**MCA II year III Semester Course Code IT 31L – Practical’s Part - B**

**Knowledge Representation, Artificial Intelligence, Machine Learning and Deep Learning**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr.**  **No.** | **Program Title** | **Page No.** | **Remark** | **Faculty Sign** |
| **1** | Find the correlation matrix. |  |  |  |
| **2** | Plot the correlation plot on dataset and visualize giving an overview of relationships among data on iris data. |  |  |  |
| **3** | Analysis of covariance: variance (ANOVA), if data have categorical variables on iris data. |  |  |  |
| **4** | Apply linear regression Model techniques to predict the data on any dataset. |  |  |  |
| **5** | Apply logical regression Model techniques to predict the data on any dataset |  |  |  |
| **6** | Clustering algorithms for unsupervised classification |  |  |  |
| **7** | Association algorithms for supervised classification on any dataset |  |  |  |
| **8** | Developing and implementing Decision Tree model on the dataset |  |  |  |
| **9** | Bayesian classification on any dataset |  |  |  |
| **10** | SVM classification on any dataset |  |  |  |
| **11** | Text Mining algorithms on unstructured dataset |  |  |  |
| **12** | Plot the cluster data using python visualizations. |  |  |  |
| **13** | Creating & Visualizing Neural Network for the given data. (Use python) |  |  |  |
| **14** | Recognize optical character using ANN. |  |  |  |
| **15** | Write a program to implement CNN |  |  |  |
| **16** | Write a program to implement CNN |  |  |  |
| **17** | Write a program to implement GAN |  |  |  |
| **18** | Web scraping experiments (by using tools) |  |  |  |

1. **Find the correlation matrix.**

Code: -

import scipy.stats as st import pandas as pd import numpy as np

import matplotlib.pyplot as plt import seaborn as sns

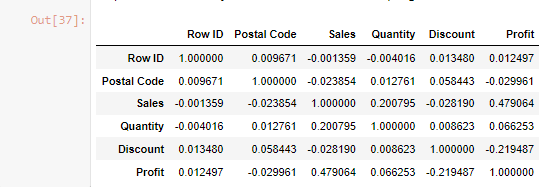
%matplotlib inline superdata=pd.read\_excel("superarketstore.xls")

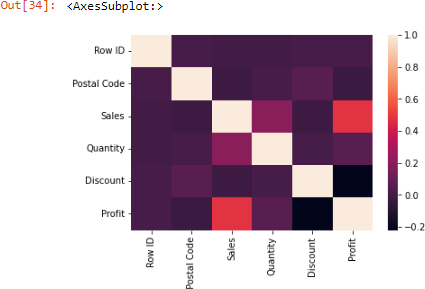
!pip install xlrd

np.corrcoef(superdata ['Sales'], superdata ['Profit']) superdata.corr()

sns.heatmap(superdata.corr())

Output: -





# Plot the correlation plot on dataset and visualize giving an overview of relationships among data on iris data.

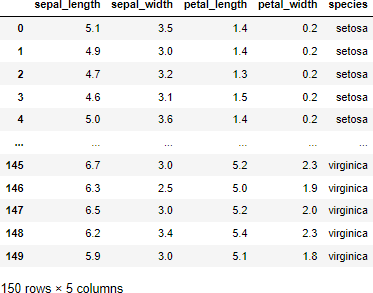
Code: -

import pandas as pd import numpy as np

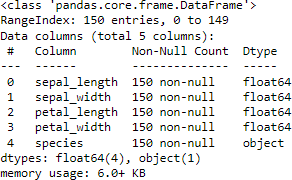
import matplotlib.pyplot as plt

%matplotlib inline import seaborn as sns

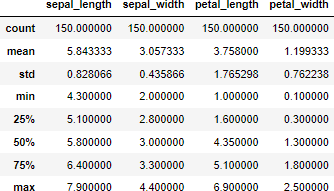
from sklearn import metrics sns.set() iris\_data=pd.read\_csv('iris.csv') iris\_data



iris\_data.info()



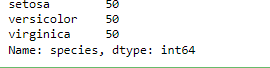
iris\_data.describe()



iris\_data[iris\_data.duplicated()]



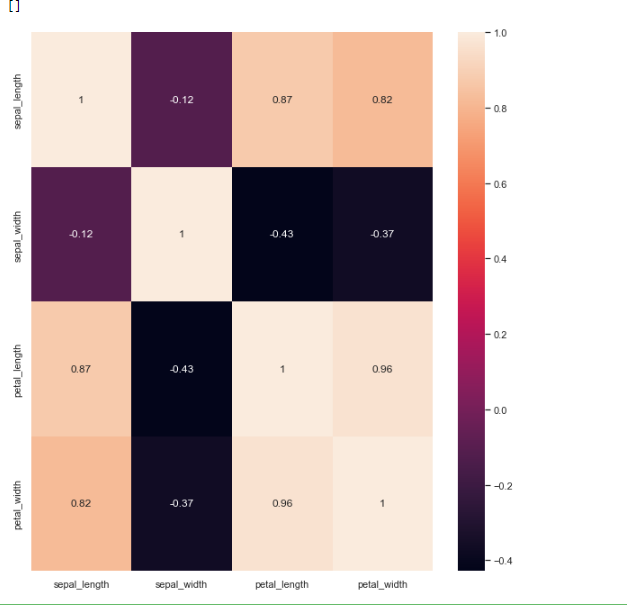
iris\_data['species'].value\_counts()



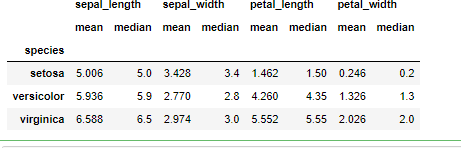
sns.pairplot(iris\_data,hue='species',height=4)



plt.figure(figsize=(10,11)) sns.heatmap(iris\_data.corr(),annot=True) plt.plot()



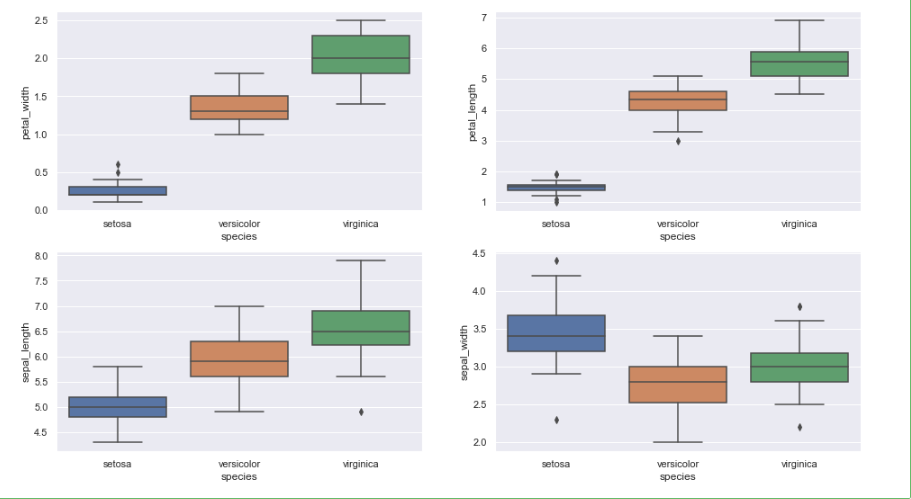
iris\_data.groupby('species').agg(['mean','median'])



fig, axes = plt.subplots(2, 2, figsize=(16,9))

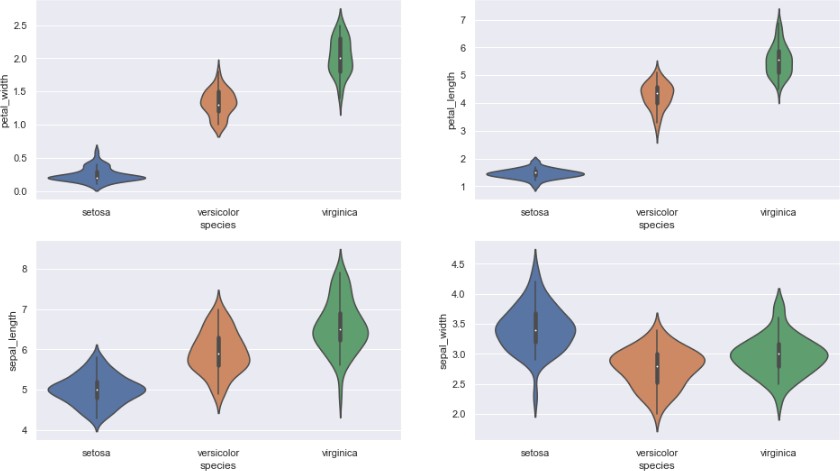
sns.boxplot( y='petal\_width', x= 'species', data=iris\_data, orient='v' , ax=axes[0, 0]) sns.boxplot( y='petal\_length', x= 'species', data=iris\_data, orient='v' , ax=axes[0, 1]) sns.boxplot( y='sepal\_length', x= 'species', data=iris\_data, orient='v' , ax=axes[1, 0])

sns.boxplot( y='sepal\_width', x= 'species', data=iris\_data, orient='v', ax=axes[1, 1]) plt.show()



fig, axes = plt.subplots(2, 2, figsize=(16,9))

sns.violinplot( y='petal\_width', x= 'species', data=iris\_data, orient='v' , ax=axes[0, 0]) sns.violinplot( y='petal\_length', x= 'species', data=iris\_data, orient='v' , ax=axes[0, 1]) sns.violinplot( y='sepal\_length', x= 'species', data=iris\_data, orient='v' , ax=axes[1, 0]) sns.violinplot( y='sepal\_width', x= 'species', data=iris\_data, orient='v', ax=axes[1, 1]) plt.show()

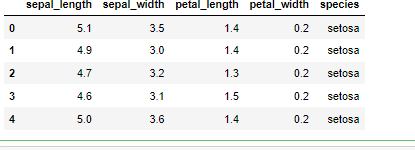


# Analysis of covariance: variance (ANOVA), if data have categorical variables on iris data.

Code: -

import numpy as np import pandas as pd

df=pd.read\_csv('iris\_data.csv') df.head()



# Apply linear regression Model techniques to predict the data on any dataset.

Code: -

import numpy as np

import matplotlib.pyplot as plt import pandas as pd data=pd.read\_csv('Salary\_Data.csv') X=data.iloc[:,:-1].values y=data.iloc[:,1].values

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

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y,test\_size=0.25,random\_state=0) from sklearn.preprocessing import StandardScaler

scaler=StandardScaler() X\_train=scaler.fit\_transform(X\_train) X\_test=scaler.fit\_transform(X\_test)

from sklearn.linear\_model import LinearRegression regressor=LinearRegression() regressor.fit(X\_train,y\_train) y\_pre=regressor.predict(X\_test[[0]])

y\_pre

Output: -



# Apply logical regression Model techniques to predict the data on any dataset.

Code: -

import pandas as pd

import matplotlib.pyplot as plt df=pd.read\_csv('Social\_Network\_Ads.csv') X=df[['Age','EstimatedSalary']] y=df['Purchased']

from sklearn.linear\_model import LogisticRegression model=LogisticRegression()

model.fit(X,y)



Scaled\_Age=(df['Age']-df['Age'].min()) / (df['Age'].max()-df['Age'].min()) Scaled\_Salary=(df['EstimatedSalary']-df['EstimatedSalary'].min()) / (df['EstimatedSalary'].max()- df['EstimatedSalary'].min())

X=pd.concat([Scaled\_Age,Scaled\_Salary],axis=1) y=df['Purchased']

model\_scaled = LogisticRegression() model\_scaled.fit(X,y)



def get\_scaled(pt): age,sal = pt[0],pt[1]

sc\_age=(age-df['Age'].min()) / (df['Age'].max()-df['Age'].min())

sc\_sal=(sal-df['EstimatedSalary'].min()) / (df['EstimatedSalary'].max()-df['EstimatedSalary'].min()) return sc\_age,sc\_sal

q1=get\_scaled([52,130000]) q2=get\_scaled([25,40000]) model\_scaled.predict([q1])



model\_scaled.predict([q2])



from sklearn.preprocessing import MinMaxScaler

X = df[['Age','EstimatedSalary']] scaler = MinMaxScaler() scaler.fit(X)

X\_scaled = scaler.transform(X)

X\_scaled



model = LogisticRegression() model.fit(X\_scaled,df['Purchased'])

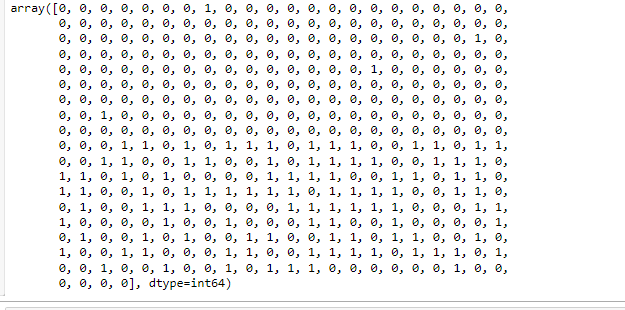


model.score(X\_scaled,df['Purchased'])



y\_pre=model.predict(X\_scaled) y\_act=df['Purchased']

y\_pre



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

X\_train, X\_test, y\_train, y\_test=train\_test\_split(X,y,test\_size=0.25) scaler = MinMaxScaler()

scaler.fit(X\_train) X\_train\_scaled=scaler.transform(X\_train) model = LogisticRegression() model.fit(X\_train\_scaled,y\_train)



train\_score=model.score(X\_train\_scaled,y\_train) train\_score



X\_test\_scaled=scaler.transform(X\_test) test\_score=model.score(X\_test\_scaled,y\_test) test\_score

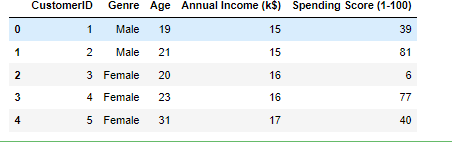
# Clustering algorithms for unsupervised classification.

Code: -

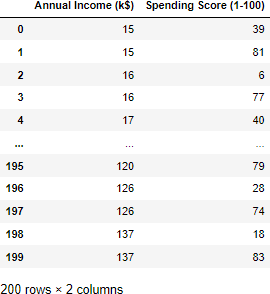
import pandas as pd

import matplotlib.pyplot as plt

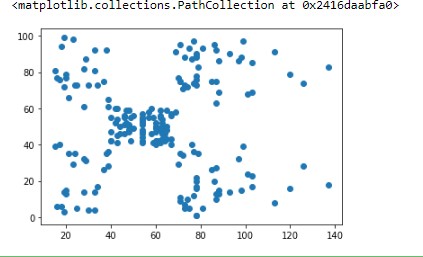
%matplotlib inline df=pd.read\_csv('Mall\_Customers\_dataset.csv') df.head()



X = df[['Annual Income (k$)','Spending Score (1-100)']] X



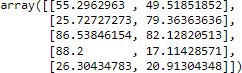
plt.scatter(X['Annual Income (k$)'],X['Spending Score (1-100)'])



from sklearn.cluster import KMeans model = KMeans(n\_clusters=5) model.fit(X)



model.cluster\_centers\_

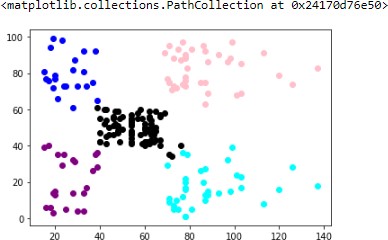


cluster\_number = model.predict(X) len(cluster\_number)



c0 = X[cluster\_number==0] c1 = X[cluster\_number==1] c2 = X[cluster\_number==2] c3 = X[cluster\_number==3] c4 = X[cluster\_number==4]

plt.scatter(c0['Annual Income (k$)'],c0['Spending Score (1-100)'],c='black') plt.scatter(c1['Annual Income (k$)'],c1['Spending Score (1-100)'],c='blue') plt.scatter(c2['Annual Income (k$)'],c2['Spending Score (1-100)'],c='pink') plt.scatter(c3['Annual Income (k$)'],c3['Spending Score (1-100)'],c='cyan') plt.scatter(c4['Annual Income (k$)'],c4['Spending Score (1-100)'],c='purple')



model.inertia\_

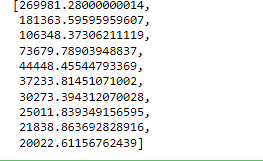


WCSS =[]

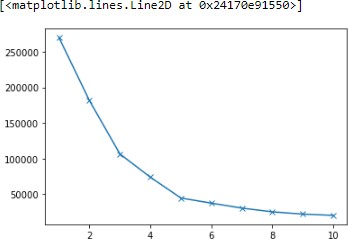
for i in range(1,11):

model = KMeans(n\_clusters=i) model.fit(X) WCSS.append(model.inertia\_)

WCSS



plt.plot(range(1,11),WCSS,marker = 'x')



# Association algorithms for supervised classification on any dataset.

Code: -

import numpy as np import pandas as pd

import matplotlib.pyplot as plt import scipy.stats as stats np.random.seed(12)

races = ["asian","black","hispanic","other","white"] voter\_race = np.random.choice(a= races,

p = [0.05, 0.15 ,0.25, 0.05, 0.5],

size=1000)

voter\_age = stats.poisson.rvs(loc=18,

mu=30, size=1000)

voter\_frame = pd.DataFrame({"race":voter\_race,"age":voter\_age}) groups = voter\_frame.groupby("race").groups

asian = voter\_age[groups["asian"]] black = voter\_age[groups["black"]]

hispanic = voter\_age[groups["hispanic"]] other = voter\_age[groups["other"]]

white = voter\_age[groups["white"]]

stats.f\_oneway(asian, black, hispanic, other, white)



import statsmodels.api as sm

from statsmodels.formula.api import ols

model = ols('age ~ race',

data = voter\_frame).fit()

anova\_result = sm.stats.anova\_lm(model, typ=2) print (anova\_result)



np.random.seed(12)

voter\_race = np.random.choice(a= races,

p = [0.05, 0.15 ,0.25, 0.05, 0.5],

size=1000)

white\_ages = stats.poisson.rvs(loc=18,

mu=32, size=1000)

voter\_age = stats.poisson.rvs(loc=18,

mu=30, size=1000)

voter\_age = np.where(voter\_race=="white", white\_ages, voter\_age)

voter\_frame = pd.DataFrame({"race":voter\_race,"age":voter\_age}) groups = voter\_frame.groupby("race").groups

asian = voter\_age[groups["asian"]] black = voter\_age[groups["black"]]

hispanic = voter\_age[groups["hispanic"]] other = voter\_age[groups["other"]]

white = voter\_age[groups["white"]]

stats.f\_oneway(asian, black, hispanic, other, white)



model = ols('age ~ race', data = voter\_frame).fit()

anova\_result = sm.stats.anova\_lm(model, typ=2) print (anova\_result)



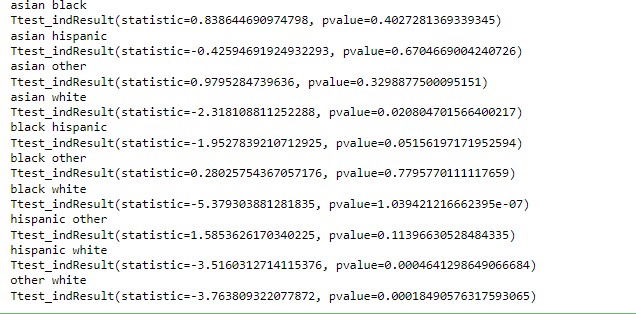
race\_pairs = []

for race1 in range(4):

for race2 in range(race1+1,5): race\_pairs.append((races[race1], races[race2]))

for race1, race2 in race\_pairs: print(race1, race2)

print(stats.ttest\_ind(voter\_age[groups[race1]], voter\_age[groups[race2]]))



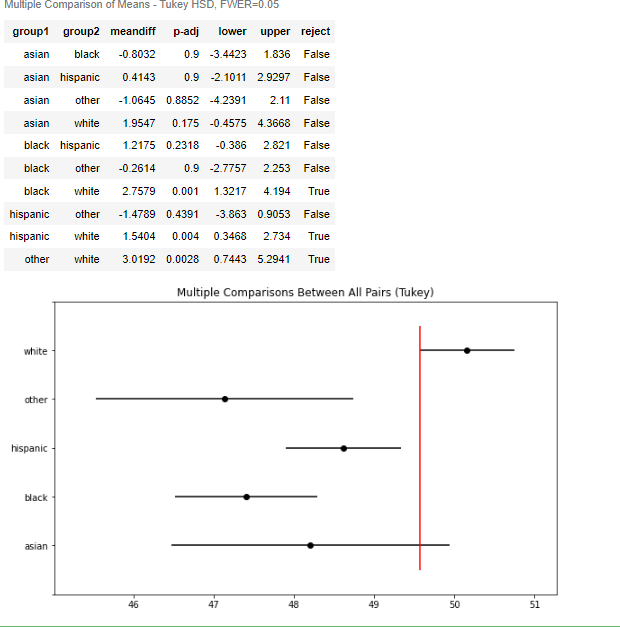
from statsmodels.stats.multicomp import pairwise\_tukeyhsd

tukey = pairwise\_tukeyhsd(endog=voter\_age, groups=voter\_race, alpha=0.05)

tukey.plot\_simultaneous()

plt.vlines(x=49.57,ymin=-0.5,ymax=4.5, color="red")

tukey.summary()



# Developing and implementing Decision Tree model on the dataset

Code: -

import numpy as np

import matplotlib.pyplot as plt import pandas as pd data=pd.read\_csv('Salary\_Data.csv') data.head()



X=data[['YearsExperience']] y=data['Salary']

from sklearn.tree import DecisionTreeRegressor regressor = DecisionTreeRegressor(random\_state=0) regressor.fit(X,y)



regressor.predict([[6.5]])



# Bayesian classification on any dataset.

Code: -

import numpy as np import pandas as pd

import matplotlib.pyplot as plt df=pd.read\_csv('iris\_data.csv')

df.columns=['sepal\_length','sepal\_width','petal\_length','petal\_width','species'] col\_names=list(df.columns)

predictors=col\_names[0:4] target=col\_names[4]

from sklearn.model\_selection import train\_test\_split train,test=train\_test\_split(df,test\_size=0.3,random\_state=0)

from sklearn.naive\_bayes import GaussianNB Gmodel=GaussianNB() Gmodel.fit(train[predictors],train[target]) train\_Gpred=Gmodel.predict(train[predictors]) test\_Gpred=Gmodel.predict(test[predictors])

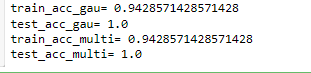
train\_acc\_gau=np.mean(train\_Gpred==train[target]) test\_acc\_gau=np.mean(test\_Gpred==test[target]) print ("train\_acc\_gau=",train\_acc\_gau)

print ("test\_acc\_gau=",test\_acc\_gau)

from sklearn.naive\_bayes import MultinomialNB Mmodel=MultinomialNB() Mmodel.fit(train[predictors],train[target]) train\_Mpred=Mmodel.predict(train[predictors]) test\_Mpred=Mmodel.predict(test[predictors])

train\_acc\_multi=np.mean(train\_Mpred==train[target]) test\_acc\_multi=np.mean(test\_Mpred==test[target])

print ("train\_acc\_multi=",train\_acc\_gau) print ("test\_acc\_multi=",test\_acc\_gau)



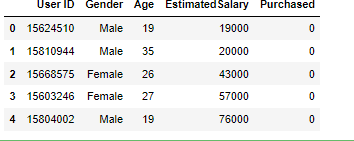
# SVM classification on any dataset

Code: -

import pandas as pd

import matplotlib.pyplot as plt

%matplotlib inline df=pd.read\_csv('Social\_Network\_Ads.csv') df.head()



X=df[['Age','EstimatedSalary']] y=df['Purchased']

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

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.23, random\_state=91)

from sklearn.preprocessing import MinMaxScaler scaler=MinMaxScaler()

scaler.fit(X\_train) X\_train\_scaled=scaler.transform(X\_train) X\_test\_scaled=scaler.transform(X\_test)

from sklearn.svm import SVC model\_lin = SVC(kernel='linear') model\_lin.fit(X\_train\_scaled,y\_train) model\_lin.score(X\_test\_scaled,y\_test)



model\_poly = SVC(kernel='poly') model\_poly.fit(X\_train\_scaled,y\_train) model\_poly.score(X\_test\_scaled,y\_test)

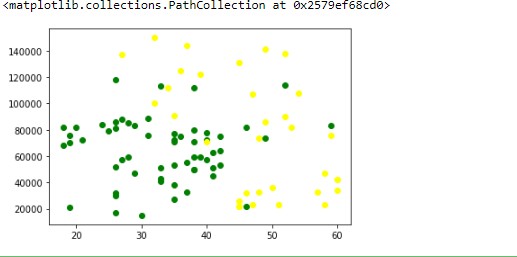


model\_rbf = SVC(kernel='rbf') model\_rbf.fit(X\_train\_scaled,y\_train) model\_rbf.score(X\_test\_scaled,y\_test)



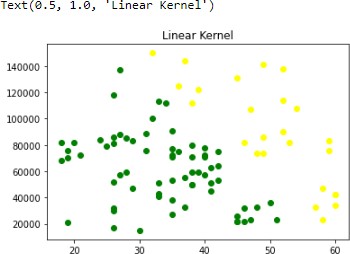
class\_0\_act = X\_test[y\_test==0] class\_1\_act = X\_test[y\_test==1]

plt.scatter(class\_0\_act['Age'],class\_0\_act['EstimatedSalary'],c='green') plt.scatter(class\_1\_act['Age'],class\_1\_act['EstimatedSalary'],c='yellow')



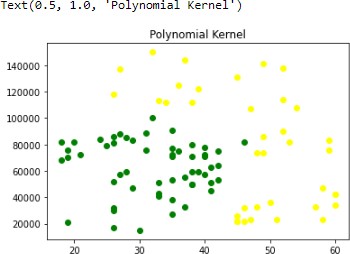
y\_pre = model\_lin.predict(X\_test\_scaled) class\_0\_pre = X\_test[y\_pre==0] class\_1\_pre = X\_test[y\_pre==1]

plt.scatter(class\_0\_pre['Age'],class\_0\_pre['EstimatedSalary'],c='green') plt.scatter(class\_1\_pre['Age'],class\_1\_pre['EstimatedSalary'],c='yellow') plt.title('Linear Kernel')



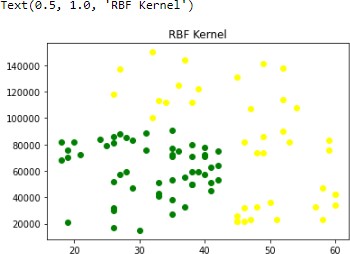
y\_pre = model\_poly.predict(X\_test\_scaled) class\_0\_pre = X\_test[y\_pre==0] class\_1\_pre = X\_test[y\_pre==1]

plt.scatter(class\_0\_pre['Age'],class\_0\_pre['EstimatedSalary'],c='green') plt.scatter(class\_1\_pre['Age'],class\_1\_pre['EstimatedSalary'],c='yellow') plt.title('Polynomial Kernel')



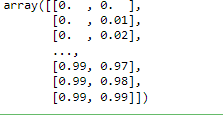
y\_pre = model\_rbf.predict(X\_test\_scaled) class\_0\_pre = X\_test[y\_pre==0] class\_1\_pre = X\_test[y\_pre==1]

plt.scatter(class\_0\_pre['Age'],class\_0\_pre['EstimatedSalary'],c='green') plt.scatter(class\_1\_pre['Age'],class\_1\_pre['EstimatedSalary'],c='yellow') plt.title('RBF Kernel')



import numpy as np plot\_data = []

for x in range(0,100,1): for y in range(0,100,1):

plot\_data.append([x,y]) plot\_data=np.array(plot\_data)/100 plot\_data

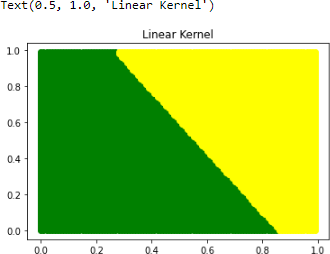
plot\_data.shape



y\_plot = model\_lin.predict(plot\_data) class\_0 = plot\_data[y\_plot==0] class\_1 = plot\_data[y\_plot==1]

plt.scatter(class\_0[:,0],class\_0[:,1],c='green')

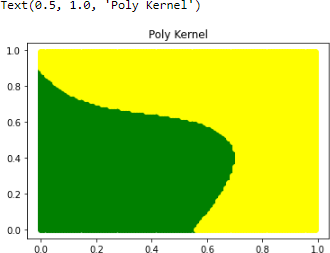
plt.scatter(class\_1[:,0],class\_1[:,1],c='yellow') plt.title('Linear Kernel')



y\_plot = model\_poly.predict(plot\_data) class\_0 = plot\_data[y\_plot==0]

class\_1 = plot\_data[y\_plot==1] plt.scatter(class\_0[:,0],class\_0[:,1],c='green')

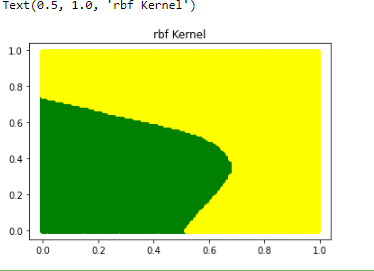
plt.scatter(class\_1[:,0],class\_1[:,1],c='yellow') plt.title('Poly Kernel')



y\_plot = model\_rbf.predict(plot\_data) class\_0 = plot\_data[y\_plot==0] class\_1 = plot\_data[y\_plot==1]

plt.scatter(class\_0[:,0],class\_0[:,1],c='green')

plt.scatter(class\_1[:,0],class\_1[:,1],c='yellow') plt.title('rbf Kernel')



pts = np.array([[25,60000],[50,120000]])

pts\_scaled = scaler.transform(pts) pts\_scaled



y = model\_rbf.predict(pts\_scaled) y

# Text Mining algorithms on unstructured dataset

Code: -

from sklearn.datasets import load\_digits from sklearn.decomposition import PCA from sklearn.cluster import KMeans import numpy as np

data = load\_digits().data pca = PCA(2)

df = pca.fit\_transform(data) df.shape



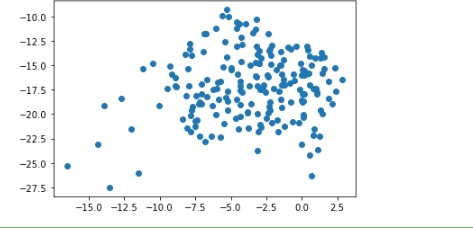
from sklearn.cluster import KMeans kmeans = KMeans(n\_clusters= 10) label = kmeans.fit\_predict(df)

print(label)



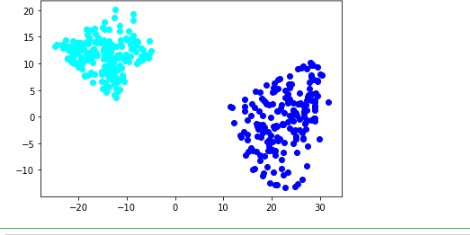
import matplotlib.pyplot as plt filtered\_label0 = df[label == 0]

plt.scatter(filtered\_label0[:,0] , filtered\_label0[:,1]) plt.show()



filtered\_label2 = df[label == 2] filtered\_label8 = df[label == 8]

plt.scatter(filtered\_label2[:,0] , filtered\_label2[:,1] , color = 'cyan') plt.scatter(filtered\_label8[:,0] , filtered\_label8[:,1] , color = 'blue') plt.show()

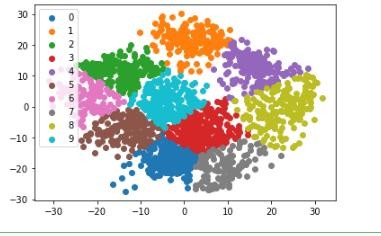


u\_labels = np.unique(label)

for i in u\_labels:

plt.scatter(df[label == i , 0] , df[label == i , 1] , label = i) plt.legend()

plt.show()

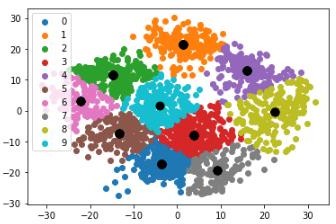


centroids = kmeans.cluster\_centers\_ u\_labels = np.unique(label)

for i in u\_labels:

plt.scatter(df[label == i , 0] , df[label == i , 1] , label = i) plt.scatter(centroids[:,0] , centroids[:,1] , s = 80, color = 'k')

plt.legend() plt.show()



# . Plot the cluster data using python visualizations.

Code: -

import tensorflow as tf

from tensorflow import keras from matplotlib.pyplot import title

from tensorflow.keras.models import Sequential,Model from tensorflow.keras.layers import Dense, Dropout, Flatten

from tensorflow.keras.layers import Conv2D, MaxPooling2D from tensorflow.keras.layers import LeakyReLU

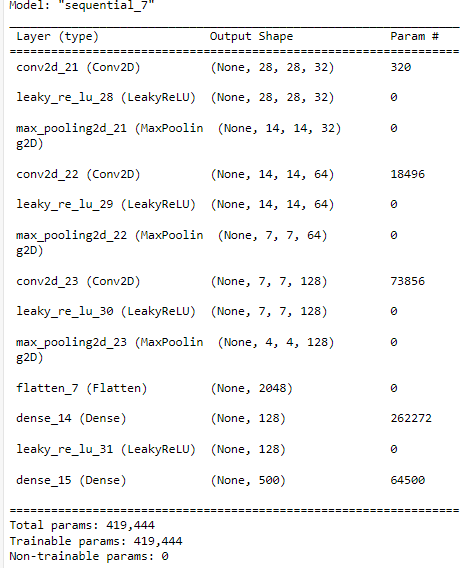
model = Sequential()

model.add(Conv2D(32, kernel\_size=(3, 3),activation='linear',input\_shape=(28,28,1),padding='same')) model.add(LeakyReLU(alpha=0.1))

model.add(MaxPooling2D((2, 2),padding='same')) model.add(Conv2D(64, (3, 3), activation='linear',padding='same')) model.add(LeakyReLU(alpha=0.1)) model.add(MaxPooling2D(pool\_size=(2, 2),padding='same')) model.add(Conv2D(128, (3, 3), activation='linear',padding='same')) model.add(LeakyReLU(alpha=0.1)) model.add(MaxPooling2D(pool\_size=(2, 2),padding='same')) model.add(Flatten())

model.add(Dense(128, activation='linear')) model.add(LeakyReLU(alpha=0.1)) model.add(Dense(500, activation='softmax'))

model.compile(loss=keras.losses.categorical\_crossentropy, optimizer=keras.optimizers.Adam(),metrics=['accuracy']) model.summary()



# Creating & Visualizing Neural Network for the given data. (Use python)

Code: -

1. Recognize optical character using ANN. Code: -

from tensorflow.keras.datasets import mnist (x\_train,y\_train),(x\_test,y\_test)=mnist.load\_data() x\_train.shape



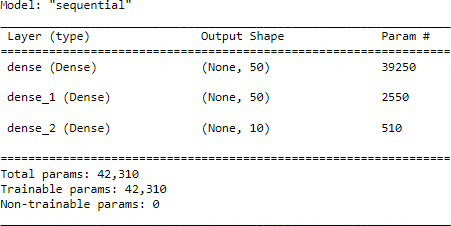
X\_train=x\_train.reshape(60000,784) X\_test=x\_test.reshape(10000,784)

from tensorflow.keras.utils import to\_categorical y\_train=to\_categorical(y\_train,num\_classes=10) y\_test=to\_categorical(y\_test,num\_classes=10) X\_train=X\_train/255

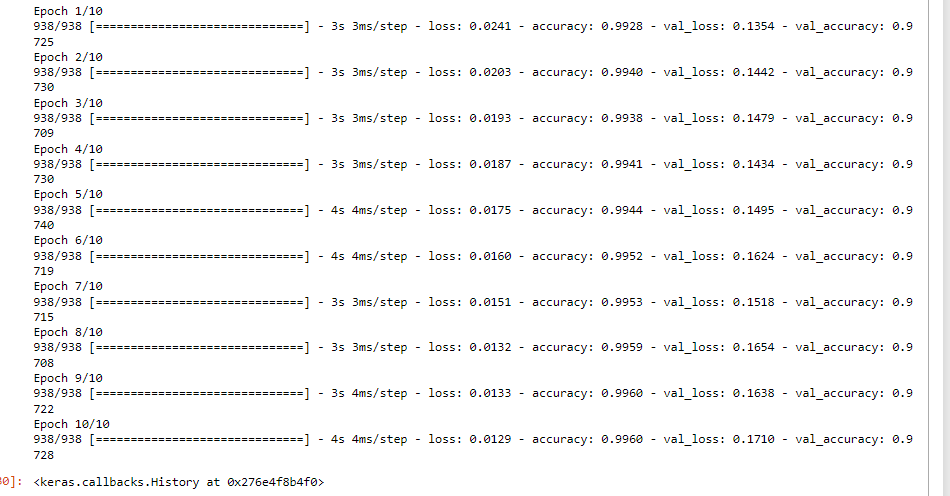
X\_test=X\_test/255

from tensorflow.keras.models import Sequential from tensorflow.keras.layers import Dense model=Sequential()

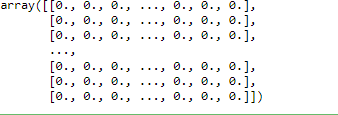
model.add(Dense(50,activation='relu',input\_shape=(784,))) model.add(Dense(50,activation='relu')) model.add(Dense(10,activation='softmax')) model.summary()



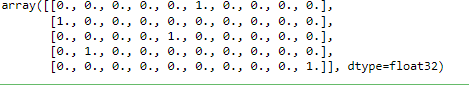
model.compile(loss='categorical\_crossentropy',metrics=['accuracy']) model.fit(X\_train,y\_train,batch\_size=64,epochs=10,validation\_data=(X\_test,y\_test))



import numpy as np X\_train



y\_train[:5,:]



img0 = np.array(X\_train[0]).reshape(1,784) model.predict(img0).argmax()



def recognise(img):

img=np.array(img).reshape(1,784) return model.predict(img).argmax()

y\_pre=model.predict(X\_test).argmax(axis=1) y\_pre



len(y\_pre)



y\_test.argmax(axis=1)



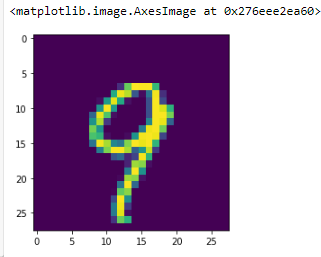
sum(y\_pre==y\_test.argmax(axis=1))



9737/10000



import matplotlib.pyplot as plt plt.imshow(np.array(X\_test[560]).reshape(28,28))



recognise(X\_test[560])



# Write a program to implement CNN

Code: -

import numpy as np import pandas as pd import os

for dirname, \_, filenames in os.walk('/kaggle/input'): for filename in filenames:

print(os.path.join(dirname, filename)) os.listdir('/kaggle/input/dogs-vs-cats/') filenames=os.listdir('../input/dogs-vs-cats/train/train') len(filenames)

filenames[:5] df=pd.DataFrame({'filename':filenames}) df.head() df['class']=df['filename'].apply(lambda X:X[:3]) df.head()

from tensorflow.keras.preprocessing.image import ImageDataGenerator data\_gen=ImageDataGenerator(zoom\_range=0.2,shear\_range=0.2,horizontal\_flip=True,rescale=1/255) train\_data=data\_gen.flow\_from\_dataframe(df,'../input/dogs-vs- cats/train/train',X='filename',y='class',target\_size=(224,224))

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv2D,MaxPool2D,Flatten,Dense model=Sequential() model.add(Conv2D(16,(3,3),activation='relu',input\_shape=(224,224,3))) model.add(MaxPool2D()) model.add(Conv2D(32,(3,3),activation='relu')) model.add(MaxPool2D()) model.add(Conv2D(64,(3,3),activation='relu')) model.add(MaxPool2D()) model.add(Conv2D(64,(5,5),activation='relu')) model.add(MaxPool2D()) model.add(Conv2D(128,(3,3),activation='relu')) model.add(MaxPool2D())

model.add(Flatten())

model.add(Dense(2,activation='softmax')) model.summary()

model.compile(optimizer='adam',loss='categorical\_crossentropy',metrics=['accuracy']) model.fit\_generator(train\_data,epochs=5)

import cv2

def get\_class(img\_path): img=cv2.imread(img\_path) img=cv2.resize(img,(224,224)) img=img/255

op=model.predict(img.reshape(1,224,224,3)).argmax() return 'cat' if op==0 else 'dog'

train\_data.class\_mode

get\_class('../input/dogs-vs-cats/train/train/cat.10002.jpg')

# Write a program to implement RNN

Code: -

from tensorflow.keras.datasets import imdb (X\_train,y\_train),(X\_test,y\_test)=imdb.load\_data(num\_words=20000) X\_train.shape,X\_test.shape



len(X\_train[0]),len(X\_train[1]),len(X\_train[2]),len(X\_train[3]),len(X\_train[4])



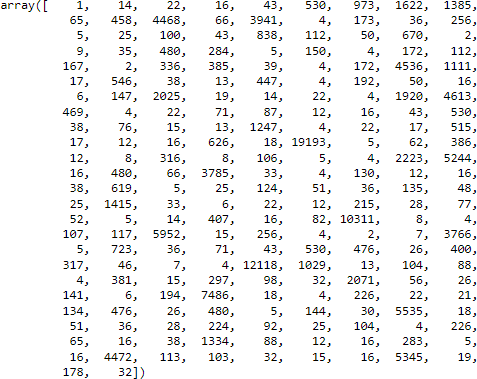
y\_train[:5]



X\_train[0]



import numpy as np np.array(X\_train[0])



from tensorflow.keras.preprocessing.sequence import pad\_sequences X=pad\_sequences(X\_train,maxlen=200) X\_val=pad\_sequences(X\_test,maxlen=200)

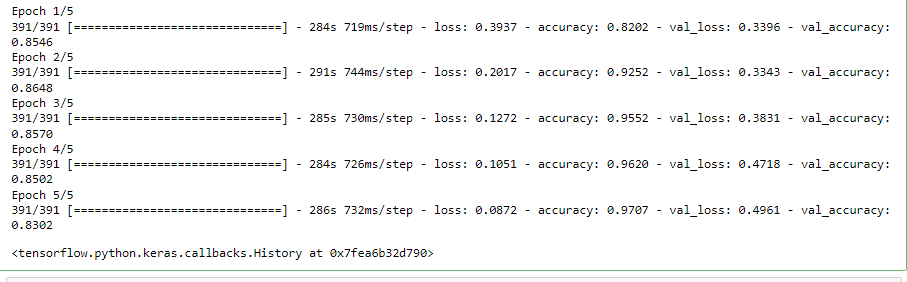
len(X[0])



from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import LSTM,Dense,Embedding model=Sequential() model.add(Embedding(20000,128,input\_shape=(200,))) model.add(LSTM(100,return\_sequences=True)) model.add(LSTM(100)) model.add(Dense(1,activation='sigmoid'))

model.compile(loss='binary\_crossentropy',optimizer='adam',metrics=['accuracy']) model.fit(X,y\_train,validation\_data=(X\_val,y\_test),epochs=5,batch\_size=64)



# Write a program to implement GAN

Code: - import os

print(os.listdir("../input"))

from future import print\_function import time

import torch

import torch.nn as nn import torch.nn.parallel import torch.optim as optim import torch.utils.data

import torchvision.datasets as dset

import torchvision.transforms as transforms import torchvision.utils as vutils

from torch.autograd import Variable import matplotlib.pyplot as plt import numpy as np

from torch import nn, optim import torch.nn.functional as F

from torchvision import datasets, transforms from torchvision.utils import save\_image import matplotlib.pyplot as plt

import matplotlib.image as mpimg

from tqdm import tqdm\_notebook as tqdm

PATH = '../input/all-dogs/all-dogs/' images = os.listdir(PATH)

print(f'There are {len(os.listdir(PATH))} pictures of dogs.')

fig, axes = plt.subplots(nrows=3, ncols=3, figsize=(12,10)) for indx, axis in enumerate(axes.flatten()):

rnd\_indx = np.random.randint(0, len(os.listdir(PATH))) img = plt.imread(PATH + images[rnd\_indx])

imgplot = axis.imshow(img) axis.set\_title(images[rnd\_indx]) axis.set\_axis\_off()

plt.tight\_layout(rect=[0, 0.03, 1, 0.95])

batch\_size = 32

image\_size = 64

random\_transforms = [transforms.ColorJitter(), transforms.RandomRotation(degrees=20)] transform = transforms.Compose([transforms.Resize(64),

transforms.CenterCrop(64), transforms.RandomHorizontalFlip(p=0.5), transforms.RandomApply(random\_transforms, p=0.2), transforms.ToTensor(),

transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

train\_data = datasets.ImageFolder('../input/all-dogs/', transform=transform) train\_loader = torch.utils.data.DataLoader(train\_data, shuffle=True,

batch\_size=batch\_size)

imgs, label = next(iter(train\_loader)) imgs = imgs.numpy().transpose(0, 2, 3, 1)

for i in range(5): plt.imshow(imgs[i]) plt.show()

def weights\_init(m): """

Takes as input a neural network m that will initialize all its weights. """

classname = m. class . name if classname.find('Conv') != -1:

m.weight.data.normal\_(0.0, 0.02) elif classname.find('BatchNorm') != -1:

m.weight.data.normal\_(1.0, 0.02) m.bias.data.fill\_(0)

class G(nn.Module): def init (self):

super(G, self). init () self.main = nn.Sequential(

nn.ConvTranspose2d(100, 512, 4, stride=1, padding=0, bias=False),

nn.BatchNorm2d(512), nn.ReLU(True),

nn.ConvTranspose2d(512, 256, 4, stride=2, padding=1, bias=False), nn.BatchNorm2d(256),

nn.ReLU(True),

nn.ConvTranspose2d(256, 128, 4, stride=2, padding=1, bias=False), nn.BatchNorm2d(128),

nn.ReLU(True),

nn.ConvTranspose2d(128, 64, 4, stride=2, padding=1, bias=False), nn.BatchNorm2d(64),

nn.ReLU(True),

nn.ConvTranspose2d(64, 3, 4, stride=2, padding=1, bias=False), nn.Tanh()

)

def forward(self, input): output = self.main(input) return output

netG = G() netG.apply(weights\_init)

class D(nn.Module): def init (self):

super(D, self). init () self.main = nn.Sequential(

nn.Conv2d(3, 64, 4, stride=2, padding=1, bias=False), nn.LeakyReLU(negative\_slope=0.2, inplace=True), nn.Conv2d(64, 128, 4, stride=2, padding=1, bias=False), nn.BatchNorm2d(128), nn.LeakyReLU(negative\_slope=0.2, inplace=True), nn.Conv2d(128, 256, 4, stride=2, padding=1, bias=False), nn.BatchNorm2d(256), nn.LeakyReLU(negative\_slope=0.2, inplace=True), nn.Conv2d(256, 512, 4, stride=2, padding=1, bias=False), nn.BatchNorm2d(512), nn.LeakyReLU(negative\_slope=0.2, inplace=True), nn.Conv2d(512, 1, 4, stride=1, padding=0, bias=False), nn.Sigmoid()

)

def forward(self, input): output = self.main(input) return output.view(-1)

netD = D() netD.apply(weights\_init)

class Generator(nn.Module):

def init (self, nz=128, channels=3): super(Generator, self). init ()

self.nz = nz self.channels = channels

def convlayer(n\_input, n\_output, k\_size=4, stride=2, padding=0): block = [

nn.ConvTranspose2d(n\_input, n\_output, kernel\_size=k\_size, stride=stride, padding=padding, bias=False),

nn.BatchNorm2d(n\_output), nn.ReLU(inplace=True),

]

return block

self.model = nn.Sequential(

\*convlayer(self.nz, 1024, 4, 1, 0),

\*convlayer(1024, 512, 4, 2, 1),

\*convlayer(512, 256, 4, 2, 1),

\*convlayer(256, 128, 4, 2, 1),

\*convlayer(128, 64, 4, 2, 1),

nn.ConvTranspose2d(64, self.channels, 3, 1, 1), nn.Tanh()

)

def forward(self, z):

z = z.view(-1, self.nz, 1, 1) img = self.model(z)

return img

class Discriminator(nn.Module): def init (self, channels=3):

super(Discriminator, self). init () self.channels = channels

def convlayer(n\_input, n\_output, k\_size=4, stride=2, padding=0, bn=False):

block = [nn.Conv2d(n\_input, n\_output, kernel\_size=k\_size, stride=stride, padding=padding, bias=False)]

if bn:

block.append(nn.BatchNorm2d(n\_output)) block.append(nn.LeakyReLU(0.2, inplace=True)) return block

self.model = nn.Sequential(

\*convlayer(self.channels, 32, 4, 2, 1),

\*convlayer(32, 64, 4, 2, 1),

\*convlayer(64, 128, 4, 2, 1, bn=True),

\*convlayer(128, 256, 4, 2, 1, bn=True),

nn.Conv2d(256, 1, 4, 1, 0, bias=False),

)

def forward(self, imgs): logits = self.model(imgs) out = torch.sigmoid(logits)

return out.view(-1, 1)

!mkdir results

!ls

EPOCH = 0

LR = 0.001

criterion = nn.BCELoss()

optimizerD = optim.Adam(netD.parameters(), lr=LR, betas=(0.5, 0.999)) optimizerG = optim.Adam(netG.parameters(), lr=LR, betas=(0.5, 0.999))

for epoch in range(EPOCH):

for i, data in enumerate(dataloader, 0):

netD.zero\_grad()

real,\_ = data

input = Variable(real)

target = Variable(torch.ones(input.size()[0])) output = netD(input)

errD\_real = criterion(output, target)

noise = Variable(torch.randn(input.size()[0], 100, 1, 1)) fake = netG(noise)

target = Variable(torch.zeros(input.size()[0])) output = netD(fake.detach())

errD\_fake = criterion(output, target)

errD = errD\_real + errD\_fake errD.backward() optimizerD.step()

netG.zero\_grad()

target = Variable(torch.ones(input.size()[0])) output = netD(fake)

errG = criterion(output, target) errG.backward() optimizerG.step()

print('[%d/%d][%d/%d] Loss\_D: %.4f; Loss\_G: %.4f' % (epoch, EPOCH, i, len(dataloader), errD.item(), errG.item()))

if i % 100 == 0:

vutils.save\_image(real, '%s/real\_samples.png' % "./results", normalize=True) fake = netG(noise)

vutils.save\_image(fake.data, '%s/fake\_samples\_epoch\_%03d.png' % ("./results", epoch), normalize=True)

batch\_size = 32

LR\_G = 0.001

LR\_D = 0.0005

beta1 = 0.5

epochs = 100

real\_label = 0.9

fake\_label = 0

nz = 128

device = torch.device("cuda" if torch.cuda.is\_available() else "cpu") netG = Generator(nz).to(device)

netD = Discriminator().to(device) criterion = nn.BCELoss()

optimizerD = optim.Adam(netD.parameters(), lr=LR\_D, betas=(beta1, 0.999)) optimizerG = optim.Adam(netG.parameters(), lr=LR\_G, betas=(beta1, 0.999))

fixed\_noise = torch.randn(25, nz, 1, 1, device=device) G\_losses = []

D\_losses = [] epoch\_time = []

def plot\_loss (G\_losses, D\_losses, epoch): plt.figure(figsize=(10,5))

plt.title("Generator and Discriminator Loss - EPOCH "+ str(epoch)) plt.plot(G\_losses,label="G")

plt.plot(D\_losses,label="D") plt.xlabel("iterations") plt.ylabel("Loss") plt.legend()

plt.show()

def show\_generated\_img(n\_images=5): sample = []

for \_ in range(n\_images):

noise = torch.randn(1, nz, 1, 1, device=device)

gen\_image = netG(noise).to("cpu").clone().detach().squeeze(0) gen\_image = gen\_image.numpy().transpose(1, 2, 0) sample.append(gen\_image)

figure, axes = plt.subplots(1, len(sample), figsize = (64,64)) for index, axis in enumerate(axes):

axis.axis('off')

image\_array = sample[index] axis.imshow(image\_array)

plt.show() plt.close()

for epoch in range(epochs): start = time.time()

for ii, (real\_images, train\_labels) in tqdm(enumerate(train\_loader), total=len(train\_loader)): netD.zero\_grad()

real\_images = real\_images.to(device) batch\_size = real\_images.size(0)

labels = torch.full((batch\_size, 1), real\_label, device=device)

output = netD(real\_images) errD\_real = criterion(output, labels) errD\_real.backward()

D\_x = output.mean().item()

noise = torch.randn(batch\_size, nz, 1, 1, device=device) fake = netG(noise)

labels.fill\_(fake\_label) output = netD(fake.detach())

errD\_fake = criterion(output, labels) errD\_fake.backward()

D\_G\_z1 = output.mean().item() errD = errD\_real + errD\_fake optimizerD.step()

netG.zero\_grad() labels.fill\_(real\_label) output = netD(fake)

errG = criterion(output, labels) errG.backward()

D\_G\_z2 = output.mean().item() optimizerG.step()

G\_losses.append(errG.item())

D\_losses.append(errD.item())

if (ii+1) % (len(train\_loader)//2) == 0:

print('[%d/%d][%d/%d] Loss\_D: %.4f Loss\_G: %.4f D(x): %.4f D(G(z)): %.4f / %.4f'

% (epoch + 1, epochs, ii+1, len(train\_loader), errD.item(), errG.item(), D\_x, D\_G\_z1, D\_G\_z2))

plot\_loss (G\_losses, D\_losses, epoch) G\_losses = []

D\_losses = []

if epoch % 10 == 0: show\_generated\_img()

epoch\_time.append(time.time()- start)

print (">> average EPOCH duration = ", np.mean(epoch\_time)) show\_generated\_img(7)

if not os.path.exists('../output\_images'): os.mkdir('../output\_images')

im\_batch\_size = 50 n\_images=10000

for i\_batch in tqdm(range(0, n\_images, im\_batch\_size)): gen\_z = torch.randn(im\_batch\_size, nz, 1, 1, device=device) gen\_images = netG(gen\_z)

images = gen\_images.to("cpu").clone().detach() images = images.numpy().transpose(0, 2, 3, 1) for i\_image in range(gen\_images.size(0)):

save\_image(gen\_images[i\_image, :, :, :], os.path.join('../output\_images', f'image\_{i\_batch+i\_image:05d}.png'))

fig = plt.figure(figsize=(25, 16)) for i, j in enumerate(images[:32]):

ax = fig.add\_subplot(4, 8, i + 1, xticks=[], yticks=[]) plt.imshow(j)

import shutil

shutil.make\_archive('images', 'zip', '../output\_images')

torch.save(netG.state\_dict(), 'generator.pth') torch.save(netD.state\_dict(), 'discriminator.pth')

1. Web scraping experiments (by using tools) Code: -

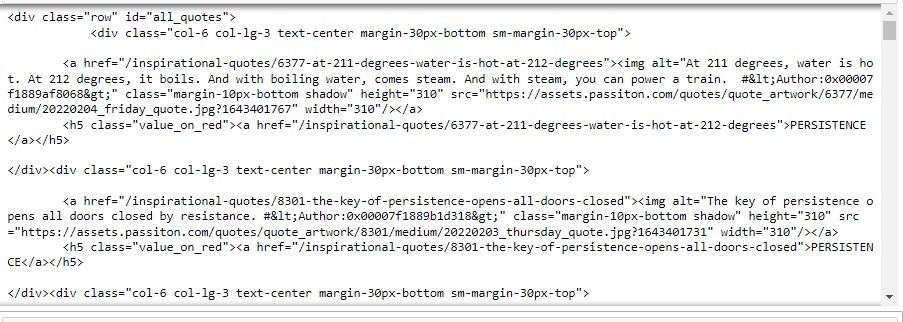
import requests

from bs4 import BeautifulSoup import csv

URL = ["http://www.values.com/inspirationa](http://www.values.com/inspirational-quotes)l[-quotes"](http://www.values.com/inspirational-quotes) r = requests.get(URL)

soup = BeautifulSoup(r.content, 'html5lib') quotes=[]

soup.find('div', attrs = {'id':'all\_quotes'})



for row in table.find\_all\_next('div', attrs = {'class': 'col-6 col-lg-3 text-center margin-30px-bottom sm-margin- 30px-top'}):

quote = {}

quote['theme'] = row.h5.text

quote['url'] = row.a['href']

quote['img'] = row.img['src']

quote['lines'] = row.img['alt'].split(" #")[0]

quote['author'] = row.img['alt'].split(" #")[1] quotes.append(quote)