



**MALAD KANDIVALI EDUCATION SOCIETY'S
NAGINDAS KHANDWALA COLLEGE OF COMMERCE,
ARTS & MANAGEMENT STUDIES & SHANTABEN NAGINDAS
KHANDWALA COLLEGE OF SCIENCE
MALAD [W], MUMBAI – 64
(AUTONOMOUS)**

**(Reaccredited 'A' Grade by NAAC)
(AFFILIATED TO UNIVERSITY OF MUMBAI)
(ISO 9001:2015)**

CERTIFICATE

Name: Mr. kuldeep sushil patel

Roll No: 574

Programme: BSc CS

Semester: V

This is certified to be a bonafide record of practical works done by the above student in the college laboratory for the course **Artificial Intelligence** (Course Code: **1851UCSPR**) for the partial fulfillment of Fifth Semester of BSc CS during the academic year 2020-2021.

The journal work is the original study work that has been duly approved in the year 2020-2021 by the undersigned.

External Examiner

**Prof. Elizabeth Leah George
(Subject-In-Charge)**

Date of Examination:

(College Stamp)

Name: Kuldeep Shushil Patel

Roll no: 574

Name: Kuldeep Shushil Patel

Roll No: 574

Subject: Artificial Intelligence (1851UCSPR)

Class: T.Y.B.SC (CS)

SEMESTER-V

NO	DATE	TITLE	SIGN
1.	17/07/2020	Blind Search Algorithms: 1. Implement Breadth First Search (BFS) algorithm 2. Implement Depth First Search (DFS) algorithm 3. Implement Iterative deepening Search (IDS) algorithm	
2.	31/07/2020	Heuristic Search Algorithm: Implement A* search algorithm	
3.	14/08/2020	Implementation of Neural Network on IRIS dataset	
4.	21/08/2020	Implementation of basic neural network model with 4 activation functions on Pima Indians onset of diabetes dataset	
5.	04/09/2020	Performing AND & OR Operations in the Neural Network	
6.	18/09/2020	Prediction Algorithm - Use of different packages on dataset of Cat and Non-Cat images	
7.	25/09/2020	Simple Linear Regression	
8.	09/10/2020	Implement Support Vector Algorithm	
9.	16/10/2020	Decision Tree Learning	

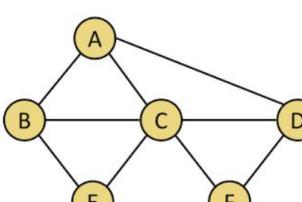
PRACTICAL 1

Aim :1]implement breadth first and depth first search algorithm

Theory :

- Breadth First Traversal (or Search) for a graph is similar to Breadth First Traversal of a tree.
- The only catch here is, unlike trees, graphs may contain cycles, so we may come to the same node again.
- To avoid processing a node more than once, we use a boolean visited array.
- For simplicity, it is assumed that all vertices are reachable from the starting vertex

Code:



```
[ ] graph_P= {'A': ['B', 'C', 'D'],
             'B': ['C', 'E'],
             'C': ['D', 'E', 'F'],
             'D': ['F'],
             'E': [],
             'F': []}
visited = []
queue = [A]
```

AI_practicals 65 - Colaboratory

File Edit View Insert Runtime Tools Help All changes saved

Table of contents

- Practical 1
 - 1 : implement breadth first and depth first search algorithm
 - 2 : implement iterative deepening search
- Practical 2
 - 1 : implement A* search algorithm
- Practical 3
 - 1 : implementation of neural net on IRIS DATASET
- Practical 4
 - 1 : pima indians onset of diabetes dataset
- Practical 5
 - 1 : performing AND & OR operations in the neural network
- Practical 6
 - 1 : prediction algorithm - use of different packages on dataset of cat and non-cat images
- Practical 7
 - 1 : simple linear regression
- Practical 8

```
[ ] graph_P= {'A': ['B', 'C', 'D'],
             'B': ['C', 'E'],
             'C': ['D','E','F'],
             'D': ['F'],
             'E': [],
             'F': []}
visited = []
queue = []

def bfs(visited, graph_P, node):
    visited.append(node)
    queue.append(node)

    while queue:
        s = queue.pop(0)
        print (s, end = " ")
        for neighbour in graph_P[s]:
            if neighbour not in visited:
                visited.append(neighbour)
                queue.append(neighbour)

bfs(visited, graph_P, 'A')
```

Type here to search

02:32 AM 10-11-2020

Output:

The screenshot shows a Jupyter Notebook environment with the following details:

- Title Bar:** AI_practicals 65 - Colaboratory, Copy of AI Practical-575.ipynb, xQDnQM6BxKZA4f+D75FrJ4ozB, ai practical - Google Drive.
- Toolbar:** Apps, Gmail, YouTube, Maps, Translate.
- Code Cell:** Contains Python code for a breadth-first search (BFS) algorithm. The code includes a helper function `bfs(visited, graph_P, 'A')` and a main loop that iterates over a queue of neighbors, adding them to the visited set and the queue if they are not already visited.
- Table of Contents:** Lists practicals from 1 to 7, each with a brief description of the task.
- Graph Diagram:** A directed graph with nodes A through H. Node A is the root node at the top. It has three children: B, C, and D. Node C has three children: E, F, and G. Node D has one child: H. The edges are colored: A to B (orange), A to C (grey), A to D (blue); C to E (green), C to F (orange), C to G (grey); D to H (blue).
- System Tray:** Shows the date (10 November 2020, Tuesday), time (02:59 AM), language (ENG), and system status (10-11-2020).

2] implement iterative deepening search

Theory :

- DFS first traverses nodes going through one adjacent of root, then next adjacent.
- The problem with this approach is, if there is a node close to root, but not in first few subtrees explored by DFS, then DFS reaches that node very late.
- Also, DFS may not find shortest path to a node (in terms of number of edges).

Code:

The screenshot shows a Google Colab notebook titled "AI_practicals 65". The left sidebar contains a "Table of contents" with sections like Practical 1, Practical 2, Practical 3, etc., and a specific section "2 : implement iterative deepening search" which is currently selected. The main workspace displays a search tree diagram with nodes S, A, D, C, B, F, E, and G. Node S is the root (green). Nodes A, D, C, B, F, E, and G are red. Edges connect S to A, A to D, D to C, D to B, D to F, and F to E. Nodes C, B, and G are isolated from the main chain. Below the tree, the code cell for "2 : implement iterative deepening search" is shown, though its content is not visible in the screenshot. The bottom status bar shows the date as 10 November 2020, Tuesday, and the time as 02:33 AM.

Output

The screenshot shows a Google Colab notebook titled "AI_practicals 65". The left sidebar contains a "Table of contents" with sections for Practical 1, 2, 3, 4, 5, 6, and 7. The main code cell contains the following Python code:

```

if depth==0:
    return
else:
    for location in start:
        listt = location
        for place in listt:
            DLS(place,depth-1)

def iddfs(source,max_height):
    for i in range(max_height+1):
        print('depth: ',str(i)+': ',end=' ')
        DLS(source,i)
        print()

iddfs('A',5)
print("-----")
def prGreen(skk): print("\033[92m {}\033[0m".format(skk))
prGreen('574 KULDEEP')

-----
574 KULDEEP

```

The output of the code is displayed below the code cell, showing the search tree and the printed message "574 KULDEEP". The bottom status bar shows the date and time as "10 November 2020 Tuesday" and "02:59 AM 10-11-2020".

Colab link :

<https://colab.research.google.com/drive/1hkVLD7CvoNHvuwu4Z-XvghvGFbCDDrXI#scrollTo=AUpTnbrAa1r>

Practical 2

Aim :1] implement A* search algorithm

Theory :

- A* Search algorithm is one of the best and popular technique used in path-finding and graph traversals.
- Why A* Search Algorithm ?

Informally speaking, A* Search algorithms, unlike other traversal techniques, it has “brains”. What it means is that it is really a smart algorithm which separates it from the other conventional algorithms.
- This fact is cleared in detail in below sections.
And it is also worth mentioning that many games and web-based maps use this algorithm to find the shortest path very efficiently (approximation).

Code

The screenshot shows a Google Colab notebook titled "AI_practicals 65 - Colaboratory". The notebook structure is as follows:

- Table of contents:**
 - Practical 1
 - 1: implement breadth first and depth first search algorithm
 - 2: implement iterative deepening search
 - Practical 2
 - 1 : implement A* search algorithm** (This section is currently selected)
 - Practical 3
 - 1: implementation of neural net on IRIS DATASET
 - Practical 4
 - 1:pima indians onset of diabetes dataset
 - Practical 5
 - 1: performing AND & OR operations in the neural network
 - Practical 6
 - 1: prediction algorithm - use of different packages on dataset of cat and non-cat images
 - Practical 7
 - 1:simple linear regression
 - Practical 8
- Practical 2**
- 1 : implement A* search algorithm**

what is A* algorithm

1. A* (A star) is a graph traversal or the path search algorithm
2. A star is a informed search algorithm (it contain the path cost, how to reach the goal node etc)
3. the main aim is finding the path to the given node having the smallest cost(minimum distance travelled, shortest time)

formula to calculate A star

$$f(n)=g(n)+h(n)$$

where n = next node in the path

$g(n)$ = it is the cost of the path from start node to n

$h(n)$ = heuristic function

```
[ ] pip install simpleai
Requirement already satisfied: simpleai in /usr/local/lib/python3.6/dist-packages (0.8.2)
```

```
[ ] from simpleai.search import SearchProblem, astar
GOAL = 'AI IS GREAT'
class HelloProblem(SearchProblem):
    pass
```

10 November 2020 Tuesday 02:34 AM ENG 10-11-2020

The screenshot shows a Google Colab notebook titled "AI_practicals 65 - Colaboratory". The code cell contains the following Python script:

```

pip install simpleai

[ ] from simpleai.search import SearchProblem, astar
GOAL = 'AI IS GREAT'
class HelloProblem(SearchProblem):
    def actions(self, state):
        if len(state) < len(GOAL):
            return list(' ABCDEFGHIJKLMNOPQRSTUVWXYZ ')
        else:
            return []
    def result(self, state, action):
        return state + action
    def is_goal(self, state):
        return state == GOAL
    def heuristic(self, state):
        wrong=sum([i if state [i]!=GOAL[i]
                  else 0
                  for i in range (len(state))])
        missing=len(GOAL) - len(state)
        return wrong + missing
problem = HelloProblem(initial_state='')
result = astar(problem)
print(result.state)
print(result.path())

```

The sidebar on the left shows a "Table of contents" with sections for Practical 1 through Practical 7. Practical 2 has a section "1: implement A* search algorithm" highlighted. The status bar at the bottom right shows the date as 10 November 2020, Tuesday, and the time as 02:35 AM.

Output

The screenshot shows the same Google Colab notebook after the code has been run. The output cell shows the results of the pip install command and the execution of the main script. The output of the script shows the package being downloaded and built successfully.

```

Collecting simpleai
  Downloading https://files.pythonhosted.org/packages/7c/c3/bb7394c293d0b844598b2ad041e3507414621cb6c44b6846dda7ebfb2eb/simpleai-0.8.2.tar.gz
Building wheels for collected packages: simpleai
  Building wheel for simpleai (setup.py) ... done
    Created wheel for simpleai: filename=simpleai-0.8.2-cp36-none-any.whl size=100979 sha256=3c619f7133e82d315bce0bd9e225dd54dfa14416e5886649542
    Stored in directory: /root/.cache/pip/wheels/eb/79/16/e69832e2d3d025b69bc6342ef5b408c15bd81c26e949155bdc
Successfully built simpleai
Installing collected packages: simpleai
Successfully installed simpleai-0.8.2

```

The sidebar on the left shows the same "Table of contents" as the previous screenshot. The status bar at the bottom right shows the date as 10 November 2020, Tuesday, and the time as 03:00 AM.

The screenshot shows a Google Colab notebook titled "AI_practicals 65". The left sidebar displays a table of contents for various practicals. The main code cell contains Python code for implementing the A* search algorithm. The output of the code is visible at the bottom of the cell.

```
from simpleai.search import SearchProblem, astar
GOAL = 'AI IS GREAT'
class HelloProblem(SearchProblem):
    def actions(self, state):
        if len(state) < len(GOAL):
            return list(' ABCDEFGHIJKLMNOPQRSTUVWXYZ ')
        else:
            return []
    def result(self, state, action):
        return state + action
    def is_goal(self, state):
        return state == GOAL
    def heuristic(self, state):
        wrong=sum([i if state[i]!=GOAL[i]
                  else 0
                  for i in range(len(state))])
        missing=len(GOAL) - len(state)
        return wrong + missing
problem = HelloProblem(initial_state='')
result = astar(problem)
print(result.state)
print(result.path())
```

AI IS GREAT
[None, '', ('A', 'A'), ('I', 'AI'), (' ', 'AI '), ('I', 'AI I'), ('S', 'AI IS'), (' ', 'AI IS '), ('G', 'AI IS G'), ('R', 'AI IS GR'), ('E',

Colab link :

<https://colab.research.google.com/drive/1hkVLD7CvoNHvuwu4Z-XvghvGFbCDDrXI#scrollTo=DhSTAc6hA6rJ>

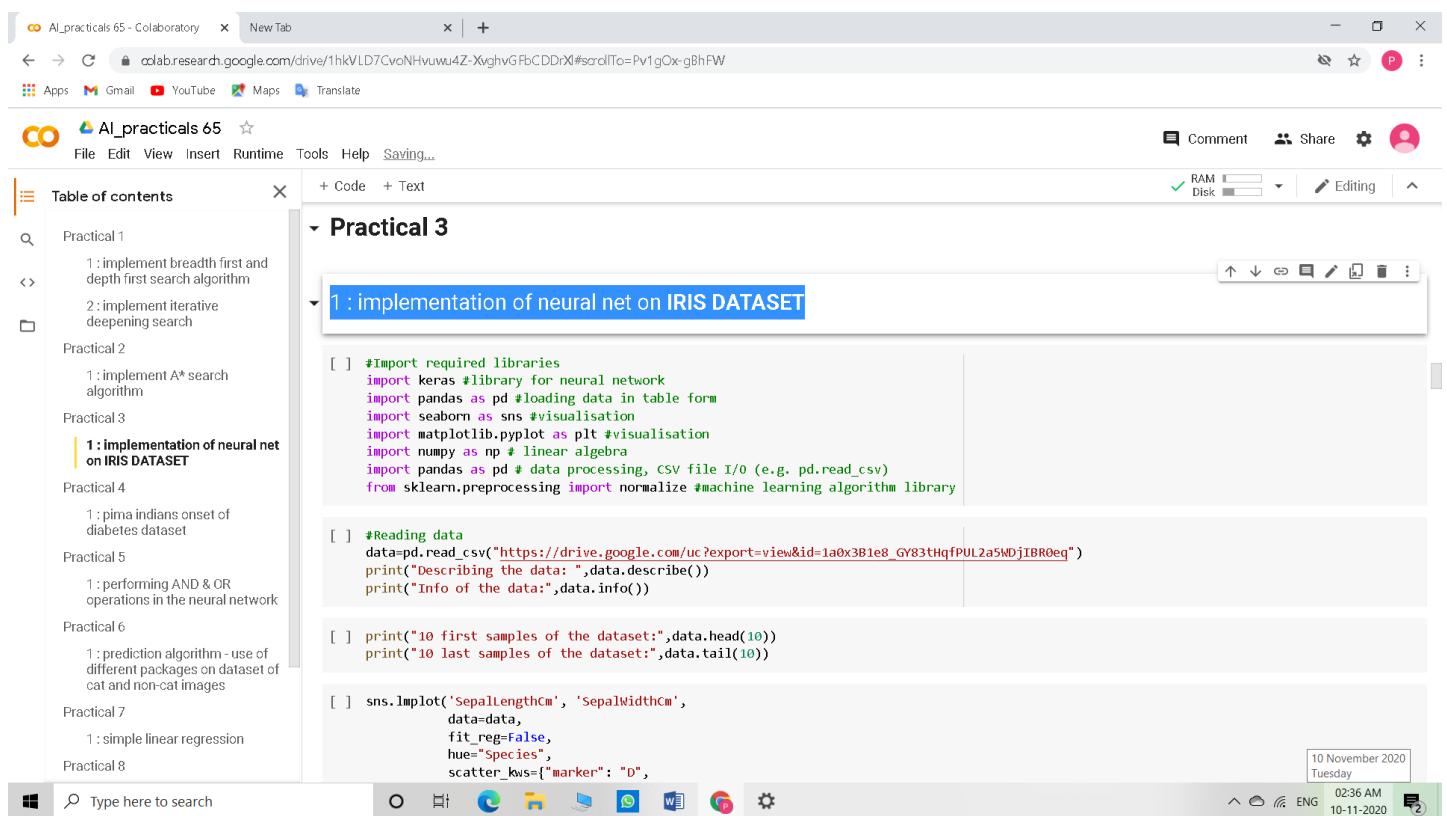
Practical 3

Aim: implementation of neural net on IRIS DATASET

Theory:

- A neuron consists of a dendrite and an axon which are responsible for collecting and sending signals.
- For our artificial neural network, the concept works similar in which a lot of neurons are connected to each layer with its own corresponding weight and biases.

Code :



The screenshot shows a Google Colab notebook titled "AI_practicals 65 - Colaboratory". The left sidebar contains a "Table of contents" with sections for Practical 1, 2, 3, 4, 5, 6, 7, and 8. Practical 3 is expanded, and "1 : implementation of neural net on IRIS DATASET" is selected. The main workspace displays the following Python code:

```

[ ] #Import required libraries
import keras #library for neural network
import pandas as pd #loading data in table form
import seaborn as sns #visualisation
import matplotlib.pyplot as plt #visualisation
import numpy as np # linear algebra
import pandas as pd # data processing, csv file I/O (e.g. pd.read_csv)
from sklearn.preprocessing import normalize #machine learning algorithm library

[ ] #Reading data
data=pd.read_csv("https://drive.google.com/uc?export=view&id=1a0x3B1e8_GY83tHqfPUL2a5W0j1BR0eq")
print("Describing the data: ",data.describe())
print("Info of the data:",data.info())

[ ] print("10 first samples of the dataset:",data.head(10))
print("10 last samples of the dataset:",data.tail(10))

[ ] sns.lmplot('SepalLengthCm', 'SepalWidthCm',
              data=data,
              fit_reg=False,
              hue="Species",
              scatter_kws={"marker": "D",
              
```

The status bar at the bottom right indicates the date as "10 November 2020 Tuesday" and the time as "02:36 AM 10-11-2020".

The screenshot shows a Google Colab notebook titled "AI_practicals 65 - Colaboratory". The code cell contains Python code for implementing various search algorithms and a neural network on the Iris dataset. The code includes imports for pandas, numpy, and sklearn, data loading, normalization, and training/test splits. A sidebar on the left shows a "Table of contents" with sections for Practical 1 through 8, including specific implementations like Breadth-First Search, Iterative Deepening Search, and A* search. The status bar at the bottom right indicates the date as 10 November 2020, Tuesday, and the time as 02:36 AM.

```

data.loc[data["Species"]=="Iris-setosa","Species"]=0
data.loc[data["Species"]=="Iris-versicolor","Species"]=1
data.loc[data["Species"]=="Iris-virginica","Species"]=2
print(data.head())

data=data.iloc[np.random.permutation(len(data))]
print(data.head())

x=data.iloc[:,1:5].values
y=data.iloc[:,5].values

print("Shape of x",x.shape)
print("Shape of y",y.shape)
print("Examples of X\n",x[1:3])
print("Examples of y\n",y[1:3])

x_normalized=normalize(x,axis=0)
print("Examples of x_normalised\n",x_normalized[1:3])

#Creating train,test and validation data
...
80% -- train data
20% -- test data
...
total_length=len(data)
train_length=int(0.8*total_length)
test_length=int(0.2*total_length)

x_train=x_normalized[:train_length]

```

Output:

The screenshot shows the output of the code from the previous Colab session. It displays the first few rows of the Iris dataset and its statistical summary. The output includes columns: sepal_length, sepal_width, petal_length, and petal_width. The summary provides counts, mean, std, min, 25%, 50%, 75%, and max values for each column. The data is described as a pandas DataFrame with 150 entries and 5 columns. The status bar at the bottom right indicates the date as 10 November 2020, Tuesday, and the time as 03:01 AM.

```

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
from sklearn.preprocessing import normalize #machine learning algorithm library

#Reading data
data=pd.read_csv("/content/drive/MyDrive/ai practical/copy of iris.csv")
print("Describing the data: ",data.describe())
print("Info of the data: ",data.info())

Describing the data:
   sepal_length  sepal_width  petal_length  petal_width
count    150.000000    150.000000    150.000000    150.000000
mean     5.843333    3.054000    3.758667    1.198667
std      0.828066    0.433594    1.764220    0.763161
min      4.300000    2.000000    1.000000    0.100000
25%     5.100000    2.800000    1.600000    0.300000
50%     5.800000    3.000000    4.350000    1.300000
75%     6.400000    3.300000    5.100000    1.800000
max     7.900000    4.400000    6.900000    2.500000

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype  
--- 
 0   sepal_length  150 non-null   float64
 1   sepal_width   150 non-null   float64
 2   petal_length  150 non-null   float64
 3   petal_width   150 non-null   float64
 4   species       150 non-null   object 
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
Info of the data: None

```

```

[19] 4 species      150 non-null   object
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
Info of the data: None

[20] print("10 first samples of the dataset:",data.head(10))
print("10 last samples of the dataset:",data.tail(10))

10 first samples of the dataset:    sepal_length  sepal_width  petal_length  petal_width  species
0           5.1          3.5          1.4         0.2   setosa
1           4.9          3.0          1.4         0.2   setosa
2           4.7          3.2          1.3         0.2   setosa
3           4.6          3.1          1.5         0.2   setosa
4           5.0          3.6          1.4         0.2   setosa
5           5.4          3.9          1.7         0.4   setosa
6           4.6          3.4          1.4         0.3   setosa
7           5.0          3.4          1.5         0.2   setosa
8           4.4          2.9          1.4         0.2   setosa
9           4.9          3.1          1.5         0.1   setosa

10 last samples of the dataset:    sepal_length  sepal_width  petal_length  petal_width  species
140          6.7          3.1          5.6         2.4  virginica
141          6.9          3.1          5.1         2.3  virginica
142          5.8          2.7          5.1         1.9  virginica
143          6.8          3.2          5.9         2.3  virginica
144          6.7          3.3          5.7         2.5  virginica
145          6.7          3.0          5.2         2.3  virginica
146          6.3          2.5          5.0         1.9  virginica
147          6.5          3.0          5.2         2.0  virginica
148          6.2          3.4          5.4         2.3  virginica
149          5.9          3.0          5.1         1.8  virginica

```

Colab link :

<https://colab.research.google.com/drive/1hkVLD7CvoNHvuwu4Z-XvghvGFbCDDrXI#scrollTo=Pv1gOx-gBhFW>

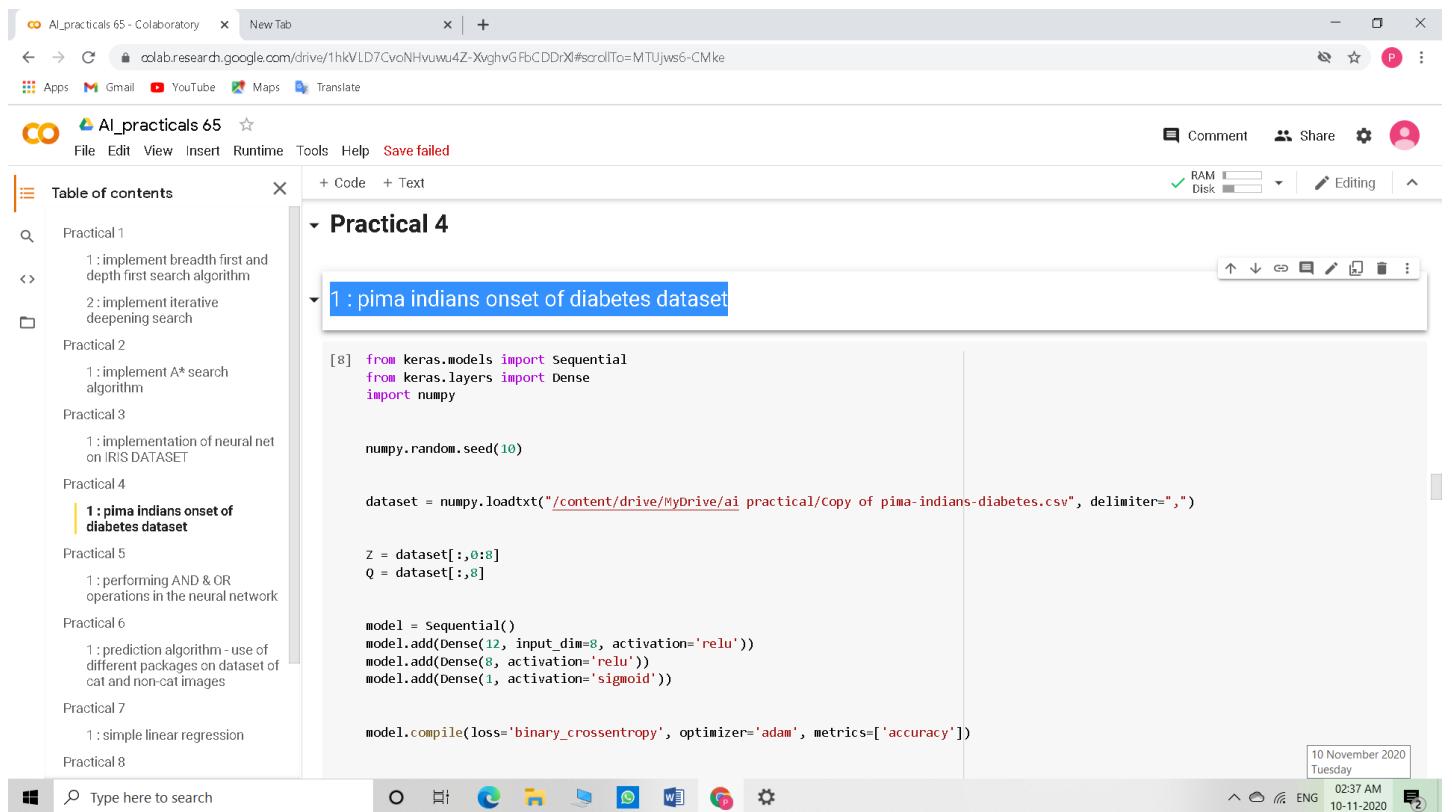
Practical 4

Aim: pima indians onset of diabetes dataset

Theory :

- This dataset is originally from the National Institute of Diabetes and Digestive and Kidney Diseases.
- The objective of the dataset is to diagnostically predict whether or not a patient has diabetes, based on certain diagnostic measurements included in the dataset.
- Several constraints were placed on the selection of these instances from a larger database. In particular, all patients here are females at least 21 years old of Pima Indian heritage.

Code:



The screenshot shows a Google Colab notebook titled "AI_practicals 65 - Colaboratory". The left sidebar contains a "Table of contents" with sections for Practical 1, 2, 3, 4, 5, 6, 7, and 8. Practical 4 is expanded, and "1 : pima indians onset of diabetes dataset" is selected. The main workspace shows the following Python code:

```

[8] from keras.models import Sequential
from keras.layers import Dense
import numpy

numpy.random.seed(10)

dataset = numpy.loadtxt("/content/drive/MyDrive/ai practical/Copy of pima-indians-diabetes.csv", delimiter=",")

Z = dataset[:,0:8]
Q = dataset[:,8]

model = Sequential()
model.add(Dense(12, input_dim=8, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

```

The status bar at the bottom right indicates the date as "10 November 2020 Tuesday" and the time as "02:37 AM 10-11-2020".

Output:

The screenshot shows a Google Colab notebook titled "AI_practicals 65 - Colaboratory". The code cell contains Python code related to model metrics and training epochs. The output pane displays the results of these executions, including accuracy and loss values for 150 epochs of a neural network. The sidebar on the left shows a "Table of contents" with sections for Practical 1 through Practical 7, with Practical 4 specifically mentioning the "pima indians onset of diabetes dataset". The bottom of the screen shows the Windows taskbar with various icons and system status.

```

print( f"\n{model.metrics_names[1]} : {scores[1]*100} ")
[8] print("\n-----")
def prGreen(skk): print("\033[92m {} \033[00m".format(skk))
prGreen('574 kuldeep')

Epoch 125/150
77/77 [=====] - 0s 1ms/step - loss: 0.4778 - accuracy: 0.7759
Epoch 126/150
77/77 [=====] - 0s 1ms/step - loss: 0.4810 - accuracy: 0.7792
Epoch 127/150
77/77 [=====] - 0s 1ms/step - loss: 0.4842 - accuracy: 0.7601
Epoch 128/150
77/77 [=====] - 0s 1ms/step - loss: 0.5083 - accuracy: 0.7358
Epoch 129/150
77/77 [=====] - 0s 1ms/step - loss: 0.4770 - accuracy: 0.7769
Epoch 130/150
77/77 [=====] - 0s 1ms/step - loss: 0.4623 - accuracy: 0.7656
Epoch 131/150
77/77 [=====] - 0s 1ms/step - loss: 0.5118 - accuracy: 0.7401
Epoch 132/150
77/77 [=====] - 0s 1ms/step - loss: 0.5363 - accuracy: 0.7431
Epoch 133/150
77/77 [=====] - 0s 1ms/step - loss: 0.5125 - accuracy: 0.7532
Epoch 134/150
77/77 [=====] - 0s 1ms/step - loss: 0.4998 - accuracy: 0.7360
Epoch 135/150
77/77 [=====] - 0s 1ms/step - loss: 0.5316 - accuracy: 0.7244
Epoch 136/150
77/77 [=====] - 0s 1ms/step - loss: 0.4952 - accuracy: 0.7571
Epoch 137/150
77/77 [=====] - 0s 1ms/step - loss: 0.5095 - accuracy: 0.7583
Epoch 138/150
77/77 [=====] - 0s 1ms/step - loss: 0.5088 - accuracy: 0.7647
Epoch 139/150

```

Colab link :

<https://colab.research.google.com/drive/1hkVLD7CvoNHvuwu4Z-XvghvGFbCDDrXI#scrollTo=MTUjws6-CMke>

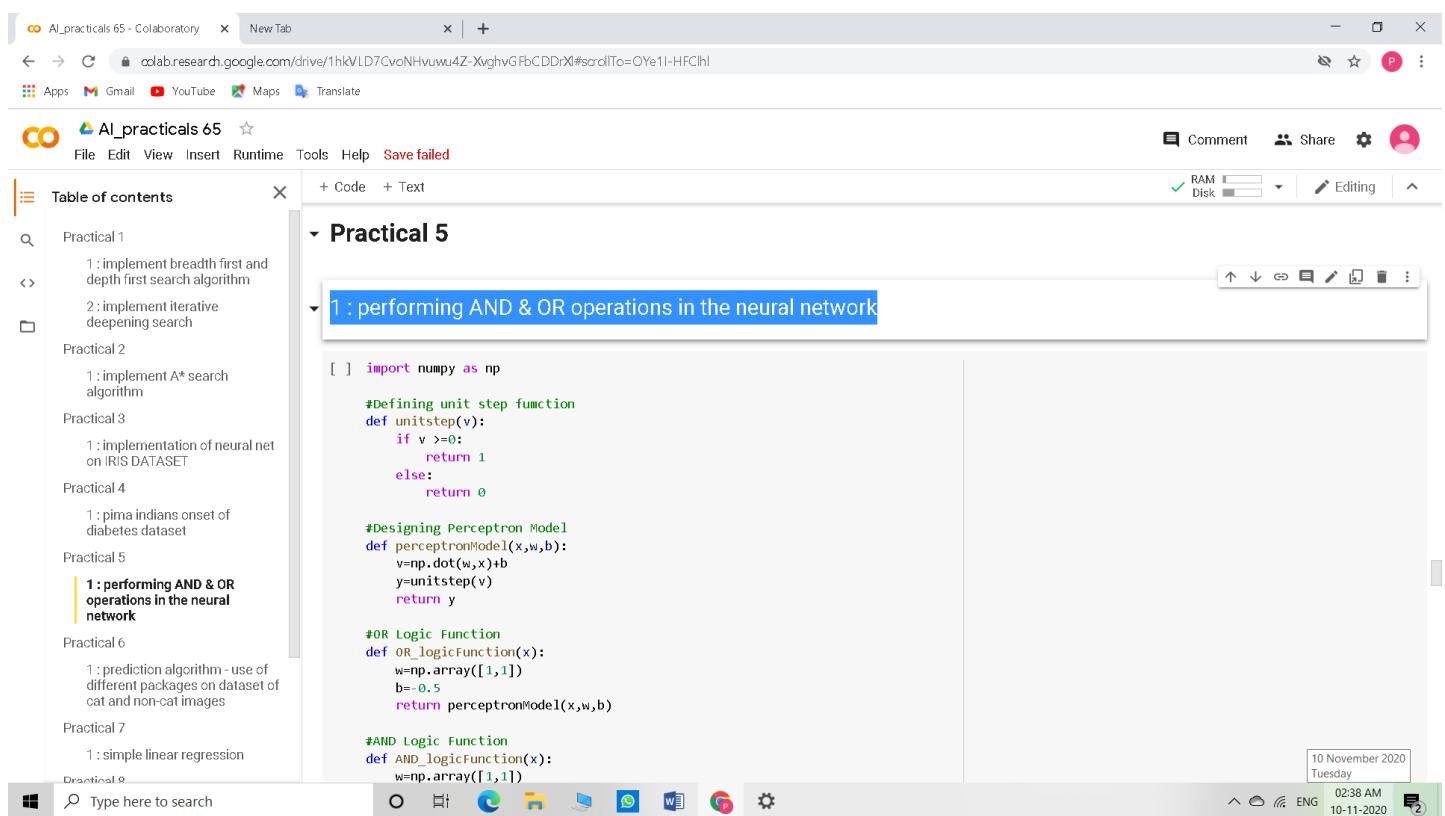
Practical 5

Aim : performing AND & OR operations in the neural network

Theory :

- A neural network is a series of algorithms that endeavors to recognize underlying relationships in a set of data through a process that mimics the way the human brain operates.
- In this sense, neural networks refer to systems of neurons, either organic or artificial in nature. Neural networks can adapt to changing input;
- so the network generates the best possible result without needing to redesign the output criteria. The concept of neural networks

Code:



The screenshot shows a Google Colab notebook titled "AI_practicals 65". The left sidebar contains a "Table of contents" with sections for Practical 1 through Practical 8. Practical 5 is expanded, showing its sub-section "1 : performing AND & OR operations in the neural network". The main workspace displays the following Python code:

```

[ ] import numpy as np

#Defining unit step function
def unitstep(v):
    if v >= 0:
        return 1
    else:
        return 0

#Designing Perceptron Model
def perceptronModel(x,w,b):
    v=np.dot(w,x)+b
    y=unitstep(v)
    return y

#OR Logic Function
def OR_LogicFunction(x):
    w=np.array([1,1])
    b=-0.5
    return perceptronModel(x,w,b)

#AND Logic Function
def AND_LogicFunction(x):
    w=np.array([1,1])
    
```

The status bar at the bottom right indicates the date as "10 November 2020 Tuesday" and the time as "02:38 AM 10-11-2020".

The screenshot shows a Google Colab notebook titled "AI_practicals 65 - Colaboratory". The left sidebar contains a "Table of contents" with practicals 1 through 9. Practical 1 has two items: "implement breadth first and depth first search algorithm" and "implement iterative deepening search". Practical 2 has one item: "implement A* search algorithm". Practical 3 has one item: "implementation of neural net on IRIS DATASET". Practical 4 has one item: "pima indians onset of diabetes dataset". Practical 5 has one item: "performing AND & OR operations in the neural network". Practical 6 has one item: "prediction algorithm - use of different packages on dataset of cat and non-cat images". Practical 7 has one item: "simple linear regression". Practical 8 is collapsed. Practical 9 has one item: "practicals".

The main area displays Python code:

```
[ ] #AND Logic Function
def AND_logicFunction(x):
    w=np.array([1,1])
    b=-1.5
    return perceptronModel(x,w,b)

#NOT Logic Function
def NOT_logicFunction(x):
    return perceptronModel(x,w=-1,b=0.5)

#XOR Logic Function
def XOR_logicFunction(x):
    g1=AND_logicFunction(x)
    g2=NOT_logicFunction(g1)
    g3=OR_logicFunction(x)
    new_x=np.array([g2,g3])
    output=AND_logicFunction(new_x)
    return output

#Testing the perceptron Model
test1=np.array([0,0])
test2=np.array([0,1])
test3=np.array([1,0])
test4=np.array([1,1])

print("OR({},{})={}".format(0,0, OR_logicFunction(test1)))
print("OR({},{})={}".format(0,1, OR_logicFunction(test2)))
print("OR({},{})={}".format(1,0, OR_logicFunction(test3)))
print("OR({},{})={}".format(1,1, OR_logicFunction(test4)))

print("====")
```

The status bar at the bottom right shows the date (10 November 2020, Tuesday), time (02:39 AM), language (ENG), and session ID (10-11-2020).

Output :

The screenshot shows a Google Colab notebook titled "AI_practicals 65 - Colaboratory". The left sidebar contains a "Table of contents" with sections for Practical 1 through Practical 7, each with sub-sections like "implement breadth first and depth first search algorithm" or "performing AND & OR operations in the neural network". The main code editor window displays Python code for logic operations:

```

print("NOT(0)={}".format(NOT_logicFunction(0)))
print("NOT(1)={}".format(NOT_logicFunction(1)))

print("=====")

print("XOR({},{}]={}.format(0,0, XOR_logicFunction(test1)))
print("XOR({},{}]={}.format(0,1, XOR_logicFunction(test2)))
print("XOR({},{}]={}.format(1,0, XOR_logicFunction(test3)))
print("XOR({},{}]={}.format(1,1, XOR_logicFunction(test4)))

=====

OR(0,0)=0
OR(0,1)=1
OR(1,0)=1
OR(1,1)=1
=====

AND(0,0)=0
AND(0,1)=0
AND(1,0)=0
AND(1,1)=1
=====

NOT(0)=1
NOT(1)=0
=====

XOR(0,0)=0
XOR(0,1)=1
XOR(1,0)=1
XOR(1,1)=0

```

The status bar at the bottom right indicates the date as "10 November 2020 Tuesday" and the time as "03:02 AM" on "10-11-2020".

Colab link :

<https://colab.research.google.com/drive/1hkVLD7CvoNHvuwu4Z-XvghvGFbCDDrXI#scrollTo=OYe1I-HFClhl>

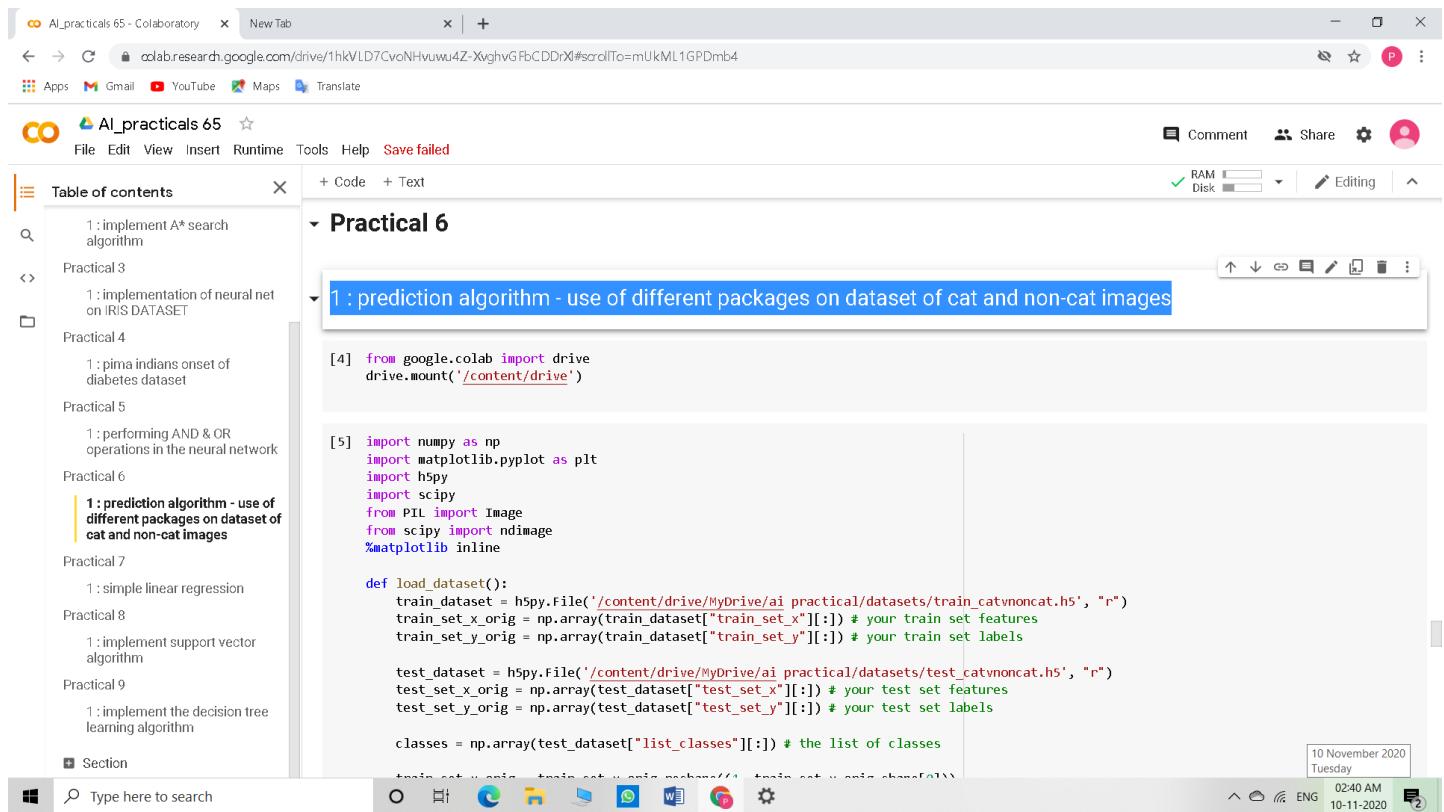
Practical 6

Aim : prediction algorithm - use of different packages on dataset of cat and non-cat images

Theory :

- Predictive analytics algorithms try to achieve the lowest error possible by either using “boosting”
- (a technique which adjusts the weight of an observation based on the last classification) or “bagging” (which creates subsets of data from training samples, chosen randomly with replacement).
- Random Forest uses bagging.

Code:



The screenshot shows a Google Colab notebook titled "AI_practicals 65 - Collaboratory". The left sidebar contains a "Table of contents" with sections for Practical 3, 4, 5, 6, and 7. Practical 6 is currently selected and expanded, showing a sub-section titled "1 : prediction algorithm - use of different packages on dataset of cat and non-cat images". The main workspace displays the following Python code:

```

[4]: from google.colab import drive
drive.mount('/content/drive')

[5]: import numpy as np
import matplotlib.pyplot as plt
import h5py
import scipy
from PIL import Image
from scipy import ndimage
%matplotlib inline

def load_dataset():
    train_dataset = h5py.File('/content/drive/MyDrive/ai practical/datasets/train_catvnoncat.h5', "r")
    train_set_x_orig = np.array(train_dataset["train_set_x"][:]) # your train set features
    train_set_y_orig = np.array(train_dataset["train_set_y"][:]) # your train set labels

    test_dataset = h5py.File('/content/drive/MyDrive/ai practical/datasets/test_catvnoncat.h5', "r")
    test_set_x_orig = np.array(test_dataset["test_set_x"][:]) # your test set features
    test_set_y_orig = np.array(test_dataset["test_set_y"][:]) # your test set labels

    classes = np.array(test_dataset["list_classes"][:]) # the list of classes
    train_set_y = train_set_y_orig.reshape((1, train_set_y_orig.shape[0]))
    test_set_y = test_set_y_orig.reshape((1, test_set_y_orig.shape[0]))

```

AI_practicals 65 - Colaboratory x New Tab x | +

colab.research.google.com/drive/1hkVLID7CvoNhuwu4Z-XvhvGFBcDDrX#scrollTo=mUkMl1GPDmb4

Apps Gmail YouTube Maps Translate

AI_practicals 65

File Edit View Insert Runtime Tools Help Save failed

Table of contents + Code + Text

RAM Disk Editing

```
[5] import numpy as np
import matplotlib.pyplot as plt
import h5py
import scipy
from PIL import Image
from scipy import ndimage
%matplotlib inline

def load_dataset():
    train_dataset = h5py.File('/content/drive/MyDrive/ai practical/datasets/train_catvnoncat.h5', "r")
    train_set_x_orig = np.array(train_dataset["train_set_x"][:]) # your train set features
    train_set_y_orig = np.array(train_dataset["train_set_y"][:]) # your train set labels

    test_dataset = h5py.File('/content/drive/MyDrive/ai practical/datasets/test_catvnoncat.h5', "r")
    test_set_x_orig = np.array(test_dataset["test_set_x"][:]) # your test set features
    test_set_y_orig = np.array(test_dataset["test_set_y"][:]) # your test set labels

    classes = np.array(test_dataset["list_classes"][:]) # the list of classes

    train_set_x_orig = train_set_x_orig.reshape((1, train_set_y_orig.shape[0]))
    test_set_y_orig = test_set_y_orig.reshape((1, test_set_y_orig.shape[0]))

    return train_set_x_orig, train_set_y_orig, test_set_x_orig, test_set_y_orig, classes

# Loading the data (non-cat)
train_set_x_orig, train_set_y, test_set_x_orig, test_set_y, classes = load_dataset()
index = 35
plt.imshow(train_set_x_orig[index])
print ("y = " + str(train_set_y[:, index]) + ", it's a " + classes[np.squeeze(train_set_y[:, index])].decode("utf-8") + " picture")
```

10 November 2020 Tuesday 02:41 AM ENG 10-11-2020

Type here to search

Section

File Edit View Insert Runtime Tools Help All changes saved

Comment Share

RAM Disk Editing

Output :

AI_practicals 65 - Colaboratory x Copy of AI Practical-575.ipynb - x xQDnQM6BxKzA4f+D75Fr4ozBj x | ai practical - Google Drive x | +

colab.research.google.com/drive/1hkVLID7CvoNhuwu4Z-XvhvGFBcDDrX#scrollTo=LWbO23AipRw

Apps Gmail YouTube Maps Translate

AI_practicals 65

File Edit View Insert Runtime Tools Help All changes saved

Table of contents + Code + Text

RAM Disk Editing

```
classes = np.array(test_dataset["list_classes"][:]) # the list of classes

train_set_y_orig = train_set_y_orig.reshape((1, train_set_y_orig.shape[0]))
test_set_y_orig = test_set_y_orig.reshape((1, test_set_y_orig.shape[0]))

return train_set_x_orig, train_set_y_orig, test_set_x_orig, test_set_y_orig, classes

# Loading the data (non-cat)
train_set_x_orig, train_set_y, test_set_x_orig, test_set_y, classes = load_dataset()
index = 35
plt.imshow(train_set_x_orig[index])
print ("y = " + str(train_set_y[:, index]) + ", it's a " + classes[np.squeeze(train_set_y[:, index])].decode("utf-8") + " picture")
```

y = [0], it's a non-cat picture

download.png

10 November 2020 Tuesday 03:02 AM ENG 10-11-2020

Type here to search

Name: Kuldeep Shushil Patel

Roll no: 574

Colab link :

<https://colab.research.google.com/drive/1hkVLD7CvoNHvuwu4Z-XvghvGFbCDDrXI#scrollTo=mUkML1GPDmb4>

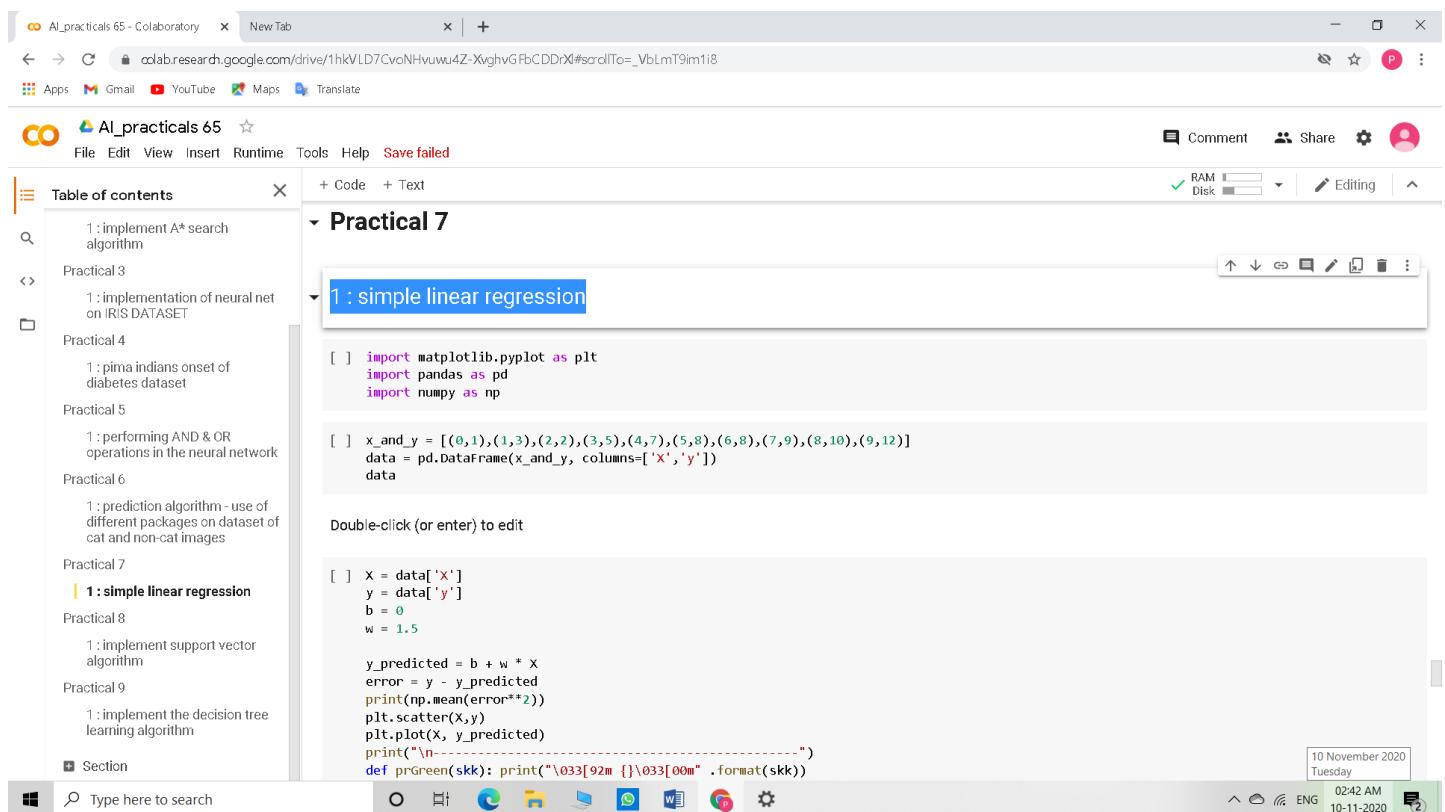
Practical 7

Aim : simple linear regression

Theory :

- **Simple linear regression** is a statistical method that allows us to summarize and study relationships between two continuous (quantitative) variables:
- One variable, denoted x , is regarded as the **predictor, explanatory, or independent variable**.
- The other variable, denoted y , is regarded as the **response, outcome, or dependent variable**.
- Because the other terms are used less frequently today, we'll use the "**predictor**" and "**response**" terms to refer to the variables encountered in this course.

Code :



The screenshot shows a Google Colab notebook titled "AI_practicals 65 - Colaboratory". The left sidebar contains a table of contents with sections like "1 : implement A* search algorithm", "Practical 3", "Practical 4", "Practical 5", "Practical 6", "Practical 7", and "Practical 8". The main area displays the code for "1 : simple linear regression".

```

import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

x_and_y = [(0,1),(1,3),(2,2),(3,5),(4,7),(5,8),(6,8),(7,9),(8,10),(9,12)]
data = pd.DataFrame(x_and_y, columns=['X', 'Y'])
data

```

Below the code, there's a section for editing:

```

x = data['X']
y = data['Y']
b = 0
w = 1.5

y_predicted = b + w * x
error = y - y_predicted
print(np.mean(error**2))
plt.scatter(x,y)
plt.plot(x, y_predicted)
print("-----")
def prGreen(skk): print("\033[92m {}\033[0m".format(skk))

```

The status bar at the bottom shows the date (10 November 2020), time (02:42 AM), and system information (ENG, 10-11-2020).

Output :

The screenshot shows a Google Colab notebook titled "AI_practicals 65 - Colaboratory". The left sidebar contains a "Table of contents" with sections for Practical 3 through Practical 9, each with a corresponding task description. The main workspace displays a code cell at index [34] that imports matplotlib.pyplot, pandas, and numpy, and then defines a data frame "x_and_y" from a list of tuples. Below the code, the resulting data frame is shown as a table:

x	y
0	1
1	3
2	2
3	5
4	7
5	8
6	8
7	9
8	10
9	12

Colab link :

<https://colab.research.google.com/drive/1hkVLD7CvoNHvuwu4Z-XvghvGFbCDDrXI#scrollTo= VbLmT9im1i8>

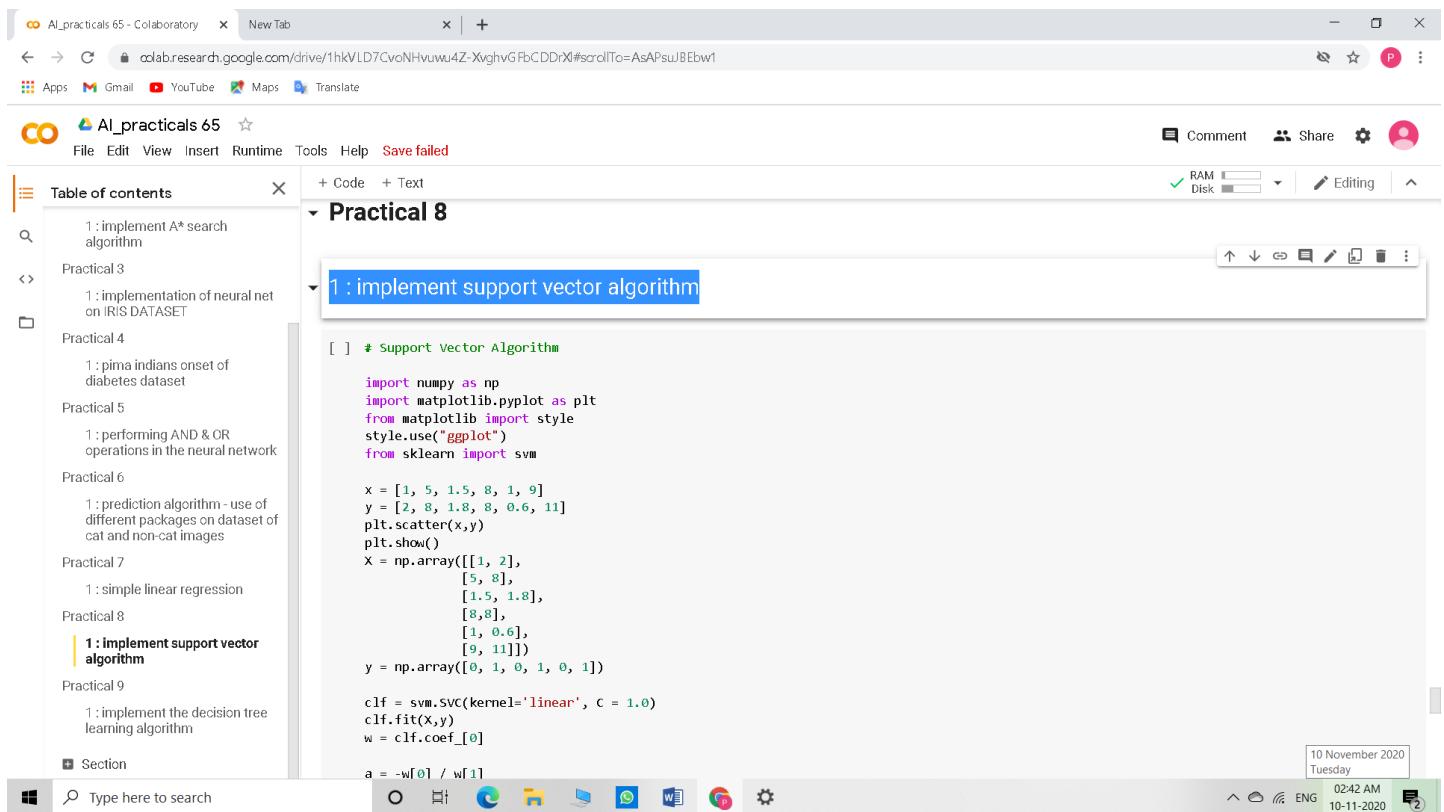
Practical 8

Aim : implement support vector algorithm

Theory :

- Support Vector Machine” (SVM) is a supervised machine learning algorithm
- which can be used for both classification or regression challenges.
- it is mostly used in classification problems.
- Support Vectors are simply the co-ordinates of individual observation

Code:



The screenshot shows a Google Colab notebook titled "AI_practicals 65 - Colaboratory". The left sidebar contains a "Table of contents" with sections for Practical 8 and Practical 9. Under Practical 8, the section "1 : implement support vector algorithm" is selected and highlighted in blue. The main workspace displays the following Python code:

```

# Support Vector Algorithm

import numpy as np
import matplotlib.pyplot as plt
from matplotlib import style
style.use("ggplot")
from sklearn import svm

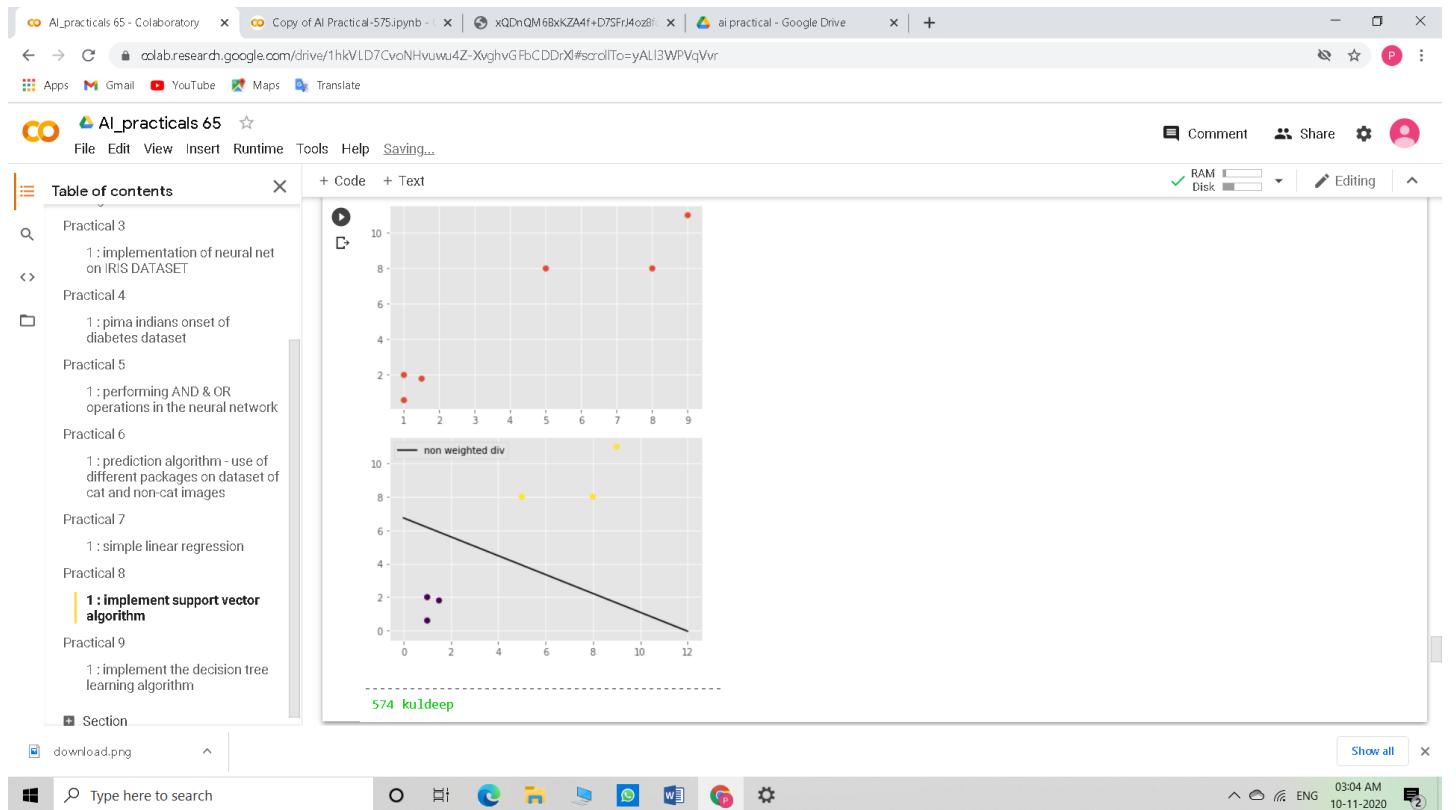
x = [1, 5, 1.5, 8, 1, 9]
y = [2, 8, 1.8, 8, 0.6, 11]
plt.scatter(x,y)
plt.show()
X = np.array([[1, 2],
              [5, 8],
              [1.5, 1.8],
              [8,8],
              [1, 0.6],
              [9, 11]])
y = np.array([0, 1, 0, 1, 0, 1])

clf = svm.SVC(kernel='linear', C = 1.0)
clf.fit(X,y)
w = clf.coef_[0]
a = -w[0] / w[1]

```

The code imports necessary libraries (numpy, matplotlib, sklearn), defines data points (x and y), plots them, and then uses the SVM classifier from scikit-learn to fit the data. The output of the code is not visible in the screenshot.

Output :



Colab link :

<https://colab.research.google.com/drive/1hkVLD7CvoNHvuwu4Z-XvghvGFbCDDrXl#scrollTo=AsAPsuJBEbw1>

Practical 9

Aim : implement the decision tree learning algorithm

Theory :

- **Decision Trees** are a type of Supervised **Machine Learning**.
- what the input is and what the corresponding output is in the training data) where the data is continuously split according to a certain parameter.
- The **tree** can be explained by two entities, namely **decision** nodes and leaves

Code :

The screenshot shows a Google Colab notebook titled "AI_practicals 65 - Colaboratory". The left sidebar contains a "Table of contents" with sections for Practical 3, 4, 5, 6, 7, 8, and 9, with "1 : implement the decision tree learning algorithm" under Practical 9 highlighted. The main workspace shows the following Python code:

```
[ ] # decision tree for Iris Dataset
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.tree import export_graphviz
from sklearn.externals.six import StringIO
from IPython.display import Image
from pydot import graph_from_dot_data
import pandas as pd
import numpy as np

print("\n-----")
def prGreen(skk): print("\033[92m {}\033[00m".format(skk))
prGreen('574 kuldeep')
print("\n-----")

iris = load_iris()
X = pd.DataFrame(iris.data, columns=iris.feature_names)
print("Following are the columns of iris dataset\n")
print(X)
```

The status bar at the bottom right indicates the date as "10 November 2020 Tuesday" and the time as "02:43 AM 10-11-2020".

Output :

The screenshot shows a Jupyter Notebook interface in Google Colab. The notebook has a table of contents on the left and a main workspace on the right.

Main Workspace Content:

- Code cell output:

```
(graph,) = graph_from_iris_data(iris_data, getvalue())
[33] Image(graph.create_png())
-----
574 kuldeep
```
- Text output:

Following are the columns of iris dataset

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2
..
145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

[150 rows x 4 columns]
- Code cell output (highlighted):

```
petal length (cm) <= 2.6
gini = 0.443
samples = 112
value = [[75, 37]
[78, 34]
[71, 41]]
```

Bottom Status Bar:

10 November 2020 Tuesday 03:04 AM ENG 10-11-2020

```

graph TD
    Root["petal length (cm) <= 2.6  
gini = 0.443  
samples = 112  
value = [[75, 37], [78, 34], [71, 41]]"] -- True --> L1_L["gini = 0.0  
samples = 37  
value = [[0, 37], [37, 0], [37, 0]]"]
    Root -- False --> L1_R["petal width (cm) <= 1.65  
gini = 0.33  
samples = 75  
value = [[75, 0], [41, 34], [34, 41]]"]
    L1_L --> L2_L["petal length (cm) <= 4.95  
gini = 0.129  
samples = 37  
value = [[37, 0], [4, 33], [33, 4]]"]
    L1_R --> L2_R["petal length (cm) <= 4.85  
gini = 0.034  
samples = 38  
value = [[38, 0], [37, 1], [1, 37]]"]
    L2_L --> Leaf_L["..."]
    L2_R --> Leaf_R["..."]
  
```

The screenshot shows a Jupyter Notebook interface with a decision tree visualization. The tree is built on the Iris dataset. The root node splits based on petal length (cm) ≤ 2.6. The left branch leads to a node where gini = 0.0, samples = 37, and value = [[0, 37], [37, 0], [37, 0]]. The right branch leads to a node where petal width (cm) ≤ 1.65, gini = 0.33, samples = 75, and value = [[75, 0], [41, 34], [34, 41]]. This right node further splits into two leaf nodes based on petal length (cm). The first leaf node (left) has gini = 0.129, samples = 37, and value = [[37, 0], [4, 33], [33, 4]]. The second leaf node (right) has gini = 0.034, samples = 38, and value = [[38, 0], [37, 1], [1, 37]]. The bottom of the screen shows the Windows taskbar with the search bar containing "download.png".

Colab link :

<https://colab.research.google.com/drive/1hkVLD7CvoNHvuwu4Z-XvghvGFbCDDrXI#scrollTo=IGsJNEE7Etlo>