Pract 1

BFS

Code

# BFS

graph = {

  '1' : ['2','3'],

  '2' : ['4', '5'],

  '3' : ['6'],

  '4' : [],

  '5' : ['6'],

  '6' : []

}

visited = []

queue = []

def bfs(visited, graph, node):

  visited.append(node)

  queue.append(node)

  while queue:

    s = queue.pop(0)

    print (s, end = " ")

    for neighbour in graph[s]:

      if neighbour not in visited:

        visited.append(neighbour)

        queue.append(neighbour)

bfs(visited, graph, '1')

DFS:

Code:

# DFS

graph = {

    '1' : ['2','3'],

    '2' : ['4', '5'],

    '3' : ['6'],

    '4' : [],

    '5' : ['6'],

    '6' : []

}

visited = set()

def dfs(visited, graph, node):

    if node not in visited:

        print (node)

        visited.add(node)

        for neighbour in graph[node]:

            dfs(visited, graph, neighbour)

dfs(visited, graph, '1')

Practical 2

Code:

class Node:

  """A simple node """

  def \_\_declare\_instance\_variables(this) -> None:

    this.parent: Node = None

    this.root: Node = None

    this.\_\_children: list = []

  def \_\_init\_\_(this, child = None, children: list = None, value: float = None, tag: str = None):

    """child: Node, children: List[Node]"""

    this.\_\_declare\_instance\_variables()

    this.tag = tag

    this.value = value

    if (child != None):

      this.add(child)

    if (children != None):

      this.add\_children(children)

  def get\_neighbors(this) -> list:

    """Returns the neighbor nodes"""

    if this.parent == None:

      return [this]

    children = this.parent.get\_children()

    if children == None:

      return []

    return children

  def get\_first(this):

    """Returns the first children of this node"""

    if (this.is\_empty): return None

    return this.\_\_children[0]

  def is\_root(this) -> bool:

    return this.parent == None

  def is\_leaf(this) -> bool:

    if (this.\_\_children == None): return True

    return this.is\_empty()

  def is\_inner(this) -> bool:

    return not (this.is\_leaf() or this.is\_root())

  def get\_children(this) -> list:

    return this.\_\_children

  def get\_root(this):

    """Returns -> Node"""

    if (this.is\_root()):

      return this

    else:

      return this.parent.root

  def get\_height(this) -> int:

    if (this.is\_empty()):

      return 0

    maxHeight: int = 0

    children: list = this.get\_children()

    for element in children:

      height: int = element.get\_height()

      if (height > maxHeight):

        maxHeight = height

    return maxHeight + 1

  def get\_depth(this) -> int:

    if (this.is\_root()):

      return 0

    return this.parent.get\_depth() + 1

  def is\_empty(this) -> bool:

    return len(this.\_\_children) == 0

  def is\_not\_empty(this) -> bool:

    return not this.is\_empty()

  def add(this, child) -> None:

    """child: Node"""

    assert child != None

    if (this.\_\_children == None):

      this.\_\_children = []

    child.parent = this

    child.root = this.get\_root()

    this.\_\_children.append(child)

  def add\_children(this, children: list) -> None:

    assert children != None

    if (len(children) == 0):

      return

    if (this.\_\_children == None):

      this.\_\_children = []

    for element in children:

      element.parent = this

      element.root = this.get\_root()

      this.\_\_children.append(element)

  def \_\_len\_\_(this) -> int:

    if (len(this.\_\_children) != 0 and this.\_\_children != None):

      maxLength: int = 1

      for child in this.\_\_children:

          maxLength += len(child)

      return maxLength

    else:

      return 1

  def \_\_str\_\_(this) -> str:

    return f"Node({this.value})"

////////////////////////////new block ///////////////

def node\_to\_string(node: Node, islast=False):

    pretab = '' if node.get\_depth() == 0 else ' ' \* (node.get\_depth())

    prefix = f'{pretab}:{node.get\_depth()} ───'

    value = node.value

    depthTab: str = ' ' \* (node.get\_depth() + 1)

    children\_str = ''

    for child in node.get\_children():

      ischildlast = node.get\_children()[-1] == child

      children\_str += f'{depthTab}{node\_to\_string(child, ischildlast)}'

    return (

        f'{prefix} {node.tag} = {value}\n'

        f'{children\_str}'

    )

////////// new line /////

def evaluate(node: Node):

  """returns the value of the node"""

  if(isinstance(node.value, float) or isinstance(node.value, int)):

    assert node.value != None, "Node must have a value"

    return node.value

  elif(isinstance(node.value, str)):

    raise NotImplementedError

//// new line ////

def evaluate(node: Node):

  """returns the value of the node"""

  if(isinstance(node.value, float) or isinstance(node.value, int)):

    assert node.value != None, "Node must have a value"

    return node.value

  elif(isinstance(node.value, str)):

    raise NotImplementedError

//// new line////

from math import inf

def hill\_climbing(start\_node) -> Node:

  """

  Pseudo-code for the algorithm

  ```

  algorithm hill Climbing is

      currentNode := startNode

      loop do

          L := NEIGHBORS(currentNode)

          nextEval := −INF

          nextNode := NULL

          for all x in L do

              if EVAL(x) > nextEval then

                  nextNode := x

                  nextEval := EVAL(x)

          if nextEval ≤ EVAL(currentNode) then

              // Return current node since no better neighbors exist

              return currentNode

          currentNode := nextNode

  ```

  """

  current\_node = start\_node

  best\_value = -inf

  best\_node = None

  while True:

    current\_value = evaluate(current\_node)

    if current\_value > best\_value:

      best\_node = current\_node

      best\_value = current\_value

    else:

      # this node has a value smaller than the best node.

      # stopping search with local maxima

      return best\_node

    childrens = current\_node.get\_children()

    for child in childrens:

      child\_value = evaluate(child)

      if child\_value > best\_value:

        best\_value = child\_value

        best\_node = child

      else:

        return best\_node

      # every neighbour of this child is traversed.

      # Setting current\_node as the last child traversed

      current\_node = child

/// new line ///

# implemented the above tree of nodes

tree1: Node = Node(

    value=10, tag='a',

    children=[

        Node(

            value=10, tag='b',

            children=[

                Node(value=4, tag='d'),

                Node(value=2, tag='c',

                     child=Node(value=0, tag='h')

                ),

            ]

        ),

        Node(value=8, tag='j',

            child=Node(value=0, tag='k')

        ),

        Node(value=7, tag='f',

            children=[

                Node(value=5, tag='e',

                    child=Node(

                        value=6, tag='i',

                        child=Node(value=0, tag='k')

                    )

                ),

                Node(value=3, tag='g')

            ]

        )

    ]

)

///new line/////

# implemented an another tree of nodes

tree2: Node = Node(

    value=2, tag='a',

    children=[

        Node(

            value=4, tag='b',

            children=[

                Node(value=5, tag='d'),

                Node(value=6, tag='c',

                     child=Node(value=8, tag='h')

                ),

            ]

        ),

        Node(value=9, tag='j',

            child=Node(value=0, tag='k')

        ),

        Node(value=7, tag='f',

            children=[

                Node(value=12, tag='e',

                    child=Node(

                        value=6, tag='i',

                        child=Node(value=0, tag='k')

                    )

                ),

                Node(value=3, tag='g')

            ]

        )

    ]

)

//// new line ///

print('Tree - 1: representation')

print('pattern -> :<depth> ─── <value>', end='\n\n')

print(node\_to\_string(tree1))

///new ////

print('Tree - 2: representation')

print('pattern -> :<depth> ─── <value>', end='\n\n')

print(node\_to\_string(tree2))

///new line ////

print('For Tree - 1')

best\_solution = hill\_climbing(tree1)

print(f"Best solution is {best\_solution.value} with tag {best\_solution.tag}")

/// new line ///

print('For Tree - 2')

best\_solution = hill\_climbing(tree2)

print(f"Best solution is {best\_solution.value} with tag {best\_solution.tag}")

Practical 3A

Code:

"""

Naive backtracking search without any heuristics or inference.

"""

VARIABLES = ["A", "B", "C", "D", "E", "F", "G"]

CONSTRAINTS = [

    ("A", "B"),

    ("A", "C"),

    ("B", "C"),

    ("B", "D"),

    ("B", "E"),

    ("C", "E"),

    ("C", "F"),

    ("D", "E"),

    ("E", "F"),

    ("E", "G"),

    ("F", "G")

]

def backtrack(assignment):

    """Runs backtracking search to find an assignment."""

    # Check if assignment is complete

    if len(assignment) == len(VARIABLES):

        return assignment

    # Try a new variable

    var = select\_unassigned\_variable(assignment)

    for value in ["Monday", "Tuesday", "Wednesday"]:

        new\_assignment = assignment.copy()

        new\_assignment[var] = value

        if consistent(new\_assignment):

            result = backtrack(new\_assignment)

            if result is not None:

                return result

    return None

def select\_unassigned\_variable(assignment):

    """Chooses a variable not yet assigned, in order."""

    for variable in VARIABLES:

        if variable not in assignment:

            return variable

    return None

def consistent(assignment):

    """Checks to see if an assignment is consistent."""

    for (x, y) in CONSTRAINTS:

        # Only consider arcs where both are assigned

        if x not in assignment or y not in assignment:

            continue

        # If both have same value, then not consistent

        if assignment[x] == assignment[y]:

            return False

    # If nothing inconsistent, then assignment is consistent

    return True

solution = backtrack(dict())

print(solution)

////////////////

Pract 3B

Code:

# working of Alpha-Beta Pruning

# Initial values of Aplha and Beta

MAX, MIN = 1000, -1000

def minimax(depth, nodeIndex, maximizingPlayer,

            values, alpha, beta):

    if depth == 3:

        return values[nodeIndex]

    if maximizingPlayer:

        best = MIN

        # Recur for left and right children

        for i in range(0, 2):

            val = minimax(depth + 1, nodeIndex \* 2 + i,

                          False, values, alpha, beta)

            best = max(best, val)

            alpha = max(alpha, best)

            # Alpha Beta Pruning

            if beta <= alpha:

                break

        return best

    else:

        best = MAX

        for i in range(0, 2):

            val = minimax(depth + 1, nodeIndex \* 2 + i,

                            True, values, alpha, beta)

            best = min(best, val)

            beta = min(beta, best)

            # Alpha Beta Pruning

            if beta <= alpha:

                break

        return best

if \_\_name\_\_ == "\_\_main\_\_":

    values = [3, 5, 6, 9, 1, 2, 0, -1]

    print("The optimal value is :", minimax(0, 0, True, values, MIN, MAX))

Practical 4:

Code:

# code for decision tree

import pandas as pd

import numpy as np

from sklearn.datasets import load\_iris

from sklearn.tree import DecisionTreeClassifier

from sklearn.model\_selection import train\_test\_split

data = load\_iris()

print('Classes to predict: ', data.target\_names)

X = data.data

y = data.target

print('Number of examples in the data:', X.shape[0])

X[:4]

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, random\_state = 47, test\_size = 0.25)

from sklearn.tree import DecisionTreeClassifier

clf = DecisionTreeClassifier(criterion = 'entropy')

clf.fit(X\_train, y\_train)

y\_pred =  clf.predict(X\_test)

from sklearn.metrics import accuracy\_score

print('Accuracy Score on train data: ', accuracy\_score(y\_true=y\_train, y\_pred=clf.predict(X\_train)))

print('Accuracy Score on test data: ', accuracy\_score(y\_true=y\_test, y\_pred=y\_pred))

Prac 4B

Code:

from sklearn.datasets import load\_breast\_cancer

data\_loaded = load\_breast\_cancer()

X = data\_loaded.data

y = data\_loaded.target

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(data\_loaded.data, data\_loaded.target, test\_size=0.2,random\_state=20)

from sklearn.naive\_bayes import GaussianNB

naive\_bayes = GaussianNB()

naive\_bayes.fit(X\_train , y\_train)

y\_predicted = naive\_bayes.predict(X\_test)

from sklearn import metrics

metrics.accuracy\_score(y\_predicted , y\_test)

Practical 5

Code:

from google.colab import files

uploaded = files.upload()

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

import sklearn as skl

heart = pd.read\_csv('heart.csv')

import io

!pip install pgmpy

from pgmpy.models import BayesianModel

from pgmpy.estimators import MaximumLikelihoodEstimator, BayesianEstimator

heart.columns

model = BayesianModel([('age','trestbps'),('age','fbs'),('sex','trestbps'),('exang','trestbps'),('trestbps','target'),('fbs','target'),

('target','restecg'),('target','thalach'),('target','chol')])

model.fit(heart,estimator=MaximumLikelihoodEstimator)

print(model.get\_cpds('age'))

(Note: data file add karna ai floder d: block heart.csv)

Practical 6:

Code:

  import numpy as np

from google.colab import files

files.upload()

trump = open('DonaldTs.txt', encoding='utf8').read()

#display the data

print(trump)

corpus = trump.split()

#Display the corpus

print(corpus)

def make\_pairs(corpus):

  for i in range(len(corpus) - 1):

    yield (corpus[i], corpus[i + 1])

pairs = make\_pairs(corpus)

word\_dict={}

for word\_1, word\_2 in pairs:

  if word\_1 in word\_dict.keys():

    word\_dict[word\_1].append(word\_2)

  else:

    word\_dict[word\_1] = [word\_2]

#randomly pick the first word

first\_word = np.random.choice(corpus)

#Pick the first word as a capitalized word so that the picked word is not taken from in between a sentence

while first\_word.islower():

#Start the chain from the picked word

   chain = [first\_word]

#Initialize the number of stimulated words

   n\_words = 60

   for i in range(n\_words):

          chain.append(np.random.choice(word\_dict[chain[-1]]))

print(" ".join(chain))

Prcatical 7

Code:

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('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('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\_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

train\_set\_x\_orig, train\_set\_y, test\_set\_x\_orig, test\_set\_y\_orig, classes = load\_dataset()

#index = 26

#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.")

index= 25

plt.imshow(test\_set\_x\_orig[index])

print ("y = " + str(test\_set\_y[:, index]) + ", it's a '" + classes[np.squeeze(test\_set\_y[:, index])].decode("utf-8") +  "' picture.")

#plt.imshow(train\_set\_y\_orig[index])

#print ("y = " + str(train\_set\_y[:, index]) + ", it's a '" + classes[np.squeeze(train\_set\_y[:, index])].decode("utf-8") +  "' picture.")

///////// new line /////////

import numpy as np

import matplotlib.pyplot as plt

import h5py

import scipy

from PIL import Image

from scipy import ndimage

#from lr\_utils import load\_dataset

%matplotlib inline

def load\_dataset():

    train\_dataset = h5py.File('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('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\_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

#Data will be loaded from the test\_catvnoncat.h5 and train\_catvnoncat.h5 files

#The load\_dataset function below is responsebile for loading the above mentioned data files.

#lr\_utils file includes the function load\_dataset()

# Loading the data (cat/non-cat)

train\_set\_x\_orig, train\_set\_y, test\_set\_x\_orig, test\_set\_y\_orig, classes = load\_dataset()

# We added "\_orig" at the end of image datasets (train and test) because we are going to preprocess them. After preprocessing, we will end up with train\_set\_x and test\_set\_x (the labels train\_set\_y and test\_set\_y don't need any preprocessing).

# Each line of your train\_set\_x\_orig and test\_set\_x\_orig is an array representing an image. You can visualize an example by running the following code. Feel free also to change the `index` value and re-run to see other images.

# Example of a picture

#change the index value below to check if the image at that particular index is cat or non cat

index = 26

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.")

#plt.imshow(train\_set\_y\_orig[index])

#print ("y = " + str(train\_set\_y[:, index]) + ", it's a '" + classes[np.squeeze(train\_set\_y[:, index])].decode("utf-8") +  "' picture.")

/// new line //////

import numpy as np

import h5py

    # Loading the data (cat/non-cat)

train\_dataset = h5py.File('train\_catvnoncat.h5', "r")

train\_set\_x\_orig = np.array(train\_dataset["train\_set\_x"][:]) # train set features

train\_set\_y\_orig = np.array(train\_dataset["train\_set\_y"][:]) # train set labels

test\_dataset = h5py.File('test\_catvnoncat.h5', "r")

test\_set\_x\_orig = np.array(test\_dataset["test\_set\_x"][:]) # test set features

test\_set\_y\_orig = np.array(test\_dataset["test\_set\_y"][:]) # 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]))

m\_train = train\_set\_x\_orig.shape[0]

m\_test = test\_set\_x\_orig.shape[0]

num\_px = train\_set\_x\_orig.shape[1]

print ("Dataset dimensions:")

print ("Number of training examples: m\_train = " + str(m\_train))

print ("Number of testing examples: m\_test = " + str(m\_test))

print ("Height/Width of each image: num\_px = " + str(num\_px))

print ("Each image is of size: (" + str(num\_px) + ", " + str(num\_px) + ", 3)")

print ("train\_set\_x shape: " + str(train\_set\_x\_orig.shape))

print ("train\_set\_y shape: " + str(train\_set\_y.shape))

print ("test\_set\_x shape: " + str(test\_set\_x\_orig.shape))

print ("test\_set\_y shape: " + str(test\_set\_y.shape))

//// new line //////

# Reshape the training and test examples

train\_set\_x\_flatten = train\_set\_x\_orig.reshape(train\_set\_x\_orig.shape[0], -1).T

test\_set\_x\_flatten = test\_set\_x\_orig.reshape(test\_set\_x\_orig.shape[0], -1).T

print ("train\_set\_x\_flatten shape: " + str(train\_set\_x\_flatten.shape))

print ("train\_set\_y shape: " + str(train\_set\_y.shape))

print ("test\_set\_x\_flatten shape: " + str(test\_set\_x\_flatten.shape))

print ("test\_set\_y shape: " + str(test\_set\_y.shape))

print ("sanity check after reshaping: " + str(train\_set\_x\_flatten[0:5,0]))

/// new line ///

train\_set\_x = train\_set\_x\_flatten/255.

test\_set\_x = test\_set\_x\_flatten/255.

from sklearn.linear\_model import LogisticRegression

lr = LogisticRegression(C=1000.0, random\_state=0)

lr.fit(train\_set\_x.T, train\_set\_y.T.ravel())

lr.coef\_.shape

lr.coef\_

lr.intercept\_

Y\_prediction = lr.predict(test\_set\_x.T)

Y\_prediction.shape

print("test accuracy: {} %".format(100 - np.mean(np.abs(Y\_prediction - test\_set\_y)) \* 100))

////new line ////

lr.coef\_.shape

//new line//

lr.coef\_

// new line //

lr.intercept\_

// new line //

Y\_prediction = lr.predict(test\_set\_x.T)

Y\_prediction.shape

// new line ///

print("test accuracy: {} %".format(100 - np.mean(np.abs(Y\_prediction - test\_set\_y)) \* 100))

(Note : add pract 7 file in Ai folder and run)

Practical 8:

Code:

import numpy as np

def unitStep(v):

  if v>=0:

    return 1

  else:

    return 0

def perceptronModel(x,w,b):

  v=np.dot(w,x)+b

  y=unitStep(v)

  return y

def OR\_logicfunction(x):

  w=np.array([1,1])

  b=-0.5

  return perceptronModel(x,w,b)

def AND\_logicFunction(x):

 w = np.array([1, 1])

 bAND = -1.5

 return perceptronModel(x, w, bAND)

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("AND({},{})={}".format(0,1,AND\_logicFunction(test1)))

print("AND({},{})={}".format(1,1,AND\_logicFunction(test2)))

print("AND({},{})={}".format(0,0,AND\_logicFunction(test3)))

print("AND({},{})={}".format(1,0,AND\_logicFunction(test4)))

Practical 9

Code:

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

import seaborn as sns

/// new line///

url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"

# Assign colum names to the dataset

names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'Class']

# Read dataset to pandas dataframe

dataset = pd.read\_csv(url, names=names)

dataset.head()

//// new line/////

X = dataset.iloc[:, :-1].values

y = dataset.iloc[:, 4].values

///// new line ///

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.20)

/// new line ///

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

scaler.fit(X\_train)

X\_train = scaler.transform(X\_train)

X\_test = scaler.transform(X\_test)

//// new line ///

from sklearn.neighbors import KNeighborsClassifier

classifier = KNeighborsClassifier(n\_neighbors=5)

classifier.fit(X\_train, y\_train)

/// new line ///

y\_pred = classifier.predict(X\_test)

from sklearn.metrics import classification\_report, confusion\_matrix

print(confusion\_matrix(y\_test, y\_pred))

print(classification\_report(y\_test, y\_pred))

//// new line ///

dataset.columns

// new line ///

sns.pairplot(dataset,hue="Class")

Practical 10

Code:

from google.colab import files

files.upload()

from numpy import loadtxt

from keras.models import Sequential

from keras.layers import Dense

/// new line ///

# load the dataset

dataset = loadtxt('pima-indians-diabetes.csv.csv', delimiter=',')

# split into input (X) and output (y) variables

X = dataset[:,0:8]

y = dataset[:,8]

///new line ////

# define the keras model

model = Sequential()

model.add(Dense(12, input\_dim=8, activation='relu'))

model.add(Dense(8, activation='relu'))

model.add(Dense(1, activation='sigmoid'))

/// new line ///

# compile the keras model

model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy'])

/// new line////

...

# fit the keras model on the dataset

model.fit(X, y, epochs=150, batch\_size=10)

# fit the keras model on the dataset

model.fit(X, y, epochs=150, batch\_size=10)

//// new line///

# evaluate the keras model

\_, accuracy = model.evaluate(X, y)

print('Accuracy: %.2f' % (accuracy\*100))

/// new line///

model.fit(X, y, epochs=150, batch\_size=10, verbose=0)

# evaluate the keras model

\_, accuracy = model.evaluate(X, y, verbose=0)

// new line//

# make probability predictions with the model

predictions = model.predict(X)

# round predictions

rounded = [round(x[0]) for x in predictions]

//// new line ///

from numpy import loadtxt

from keras.models import Sequential

from keras.layers import Dense

# load the dataset

dataset = loadtxt('pima-indians-diabetes.csv.csv', delimiter=',')

# split into input (X) and output (y) variables

X = dataset[:,0:8]

y = dataset[:,8]

# define the keras model

model = Sequential()

model.add(Dense(12, input\_dim=8, activation='relu'))

model.add(Dense(8, activation='relu'))

model.add(Dense(1, activation='sigmoid'))

# compile the keras model

model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy'])

# fit the keras model on the dataset

model.fit(X, y, epochs=150, batch\_size=10, verbose=0)

# make class predictions with the model

# summarize the first 5 cases

for i in range(5):

  print('%s => %d (expected %d)' % (X[i].tolist(), predictions[i], y[i]))

(Note : add file **pima-indians-diabetes.csv Pract 10 )**