2
del_Etotal_w2 = del_Etotal_h1out * del_h1out_h1net * del_h1net_w2

del_o1net_h2out = w6
del_Eo1_h2out = del_Eo1_o1net * del_o1net_h2out

del_o2net_h2out = w8
del_Eo2_h2out = del_Eo2_o2net * del_o2net_h2out

del_Etotal_h2out = del_Eo1_h2out + del_Eo2_h2out

del_h2net_w3 = input1
del_h2out_h2net = h2out * (1 - h2out)
del_Etotal_w3 = del_Etotal_h2out * del_h2out_h2net * del_h2net_w3

del_h2net_w3 = input2
del_Etotal_w4 =  del_Etotal_h2out * del_h2out_h2net * del_h2net_w3

w1New = w1 - eta*del_Etotal_w1
w2New = w2 - eta*del_Etotal_w2
w3New = w3 - eta*del_Etotal_w3
w4New = w4 - eta*del_Etotal_w4
```

```
w5New = w5 - eta*del_Etotal_w5
w6New = w6 - eta*del_Etotal_w6
w7New = w7 - eta*del_Etotal_w7
w8New = w8 - eta*del_Etotal_w8



init = tf.global_variables_initializer()


inp_out_dict = {input1 : np.asarray([0.05]).reshape(1,1),\
 input2 : np.asarray([0.10]).reshape(1,1),
                output1 : np.asarray([0.01]).reshape(1,1),\
                 output2 : np.asarray([0.99]).reshape(1,1)}



def update_WB(sess, w1a, w2a, w3a, w4a, w5a, w6a, w7a, w8a, b1a, b2a):
    global w1, w2, w3, w4, w5, w6, w7, w8, b1, b2
    sess.run(w1.assign(sess.run(w1a , feed_dict=inp_out_dict)[0][0]))
    sess.run(w2.assign(sess.run(w2a , feed_dict=inp_out_dict)[0][0]))
    sess.run(w3.assign(sess.run(w3a , feed_dict=inp_out_dict)[0][0]))
    sess.run(w4.assign(sess.run(w4a , feed_dict=inp_out_dict)[0][0]) )
    sess.run(w5.assign(sess.run(w5a , feed_dict=inp_out_dict)[0][0]))
    sess.run(w6.assign(sess.run(w6a , feed_dict=inp_out_dict)[0][0] ))
    sess.run(w7.assign(sess.run(w7a , feed_dict=inp_out_dict)[0][0] ))
    sess.run(w8.assign(sess.run(w8a , feed_dict=inp_out_dict)[0][0] ))
    sess.run(b1.assign(sess.run(b1a , feed_dict=inp_out_dict)))
    sess.run(b2.assign(sess.run(b2a , feed_dict=inp_out_dict)))


def print_WB(sess):
    global w1, w2, w3, w4, w5, w6, w7, w8, b1, b2
    print('W1 : ',sess.run(w1 , feed_dict=inp_out_dict))
    print('W2 : ',sess.run(w2 , feed_dict=inp_out_dict))
    print('W3 : ',sess.run(w3 , feed_dict=inp_out_dict))
    print('W4 : ',sess.run(w4 , feed_dict=inp_out_dict))
    print('W5 : ',sess.run(w5 , feed_dict=inp_out_dict))
    print('W6 : ',sess.run(w6 , feed_dict=inp_out_dict))
    print('W7 : ',sess.run(w7 , feed_dict=inp_out_dict))
    print('W8 : ',sess.run(w8 , feed_dict=inp_out_dict))
    print('B1 : ',sess.run(b1 , feed_dict=inp_out_dict))
    print('B2 : ',sess.run(b2 , feed_dict=inp_out_dict))


print('Weights and Biases before Backpropagation ')

with tf.Session() as sess:
    sess.run(init)
    print_WB(sess)
    print('Total error before Backpopagation is %.6f'%\
    (sess.run(Etotal, feed_dict = inp_out_dict)))

    update_WB(sess, w1New, w2New, w3New, w4New, w5New, w6New,\
     w7New, w8New, b1, b2)

    print('Weights and Biases After Backpropagation ')
    print_WB(sess)
    print('Total error after Backpopagation is %.6f'%\
    (sess.run(Etotal, feed_dict = inp_out_dict)))
```

2.4 RESULT

The back propagation has been implemented by tensorflow and we can observe that the error after back propagation is lesser than error before back propagation



```
Weights and Biases before Backpropagation
W1 :  0.15
W2 :  0.2
W3 :  0.25
W4 :  0.3
W5 :  0.4
W6 :  0.45
W7 :  0.5
W8 :  0.55
B1 :  0.35
B2 :  0.6
Total error before Backpopagation is 0.298371
Weights and Biases After Backpropagation
W1 :  0.14978072
W2 :  0.19956143
W3 :  0.24975115
W4 :  0.2995023
W5 :  0.35891727
W6 :  0.4084207
W7 :  0.48602253
W8 :  0.53574777
B1 :  0.35
B2 :  0.6
Total error after Backpopagation is 0.292164
```

Figure 6: Back Propagation Output

3 Tamil Handwritten Character Recognition

3.1 ABSTRACT

Handwritten character is constructed by executing a sequence of strokes. A structure- or shape-based representation of a stroke is used in which a stroke is represented as a string of shape features. Using this string representation, an unknown stroke is identified by comparing it with a database of strokes using a flexible string matching procedure.

A full character is recognized by identifying all the component strokes. Handwritten Text Recognition is the ability of a computer to receive and interpret intelligible hand-written input from sources such as paper documents, photographs,touch-screens and other devices.

A **Convolutional Neural Network (CNN)** is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other.

The recognition of the Tamil language letters are challenging because of its various writing styles and very large number of characters. Here we wish to use CNN in recognizing handwritten Tamil characters. CNNs differ from traditional approach of Handwritten Tamil Character Recognition in extracting the features automatically and thereby increasing the chance of automation of such recognition.

3.2 THEORY

Machine simulation of human functions has been a very challenging research field since the advent of digital computers. In some areas, which entail certain amount of intelligence, such as number crunching or chess playing, tremendous improvements are achieved.

In this regard, we came across this Tamil Handwritten character Recognition. Handwritten text recognition is the ability of a machine or computer to receive and interpret handwritten text input from sources such as paper documents, photographs, Touch Screens and other devices.

A handwriting recognition system handles formatting, performs correct segmentation into characters, and finds the most plausible words. This Tamil Language had about 247 characters. This creates recognition of the character is harder due to large category set and confusion between character which are similar. This makes this project to be one of the toughest handwritten recognition projects.

3.3 DATASET

The dataset is from HP Labs India collected from native Tamil writers. There are about 156 characters with each character consisting of approximately 500 samples making the dataset about 82929 images in total.

The characters used for recognition are:

```
['அ', 'ஆ', 'இ', 'ஈ', 'உ', 'ஊ', 'எ', 'ஏ', 'ஐ', 'ஒ', 'ஓ', 'ஃ', 'க', 'ங', 'ச', 'ஞ', '
ட', 'ண', 'த', 'ந', 'ப', 'ம', 'ய', 'ர', 'ல', 'வ', 'ழ', 'ள', 'ற', 'ன', 'ஸ', 'ஷ', 'ஜ',
'ஹ', 'க்ஷ', 'கி', 'ஙி', 'சி', 'ஞி', 'டி', 'ணி', 'தி', 'நி', 'பி', 'மி', 'யி', 'ரி', 'லி', 'வி',
'ழி', 'ளி', 'றி', 'னி', 'ஸி', 'ஷி', 'ஜி', 'ஹி', 'க்ஷி', 'கீ', 'ஙீ', 'சீ', 'ஞீ', 'டீ', 'ணீ', '
தீ', 'நீ', 'பீ', 'மீ', 'யீ', 'ரீ', 'லீ', 'வீ', 'ழீ', 'ளீ', 'றீ', 'னீ', 'ஸீ', 'ஷீ', 'ஜீ', 'ஹீ', 'க்ஷ
ீ', 'கு', 'ஙு', 'சு', 'ஞு', 'டு', 'ணு', 'து', 'நு', 'பு', 'மு', 'யு', 'ரு', 'லு', 'வு', 'ழு', '
ளு', 'று', 'னு', 'கூ', 'ஙூ', 'சூ', 'ஞூ', 'டூ', 'ணூ', 'தூ', 'நூ', 'பூ', 'மூ', 'யூ', 'ரூ', 'லூ',
'வூ', 'ழூ', 'ளூ', 'றூ', 'னூ', 'ா', 'ெ', 'ே', 'ை', 'ஸ்ரீ', 'ஸு', 'ஷு', 'ஜு', 'ஹு', '
க்ஷு', 'ஸூ', 'ஷூ', 'ஜ0BC2', 'ஹூ', 'க்ஷூ', 'க்', 'ங்', 'ச்', 'ஞ்', 'ட்', 'ண்', 'த்', 'ந்', '
ப்', 'ம்', 'ய்', 'ர்', 'ல்', 'வ்', 'ழ்', 'ள்', 'ற்', 'ன்', 'ஸ்', 'ஷ்', 'ஜ்', 'ஹ்', 'க்ஷ்', 'ஔ']
```

3.4 METHOD USED

3.4.1 Pre-Processing

- In general, the images which are taken from the data set will be random resolutions. But we need to feed this data only to the CNN. So we should be able to change and resize the data according to our requirements. So we had taken the average of entire dataset, then we got the average resolution value as 128x128.

- Next the dataset is split into training and testing data. About 85% of the total data i.e., 70k is given for the purpose of training the CNN Model and remaining 15% (Approximately 13k) will be given for the testing purpose. As the splitting of data increases for the Training, the performance efficiency of the Model will increases.
- Here, we used One hot encoding. This allows the representation of categorical data to be more expressive. One-hot encoding ensures that machine learning does not assume that higher numbers are more important. Many machine learning algorithms cannot work with categorical data directly. The categories must be converted into numbers.
- The 156 character labels will be assigned to a 156x1 array. Whenever, the character is detected then that particular character's position will be occupied by the binary bit 1, other elements positions will be represented with 0.
- The data set is stored as python pickle Object so as to retrieve that data without reading each and every time of simulation.
- Python pickle module is used for serializing and de-serializing a Python object structure. ... Pickling is a way to convert a python object (list, dictionary, etc.) into a character stream. The idea is that this character stream contains all the information necessary to reconstruct the object in another python script.

3.4.2 Architecture

The input to the CNN is 128x128 image which is passed to the architecture shown below.

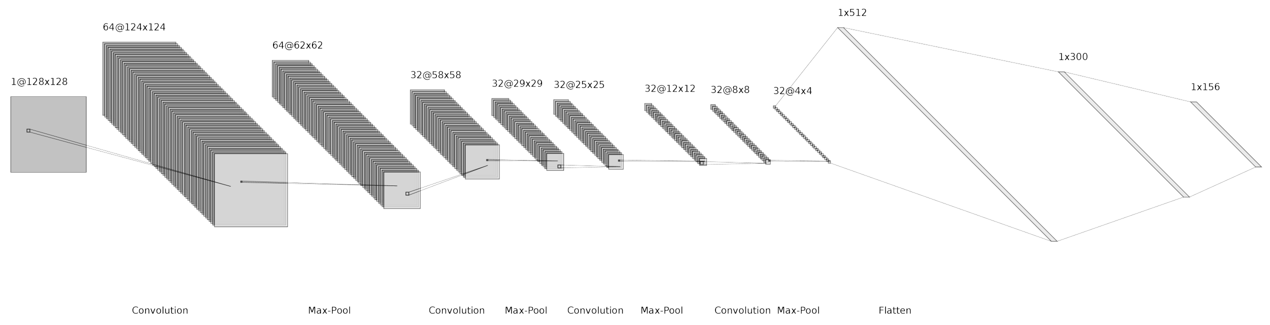


Figure 7: Architecture used for Tamil Handwritten character recognition

- As there are 82929 images of 156 characters, which is of equal average resolution of 128x128.
- This resolution of image is passed through the convolution of the particular filter, where the weight and biases which is attached with ReLU activation function learns during this process. After convolution it is passed through the max polling. During this Max. pooling operation, it calculates the max. Value in each patch of each feature map. Through this process dimension of the image decreases.

- Here, we pass 1@128*128 image for the convolution which gives output of 64@124*124 and after max polling it gives output of 64@62*62. We process this convolution for 4 times followed by max polling of 4 times. Finally we get 32@4*4 image which is flattening to hidden layer of 300 neurons. After that it is passed to output layer of 156 outputs.
- For output, we use Softmax activation function where as for convolution and hidden layer we use ReLU activation function.

3.4.3 Brief Explanation of Architecture

We designed Our own CNN which has the following layers:-

1. **Input** Images taken from the dataset, reshape. The same images used and of size 128x128x1.
2. **Conv-1** The first convolutional layer consists of 64 kernels of size 5x5 applied with a stride of 1 and padding of 0.
3. **MaxPool-1** The max-pool layer following Conv-2 consists of pooling size of 2x2 and a stride of 1.
4. **Conv-2** The second convolution layer consists of 32 kernels of size 5x5 applied with a stride of 1 and padding of 0.
5. **MaxPool-2** The max-pool layer following Conv-2 consists of pooling size of 2x2 and a stride of 1.
6. **Conv-3** The third conv layer consists of 32 kernels of size 5x5 applied with a stride of 1 and padding of 0.
7. **MaxPool-3** The max-pool layer following Conv-3 consists of pooling size of 2x2 and a stride of 1.
8. **Conv-4** The fourth conv layer consists of 32 kernels of size 5x5 applied with a stride of 1 and padding of 0.
9. **MaxPool-4** The maxpool layer following Conv-4 consists of pooling size of 2x2 and a stride of 0.
10. **Flattening Layer** The output of CNN is flattened to get 1x512 output.
11. **FC-1 (Dense Layer 1)** The flattened output is fed to a hidden layer of 300 neurons.
12. **Output (Dense Layer 2)** Finally, the output of hidden layer is fed to the output layer of 156 to get the final output.

    All these 12 layers constitute the CNN for Tamil Hand written Character recognition Project.

Input shape	Layer	Output shape
128x128	Convolution Layer - 64C5x5	124x124x64
124x124x64	MaxPooling Layer - P2x2	62x62x64
62x62x64	Convolution Layer - 32C5x5	58x58x32
58x58x32	MaxPooling Layer - P2x2	29x29x32
29x29x32	Convolution Layer - 32C5x5	25x25x32
25x25x32	MaxPooling Layer - P2x2	12x12x32
12x12x32	Convolution Layer - 32C5x5	8x8x32
8x8x32	MaxPooling Layer - P2x2	4x4x32
4x4x32	Flatten Layer	1x512
1x512	Hidden Layer	1x300
1x300	Output Layer	1X156

We obtain the output shape after each layer as shown above.

3.4.4 Training and Testing

The training is done with a total of 70,498 datas with 13 epochs. Each epochs consist of batch size of 100. The learning is done using adaptive learning rate with loss function of Categorical Cross Entropy.

The testing is done on the remaining 12,440 datas from the dataset. Also, a webpage is created to test the online character recognition.

3.5 PYTHON CODE IMPLEMENTATION

3.6 Preprocessing

```
# -*- coding: utf-8 -*-
"""preProcessing.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1cWIGy14r_nthfeM41OmgtPzt_Ebk7phI

# Dataset Location
http://shiftleft.com/mirrors/www.hpl.hp.com/ind⌟
↪  ia/research/penhw-resources/hpl-tamil-iso-char-offline-1.0.tar.gz
"""

!wget http://shiftleft.com/mirrors/www.hpl.hp.com/ind⌟
↪  ia/research/penhw-resources/hpl-tamil-iso-char-offline-1.0.tar.gz
!tar -xvf ./hpl-tamil-iso-char-offline-1.0.tar.gz

import os
import pickle
import cv2
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
import shutil

def plotIm(img_):
  plt.imshow(img_, cmap='gray')
  plt.show()
def bbox2(img1):
  img = 1 - img1
  rows = np.any(img, axis=1)
  cols = np.any(img, axis=0)
  rmin, rmax = np.where(rows)[0][[0, -1]]
  cmin, cmax = np.where(cols)[0][[0, -1]]
  return rmin, rmax, cmin, cmax
def RR(img):
    rmin, rmax, cmin, cmax = bbox2(img)
    # print(rmin, rmax, cmin, cmax)
    npArr = img[rmin:rmax, cmin:cmax]
    npArr = cv2.resize(npArr, dsize=(100, 100))
    jinga = np.ones((128,128))
    jinga[14:114,14:114] = npArr
    npArr = jinga.reshape(128, 128 , 1)
    return npArr

images=[]
labels=[]
numCategory = 12
datasetsLoc = './tamil_dataset_offline/'
w,h = 128,128
i = 0
shapeL=[]

for folders in  os.listdir(datasetsLoc):
    for files in  os.listdir(datasetsLoc+str(folders)):
        if str(files) == 'Thumbs.db' or str(files) == 't03.tiff':
          continue
        # print(str(folders))
        indx = int(str(files)[:3])
        if indx < numCategory:
            image = Image.open(datasetsLoc+str(folders)+'/'+str(files))
            img=np.asarray(image, dtype=np.uint8)
            shapeL.append(img.shape)
            img2 = RR(img)
            img2 = np.asarray(img2, dtype=np.uint8)
            images.append(img2)
            labels.append(indx)
        if i%8000 == 0:
            print(str(i)+ '   ' + str(len(labels)))
        i = i + 1
filIm = open('./image_uyir_128x128.obj', 'wb')
pickle.dump(images, filIm)
```

```
filLab = open('./label_uyir_128x128.obj', 'wb')
pickle.dump(labels, filLab)

!cp /content/image_uyir_128x128.obj /content/drive/My\
 ↪  Drive/wkDir/image_uyir_128x128.obj
!cp /content/label_uyir_128x128.obj /content/drive/My\
 ↪  Drive/wkDir/label_uyir_128x128.obj

images=[]
labels=[]
numCategory = 156
datasetsLoc = './tamil_dataset_offline/'
w,h = 128,128
i = 0
shapeL=[]

for folders in os.listdir(datasetsLoc):
    for files in  os.listdir(datasetsLoc+str(folders)):
        if str(files) == 'Thumbs.db' or str(files) == 't03.tiff' or str(files) ==
         ↪  '036t01.png':
          continue
        # print(str(folders))
        indx = int(str(files)[:3])
        if indx < numCategory:
            image = Image.open(datasetsLoc+str(folders)+'/'+str(files))
            img=np.asarray(image, dtype=np.uint8)
            shapeL.append(img.shape)
            img2 = RR(img)
            img2 = np.asarray(img2, dtype=np.uint8)
            images.append(img2)
            labels.append(indx)
        if i%8000 == 0:
            print(str(i)+ '   ' + str(len(labels)))
        i = i + 1
filIm = open('./image_ALL_128x128.obj', 'wb')
pickle.dump(images, filIm)
filLab = open('./label_ALL_128x128.obj', 'wb')
pickle.dump(labels, filLab)

!cp /content/image_ALL_128x128.obj /content/drive/My\
 ↪  Drive/wkDir/image_ALL_128x128.obj
!cp /content/label_ALL_128x128.obj /content/drive/My\
 ↪  Drive/wkDir/label_ALL_128x128.obj
```

3.7 Model Creation

```
# -*- coding: utf-8 -*-
"""tamilUyirEzhuthukalKeras.ipynb

Automatically generated by Colaboratory.


"""
```

```
import os
import cv2
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
import re
import csv
import pickle
import keras
from keras.models import Sequential
from keras.layers import Dense
from keras.utils import to_categorical
from keras.layers import Dropout
from keras.layers import Flatten
from keras.layers.convolutional import Conv2D
from keras.layers.convolutional import MaxPooling2D
from keras.utils import np_utils
from sklearn.model_selection import train_test_split
from keras.preprocessing.image import ImageDataGenerator

w, h = 128, 128
numCategory = 12
filIm = open('/content/drive/My Drive/wkDir/image_uyir_128x128.obj', 'rb')
images = pickle.load(filIm)
filLab = open('/content/drive/My Drive/wkDir/label_uyir_128x128.obj', 'rb')
labels = pickle.load(filLab)

def plotIm(img_):
  plt.imshow(img_, cmap='gray')
  plt.show()

tamilCharacterCode = []
w,h=128,128
with open('/content/drive/My Drive/wkDir/unicodeTamil.csv', newline='') as f:
  reader = csv.reader(f)
  data = list(reader)
  for i in data:
    go = i[1].split(' ')
    charL = ""
    for gg in go:
      charL = charL + "\\u"+str(gg)
    tamilCharacterCode.append(charL.encode('utf-8').decode('unicode-escape'))
print(tamilCharacterCode)

images=np.array(images)
labels=np.array(labels,dtype=np.uint8)
y_labels=to_categorical(labels)
X_train, X_test, y_train, y_test = train_test_split(images, y_labels, test_size=0.33,
↪  random_state=42)
keras.initializers.lecun_uniform(seed=None)
print(X_train.shape, y_train.shape)
```

```
ridx = np.random.randint(X_train.shape[0])
print(tamilCharacterCode[np.argmax(y_train[ridx])])
plotIm(X_train[ridx].reshape(w,h))

model = Sequential()
model.add(Conv2D(64, (5, 5), input_shape=(w,h,1), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))
model.add(Conv2D(32, (5, 5), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(32, (5, 5), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))
model.add(Conv2D(32, (5, 5), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dense(numCategory, activation='softmax'))
model.summary()
model.compile(loss='categorical_crossentropy', optimizer='adam',
 ↪  metrics=['accuracy'])

model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=10,
 ↪  batch_size=100, verbose=1)
TrainAccuracy = model.evaluate(X_train, y_train, verbose=1)
TestAccuracy = model.evaluate(X_test, y_test, verbose=1)

model.save("/content/drive/My Drive/wkDir/tamilUyirEzhuthukalKeras_Model.h5")
print("Saved model to disk")
```

3.8 Online Testing using Flash APP

```
import os
from math import floor
import io
import requests
from PIL import Image
from flask˙wtf import FlaskForm
from flask import request
from flask import Flask
from wtforms import StringField, PasswordField, SubmitField, BooleanField,
 ↪  TextAreaField
from wtforms.validators import DataRequired, Length, Email,EqualTo, ValidationError
from flask import render_template, url_for, flash, redirect, request, abort
import numpy as np
import cv2
import csv
from matplotlib import pyplot as plt
from tensorflow.keras.models import load_model
import pickle
```

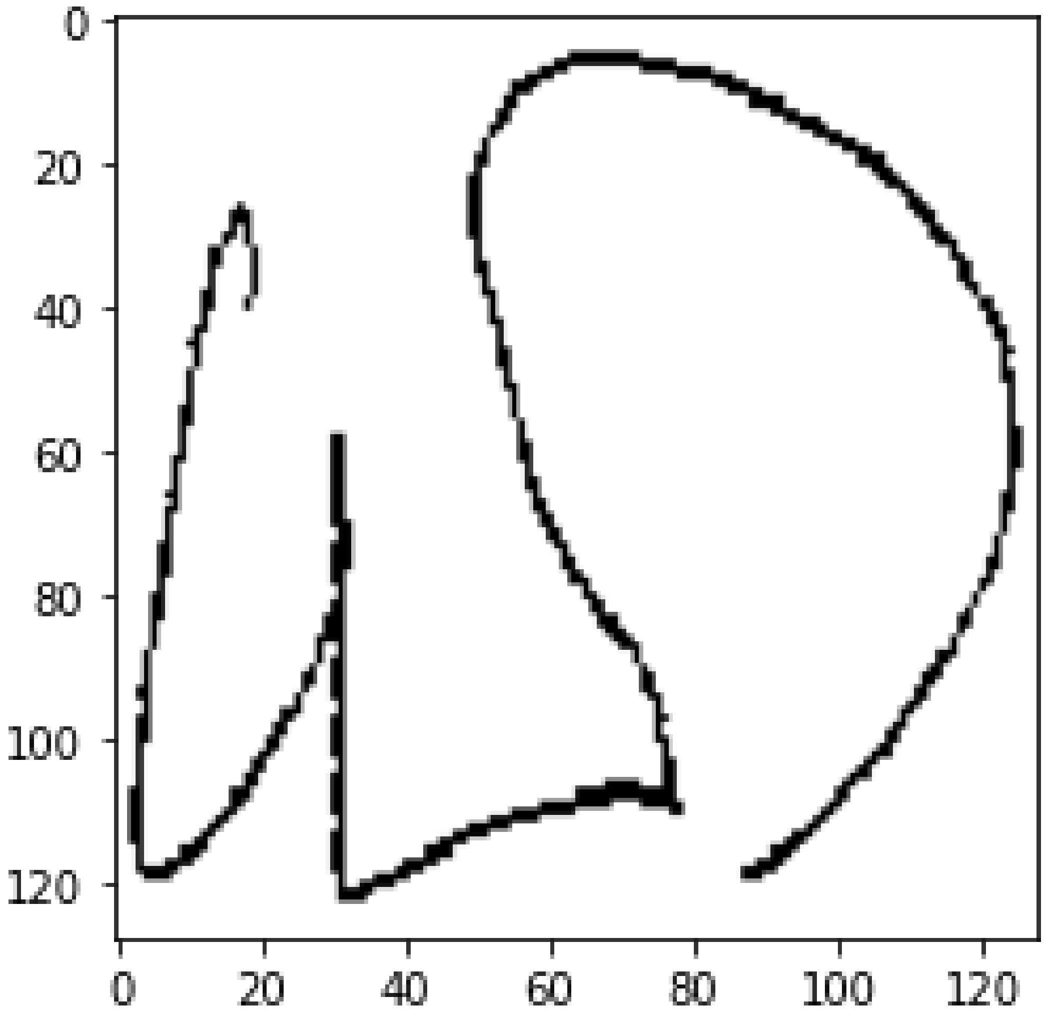
```python
tamilCharacterCode = []
model = None

app = Flask(__name__)



@app.route('/')
@app.route('/home')
def homepage_func():
        return render_template('homepage.html')

def bbox2(img1):
  img = 1 - img1
  rows = np.any(img, axis=1)
  cols = np.any(img, axis=0)
  rmin, rmax = np.where(rows)[0][[0, -1]]
  cmin, cmax = np.where(cols)[0][[0, -1]]
  return rmin, rmax, cmin, cmax
def RR(img):
    rmin, rmax, cmin, cmax = bbox2(img)
    # print(rmin, rmax, cmin, cmax)
    npArr = img[rmin:rmax, cmin:cmax]
    npArr = cv2.resize(npArr, dsize=(100, 100))
    jinga = np.ones((128,128))
    jinga[14:114,14:114] = npArr
    npArr = jinga.reshape(128, 128 , 1)
    return npArr

def getTamilChar(tamilCharacterCode, indx):
        return tamilCharacterCode[indx]


@app.route('/postmethod', methods = ['POST'])
def get_post_javascript_data():
        global tamilCharacterCode, model
        att = request.data

        imgStr = att.decode('utf-8')
        imgArr = imgStr.split(',')
        npArr = np.asarray(imgArr, dtype=np.uint8).reshape(400,400)


        npArr = RR(npArr)
        npArr = npArr.reshape(1, 128, 128 , 1)
        atc = model.predict(npArr)

        percentage = atc[0]

        valsss = atc[0].argsort()[-3:][::-1]
        responseTextSt = getTamilChar(tamilCharacterCode,valsss[0])+","+
         ↪  getTamilChar(tamilCharacterCode,valsss[1])+ ","+
         ↪  getTamilChar(tamilCharacterCode,valsss[2])

        responseTextSt = responseTextSt + ',%.3f,%.3f,%.3f'%(percentage[valsss[0]]
         ↪  *100.0,percentage[valsss[1]] *100.0,percentage[valsss[2]]*100.0)

        return responseTextSt


def init_somethings():

        global tamilCharacterCode, model

        with open('unicodeTamil.csv', newline='') as f:
                reader = csv.reader(f)
                data = list(reader)
                for i in data:
                        go = i[1].split(' ')
                        charL = ""
                        for gg in go:
                                charL = charL + "\\u"+str(gg)
                        tamilCharacterCode.append(charL.encode('utf-8').decod␣
                         ↪  e('unicode-escape'))


        model = load_model('tamilALLEzhuthukalKeras_Model.h5')
        print(model.summary())

if __name__ == '__main__':
        init_somethings()
        #print(tamilCharacterCode)
        print("\n******************App started****************\n")
        # run overall
        # app.run(debug=True,  host='0.0.0.0')
        # run in localhost
        app.run(debug=True)
```

3.9 RESULT

The results obtained are shown below:

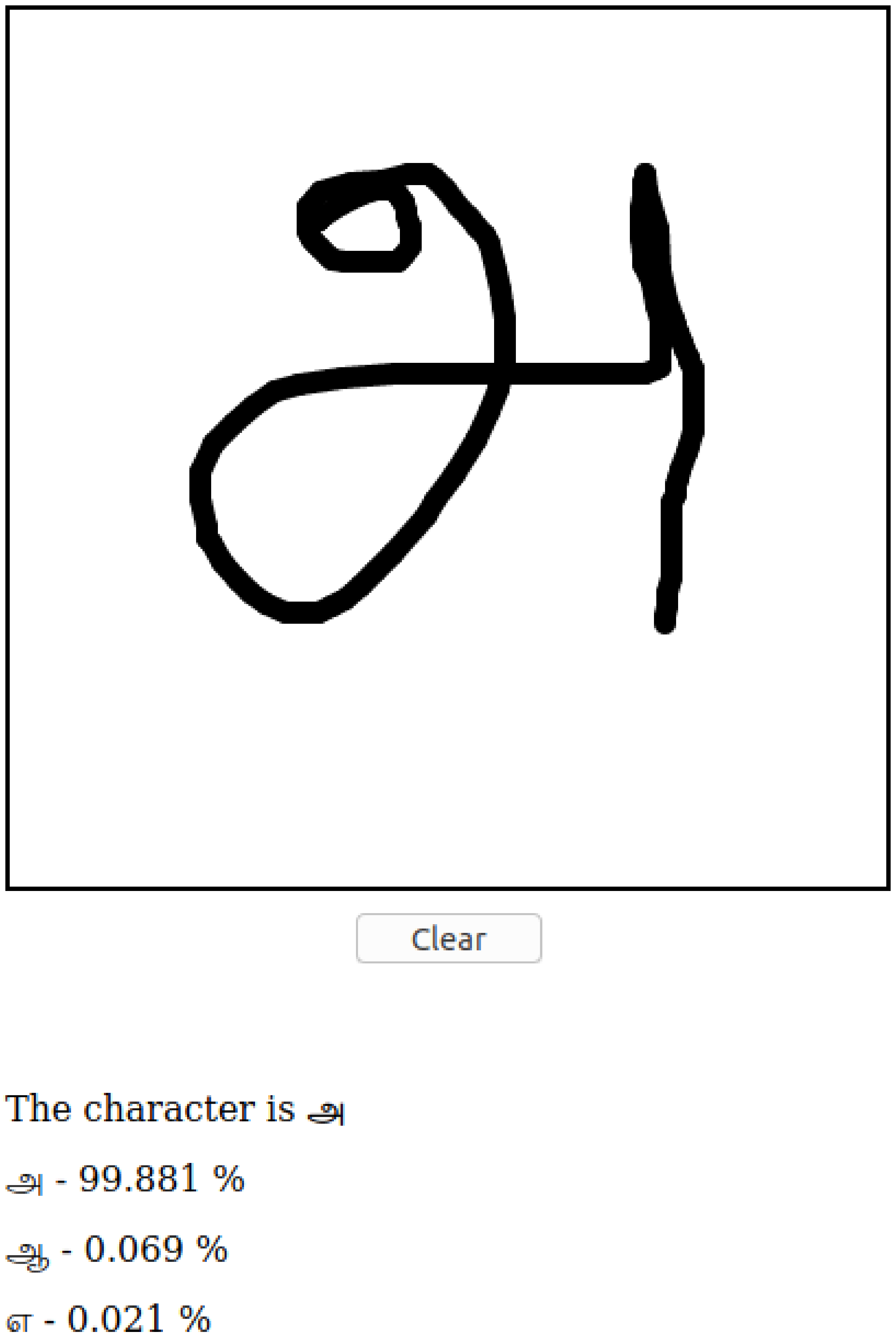
1. Training Accuracy - 96.334
2. Testing Accuracy - 91.439
3. Classification Loss - 0.249

```
Given result : ழ
```



```
predicted result: ழ
```

Figure 8: Testing data output



```
The character is அ
அ - 99.881 %
ஆ - 0.069 %
எ - 0.021 %
```

Figure 9: Online data output