

**Machine Learning with Graphs**

**LAB FILE**

**Bachelor OF Technology**

**(Academic Session: 2019-2023)**

**Name of Student:** Divvela Vishnu Sai Kumar

**Enrollment No. :** A076119819005

**Branch:** B.Tech (AI) 7th Sem

**Course** **Name**: Machine learning with graphs

**Course Code**: CSE467

**Faculty Name:** Prof. Dr. Syed Wajahat Abbas Rizvi

**Faculty** **Name**: Dr Pooja Khanna

1

**Department of Computer Science & Engineering**

**Amity School of Engineering & Technology**

**Amity University, Uttar Pradesh**

**INDEX**

|  |  |  |  |
| --- | --- | --- | --- |
| **S.NO.** | **Program Title** | **Date** | **Remarks** |
| 1 | Use Dijkstra’s algorithm to find the shortest path in a weighted and unweighted network |  |  |
| 2 | Write a program for XOR network using Artificial Neural Networks |  |  |
| 3 | Implementing the Girvan-Newman Algorithm for Community Detection in Python |  |  |
| 4 | Implement image Classification using TensorFlow |  |  |
| 5 | Use Recurrent Neural networks to predict the sentiment of various tweets |  |  |
| 6 | Use SlashBurn for social network graph compression |  |  |
| 7 | Implement knowledge graph construction for general human knowledge |  |  |
| 8 | Implement of a value iteration algorithm for calculating an optimal MDP policy |  |  |
| 9 | Create a GNN recommendation system for online shopping. |  |  |
| 10 | Open Ended Problems |  |  |

**PRACTICAL-WORK**

Practical-1

OBJECTIVE:

Use Dijkstra’s algorithm to find the shortest path in a weighted graph.

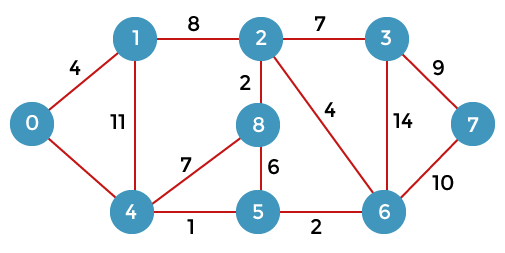
Tools Used:

* Python
* Vs code editor

Description:

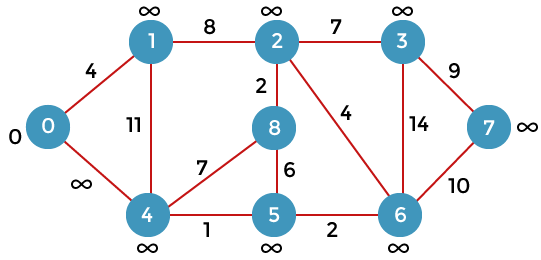
In this practical we will write a program on finding shortest path from source node to every node of the weighted graph using Dijkstra’s algorithm in python programming language. Dijkstra algorithm is a single-source shortest path algorithm. Here, single-source means that only one source is given, and we have to find the shortest path from the source to all the nodes.

**Let's understand the working of Dijkstra's algorithm. Consider the below graph.**



First, we have to consider any vertex as a source vertex. Suppose we consider vertex 0 as a source vertex.

Here we assume that 0 as a source vertex, and distance to all the other vertices is infinity. Initially, we do not know the distances. First, we will find out the vertices which are directly connected to the vertex 0. As we can observe in the above graph that two vertices are directly connected to vertex 0.



Let's assume that the vertex 0 is represented by 'x' and the vertex 1 is represented by 'y'. The distance between the vertices can be calculated by using the below formula:

**d(x, y) = d(x) + c(x, y) < d(y)**

= **(0 + 4)** < ∞

= **4** < ∞

Since 4<∞ so we will update d(v) from ∞ to 4.

Therefore, we come to the conclusion that the formula for calculating the distance between the vertices:

**{if( d(u) + c(u, v) < d(v))**

**d(v) = d(u) +c(u, v) }**

Now we consider vertex 0 same as 'x' and vertex 4 as 'y'.

**d(x, y) = d(x) + c(x, y) < d(y)**

= **(0 + 8)** < ∞

= **8** < ∞

Therefore, the value of d(y) is 8. We replace the infinity value of vertices 1 and 4 with the values 4 and 8 respectively. Now, we have found the shortest path from the vertex 0 to 1 and 0 to 4. Therefore, vertex 0 is selected. Now, we will compare all the vertices except the vertex 0. Since vertex 1 has the lowest value, i.e., 4; therefore, vertex 1 is selected.

Since vertex 1 is selected, so we consider the path from 1 to 2, and 1 to 4. We will not consider the path from 1 to 0 as the vertex 0 is already selected.

First, we calculate the distance between the vertex 1 and 2. Consider the vertex 1 as 'x', and the vertex 2 as 'y'.

**d(x, y) = d(x) + c(x, y) < d(y)**

= **(4 + 8)** < ∞

= **12** < ∞

Since 12<∞ so we will update d(2) from ∞ to 12.

Now, we calculate the distance between the vertex 1 and vertex 4. Consider the vertex 1 as 'x' and the vertex 4 as 'y'.

**d(x, y) = d(x) + c(x, y) < d(y)**

**= (4 + 11) < 8**

**= 15 < 8**

Since 15 is not less than 8, we will not update the value d(4) from 8 to 12.

Till now, two nodes have been selected, i.e., 0 and 1. Now we have to compare the nodes except the node 0 and 1. The node 4 has the minimum distance, i.e., 8. Therefore, vertex 4 is selected.

CODE:

**import** sys

**class** Graph():

**def** \_\_init\_\_(self, vertices):

        self.V **=** vertices

        self.graph **=** [[0 **for** column **in** range(vertices)]

**for** row **in** range(vertices)]

**def** printSolution(self, dist):

**print**("Vertex \tDistance from Source")

**for** node **in** range(self.V):

            print(node, "\t", dist[node])

**def** minDistance(self, dist, sptSet):

        min **=** sys.maxsize

**for** u **in** range(self.V):

**if** dist[u] < min **and** sptSet[u] **==** False:

                min **=** dist[u]

                min\_index **=** u

**return** min\_index

**def** dijkstra(self, src):

        dist **=** [sys.maxsize] **\*** self.V

        dist[src] **=** 0

        sptSet **=** [False] **\*** self.V

**for** cout **in** range(self.V):

            x **=** self.minDistance(dist, sptSet)

            sptSet[x] **=** True

**for** y **in** range(self.V):

**if** self.graph[x][y] > 0 **and** sptSet[y] **==** False **and** \

                        dist[y] > dist[x] **+** self.graph[x][y]:

                    dist[y] **=** dist[x] **+** self.graph[x][y]

        self.printSolution(dist)

**if** \_\_name\_\_ **==** "\_\_main\_\_":

    g **=** Graph(9)

    g.graph **=** [[0, 4, 0, 0, 0, 0, 0, 8, 0],

               [4, 0, 8, 0, 0, 0, 0, 11, 0],

               [0, 8, 0, 7, 0, 4, 0, 0, 2],

               [0, 0, 7, 0, 9, 14, 0, 0, 0],

               [0, 0, 0, 9, 0, 10, 0, 0, 0],

               [0, 0, 4, 14, 10, 0, 2, 0, 0],

               [0, 0, 0, 0, 0, 2, 0, 1, 6],

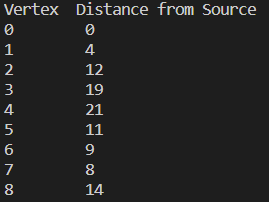
               [8, 11, 0, 0, 0, 0, 1, 0, 7],

               [0, 0, 2, 0, 0, 0, 6, 7, 0]

               ]

    g.dijkstra(0)

OUTPUT:



Practical-2

OBJECTIVE:

Write a program for XOR network using Artificial Neural Networks

Tools Used:

* NUMPY
* Jupyter Notebook
* Pandas
* Scikit-learn

Description:

In this practical we develop a neural network for XOR function in which the neural network is having one input layer, one hidden layer and one output layer. The input layer will have two neurons, the output layer will have one neuron, and the hidden layer will have two neurons. The activation function used for hidden layer is relu and the activation function used in the output layer is threshold activation function. The dataset which we pass is related to XOR function it is having two features and one classification column.

CODE:

def threshold(x):

if x>0:

return 1

else:

return 0

def relu(x):

if x<0:

return 0

else:

return x

class NeuralNetwork\_for\_twoinputs\_oneoutput:

def \_\_init\_\_(self,data):

self.data = data

self.w1=0.1

self.w2=0.5

self.w3=1.3

self.w4=0.7

self.w5=1.2

self.w6=1.8

self.b1=1.5

self.b2=1.1

self.train\_errors=[]

self.test\_errors=[]

def split\_data(self):

X=self.data.drop(columns='t',axis=1)

Y=self.data['t']

self.X\_train, self.X\_test, self.Y\_train, self.Y\_test = train\_test\_split(X, Y, test\_size=0.1, random\_state=101)

def train\_neuralnetwork(self,numofepochs):

self.numofepochs=numofepochs

self.train\_errors=[]

n=0

errors\_in\_each\_epoch=[]

for k in range(numofepochs):

n += 1

for i,j in zip(list(self.X\_train.index),self.Y\_train):

x1 = self.X\_train['x1'][i]

x2 = self.X\_train['x2'][i]

t=j

h1=x1\*self.w1+x2\*self.w3+self.b1

h2=x1\*self.w2+x2\*self.w4+self.b1

h1f=relu(h1)

h2f=relu(h2)

y=h1f\*self.w5+h2f\*self.w6+self.b2

yf=threshold(y)

error=(t-yf)\*\*2

der\_of\_error\_w1=-1\*2\*(t-yf)\*self.w5\*x1

w1\_new=self.w1-0.5\*der\_of\_error\_w1

der\_of\_error\_w2=-1\*2\*(t-yf)\*self.w6\*x1

w2\_new=self.w2-0.5\*der\_of\_error\_w2

der\_of\_error\_w3=-1\*2\*(t-yf)\*self.w5\*x2

w3\_new=self.w3-0.5\*der\_of\_error\_w3

der\_of\_error\_w4=-1\*2\*(t-yf)\*self.w6\*x2

w4\_new=self.w4-0.5\*der\_of\_error\_w4

der\_of\_error\_w5=-1\*2\*(t-yf)\*h1f

w5\_new=self.w5-0.5\*der\_of\_error\_w5

der\_of\_error\_w6=-1\*2\*(t-yf)\*h2f

w6\_new=self.w5-0.5\*der\_of\_error\_w6

der\_of\_error\_b1=-1\*2\*(t-yf)\*self.w5

b1\_new=self.b1-der\_of\_error\_b1

der\_of\_error\_b2=-1\*2\*(t-yf)

b2\_new=self.b2-der\_of\_error\_b2

self.w1=w1\_new

self.w2=w2\_new

self.w3=w3\_new

self.w4=w4\_new

self.w5=w5\_new

self.w6=w6\_new

self.b1=b1\_new

self.b2=b2\_new

h1=x1\*self.w1+x2\*self.w3+self.b1

h2=x1\*self.w2+x2\*self.w4+self.b1

h1f=relu(h1)

h2f=relu(h2)

y=h1f\*self.w5+h2f\*self.w6+self.b2

yf=threshold(y)

error=(t-yf)\*\*2

errors\_in\_each\_epoch.append(error)

print("Average of Errors of all data points in ",n," epoch is ",sum(errors\_in\_each\_epoch)/len(errors\_in\_each\_epoch))

self.train\_errors.append(sum(errors\_in\_each\_epoch)/len(errors\_in\_each\_epoch))

errors\_in\_each\_epoch=[]

def test\_neuralnetwork(self):

n=0

for i,j in zip(list(self.X\_test.index),self.Y\_test):

x1 = self.X\_test['x1'][i]

x2 = self.X\_test['x2'][i]

t=j

h1=x1\*self.w1+x2\*self.w3+self.b1

h2=x1\*self.w2+x2\*self.w4+self.b1

h1f=relu(h1)

h2f=relu(h2)

y=h1f\*self.w5+h2f\*self.w6+self.b2

yf=threshold(y)

error=(t-yf)\*\*2

n += 1

print("Error in ",n," row input of test data is ",error)

self.test\_errors.append(error)

avgoferror=sum(self.test\_errors)/len(self.test\_errors)

print("Average of Errors after testing the data is ",avgoferror)

def predict(self,x1,x2):

h1=x1\*self.w1+x2\*self.w3+self.b1

h2=x1\*self.w2+x2\*self.w4+self.b1

h1f=relu(h1)

h2f=relu(h2)

y=h1f\*self.w5+h2f\*self.w6+self.b2

yf=threshold(y)

print("The Prediction for following input is: ",yf)

data\_xor\_gate = pd.DataFrame([[0,0,0],[0,1,0],[1,0,0],[1,1,1]],columns=['x1','x2','t'])

model1 = NeuralNetwork\_for\_twoinputs\_oneoutput(data\_xor\_gate)

model1.split\_data()

model1.X\_train

model1.train\_neuralnetwork(5)

model1.test\_neuralnetwork()

model1.predict(0,1)

OUTPUT:

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Practical-3

OBJECTIVE:

Implementing the Girvan-Newman Algorithm for Community Detection in Python

Tools Used:

* Python
* Vs code editor

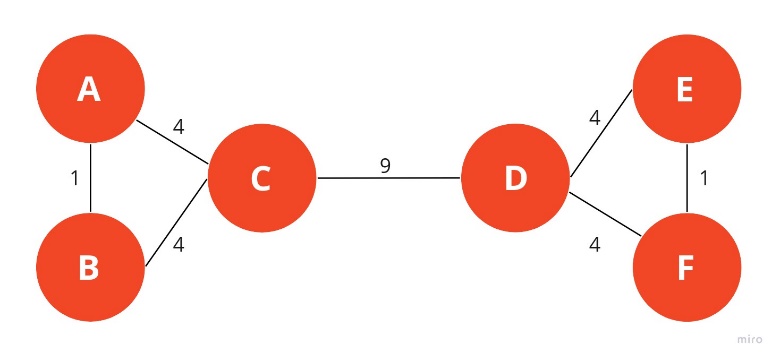
Description:

We are going to divide the nodes of the graph into two or more communities using the Girvan Newman algorithm. The Girvan Newman Algorithm removes the edges with the highest betweenness until there are no edges remain. Betweenness is the number of the shortest paths between pairs of nodes that run through it.

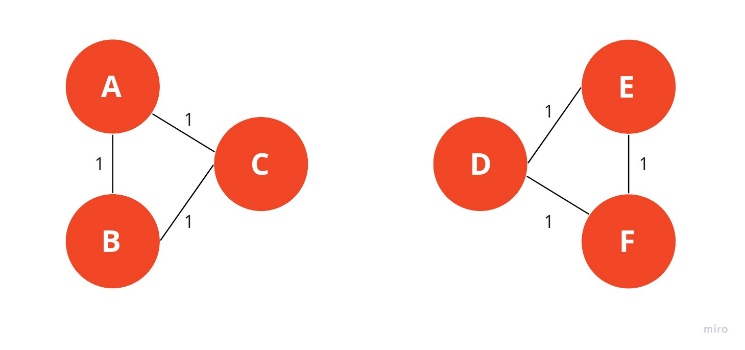
The Girvan-Newman algorithm for the detection and analysis of community structure relies on the iterative elimination of edges that have the highest number of shortest paths between nodes passing through them. By removing edges from the graph one-by-one, the network breaks down into smaller pieces, so-called communities. The algorithm was introduced by Michelle Girvan and Mark Newman.

The Girvan-Newman algorithm can be divided into four main steps:

1. For every edge in a graph, calculate the edge betweenness centrality.
2. Remove the edge with the highest betweenness centrality.
3. Calculate the betweenness centrality for every remaining edge.
4. Repeat steps 2-4 until there are no more edges left.



In this example, you can see how a typical graph looks like when **edges are assigned weights based on the number of shortest paths passing through them**. To keep things simple, we only calculated the number of undirected shortest paths that pass through an edge. The edge between nodes **A** and **B** has a strength of 1 because we don’t count **A->B** and **B->A** as two different paths.



The Girvan-Newman algorithm would remove the edge between nodes **C** and **D** because it is the one with the highest strength. As you can see intuitively, this means that the edge is located between communities. After removing an edge, the betweenness centrality has to be recalculated for every remaining edge. In this example, we have come to the point where every edge has the same betweenness centrality.

CODE:

**import** networkx as nx

**def** edge\_to\_remove(g):

    d1 **=** nx.edge\_betweenness\_centrality(g)

    list\_of\_tuples **=** list(d1.items())

    sorted(list\_of\_tuples, key **=** **lambda** x:x[1], reverse **=** True)

**return** list\_of\_tuples[0][0]

**def** girvan(g):

    a **=** nx.connected\_components(g)

    lena **=** len(list(a))

    print (' The number of connected components are ', lena)

**while** (lena **==** 1):

        u, v **=** edge\_to\_remove(g)

        g.remove\_edge(u, v)

        a **=** nx.connected\_components(g)

        lena**=**len(list(a))

**print** (' The number of connected components are ', lena)

**return** a

g **=** nx.barbell\_graph(5,0)

a **=** girvan(g)

print ('Barbell Graph')

**for** i **in** a:

    print (i.nodes())

    print ('.............')

g1 **=** nx.karate\_club\_graph()

a1 **=** girvan(g1)

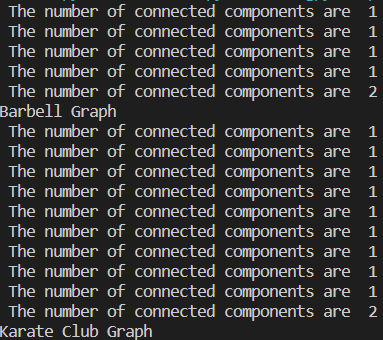
print ('Karate Club Graph')

**for** i **in** a1:

**print** (i.nodes())

    print ('.............')

OUTPUT:



Practical-4

OBJECTIVE:

Implement image Classification using TensorFlow

Tools Used:

* NUMPY
* Jupyter Notebook
* Pandas
* Tensorflow-keras
* Matplotlib

Description:

In this practical we implemented a neural network with a convolutional layers which is containing convolution, pooling layers. First the data will be passed to this convolutional layer and after that we will send that data to the Neural network. The dataset which we considered is having 10 classes to classify the data which are airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck.

CODE:

# Import TensorFlow

import tensorflow as tf

from tensorflow.keras import datasets, layers, models

import matplotlib.pyplot as plt

(train\_images, train\_labels), (test\_images, test\_labels) = datasets.cifar10.load\_data()

# Normalize pixel values to be between 0 and 1

train\_images, test\_images = train\_images / 255.0, test\_images / 255.0

class\_names = ['airplane', 'automobile', 'bird', 'cat', 'deer','dog', 'frog', 'horse', 'ship', 'truck']

plt.figure(figsize=(8,8))

for i in range(25):

plt.subplot(5,5,i+1)

plt.xticks([])

plt.yticks([])

plt.grid(False)

plt.imshow(train\_images[i])

# The CIFAR labels happen to be arrays,

#which is why we need the extra index

plt.xlabel(class\_names[train\_labels[i][0]])

plt.show()

len(train\_images[0])

train\_images[0].shape

model = models.Sequential()

model.add(layers.Conv2D(32, (3, 3), activation='relu', input\_shape=(32, 32, 3)))

model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Conv2D(64, (3, 3), activation='relu'))

model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Conv2D(64, (3, 3), activation='relu'))

model.add(layers.Flatten())

model.add(layers.Dense(64, activation='relu'))

model.add(layers.Dense(10,activation='softmax'))

# Adam is the best among the adaptive optimizers in most of the cases

model.compile(optimizer='adam',

loss=tf.keras.losses.SparseCategoricalCrossentropy(from\_logits=True),

metrics=['accuracy'])

# An epoch means training the neural network with all the

# training data for one cycle. Here I use 10 epochs

history = model.fit(train\_images, train\_labels, epochs=10,

validation\_data=(test\_images, test\_labels))

plt.plot(history.history['accuracy'],label='accuracy')

plt.plot(history.history['val\_accuracy'],label = 'val\_accuracy')

plt.xlabel('Epoch')

plt.ylabel('Accuracy')

plt.ylim([0.5, 1])

plt.legend(loc='lower right')

test\_loss, test\_acc = model.evaluate(test\_images,

test\_labels,

verbose=2)

print('Test Accuracy is',test\_acc)

class\_names[model.predict(np.asarray([train\_images[0]])).argmax()]

OUTPUT:

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated A screenshot of a computer

Description automatically generated A screenshot of a computer

Description automatically generated A screenshot of a computer

Description automatically generated

Practical-5

OBJECTIVE:

Use Recurrent Neural networks to predict the sentiment of various tweets

Tools Used:

* NUMPY
* Jupyter Notebook
* Pandas
* Tensorflow-keras

Description:

In this practical we implemented a recurrent neural network which tries to classify the sentiment of the tweets by training on the tweets dataset. In this practical we will tokenize the sentence in which all the lemmatization, stemming, removing stop words and converting to count vector are included, and we will padd the sequences generated by tokenizer and we will train the dataset with its sentiment label to a LSTM network.

Code:

import numpy as np

import pandas as pd

from tensorflow.keras.preprocessing.text import Tokenizer

from tensorflow.keras.preprocessing.sequence import pad\_sequences

import tensorflow as tf

import speech\_recognition as sr

data = pd.read\_csv('tweet\_emotions.csv')

data.drop(['tweet\_id'],inplace=True,axis=1)

data['sentiment'].value\_counts()

data.isnull().sum().sum()

vocab\_size = 1000

max\_length = 70

embedding\_dim = 16

tokenizer = Tokenizer(oov\_token="<OOV>")

tokenizer.fit\_on\_texts(data['content'])

word\_index = tokenizer.word\_index

# Generate and pad the training sequences

sequences = tokenizer.texts\_to\_sequences(data['content'])

padded = pad\_sequences(sequences,maxlen=max\_length, truncating='post')

vocab\_size = len(word\_index)

from sklearn import preprocessing

le = preprocessing.LabelEncoder()

le.fit(data['sentiment'])

data['emotion']=le.transform(data['sentiment'])

le.classes\_

data.head()

from tensorflow.keras.utils import to\_categorical

y = to\_categorical(data['emotion']).astype(int)

y[0]

train\_size = int(data.shape[0]\*0.9)

X\_train = padded[:train\_size]

y\_train = y[:train\_size]

X\_test = padded[train\_size:]

y\_test = y[train\_size:]

# Build the model

model = tf.keras.Sequential([

    tf.keras.layers.Embedding(vocab\_size,embedding\_dim, input\_length=max\_length),

    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(128,return\_sequences=True)),

    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(64,return\_sequences=True)),

    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(32)),

    tf.keras.layers.Dense(64, activation='relu'),

    tf.keras.layers.Dense(13, activation='softmax')

])

# Setup the training parameters

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

# Print the model summary

model.summary()

NUM\_EPOCHS = 20

# Train the model

history = model.fit(X\_train,y\_train, epochs=NUM\_EPOCHS, validation\_data=(X\_test,y\_test))

text4 = "charviray Charlene my love. I miss you"

data4 = [text4]

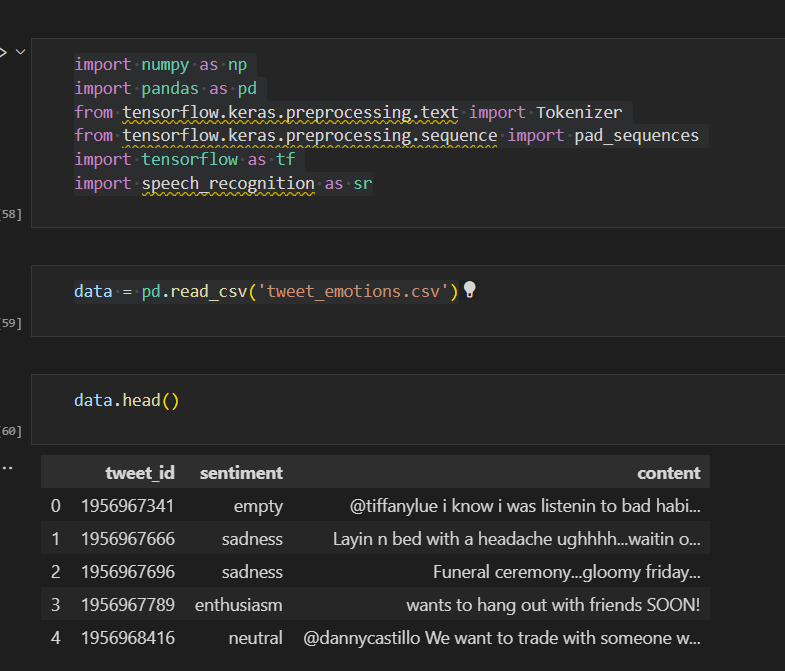
data4=tokenizer.texts\_to\_sequences(data4)

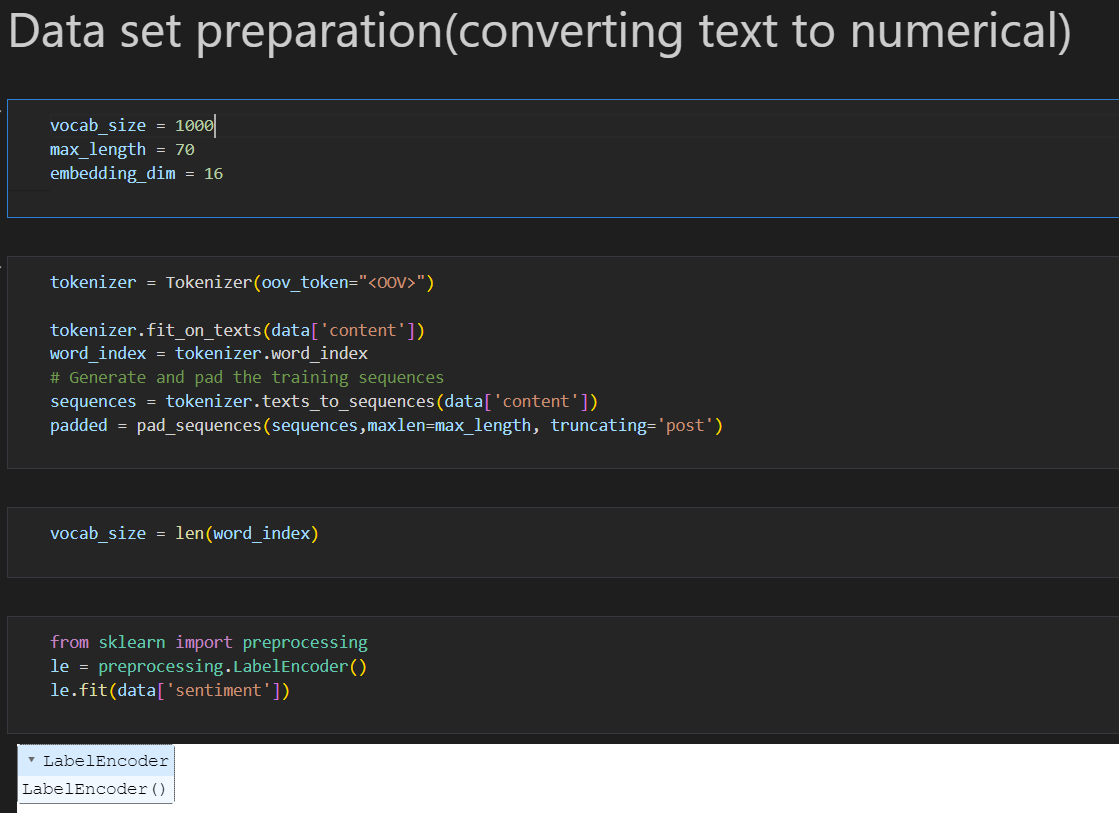
data4=pad\_sequences(data4,maxlen=max\_length,truncating='post')

prediction4 = model.predict(data4)

print(le.classes\_[np.argmax(prediction4)])

Output:





A screenshot of a computer

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Chart, line chart

Description automatically generated

Text

Description automatically generated

Practical-6

OBJECTIVE:

Use SlashBurn for social network graph compression.

Tools Used:

* Python
* Vs code editor

Description:

A social network is a network of individuals (such as friends, acquaintances, and coworkers) connected by interpersonal relationships. The number of active users in social media networks is in an exponential rise, creating the need for complex mechanisms to analyze and understand user behavior. One important tool to aid this being the ability to compress these social media networks into graphs that can fit in main memory. This project aims to provide implementations for the state of the art social network compression algorithms (i.e graph compression) based on current research.

There are working implementations based on three methods,

Backlinks compression scheme (C++,[1])

Slashburn technique (Python,[2])

Abstract Binary Tree compression (C++,[3]).

Slashburn tried to exploit the hubs and the neighbors (‘spokes’) of the hubs to define an alternative community different from the traditional community. It is based on the observation that real world graphs are easily disconnected by hubs, or high degree nodes: removing hubs from a graph creates many small disconnected components, and the remaining giant connected component is substantially smaller than the original graph. The method aims to order these hubs and spokes to get such a compact representation of the adjacency matrix, which in turn leads to good compression.

In its essence Slashburn is a reodering algorithm following which running standard compression algorithms such as gzip, bz2 and 7z yield better compression.

Practical-7

OBJECTIVE:

Implement knowledge graph construction for general human knowledge.

Tools Used:

* Python
* Vs code editor

Description:

The advent of the internet has granted access to a large number of content creators to generate information. Owing to this, there is a massive amount of data that is now present on the web. In order to provide useful insights, we need an efficient way to represent all this data. One such efficient knowledge representation method is via knowledge graphs. In brief, a knowledge graph is a large network of interconnected data. Knowledge graphs are constructed from knowledge bases. Knowledge bases gather their information from free text on web pages, databases, and audio and video content. The basic pipeline of a knowledge graph’s construction process is shown in Figure 1.

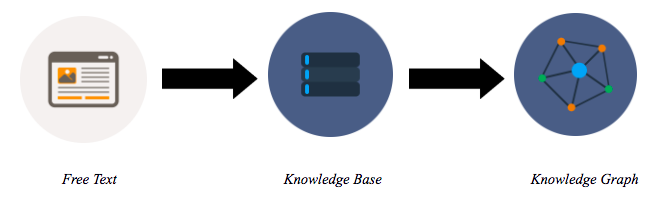


Figure 1. The knowledge graph construction pipeline

**First step**

During the first phase of the pipeline, we identify facts from free text. Initially, we scour the internet to filter useful information by identifying the entities and the relationships that the entities are involved in from free text. This identification process takes place using natural language processing techniques, such as named entity resolution, lemmatization, and stemming. Hence, the data extracted from free text in the first step may resemble the form of the following statement.

“The Louvre is located in Paris”

**Second step**

Proceeding to the second phase of the pipeline, the statements are generalized in the form of triples within knowledge bases; these triples will be categorized under different ontologies using an ontology extraction process that can harness the capabilities of natural language processing techniques as well. A triple is composed of a subject, the predicate, and its object. The subject and object are entities that are involved in a relationship defined by the predicate. Hence, for the previous statement identified from free text, we break this down in the following form of a triple for the knowledge base.

Subject : Louvre

Predicate : is located

Object : Paris

So within a knowledge base, we will have the above relationship in the form of islocated(Louvre, Paris). This is a single triple within a knowledge base. In practice, knowledge bases include millions of such triples, which we also term as facts. These facts are grouped under ontologies in knowledge bases. An ontology is an identifying category for a particular domain of facts.

**Knowledge Graphs**

A knowledge graph is a large network of interconnected entities. The connections are created based on the triples from knowledge bases. The main intent of the knowledge graph is to identify the missing links between entities. In order to explicate this further, let’s consider the following sample relationships that we have gathered from the knowledge base.

* *Friends (Anne, Jane)*
* *Friends (Jane, Jim)*
* *LivesIn (Anne, Paris)*
* *LivesIn (Jim, Brazil)*
* *LivesIn (Jane, Brazil)*
* *BornIn (Anne, Paris)*
* *BornIn (Jim, Paris)*

If we try to build a basic knowledge graph based on only the above relationships, we will be able to visualize the following graph.



Figure 3. A knowledge graph constructed only using the observed facts

On the other hand, there are some unknown relationships that were not explicitly retrieved from the knowledge bases, such as, Are Anne and Jim friends?, What is Jane’s birthplace? This means that such relationships can be considered as missing links.

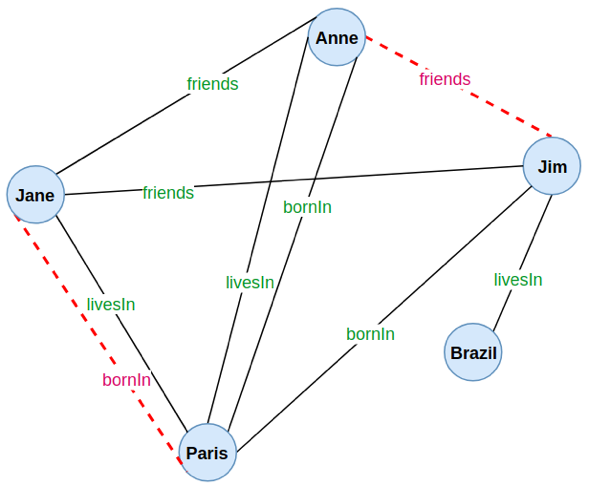


Figure 4. The missing links in the knowledge graph

Practical-8

OBJECTIVE:

Implement a value iteration algorithm for calculating an optimal MDP policy.

Tools Used:

* Python
* Vs code editor

Description:

Reinforcement learning is an area of Machine Learning that focuses on having an agent learn how to behave/act in a specific environment. MDPs are simply meant to be the framework of the problem, the environment itself.

**What constitutes a MDP?**

MDPs are composed of 5 elements.

A set of states. This represents all the possible locations for the agent in the environment.

A set of actions. This represents all the actions the agent can take at any given state.

Transition probabilities. This represents the probability that the action the agent attempts will be successful (or not).

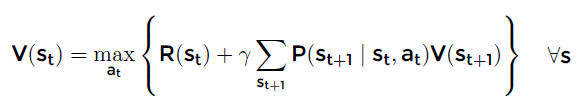
Rewards, which are the values of arriving in a specific state.

A discount factor γ which is meant to diminish the value of future rewards compared to instant rewards.

The goal of the agent in a MDP is to find the optimal policy, which is the set of optimal actions to take at any given state.

**Value Iteration Algorithm**

The algorithm tries to find the value V(st) of being in any given state. It uses the Bellman equation.



Don’t worry, it really isn’t that complicated. All this means is that the value of being in a state is equal to the maximum of the immediate reward of that state (R) plus the discounted rewards of every adjacent state (St+1), considering the transition probabilities. Therefore, we only look one step ahead in this algorithm.

Code:

|  |  |
| --- | --- |
|  | '''================================================== |
|  | Value Iteration |
|  | ==================================================''' |
|  |  |
|  | iteration = 0 |
|  | while True: |
|  | biggest\_change = 0 |
|  | for s in all\_states: |
|  | if s in policy: |
|  |  |
|  | old\_v = V[s] |
|  | new\_v = 0 |
|  |  |
|  | for a in actions[s]: |
|  | if a == 'U': |
|  | nxt = [s[0]-1, s[1]] |
|  | if a == 'D': |
|  | nxt = [s[0]+1, s[1]] |
|  | if a == 'L': |
|  | nxt = [s[0], s[1]-1] |
|  | if a == 'R': |
|  | nxt = [s[0], s[1]+1] |
|  |  |
|  | #Choose a new random action to do (transition probability) |
|  | random\_1=np.random.choice([i for i in actions[s] if i != a]) |
|  | if random\_1 == 'U': |
|  | act = [s[0]-1, s[1]] |
|  | if random\_1 == 'D': |
|  | act = [s[0]+1, s[1]] |
|  | if random\_1 == 'L': |
|  | act = [s[0], s[1]-1] |
|  | if random\_1 == 'R': |
|  | act = [s[0], s[1]+1] |
|  |  |
|  | #Calculate the value |
|  | nxt = tuple(nxt) |
|  | act = tuple(act) |
|  | v = rewards[s] + (GAMMA \* ((1-NOISE)\* V[nxt] + (NOISE \* V[act]))) |
|  | if v > new\_v: #Is this the best action so far? If so, keep it |
|  | new\_v = v |
|  | policy[s] = a |
|  |  |
|  | #Save the best of all actions for the state |
|  | V[s] = new\_v |
|  | biggest\_change = max(biggest\_change, np.abs(old\_v - V[s])) |
|  |  |
|  | #See if the loop should stop now |
|  | if biggest\_change < SMALL\_ENOUGH: |
|  | break |
|  | iteration += 1 |

Practical-9

OBJECTIVE:

Perform basic operations on matrices (like addition, Subtraction, Multiplication) and display specific rows or columns of the matrix.

Tools Used:

* NUMPY
* SPYDER (IDE)

PROGRAM:

* MATRIX ADDITION

import numpy as np

#Matrix x

x = np.array([[1,1,1],[1,1,1],[1,1,2]])

#Matrix y

y = np.array([[1,3,1],[1,3,1],[1,3,8]])

#Add two matrices

Add = np.add(x,y)

print("Addition of Two Matrices \n",Add)

OUTPUT:

Text

Description automatically generated

* MATRIX SUBTRACTION

import numpy as np

#Matrix x

x = np.array([[1,1,1],[1,1,1],[1,1,2]])

#Matrix y

y = np.array([[1,3,1],[1,3,1],[1,3,8]])

#Subtraction of two matrices

Subtract = np.subtract(x,y)

print("subtraction of two matrices : \n",Subtract)

OUTPUT:

A picture containing graphical user interface

Description automatically generated

* MATRIX MULTIPLICATION

import numpy as np

#Matrix x

x = np.array([[1,1,1],[1,1,1],[1,1,2]])

#Matrix y

y = np.array([[1,3,1],[1,3,1],[1,3,8]])

#Mutliplication of two matrices

Multiply = np.multiply(x,y)

print("Mutliplication of two matrices:\n",Multiply)

Output:

Text

Description automatically generated

Practical-10

Objective:

Perform other matrix operations like converting matrix data to absolute values, taking the negative of matrix values, adding/removing rows/Column from a matrix, finding the maximum or minimum values in a matrix or in a row/column, and finding the sum of some/all elements in a matrix.

Tools Used:

* NumPy
* Spyder (IDE)

Program:

* Converting matrix data to Absolute values

import numpy as np

matrix = np.array([[1,-2,3,4],[5,6,7,-8],[9,-10,11,-12]])

print("Matix Data : \n",matrix)

print("After converting matrix data into Absolute Values : ")

ab\_matrix = np.absolute(matrix)

print(ab\_matrix)

OUTPUT:

Graphical user interface, text

Description automatically generated with medium confidence

* Negative of the Matrix.

import numpy as np

matrix = np.array([[1,2,3,4],[5,6,7,8],[9,10,11,12]])

print("Matix Data : \n",matrix)

print("After converting matrix data into Negative Values : ")

neg\_matrix = np.negative(matrix)

print(neg\_matrix)

OUTPUT:

Text

Description automatically generated

* Adding row/columns in a Matrix.

import numpy as np

matrix = np.array([[1,2,3,4],[5,6,7,8],[9,10,11,12]])

#Adding columns to a matrix

print("Matrix Before adding column and row :\n",matrix)

add\_column =np.array([10,20,30,40])

col\_result = np.vstack((matrix,add\_column))

print("After Adding a column to the Matrix : \n",col\_result)

#Adding a row to matrix

add\_row = np.array([[13,14,15,16]])

row\_result = np.append(matrix,add\_row,axis = 0)

print("After adding a row to the Matrix:\n",row\_result)

OUTPUT:

Text

Description automatically generated

* Removing row/column from the matrix

import numpy as np

matrix = np.array([[1,2,3,4],[5,6,7,8],[9,10,11,12]])

print(matrix)

#removing row from the matrix

remove\_row= np.delete(matrix,1,0)

#removing column from the matrix

remove\_column = np.delete(matrix,1,1)

print("After Deleting row 2 From the Matrix: \n",remove\_row)

print("After Deleting column 2 From the Matrix: \n",remove\_column)

OUTPUT:

Text

Description automatically generated

* Maximum and Minimum Value in a Matrix

import numpy as np

matrix = np.array([[1,2,3,4],[5,6,7,8],[9,10,11,12]])

print(matrix)

#Finding maximum and minimum value in a matrix

print("Maximum Value in the Matrix : ",np.max(matrix))

print("Minimum Value in the Matrix : ",np.min(matrix))

#Maximum values in a row

for i in range(len(matrix)):

print("Maximum Value in row",i+1,":",np.max(matrix[i]))

OUTPUT:

Text

Description automatically generated

* Sum of All values in a Matrix

import numpy as np

matrix = np.array([[1,2,3,4],[5,6,7,8],[9,10,11,12]])

print(matrix)

print("Sum of all values in a Matrix :",np.sum(matrix))

OUTPUT:

Graphical user interface

Description automatically generated

Practical-11

Objective:

Create various type of plot/charts like histograms, plot based on sine/cosine function based on data from a matrix. Further label different axes in a plot and data in a plot.

Tools Used:

* NumPy
* Jupyter Notebook

Program:

* Plot of sine of a matrix

x=np.array([[1,2,3],[4,5,6],[7,8,9]])

y=np.sin(x)

plt.plot(x,y)

plt.xlabel('X-axis(matrix)')

plt.ylabel('Y-axis(sin-plot)')

plt.title('Sine plot of matrix')

plt.show()

OUTPUT:

Graphical user interface, application

Description automatically generated

* Plot of cosine of a matrix

x2=np.array([[ 6, 7, 8],

[ 9, 10, 11],

[12, 13, 14]])

y2=np.cos(x2)

plt.plot(x2,y2)

plt.xlabel('X-axis(matrix)')

plt.ylabel('Y-axis(cos-plot)')

plt.title('Cosine plot of matrix')

plt.show()

OUTPUT:

Graphical user interface, application

Description automatically generated

* Plot of histogram of an array

#hist plots

from random import sample

data = sample(range(1,1000),100)

plt.hist(data)

plt.xlabel('X-axis')

plt.ylabel('y-axis')

plt.title('Histogram')

plt.show()

OUTPUT

Graphical user interface, application

Description automatically generated

Practical-12

Objective:

Perform vectorized implementation of simple matrix operation like finding the transpose of a matrix, adding, subtracting or multiplying two matrices.

Tools Used:

* NumPy
* Jupyter Notebook

Program:

* Vectorized Implementation of simple matrix for Adding Matrices

import numpy as np

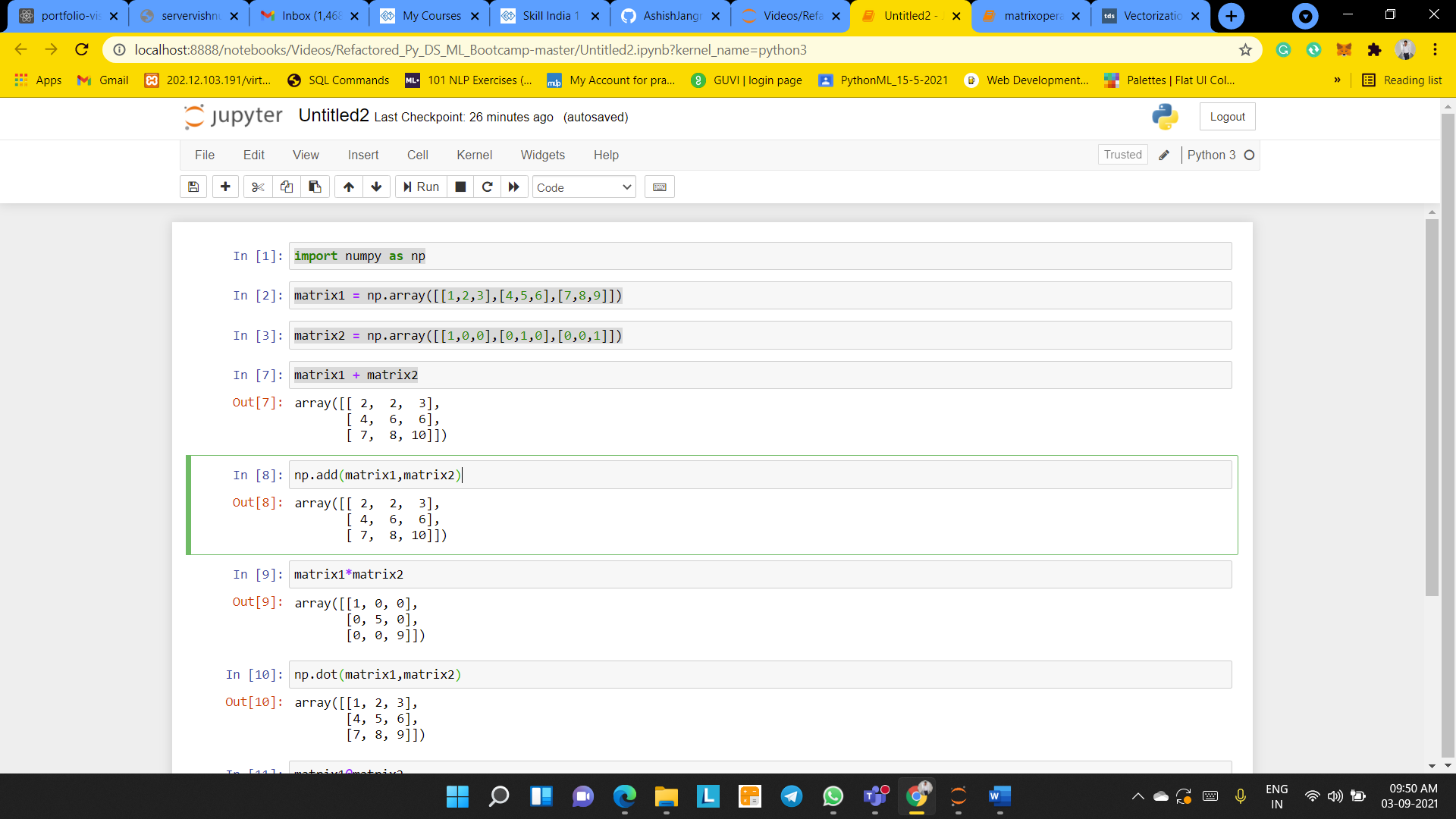
matrix1 = np.array([[1,2,3],[4,5,6],[7,8,9]])

matrix2 = np.array([[1,0,0],[0,1,0],[0,0,1]])

matrix1 + matrix2

np.add(matrix1,matrix2)

OUTPUT:



* Vectorized Implementation of simple matrix for Multiplication Matrices

matrix1\*matrix2

np.dot(matrix1,matrix2)

matrix1@matrix2

OUTPUT:

Graphical user interface, text, application, email

Description automatically generated

* Vectorized Implementation of row and column Matrices

matrix\_row=np.array([[1,2,3]])

matrix\_row.shape

matrix\_column=np.array([[1],[2],[3]])

matrix\_column.shape

OUTPUT:

Graphical user interface, text, application, email

Description automatically generated

* Vectorized Implementation for Matrix slicing.

matrix1[0,0]

matrix1[2,2]

matrix1[:]

matrix1[:2,:2]

matrix1[1:2,:3]

matrix1[:,1:2]

#first row of array

matrix1[0,:]

#second row of array

matrix1[:,1]

OUTPUT:

Graphical user interface, text, application, email

Description automatically generated

* Vectorized Implementation for Transposing Matrix.

matrix1.T

OUTPUT:

Graphical user interface, text, application, email

Description automatically generated