Backpropogation algorithm is implemented on the Vowel Dataset. The dataset is attached with the file. Number of input features are three and output classes are six and sample size is 871.
The input features are fuzzified using PI membership function and fed into the neural network. The fuzzification is class independent. The output layer is also fuzzified.
5-fold cross validation is used to achieve better accuracy
Confusion matrix, precision, recall, accuracy and Fscore is computed using sklearn package

```
from random import seed
from random import randrange
from random import random
from csv import reader
from math import exp
from sklearn.metrics import confusion_matrix
import numpy as np
import math

# Load a CSV file
def load_csv(filename):
	dataset = list()
	with open(filename, 'r') as file:
		csv_reader = reader(file)
		for row in csv_reader:
			if not row:
				continue
			dataset.append(row)
	return dataset

def minmax(dataset):
	minmax = list()
	stats = [[min(column), max(column)] for column in zip(*dataset)]
	return stats

# Rescale dataset columns to the range 0-1
def normalize(dataset, minmax):
	for row in dataset:
		for i in range(len(row)-1):
			row[i] = (row[i] - minmax[i][0]) / (minmax[i][1] - minmax[i][0])

# Convert string column to float
def column_to_float(dataset, column):
	for row in dataset:
		try:
			row[column] = float(row[column].strip())
		except ValueError:
			print("Error with row",column,":",row[column])
```

Copyright rests with Archana Mathur, astrirg lab, PESU and SSIU, ISI Banglore Please feel free to use the code and please acknowledge while doing so. The code is hosted on astrirg.org

```
			pass

# Convert string column to integer
def column_to_int(dataset, column):
	for row in dataset:
		row[column] = int(row[column])

# Find the min and max values for each column
# Split a dataset into k folds
def cross_validation_split(dataset, n_folds):
	dataset_split = list()
	dataset_copy = list(dataset)
	fold_size = int(len(dataset) / n_folds)
	for i in range(n_folds):
		fold = list()
		while len(fold) < fold_size:
			index = randrange(len(dataset_copy))
			fold.append(dataset_copy.pop(index))
		dataset_split.append(fold)
	return dataset_split

# Calculate accuracy percentage
def accuracy_met(actual, predicted):
	correct = 0
	for i in range(len(actual)):
		if actual[i] == predicted[i]:
			correct += 1
	return correct / float(len(actual)) * 100.0

# Evaluate an algorithm using a cross validation split
def run_algorithm(dataset, algorithm, n_folds, *args):
	#print(dataset)
	folds = cross_validation_split(dataset, n_folds)
	#for fold in folds:
		#print("Fold {} \n \n".format(fold))
	scores = list()
	for fold in folds:
		#print("Test Fold {} \n \n".format(fold))
		train_set = list(folds)
		train_set.remove(fold)
		train_set = sum(train_set, [])
		test_set = list()
		for row in fold:
			row_copy = list(row)
			test_set.append(row_copy)
			row_copy[-1] = None
```

Copyright rests with Archana Mathur, astrirg lab, PESU and SSIU, ISI Banglore Please feel free to use the code and please acknowledge while doing so. The code is hosted on astrirg.org

```
        predicted = algorithm(train_set, test_set, *args)
        actual = [row[-1] for row in fold]
        #print(predicted)
        #print(actual)
        accuracy = accuracy_met(actual, predicted)
        cm = confusion_matrix(actual, predicted)
        print('\n'.join([''.join(['{:4}'.format(item) for item in row]) for row in cm]))
        #confusionmatrix = np.matrix(cm)
        FP = cm.sum(axis=0) - np.diag(cm)
        FN = cm.sum(axis=1) - np.diag(cm)
        TP = np.diag(cm)
        TN = cm.sum() - (FP + FN + TP)
        print('False Positives\n {}'.format(FP))
        print('False Negetives\n {}'.format(FN))
        print('True Positives\n {}'.format(TP))
        print('True Negetives\n {}'.format(TN))
        TPR = TP/(TP+FN)
        print('Sensitivity \n {}'.format(TPR))
        TNR = TN/(TN+FP)
        print('Specificity \n {}'.format(TNR))
        Precision = TP/(TP+FP)
        print('Precision \n {}'.format(Precision))
        Recall = TP/(TP+FN)
        print('Recall \n {}'.format(Recall))
        Acc = (TP+TN)/(TP+TN+FP+FN)
        print('Áccuracy \n{}'.format(Acc))
        Fscore = 2*(Precision*Recall)/(Precision+Recall)
        print('FScore \n{}'.format(Fscore))
        scores.append(accuracy)


# Calculate neuron activation for an input
def activate(weights, inputs):
    activation = weights[-1]
    for i in range(len(weights)-1):
        activation += weights[i] * inputs[i]
    return activation

# Transfer neuron activation
def function(activation):
    return 1.0 / (1.0 + exp(-activation))

# Forward propagate input to a network output
def forward_propagate(network, row):
    inputs = row
    #print("input row{}\n".format(inputs))
```

Copyright rests with Archana Mathur, astrirg lab, PESU and SSIU, ISI Banglore Please feel free to use the code and please acknowledge while doing so. The code is hosted on astrirg.org

```
    for layer in network:
        new_inputs = []
        for neuron in layer:
            activation = activate(neuron['weights'], inputs)
            neuron['output'] = function(activation)
            new_inputs.append(neuron['output'])
        inputs = new_inputs
    #print("output row{}\n".format(inputs))
    return inputs

# Calculate the derivative of an neuron output
def function_derivative(output):
    return output * (1.0 - output)

# Backpropagate error and store in neurons
def backprop_error(network, expected):
    for i in reversed(range(len(network))):
        layer = network[i]
        errors = list()
        if i != len(network)-1:
            for j in range(len(layer)):
                error = 0.0
                for neuron in network[i + 1]:
                    error += (neuron['weights'][j] * neuron['delta'])
                errors.append(error)
        else:
            for j in range(len(layer)):
                neuron = layer[j]
                errors.append(expected[j] - neuron['output'])
        for j in range(len(layer)):
            neuron = layer[j]
            neuron['delta'] = errors[j] * function_derivative(neuron['output'])

# Update network weights with error
def change_weights(network, row, l_rate):
    for i in range(len(network)):
        inputs = row[:-1]
        if i != 0:
            inputs = [neuron['output'] for neuron in network[i - 1]]
        for neuron in network[i]:
            for j in range(len(inputs)):
                neuron['weights'][j] += l_rate * neuron['delta'] * inputs[j]
            neuron['weights'][-1] += l_rate * neuron['delta']

#To fuzzify the output layer
def fuzzyout(row,mean,stdev,n_outputs):
```

Copyright rests with Archana Mathur, astrirg lab, PESU and SSIU, ISI Banglore Please feel free to use the code and please acknowledge while doing so. The code is hosted on astrirg.org

```
   z=list()
   mu=list()
   muINT=list()
   rowclass=row[-1]-1
   #print(rowclass)
   for k in range(n_outputs):
      sumz=0
      for j in range(9):
         interm=pow((row[j]-mean[k][j])/stdev[k][j],2)
         sumz=sumz+interm
         #print("row{}".format(row[j]))
         #print("mean{}".format(mean[rowclass][j]))
         #print("sum{}".format(sumz))
      weightedZ=math.sqrt(sumz)
      memMU=1/(1+(weightedZ/5))
      if 0 <= memMU <= 0.5:
         memMUINT=2*pow(memMU,2)
      else:
         temp=1-memMU
         memMUINT=1-(2*pow(temp,2))
      mu.append(memMU)
      z.append(weightedZ)
      muINT.append(memMUINT)
   return muINT

# Train a network for a fixed number of epochs
def neural_network_train(network, train, l_rate, n_epoch, n_outputs):
	#print(dataset)
	for epoch in range(n_epoch):
		#print(train)
		for row in train:
			outputs = forward_propagate(network, row)
			#print(outputs)
			expected = fuzzyout(row,mean,stdev,n_outputs)
			#print("input row{}\n".format(row))
			#expected[row[-1]-1] = 1
			#print("expected row{}\n".format(expected))
			backprop_error(network, expected)
			change_weights(network, row, l_rate)

# Initialize a network
def init_net(n_inputs, n_hidden, n_outputs):
	network = list()
	hidden_layer = [{'weights':[random() for i in range(n_inputs + 1)]} for i in range(n_hidden)]
	network.append(hidden_layer)
	output_layer = [{'weights':[random() for i in range(n_hidden + 1)]} for i in range(n_outputs)]
```

Copyright rests with Archana Mathur, astrirg lab, PESU and SSIU, ISI Banglore Please feel free to use the code and please acknowledge while doing so. The code is hosted on astrirg.org

```
	network.append(output_layer)
	return network

# Make a prediction with a network
def predict(network, row):
	outputs = forward_propagate(network, row)
	#print(outputs)
	indexOut=outputs.index(max(outputs))+1
	#print(indexOut)
	return indexOut

# Backpropagation Algorithm With Stochastic Gradient Descent
def back_propagation(train, test, l_rate, n_epoch, n_hidden):
	n_inputs = len(train[0]) - 1
	 n_outputs = len(set([row[-1] for row in train]))
	network = init_net(n_inputs, n_hidden, n_outputs)
	#print("initialize network {}\n".format(network))
	neural_network_train(network, train, l_rate, n_epoch, n_outputs)
	#print("network {}\n".format(network))
	predictions = list()
	for row in test:
		prediction = predict(network, row)
		predictions.append(prediction)
	return(predictions)

# Test Backprop on Seeds dataset
seed(1)
# load and prepare data
filename = 'data.csv'
dataset = load_csv(filename)
for i in range(len(dataset[0])-1):
	column_to_float(dataset, i)
# convert class column to integers
column_to_int(dataset, len(dataset[0])-1)
#normalize input variables
#minmax = minmax(dataset)
#normalize(dataset, minmax)
# evaluate algorithm
n_folds = 5		#k=5
l_rate = 0.2		#learning rate
n_epoch = 500		#epochs
n_hidden = 7		#since the number of inputs are 3*3, and number of classes are 6

#this part of the computation is to fuzzify inputs, PI membership function is taken for fuzzyfication. each feature vector is fuzzyfied into low, meadium and high degree of membership
center=[[250,575,900],[700,1625,2550],[1800,2500,3200]]
```

Copyright rests with Archana Mathur, astrirg lab, PESU and SSIU, ISI Banglore Please feel free to use the code and please acknowledge while doing so. The code is hosted on astrirg.org

```
#print(center)
rad=[[650,325,650],[1850,925,1850],[1400,700,1400]]
fuzzy=list()

for i in range(870):
   data=dataset[i]
   #print(data)
   l=0
   value=list()
   for j in range(3):
      d=data[j]
      for k in range(3):
         r=rad[j][k]/2
         eucleanD=math.pow((d-center[j][k]),2)
         eDist=math.sqrt(eucleanD)
         if eDist <= r :
            val = 1- 2*math.pow(eDist/(r*2),2)
            value.insert(l,val)
         else:
            y=eDist/rad[j][k]
            x=(1-(eDist/rad[j][k]))
            val = 2*math.pow(x,2)
            value.insert(l,val)
         l=l+1
   value.insert(l,data[-1])
   fuzzy.insert(i,value)
#fuzzy is the modified dataset after fuzzification
#print(fuzzy)

#This part of the code computes Mean and standard deviation of every feature of all classes to fuzzyfies
the output layer of the network
classes=[row[-1] for row in fuzzy]
Unique=np.unique(classes)
dataset_split=list()
fold_size=int(len(Unique))
for i in range(fold_size):
      fold=list()
      for row in fuzzy:
            if row[-1] == Unique[i]:
                  fold.append(row)
      dataset_split.append(fold)
i=0
mean=list()
stdev=list()
j=0
for fold in dataset_split:
```

Copyright rests with Archana Mathur, astrirg lab, PESU and SSIU, ISI Banglore Please feel free to use the code and please acknowledge while doing so. The code is hosted on astrirg.org

```
x=list()
y=list()
z=list()
x1=list()
y1=list()
z1=list()
x2=list()
y2=list()
z2=list()
for row in fold:
   if row[-1] == Unique[j]:
      x.append(row[0])
      y.append(row[1])
      z.append(row[2])
      x1.append(row[3])
      y1.append(row[4])
      z1.append(row[5])
      x2.append(row[6])
      y2.append(row[7])
      z2.append(row[8])
m1=sum(x)/float(len(x))
m2=sum(y)/float(len(y))
m3=sum(z)/float(len(z))
m4=sum(x1)/float(len(x1))
m5=sum(y1)/float(len(y1))
m6=sum(z1)/float(len(z1))
m7=sum(x2)/float(len(x2))
m8=sum(y2)/float(len(y2))
m9=sum(z2)/float(len(z2))
mean.append([m1,m2,m3,m4,m5,m6,m7,m8,m9])
st1=sum([pow(val-m1,2) for val in x])/float(len(x)-1)
st2=sum([pow(val-m2,2) for val in y])/float(len(y)-1)
st3=sum([pow(val-m3,2) for val in z])/float(len(z)-1)
st4=sum([pow(val-m4,2) for val in x1])/float(len(x1)-1)
st5=sum([pow(val-m5,2) for val in y1])/float(len(y1)-1)
st6=sum([pow(val-m6,2) for val in z1])/float(len(z1)-1)
st7=sum([pow(val-m7,2) for val in x2])/float(len(x2)-1)
st8=sum([pow(val-m8,2) for val in y2])/float(len(y2)-1)
st9=sum([pow(val-m9,2) for val in z2])/float(len(z2)-1)
std1=math.sqrt(st1)
std2=math.sqrt(st2)
std3=math.sqrt(st3)
std4=math.sqrt(st4)
std5=math.sqrt(st5)
std6=math.sqrt(st6)
std7=math.sqrt(st7)
```

Copyright rests with Archana Mathur, astrirg lab, PESU and SSIU, ISI Banglore Please feel free to use the code and please acknowledge while doing so. The code is hosted on astrirg.org

```
    std8=math.sqrt(st8)
    std9=math.sqrt(st9)
    stdev.append([std1,std2,std3,std4,std5,std6,std7,std8,std9])
    j=j+1
#print(mean)
#print(stdev)
run_algorithm(fuzzy, back_propagation, n_folds, l_rate, n_epoch, n_hidden)
```

Copyright rests with Archana Mathur, astrirg lab, PESU and SSIU, ISI Banglore Please feel free to use the code and please acknowledge while doing so. The code is hosted on astrirg.org