Mushroom Edibility Classification
Using Back propagation

Report submitted to the SASTRA Deemed to be University
as the requirement for the course

BEIDEI708: SOFT COMPUTING

Submitted by

Arun Balaji T S R
(Reg. No.: 121006097)

February 2021

SCHOOL OF ELECTRICAL AND ELECTRONICS ENGINEERING
THANJAVUR, TAMIL NADU, INDIA – 613 401

SCHOOL OF ELECTRICAL AND ELECTRONICS ENGINEERING

THANJAVUR – 613 401

Bonafide Certificate

This is to certify that the report titled "**Mushroom Edibility Classification using Back Propagation"** submitted as a requirement for the course, **BEIDEI708: SOFT COMPUTING** for B.Tech. EIE Programme, is a bonafide record of the work done by **Mr.Arun Balaji T S R (Reg. No.121006097)** during the academic year 2020-21, in the School of EEE, under my supervision.

Signature of Project Supervisor :

Name with Affiliation : Dr.K.Ramkumar, Professor, SEEE

Date :

Project *Viva voce* held on _____

Examiner 1 **Examiner 2**

SCHOOL OF ELECTRICAL AND ELECTRONICS ENGINEERING

THANJAVUR – 613 401

Declaration

I declare that the thesis titled "**Mushroom Edibility Classification using Back Propagation**" submitted by me is an original work done by me under the guidance of **Dr.K.Ramkumar, Professor, School of Electrical and Electronics Engineering, SASTRA Deemed to be University** during the seventh semester in the academic year 2020-21, in the **School of Electrical and Electronics Engineering**. The work is original and wherever I have used materials from other sources, I have given due credit and cited them in the text of the thesis. This thesis has not formed the basis for the award of any degree, diploma, associate-ship, fellowship or other similar title to any candidate of any University.

Signature of the candidate :

Name of the candidate : Arun Balaji T S R (Reg.no.: 121006097)

Date : 11th February, 2021

ACKNOWLEDGEMENT

First of all, I am grateful to The Almighty God for giving me the opportunity and mental strength for making me choose the right path in spite of all ups and downs. I express my gratitude to **Dr.S.Vaidhyasubramaniam**, Vice Chancellor, SASTRA Deemed University who provided all facilities and necessary encouragement during the course of study. I extend my sincere thanks to **Dr.S.Swaminathan**, Dean of Planning and Development, **Dr.R.Chandramouli**, Registrar, SASTRA Deemed University for providing the opportunity to pursue this project. I dedicate my whole hearted thanks to **Dr.K.Thenmozhi**, Dean (SEEE) and **Dr.A.Krishnamoorthy**, Associate Dean (EIE) who motivated me during this project work.

I would also like to thank our guide **Dr.K.Ramkumar** , Professor, School of Electrical and Electronics Engineering for sharing his immense knowledge and always keeping us motivated and sharing his valuable feedback.

Table of Contents

List of Figures

List of Tables

ABSTRACT

Mushrooms can either be edible or poisonous. This project is aimed at predicting the edibility of the mushroom using attributes such as its colour, odour, habitat, shape etc., This prediction is achieved through multi layered neural networks trained using back propagation algorithm. The data set used to train the network is obtained from UCI Machine Learning repository and Kaggle.

The dataset has 22 attributes and 8124 instances of data related to 23 different species of Mushrooms in Agaricus and Lepiota family. The instances of data has different character to represent different parameters of mushroom. Those characters are converted into numerical values using LabelEncoder library and the data is pre-processed to required form.

Then the neural network structure is initialized with random weights. First we propagate forward from input layer to output layer and compute the outputs and cost and save them. The neuron output is calculated by multiplying the inputs with weights, adding bias and then giving it to an activation function such as sigmoid function. Then the back propagation propagates backward from output layer to input layer and adjusts the weights of it using gradient descent to achieve less cost function output which in turn gives high accuracy of prediction. This takes place in iteration till least possible cost function output is achieved.

The accuracy values are printed out and different graphs like Accuracy vs iteration curve and Cost vs iteration curve are plotted to learn the characteristics of the neural network.

CHAPTER 1

INTRODUCTION

Mushrooms serve as a good source of Vitamin D and several other Vitamin B nutrients such as riboflavin, thiamine, folate, niacin etc. It is also free from cholesterol and there are proven studies that it reduces cancer and improves the life of diabetes patients and cancer survivors.

Mushrooms have been used for purposes like cooking, making medicines, making cosmetic products for centuries. They are a type of Fungi which grows overnight rapidly. There are about 14000+ species of mushrooms in existence whereas only around 25 species are widely accepted to be edible and used for cooking purposes.

1.1 Need for edibility classification using neural network

Mushrooms can be edible or poisonous. Poisonous mushrooms has toxins which can cause food poisoning and sometimes can even be deadly. Fungi being a living particle, mutates often and thereby about 800 species of mushrooms are found newly every year. So, it is almost impossible to manually identify the mushrooms which are edible.

Currently mushrooms are harvested by using local mushroom guides who are aware of different types of species in that area and avoid the ones which seem to be poisonous. But on the long run it is not possible as large number of newly formed species similar to each other are evolved. So, a machine learning technique for mushroom edibility detection becomes necessary.

Some of the following points are noted while segregating edible mushrooms manually:

1) Avoiding mushrooms with white gills, a ring on stem or sack at base
2) Avoiding mushrooms with red colour cap or stem

Such techniques can eliminate some of the edible mushrooms in error which leads to loss of cost to farmers. So, a neural network with optimum weights is the best way to segregate edible mushrooms.

CHAPTER 2

DATASET

2.1 SOURCE OF DATASET AND TYPE OF DATA

The data set is obtained from UCI Machine learning repository and Kaggle. It has 22 attributes and 8124 instances of data of mushrooms from 23 species of gilled mushrooms in Agaricus and Lepiota family. The source of data is the records drawn from the research paper " The Audubon Society field guide to North America (1981) ".

It is a Comma Seperated Values (.csv) file with attributes as columns and instances as rows. The first column consists of the output data - edible or poisonous. The remaining columns has the other attributes.

2.2 DATASET ATTRIBUTES

There are different parts in a mushroom as follows:

Cap	It is the upper part of mushroom
Gill	It is the part below the cap of mushroom
Stalk (Stem)	It is the part supporting the cap of mushroom
Veil	Thin membrane covering the cap of mushroom
Ring	Membrane located under cap of mushroom
Volva	Membrane that covers mushroom when its immature
Spores	Microscopic particles that acts as reproductive agents of mushroom

Table.2.1 Parts of Mushroom

The data about the size of these parts, the mushroom's habitat, colour, odour etc can be used as attributes for detecting the edibility of mushrooms.

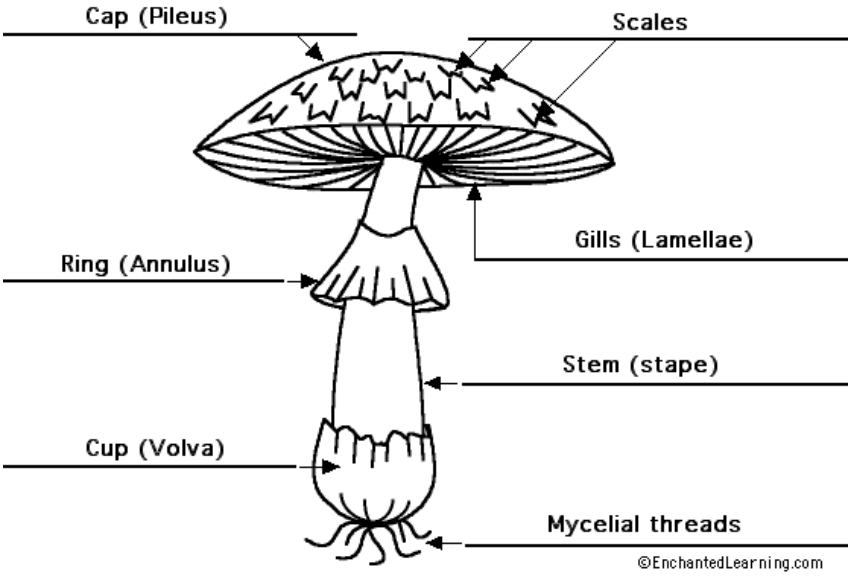


Fig.2.1 Parts of mushroom

Attribute	Values that can be taken
class	edible=e, poisonous=p
Cap-shape	bell=b,conical=c,convex=x,flat=f, knobbed=k,sunken=s
Cap-surface	fibrous=f,grooves=g,scaly=y,smooth=s
cap-surface	fibrous=f,grooves=g,scaly=y,smooth=s
cap-color	brown=n,buff=b,cinnamon=c,gray=g,green=r,pink=p,purple=u,red=e, white=w,yellow=y
bruises	bruises=t,no=f
odor	almond=a,anise=l,creosote=c,fishy=y,foul=f,musty=m,none=n,pungent=p,spicy=s
gill-attachment	attached=a, descending=d, free=f, notched=n
gill-spacing	close=c,crowded=w,distant=d
gill-size	broad=b,narrow=n

gill-color	black=k,brown=n,buff=b,chocolate=h,gray=g, green=r,orange=o,pink=p,purple=u,red=e,white=w,yellow=y
stalk-shape	enlarging=e,tapering=t
stalk-root	bulbous=b,club=c,cup=u,equal=e, rhizomorphs=z,rooted=r,missing=?
stalk-surface-above-ring	fibrous=f,scaly=y,silky=k,smooth=s
stalk-surface-below-ring	fibrous=f,scaly=y,silky=k,smooth=s
stalk-color-above-ring	brown=n,buff=b,cinnamon=c,gray=g,orange=o, pink=p,red=e,white=w,yellow=y
stalk-color-below-ring	brown=n,buff=b,cinnamon=c,gray=g,orange=o, pink=p,red=e,white=w,yellow=y
veil-type	partial=p,universal=u
veil-color	brown=n,orange=o,white=w,yellow=y
ring-number	none=n,one=o,two=t
ring-type	cobwebby=c,evanescent=e,flaring=f,large=l, none=n,pendant=p,sheathing=s,zone=z
spore-print-color	black=k,brown=n,buff=b,chocolate=h,green=r, orange=o,purple=u,white=w,yellow=y
population	abundant=a,clustered=c,numerous=n, scattered=s,several=v,solitary=y
habitat	grasses=g,leaves=l,meadows=m,paths=p, urban=u,waste=w,woods=d

Table.2.2 Attributes in dataset

CHAPTER 3

BACK PROPAGATION

Neural network is a set of layers which has a input layer, output layers and ‘n’ number of hidden layers between them connected to each other. The connections between them has a weight. The neurons in a neural network give an output based on the activation function.

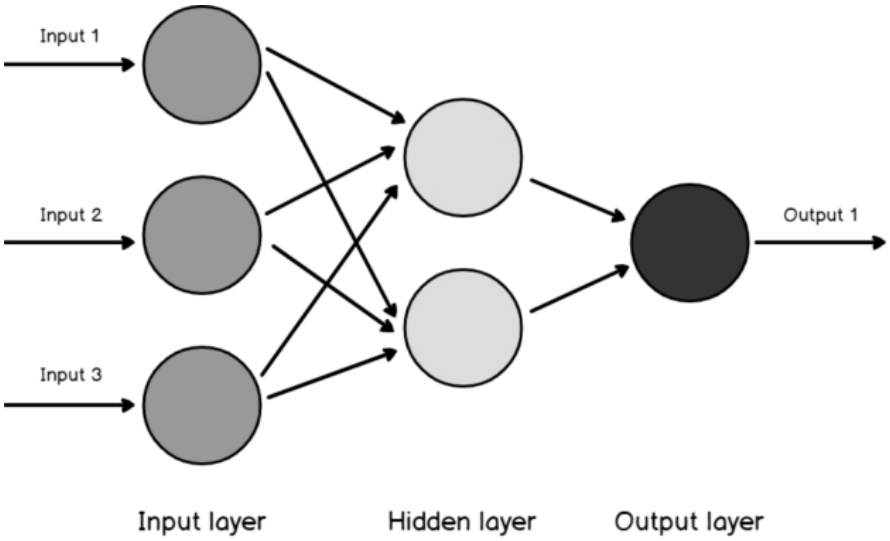


Fig.3.1 Feed forward Neural Network

The activation function is mostly a sigmoid function which returns ‘1’ if the input is positive and ‘0’ elsewhere. The input to the neuron is multiplied with the weights and added bias before passing it to the activation function.

$$Z=wx+B$$

where w stands for weight, b stands for bias and Z is the input to be given to activation function.

It is necessary that the weights are set appropriate so that the cost function is less and maximum accuracy is obtained. Back Propagation is a method to determine the weights of a neural network to obtain least cost. By this method the weights are initially set with random values. Then we propagate forward, calculate the output of neuron and the cost and save them. Then we again propagate backward from output layer to input layer and at each neuron we try to get optimum output by changing the weights so as to reduce the cost function output. These weights are calculated by using gradient descent method. This is performed for several iterations and the weights are adjusted every time. Thus the cost reduces as iteration keeps on increasing.

The gradient descent considers this as a slope (rate of change of weights with respect to rate of change of output) and keeps on updating the values of weights until the point where the slope is zero. At this point we can stop the iteration as the result converges.

ADVANTAGES OF BACK PROPAGATION:

- Prior knowledge about the network is not needed
- Can be used for any number of hidden layers/nodes
- Easy to program and fast

PROBLEM WITH BACK PROPAGATION:

- The gradient descent method to update the weights may some time get struck with local minima. This is called local minima problem. This occurs because it cant go uphill with this method.

CHAPTER 4

IMPLEMENTATION

4.1 PRE-PROCESSING OF DATA

The data has to be converted to required format before it can be used. So pre processing is done. In the data set, the information is present as characters which represent a feature in mushroom. To use the data in neural network it has to be converted to relevant numerical value.

.

class	cap-shape	cap-surface	cap-color	bruises
p	x	s	n	t
e	x	s	y	t
e	b	s	w	t
p	x	y	w	t
e	x	s	g	f
e	x	y	y	t
e	b	s	w	t

class	cap-shape	cap-surface	cap-color	bruises
1	5	2	4	1
0	5	2	9	1
0	0	2	8	1
1	5	3	8	1
0	5	2	3	0
0	5	3	9	1
0	0	2	8	1

Fig.4.1 Data preprocessing

This is achieved by using the Label Encoder library of the Scikit Learn package. The fit() method of this library encodes the characters into numbers which can be decoded back later. The following code snippet is used for this purpose.

```
for col in data.columns:
    data[col]=label.fit_transform(data[col])
```

4.2 PREPARING THE TRAINING AND TESTING DATASET

The dataset downloaded from the UCI Machine Learning Repository has to be split into two, one for training purpose and another for testing the neural network. The splitting of data set should be randomized. For this, train_test_split library of Scikit Learn package is used. It splits the dataset into two based on the input code as shown below.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

4.3 DEFINING THE STRUCTURE OF NEURAL NETWORK

The number of nodes in input layer , the number of hidden layers and nodes in each hidden layer, the number of nodes in output layer are specified. In this data set, the input layer has 22 nodes as there are 22 attributes in input training data and one node in output layer which specifies whether the mushroom is poisonous or not.

Only one hidden layer is used here which has 22 nodes. These layers are specified in the form of arrays.

4.4 INITIALISING RANDOM WEIGHTS AND BIAS

The weights between the nodes are initialized randomly at beginning. Similarly the bias is set to zero initially. This is an assumption which will be modified later by the network using back propagation algorithm.

4.5 DEFINING THE SIGMOID FUNCTION

Sigmoid function is used as the activation function in this neural network. A function that takes in a value, performs sigmoid function and returns an output is written.

$$f(x) = \frac{1}{1+e^{-(x)}}$$

Following is the python code to perform this function,

```
def sigmoid(z):
    return 1/(1+np.exp(-z))
```

4.6 FORWARD PROPAGATION

It is also called as forward pass. In this process, we are moving from input layer to output layer through the hidden layer to know the output of each neuron for the given input. The output is based on activation function, weights and biases. Either the sigmoid function or tanh function can be used as an activation function. The output of the neurons are stored in an array.

4.8 COST CALCULATION

Cost function gives the difference between the expected output and actual output. Lesser the cost function output, higher is the accuracy of the network. There are many types of cost functions and here cross entropy cost function is used.

4.9 BACK PROPAGATION FUNCTION

This is a reverse process of forward propagation. Here we start from output layer and move backwards to input layer. While moving backwards, the weights and biases between the nodes are adjusted to get the desired output of the neuron thereby reducing the cost function output. The updation of weights and biases is done by Gradient descent method which is implemented after this. This process continues for several iteration till the cost function reduces.

4.10 GRADIENT DESCENT

Gradient Descent is a method used to find the values of weights during back propagation. It determines the weights so as to reduce the cost function. It is the slope of function which measures the rate of change of output with respect to input. The learning rate is also specified in this process. Learning rate determines how fast the algorithm will complete training.

4.11 CALLING ALL THE ABOVE DEFINED FUNCTIONS

A function is written to call all the above functions and perform training of the neural network. First the neural network is defined, then random weights are initialized. After that, forward propagation and back propagation takes place. After the completion of training using the training dataset the final weights will be assigned. Then, the test data set is given to test the output and find the accuracy of the network. It can also be seen that the output of cost function reduces with each iteration. The final accuracy of the network is printed out.

4.12 PLOTTING GRAPHS TO GET INSIGHTS

Using matplotlib library and ScikitLearn package, the graph between number of iterations and output of cost function and graph between accuracy and number of iterations is plotted to get an idea about the neural network.

CHAPTER 5

RESULTS AND DISCUSSION

5.1 ACCURACY

After 2000 iterations, the accuracy obtained is about 87.75% using tanh activation function. If sigmoid function is used as an activation function, the accuracy reduces considerably.

```
Accuracy for test data set is 87.75384615384615 %
```

Fig.5.1 Output of network accuracy

5.2 COST FUNCTION OUTPUT ANALYSIS

As the iteration increases, the cost function output decreases which shows that the training happens correctly. A graph is plotted between the cost function output and number of iterations as shown below. For plotting graphs, Scikit Learn and Matplotlib packages are used.

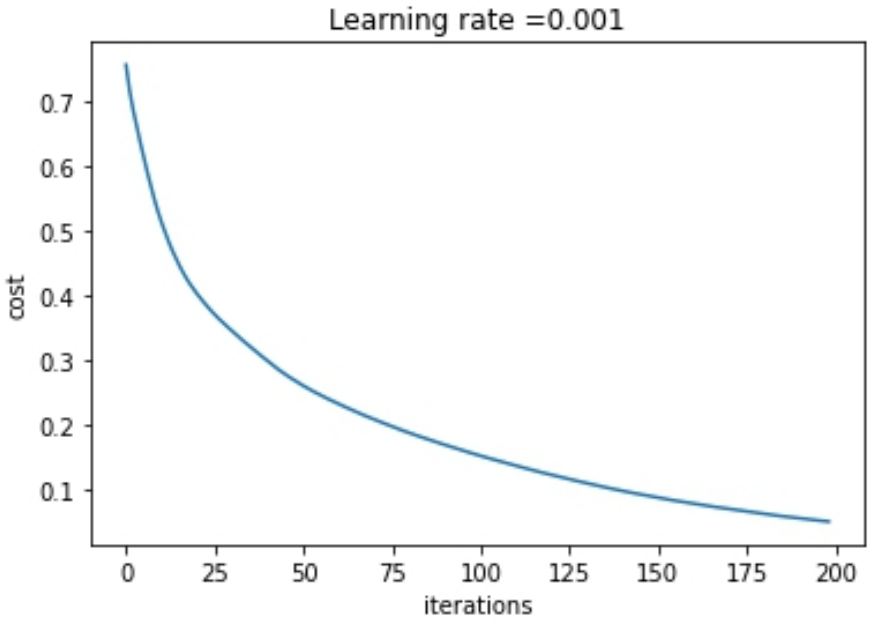


Fig.5.2 Cost vs Iteration curve

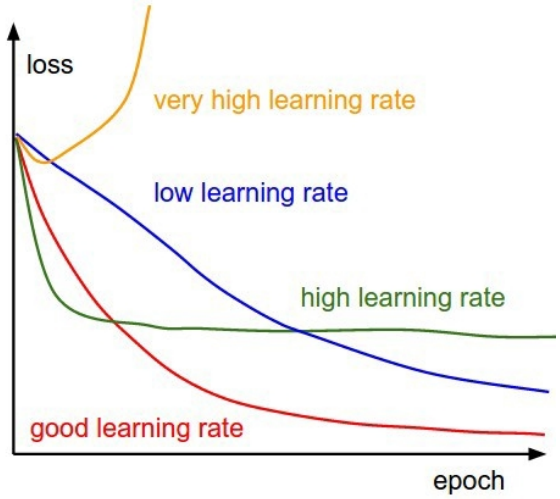


Fig.5.3 Learning rate prediction using Cost vs iteration curve

From Fig.5.3, the good learning rate curve matches with our Fig.5.2 Cost vs iteration curve. So this proves that neural network is performing as expected.

```
Cost after iteration 315: 0.636986
Cost after iteration 320: 0.634288
Cost after iteration 325: 0.631547
Cost after iteration 330: 0.628766
Cost after iteration 335: 0.625948
Cost after iteration 340: 0.623095
Cost after iteration 345: 0.620211
Cost after iteration 350: 0.617297
Cost after iteration 355: 0.614357
Cost after iteration 360: 0.611394
Cost after iteration 365: 0.608409
Cost after iteration 370: 0.605405
Cost after iteration 375: 0.602385
Cost after iteration 380: 0.599351
Cost after iteration 385: 0.596305
Cost after iteration 390: 0.593249
Cost after iteration 395: 0.590186
Cost after iteration 400: 0.587117
Cost after iteration 405: 0.584046
```

Fig.5.4 Portion of output showing reducing values of cost after each iteration

5.3 ACCURACY vs NUMBER OF ITERATION ANALYSIS

A graph is plotted with x-axis as number of iterations and y-axis as accuracy. Following is obtained,

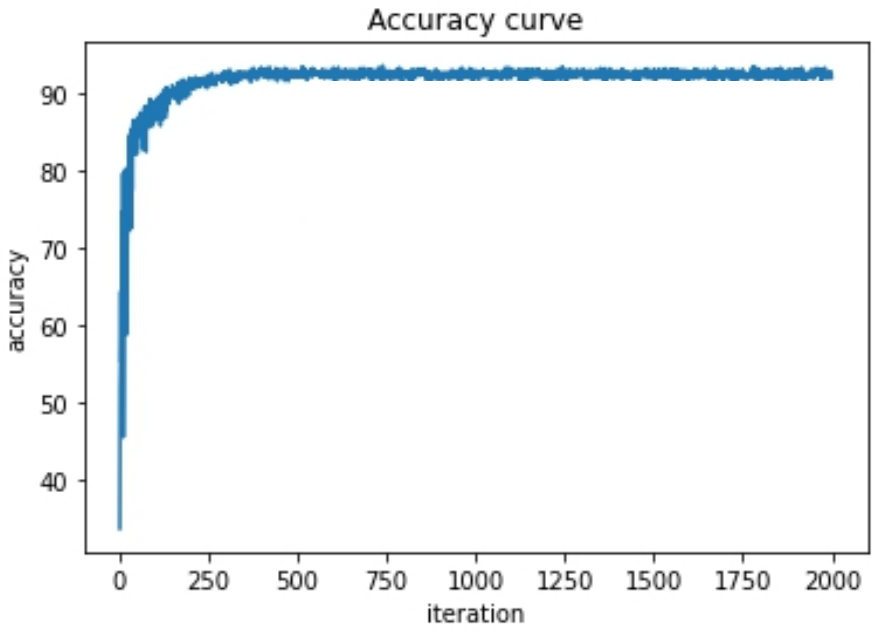


Fig.5.5 Accuracy vs iteration curve

This shows that accuracy increases with increase in iteration and becomes stable after a certain number of iterations.

CHAPTER 6

PROGRAM CODE

Uploading dataset:

In [3]:
```
from google.colab import files
uploaded = files.upload()
```

```
Saving mushrooms.csv to mushrooms.csv
```

Importing libraries:

In [53]:
```
import numpy as np
import pandas as pd
import sklearn
from sklearn.preprocessing import LabelEncoder
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
label = LabelEncoder()
```

Preprocessing of data into necessary format:

In [54]:
```
data = pd.read_csv("mushrooms.csv")
```

In [36]:
```
data.shape
```

Out[36]:
```
(8124, 23)
```

In [55]:
```
data.head()
```

Out[55]:

	class	cap-shape	cap-surface	cap-color	bruises	odor	gill-attachment	gill-spacing	gill-size	gill-color	stalk shap
0	p	x	s	n	t	p	f	c	n	k	e
1	e	x	s	y	t	a	f	c	b	k	e
2	e	b	s	w	t	l	f	c	b	n	e
3	p	x	y	w	t	p	f	c	n	n	e
4	e	x	s	g	f	n	f	w	b	k	t

Converting the labelled data into numerical using LabelEncoder of ScikitLearn

In [56]:
```
for col in data.columns:
    data[col]=label.fit_transform(data[col])

data.head()
```

Out[56]:

	class	cap-shape	cap-surface	cap-color	bruises	odor	gill-attachment	gill-spacing	gill-size	gill-color	stalk-shape	stalk-root	stalk-surface-above-ring	stalk-surface-below-ring	sta col abo rin
0	1	5	2	4	1	6	1	0	1	4	0	3	2	2	7
1	0	5	2	9	1	0	1	0	0	4	0	2	2	2	7
2	0	0	2	8	1	3	1	0	0	5	0	2	2	2	7
3	1	5	3	8	1	6	1	0	1	5	0	3	2	2	7
4	0	5	2	3	0	5	1	1	0	4	1	3	2	2	7

In [50]:
```
df.to_csv('mushroom_converted.csv')
```

Splitting input and output data into seperate arrays:

In [51]:
```
data = np.genfromtxt('mushrooms_converted.csv', delimiter = ',')
X = data[:,1:23]
y = data[:, 0]
print(X)
print(y)
```

```
[[5. 2. 4. ... 2. 3. 5.]
 [5. 2. 9. ... 3. 2. 1.]
 [0. 2. 8. ... 3. 2. 3.]
 ...
 [2. 2. 4. ... 0. 1. 2.]
 [3. 3. 4. ... 7. 4. 2.]
 [5. 2. 4. ... 4. 1. 2.]]
[1. 0. 0. ... 0. 1. 0.]
```

Splitting the data into test and train datasets randomly using train_test_split library

In [57]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
X_train = X_train.T
y_train = y_train.reshape(1, y_train.shape[0])
X_test = X_test.T
y_test = y_test.reshape(1, y_test.shape[0])
print ('Train X size is ', X_train.shape)
print ('Train Y size is ', y_train.shape)
print ('\nTest Dataset size is ', X_test.shape)
```

```
Train X size is  (22, 6499)
Train Y size is  (1, 6499)

Test Dataset size is  (22, 1625)
```

Defining the neural network structure (Number of nodes in input and hidden layer)

In [58]:
```
def neural_network(X, Y):
    input_unit = X.shape[0]
    hidden_unit = 22
    output_unit = Y.shape[0]
    return (input_unit, hidden_unit, output_unit)
```

In [59]:
```
(input_unit, hidden_unit, output_unit) = neural_network(X_train, y_train)
print("There are "+ str(input_unit)+" input layers and "+str(hidden_unit)+" hidden layers")
```

```
There are 22 input layers and 22 hidden layers
```

Allotting random weights and zero bias initially

In [60]:
```
def random_weights(input_unit, hidden_unit, output_unit):
    np.random.seed(2)
    W1 = np.random.randn(hidden_unit, input_unit)*0.01
    b1 = np.zeros((hidden_unit, 1))
    W2 = np.random.randn(output_unit, hidden_unit)*0.01
    b2 = np.zeros((output_unit, 1))
    weights = {"W1": W1, "b1": b1, "W2": W2, "b2": b2}
    return weights
```

Function to perform Sigmoid function

In [61]:
```
def sigmoid(z):
    return 1/(1+np.exp(-z))
```

Forward Propagation

In [75]:
```
def forward_propagation(X, weights):
    W1 = weights['W1']
    b1 = weights['b1']
    W2 = weights['W2']
    b2 = weights['b2']
    Z1 = np.dot(W1, X) + b1
    Activation_one = np.tanh(Z1)
    #Activation_one = sigmoid(Z1) (Sigmoid had less accuracy so used tanh activation)
    Z2 = np.dot(W2, Activation_one) + b2
    Activation_two = sigmoid(Z2)
    cache = {"Z1": Z1,"Activation1": Activation_one,"Z2": Z2,"Activation2": Activation_two}
    return Activation_two, cache
```

Calculating cost

In [88]:
```
def cost_function(Activation_two, Y):
    m = Y.shape[1]
    logprobs = np.multiply(np.log(Activation_two), Y) + np.multiply((1-Y), np.log(1 - Activation_two))
    cost = - np.sum(logprobs) / m
    cost = float(np.squeeze(cost))
    return cost
```

Propagating Backwards

In [66]:
```
def backward_propagation(weights, cache, X, Y):
    #number of training example
    m = X.shape[1]

    W1 = weights['W1']
    W2 = weights['W2']
    Activation_one = cache['Activation1']
    Activation_two = cache['Activation2']

    dZ2 = Activation_two-Y
    dW2 = (1/m) * np.dot(dZ2, Activation_one.T)
    db2 = (1/m) * np.sum(dZ2, axis=1, keepdims=True)
    dZ1 = np.multiply(np.dot(W2.T, dZ2), 1 - np.power(Activation_one, 2))
    dW1 = (1/m) * np.dot(dZ1, X.T)
    db1 = (1/m)*np.sum(dZ1, axis=1, keepdims=True)

    grads = {"dW1": dW1, "db1": db1, "dW2": dW2,"db2": db2}

    return grads
```

Gradient Descent (Updating weights) and setting learning rate

In [68]:
```
def gradient_descent(weights, grads, learning_rate = 0.01):
    W1 = weights['W1']
    b1 = weights['b1']
    W2 = weights['W2']
    b2 = weights['b2']

    dW1 = grads['dW1']
    db1 = grads['db1']
    dW2 = grads['dW2']
    db2 = grads['db2']

    W1 = W1 - learning_rate * dW1
    b1 = b1 - learning_rate * db1
    W2 = W2 - learning_rate * dW2
    b2 = b2 - learning_rate * db2

    updated_weights = {"W1": W1, "b1": b1,"W2": W2,"b2": b2}

    return updated_weights
```

Model program for testing dataset

In [93]:
```
def neural_network_model(X, Y, hidden_unit, num_iterations = 1000):
    np.random.seed(3)
    input_unit = neural_network(X, Y)[0]
    output_unit = neural_network(X, Y)[2]

    weights = random_weights(input_unit, hidden_unit, output_unit)

    W1 = weights['W1']
    b1 = weights['b1']
    W2 = weights['W2']
    b2 = weights['b2']

    for i in range(0, num_iterations):
        A2, cache = forward_propagation(X, weights)
        cost = cost_function(A2, Y)
        grads = backward_propagation(weights, cache, X, Y)
        weights = gradient_descent(weights, grads)
        if i % 5 == 0:
            print ("Cost after iteration %i: %f" %(i, cost))
    return weights
```

Cost reduces after each iteration of training

In [94]:
```
final_weights = neural_network_model(X_train, y_train, 4, num_iterations=1000)
```

```
Cost after iteration 0: 0.693059
Cost after iteration 5: 0.693005
Cost after iteration 10: 0.692950
Cost after iteration 15: 0.692893
Cost after iteration 20: 0.692835
Cost after iteration 25: 0.692774
Cost after iteration 30: 0.692710
Cost after iteration 35: 0.692642
Cost after iteration 40: 0.692571
Cost after iteration 45: 0.692494
Cost after iteration 50: 0.692412
Cost after iteration 55: 0.692323
Cost after iteration 60: 0.692228
Cost after iteration 65: 0.692125
Cost after iteration 70: 0.692013
Cost after iteration 75: 0.691892
Cost after iteration 80: 0.691760
Cost after iteration 85: 0.691618
Cost after iteration 90: 0.691462
Cost after iteration 95: 0.691293
Cost after iteration 100: 0.691109
Cost after iteration 105: 0.690909
Cost after iteration 110: 0.690691
Cost after iteration 115: 0.690454
```

In [95]:
```
def prediction(weights, X):
    Activation2, cache = forward_propagation(X, weights)
    predictions = np.round(Activation2)
    return predictions
```

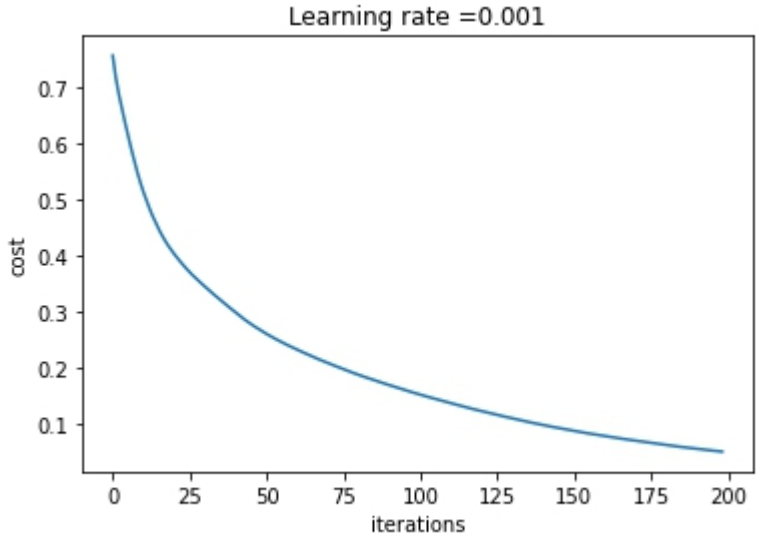
Accuracy Calculations

In [98]:
```
predictions = prediction(final_weights, X_test)
accuracy=float((np.dot(y_test, predictions.T) + np.dot(1 - y_test, 1 - predictions.T))/float(y_test.size)*100)
print ("Accuracy for test data set is "+str(accuracy)+' %')
```

```
Accuracy for test data set is 87.75384615384615 %
```

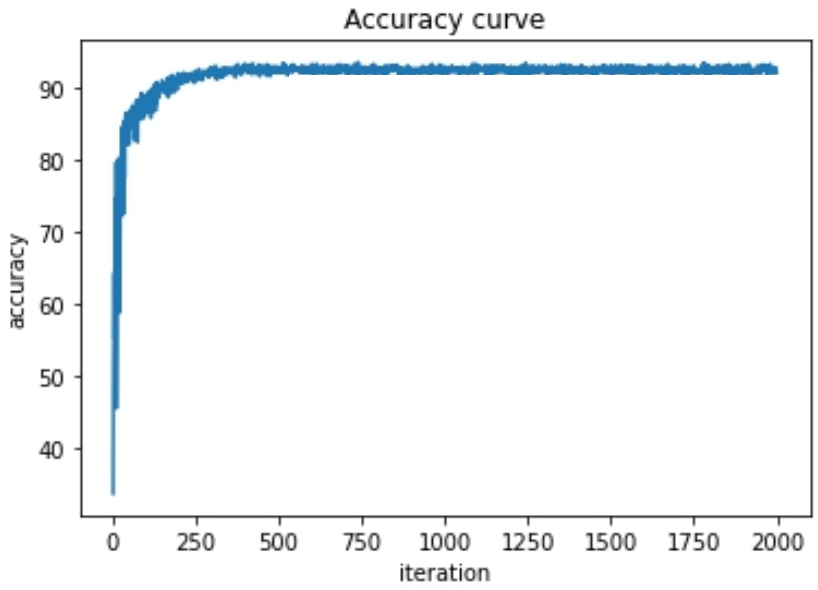
[ ]
```
from sklearn.neural_network import MLPClassifier
mlp = MLPClassifier(max_iter=2000, learning_rate_init=0.001,)
history=mlp.fit(X_train, y_train)
mlp_train_score = mlp.score(X_train, y_train)
mlp_test_score = mlp.score(X_test, y_test)
```

[ ]
```
import numpy as np
import matplotlib.pyplot as plt
plt.ylabel('cost')
plt.xlabel('iterations')
plt.title("Learning rate =" + str(0.001))
plt.plot(mlp.loss_curve_)
plt.show()
```



```
[ ]  from warnings import simplefilter
     from sklearn.exceptions import ConvergenceWarning
     simplefilter("ignore", category=ConvergenceWarning)
     accuracy1=[]
     iteration=[]
     for i in range(1,2000):
       mlp = MLPClassifier(max_iter=i, learning_rate_init=0.001,)
       history=mlp.fit(X_train, y_train)
       mlp_test_score = mlp.score(X_test, y_test)
       accuracy1.append(mlp_test_score*100)
       iteration.append(i)

     plt.ylabel('accuracy')
     plt.xlabel('iteration')
     plt.title("Accuracy curve")
     plt.plot(iteration, accuracy1,)
     plt.show()
```



CHAPTER 7

CONCLUSION

Thus it can be seen that the mushroom can be classified into two types - edible and poisonous using multi layer neural networks trained using back propagation. It can also be seen that accuracy increases as iteration increases and cost decreases as iteration increases. Such machine learning models combined with image processing can be used real-time to detect and remove poisonous mushrooms thereby reducing food poisoning. The network outputs ‘1’ if its edible and ‘0’ if its poisonous.

REFERENCES

1) **Eyad Sameh Alkronz et.al,** "Classification of Mushroom using Artificial Neural Network", *International Journal of Academic and Applied Research (IJAAR),* Vol. 3 Issue 2, February 2019

2) **Rakesh Kumar Y et.al**, "Machine Learning Methods to Classify Mushrooms for Edibility-A Review", *International Journal for Modern Trends in Science and Technology*, 2020, https://doi.org/10.46501/IJMTST060909

3) **S.K. Verma et.al,** "Mushroom Classification Using ANN and ANFIS Algorithm", *IOSR Journal of Engineering (IOSRJEN),* Vol. 8 Issue 1, January 2018

4) Building a Neural Network with a Single Hidden Layer using Numpy, Towards Data Science
https://towardsdatascience.com/building-a-neural-network-with-a-single-hidden-layer-using-numpy-923be1180dbf

5) Andrew Ng's courses in Coursera and Deep Learning.AI and his blogs

6) Mushroom Data Set - UCI Machine Learning Repository
https://archive.ics.uci.edu/ml/datasets/mushroom

7) Mushroom Classification - Kaggle
https://www.kaggle.com/uciml/mushroom-classification