**Program-1**

# Linear Regression using Abalone dataset using Keras

import os

import numpy as np

import pandas as pd

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

from keras.models import Sequential

from keras.layers import Dense, Dropout,LeakyReLU

from keras import metrics

import matplotlib.pyplot as plt

from keras.optimizers import Adam, RMSprop

from keras.callbacks import TensorBoard

#Dataframe is created using abalone.csv

abalone\_Data = pd.read\_csv('abalone.csv')

# We need to calculate Age , lets first compute the 'Age' and assign it to dataset abalone\_Data.

abalone\_Data['Age'] = abalone\_Data['Rings']+1.5

abalone\_Data.drop('Rings', axis=1, inplace=True)

# Feature wise statistics using builtin tools

print(abalone\_Data.columns)

print(abalone\_Data.head())

print(abalone\_Data.info())

print(abalone\_Data.describe())

#There are no Missing values and all Feature are numeric except sex

# Creating X and y

feature\_column = ['Length', 'Diameter', 'Height', 'Whole weight', 'Shucked weight', 'Viscera weight', 'Shell weight']

X = abalone\_Data[feature\_column]

y = abalone\_Data['Age']

X\_train, X\_valid, y\_train, y\_valid = train\_test\_split(X, y, test\_size=0.30, random\_state=42)

print(X\_train.head())

print(y\_valid.head())

np.random.seed(155)

# Normalization

def norm\_s(df1, df2):

dfs = df1.append(df2)

min = np.min(dfs)

max = np.max(dfs)

me = np.mean(dfs)

sigma = np.std(dfs)

return (min, max, me, sigma)

def z\_score(col, stats):

m, M, me, s = stats

df2 = pd.DataFrame()

for c in col.columns:

df2[c] = (col[c]-me[c])/s[c]

return df2

stats = norm\_s(X\_train, X\_valid)

arr\_x\_train = np.array(z\_score(X\_train, stats))

arr\_y\_train = np.array(y\_train)

arr\_x\_valid = np.array(z\_score(X\_valid, stats))

arr\_y\_valid = np.array(y\_valid)

print('Training shape:', arr\_x\_train.shape)

print('Validation',arr\_y\_train.shape)

print('Training samples: ', arr\_x\_train.shape[0])

print('Validation samples: ', arr\_x\_valid.shape[0])

# Defining the Model

def model(x\_size, y\_size):

t\_model = Sequential()

t\_model.add(Dense(100, activation="tanh", input\_shape=(x\_size,)))

t\_model.add(Dropout(0.1))

t\_model.add(Dense(50, activation="relu"))

t\_model.add(Dense(20, activation="relu"))

t\_model.add(Dense(y\_size))

t\_model.compile(loss='mean\_squared\_error', optimizer=RMSprop(lr=0.004), metrics=[metrics.mae])

return t\_model

model = model(arr\_x\_train.shape[1], 1)

model.summary()

# Batch Size and Epoch

epochs = 800

batch\_size = 128

# Tensoboard Logic

LOG\_DIR = os.getcwd()

tensorboard = TensorBoard(log\_dir='LOG\_DIR', histogram\_freq=0,

write\_graph=True, write\_images=True)

# Fitting the Model

history = model.fit(arr\_x\_train, arr\_y\_train, batch\_size=batch\_size, epochs=epochs, shuffle=True, verbose=2,

callbacks=[tensorboard], validation\_data=(arr\_x\_valid, arr\_y\_valid),)

train\_score = model.evaluate(arr\_x\_train, arr\_y\_train, verbose=0)

valid\_score = model.evaluate(arr\_x\_valid, arr\_y\_valid, verbose=0)

print('Train MAE: ', round(train\_score[1], 4), ', Train Loss: ', round(train\_score[0], 4))

print('Val MAE: ', round(valid\_score[1], 4), ', Val Loss: ', round(valid\_score[0], 4))

print(os.getcwd())

# Logic for Plotting the Loss

def plot\_loss(h):

plt.figure()

plt.plot(h['loss'])

plt.plot(h['val\_loss'])

plt.title('Training vs Validation Loss')

plt.ylabel('Loss')

plt.xlabel('Epoch')

plt.legend(['Train', 'Validation'])

plt.draw()

plt.show()

return

plot\_loss(history.history)

**Program-2**

from tensorboardcolab import \*

from \_\_future\_\_ import print\_function

import os

from datetime import time

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import keras

from keras import metrics

from keras.models import Sequential

from keras.layers import Dense, Dropout, Flatten, Activation

from keras.layers import Conv2D, MaxPooling2D

from keras.optimizers import Adam, RMSprop

from keras.callbacks import TensorBoard, EarlyStopping, ModelCheckpoint

from keras.utils import plot\_model

from keras.models import load\_model

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

tbc=TensorBoardColab()

df = pd.read\_csv('heart.csv')

kc\_data = pd.DataFrame(df, columns=["age","sex","cp","trestbps","chol","fbs","restecg","thalach","exang","oldpeak","slope","ca","thal","target"])

label\_col = 'Class'

print(kc\_data.describe())

kc\_x\_train, kc\_x\_valid, kc\_y\_train, kc\_y\_valid = train\_test\_split(kc\_data.iloc[:,0:13], kc\_data.iloc[:,13],

                                                    test\_size=0.3, random\_state=87)

np.random.seed(155)

def norm\_stats(df1, df2):

    dfs = df1.append(df2)

    minimum = np.min(dfs)

    maximum = np.max(dfs)

    mu = np.mean(dfs)

    sigma = np.std(dfs)

    return (minimum, maximum, mu, sigma)

def z\_score(col, stats):

    m, M, mu, s = stats

    df2 = pd.DataFrame()

    for c in col.columns:

        df2[c] = (col[c]-mu[c])/s[c]

    return df2

stats = norm\_stats(kc\_x\_train, kc\_x\_valid)

arr\_x\_train = np.array(z\_score(kc\_x\_train, stats))

arr\_y\_train = np.array(kc\_y\_train)

arr\_x\_valid = np.array(z\_score(kc\_x\_valid, stats))

arr\_y\_valid = np.array(kc\_y\_valid)

print('Training shape:', arr\_x\_train.shape)

print('ddd',arr\_y\_train.shape)

print('Training samples: ', arr\_x\_train.shape[0])

print('Validation samples: ', arr\_x\_valid.shape[0])

def basic\_model\_1(x\_size, y\_size):

    t\_model = Sequential()

    t\_model.add(Dense(100, activation='sigmoid', input\_shape=(x\_size,)))

    t\_model.add(Dense(50, activation="sigmoid"))

    t\_model.add(Dense(y\_size))

    t\_model.compile(loss='binary\_crossentropy',

        optimizer='rmsprop',

        metrics=[metrics.mae])

    return(t\_model)

def basic\_model\_2(x\_size, y\_size):

    t\_model = Sequential()

    t\_model.add(Dense(100, activation='sigmoid', input\_shape=(x\_size,)))

    t\_model.add(Dropout(0.1))

    t\_model.add(Dense(50, activation='sigmoid'))

    t\_model.add(Dense(20, activation='sigmoid'))

    t\_model.add(Dense(y\_size))

    keras.optimizers.Adam(lr=0.001, beta\_1=0.9, beta\_2=0.999, epsilon=None, decay=0.0, amsgrad=False)

    optimizers = ['rmsprop', 'adam']

    t\_model.compile(loss='binary\_crossentropy',

        optimizer=Adam(),

        metrics=[metrics.mae])

    return(t\_model)

model = basic\_model\_2(arr\_x\_train.shape[1], 1)

model.summary()

epochs = 20

batch\_size =32

history = model.fit(arr\_x\_train, arr\_y\_train,

    batch\_size=batch\_size,

    epochs=epochs,

    shuffle=True,

    verbose=2,

    validation\_data=(arr\_x\_valid, arr\_y\_valid),callbacks=[TensorBoardColabCallback(tbc)])

train\_score = model.evaluate(arr\_x\_train, arr\_y\_train, verbose=0)

valid\_score = model.evaluate(arr\_x\_valid, arr\_y\_valid, verbose=0)

print('Train MAE: ', round(train\_score[1], 4), ', Train Loss: ', round(train\_score[0], 4))

print('Val MAE: ', round(valid\_score[1], 4), ', Val Loss: ', round(valid\_score[0], 4))

keras\_callbacks = [

    ModelCheckpoint('/tmp/keras\_checkpoints/model.{epoch:02d}-{val\_loss:.2f}.hdf5', monitor='val\_loss', save\_best\_only=True, verbose=2),

    ModelCheckpoint('/tmp/keras\_checkpoints/model.{epoch:02d}.hdf5', monitor='val\_loss', save\_best\_only=True, verbose=0),

    TensorBoard(log\_dir='./model\_3', histogram\_freq=0, write\_graph=True, write\_images=True, embeddings\_freq=0, embeddings\_layer\_names=None, embeddings\_metadata=None),

    EarlyStopping(monitor='val\_mean\_absolute\_error', patience=20, verbose=0)

]

def plot\_hist(h, xsize=6, ysize=10):

    fig\_size = plt.rcParams["figure.figsize"]

    plt.rcParams["figure.figsize"] = [xsize, ysize]

    fig, axes = plt.subplots(nrows=4, ncols=4, sharex=True)

    plt.subplot(211)

    plt.plot(h['mean\_absolute\_error'])

    plt.plot(h['val\_mean\_absolute\_error'])

    plt.title('Training vs Validation MAE')

    plt.ylabel('MAE')

    plt.xlabel('Epoch')

    plt.legend(['Train', 'Validation'], loc='upper left')

    plt.subplot(212)

    plt.plot(h['loss'])

    plt.plot(h['val\_loss'])

    plt.title('Training vs Validation Loss')

    plt.ylabel('Loss')

    plt.xlabel('Epoch')

    plt.legend(['Train', 'Validation'], loc='upper left')

    plt.draw()

    plt.show()

    return

plot\_hist(history.history, xsize=8, ysize=12)

score = model.evaluate(arr\_x\_valid,arr\_y\_valid)

print('test accuracy',score[1])

epochs = [5, 10]

batches = [5, 10, 100]

optimizers = ['rmsprop', 'adam']

hyperparameters = dict(optimizer=optimizers, epochs=epochs, batch\_size=batches)

**Program-3**

# Import appropriate libraries

import numpy as np

import os

import cv2

import matplotlib.pyplot as plt

from keras.callbacks import TensorBoard

from keras.models import Sequential

from keras.layers import Dense, MaxPool2D

from keras.layers import Dropout

from keras.layers import Flatten

from keras.constraints import maxnorm

from keras.optimizers import SGD

from keras.layers.convolutional import Conv2D

from keras.layers.convolutional import MaxPooling2D

from keras.utils import np\_utils, to\_categorical

from keras import backend as K

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

from sklearn.preprocessing import LabelEncoder

# Load data of natural images

labels = os.listdir('natural\_images/')

x = [] # Feature predictor variables array

y = [] # Target variables array

for label in labels:

pics = os.listdir('natural\_images/{}/'.format(label))

for pic in pics:

image = cv2.imread('natural\_images/{}/{}'.format(label, pic))

image\_resized = cv2.resize(image, (32, 32))

x.append(np.array(image\_resized))

y.append(label)

x = np.array(x)

y = np.array(y)

x = x.astype('float32') / 255

enc = LabelEncoder().fit(y)

y\_encoded = enc.transform(y)

y = to\_categorical(y\_encoded)

# Splitting data set into training and test data set

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.33)

# Build the neural network model

# Define the model being built

model = Sequential()

# Convolutional layer

model.add(Conv2D(filters=32, kernel\_size=(5, 5), activation='relu', input\_shape=x\_train.shape[1:]))

model.add(MaxPool2D(pool\_size=(2, 2)))

model.add(Conv2D(filters=64, kernel\_size=(3, 3), activation='relu'))

model.add(MaxPool2D(pool\_size=(2, 2)))

model.add(Dropout(rate=0.25))

# Flatten layer

model.add(Flatten())

model.add(Dense(256, activation='relu'))

model.add(Dropout(rate=0.5))

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

# Compile the model defined

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

print(model.summary())

# Fit the model defined on the training data set

history = model.fit(x\_train, y\_train, validation\_data=(x\_test, y\_test), epochs=5, batch\_size=2000)

# Final evaluation of the model using the test data set

scores = model.evaluate(x\_test, y\_test, verbose=0)

print("Accuracy: %.2f%%" % (scores[1]\*100))

[test\_loss, test\_acc] = model.evaluate(x\_test, y\_test)

print("Evaluation result on Test Data : Loss = {}, accuracy = {}".format(test\_loss, test\_acc))

# Listing all the components of data present in history

print('The data components present in history are', history.history.keys())

# Graphical evaluation of accuracy associated with training and validation data

plt.plot(history.history['acc'])

plt.plot(history.history['val\_acc'])

plt.title('Evaluation of Data Accuracy')

plt.xlabel('epoch')

plt.ylabel('Accuracy of Data')

plt.legend(['TrainData', 'ValidationData'], loc='upper right')

plt.show()

# Graphical evaluation of loss associated with training and validation data

plt.plot(history.history['loss'])

plt.plot(history.history['val\_loss'])

plt.xlabel('epoch')

plt.ylabel('Loss of Data')

plt.title('Evaluation of Data Loss')

plt.legend(['TrainData', 'ValidationData'], loc='upper right')

plt.show()

# Visualization of the model using tensor board

tbCallBack = TensorBoard(log\_dir='./lab2\_1', histogram\_freq=0, write\_graph=True, write\_images=True)

# Fitting the model defined using the training data along with validation using test data

history = model.fit(x\_train, y\_train, validation\_data=(x\_test, y\_test), epochs=10, verbose=0, initial\_epoch=0, callbacks=[tbCallBack])

# Evaluation of the loss and accuracy associated to the test data set

[test\_loss, test\_acc] = model.evaluate(x\_test, y\_test)

print("Evaluation result on Test Data using Tensorflow : Loss = {}, accuracy = {}".format(test\_loss, test\_acc))

# Listing all the components of data present in history

print('The data components present in history using Tensorflow are', history.history.keys())

# Graphical evaluation of accuracy associated with training and validation data

plt.plot(history.history['acc'])

plt.plot(history.history['val\_acc'])

plt.title('Evaluation of Data Accuracy using Tensorflow')

plt.xlabel('epoch')

plt.ylabel('Accuracy of Data')

plt.legend(['TrainData', 'ValidationData'], loc='upper right')

plt.show()

# Graphical evaluation of loss associated with training and validation data

plt.plot(history.history['loss'])

plt.plot(history.history['val\_loss'])

plt.xlabel('epoch')

plt.ylabel('Loss of Data')

plt.title('Evaluation of Data Loss using Tensorflow')

plt.legend(['TrainData', 'ValidationData'], loc='upper right')

plt.show()

**Program-4**

import re

from keras.layers.convolutional import MaxPooling1D

from keras.optimizers import Adam

from keras.layers import Dense, Dropout, Reshape, Flatten, concatenate, Input, Conv1D, GlobalMaxPooling1D, Embedding

import matplotlib

import numpy as np

import pandas as pd

from keras.preprocessing import sequence

from keras.models import Sequential

from keras.layers import Dense, Embedding

from keras.layers import LSTM

from keras.utils import to\_categorical

from keras.preprocessing.text import Tokenizer

from keras.preprocessing.sequence import pad\_sequences

import nltk

nltk.download('stopwords')

from nltk.corpus import stopwords

from nltk.tokenize import word\_tokenize

from sklearn.feature\_extraction.text import CountVectorizer

vect = CountVectorizer()

from sklearn import metrics

import seaborn as sns

import matplotlib.pyplot as plt

df\_train = pd.read\_csv('train.tsv', sep='\t')

df\_test = pd.read\_csv('test.tsv', sep='\t')

#The above part should help you to decide whether to use STOP WORDS or not.

#This part of code is just great analytical tool

#The data preprocessing

#we gonna make the text lower case and leave only the letters from a to z and digits.

df\_train['Phrase'] = df\_train['Phrase'].str.lower()

df\_train['Phrase'] = df\_train['Phrase'].apply((lambda x: re.sub('[^a-zA-z0-9\s]','',x)))

df\_test['Phrase'] = df\_test['Phrase'].str.lower()

df\_test['Phrase'] = df\_test['Phrase'].apply((lambda x: re.sub('[^a-zA-z0-9\s]','',x)))

X\_train = df\_train.Phrase

y\_train = df\_train.Sentiment

max\_fatures = 2000

tokenize = Tokenizer(num\_words=max\_fatures, split=' ')

tokenize.fit\_on\_texts(X\_train.values)

X\_test = df\_test.Phrase

X\_train = tokenize.texts\_to\_sequences(X\_train)

X\_test = tokenize.texts\_to\_sequences(X\_test)

max\_lenght = max([len(s.split()) for s in df\_train['Phrase']])

X\_train = pad\_sequences(X\_train, max\_lenght)

X\_test = pad\_sequences(X\_test, max\_lenght)

print(X\_train.shape)

print(X\_test.shape)

#model building

model=Sequential()

model.add(Embedding(max\_fatures, output\_dim=100,input\_length=48))

model.add(Conv1D(filters=64, kernel\_size=5, activation='relu', padding='causal'))

model.add(MaxPooling1D(pool\_size=2))

model.add(Conv1D(filters=64, kernel\_size=5, activation='relu', padding='causal'))

model.add(MaxPooling1D(pool\_size=2))

model.add(Dropout(0.7))

model.add(Dense(100,activation='relu'))

model.add(Dropout(0.5))

model.add(Flatten())

model.add(Dense(5,activation='softmax'))

model.compile(loss='sparse\_categorical\_crossentropy',optimizer=Adam(lr=0.001),metrics=['accuracy'])

model.summary()

history = model.fit(X\_train, y\_train, epochs=40, verbose=True,  batch\_size=1024)

**Program-5**

import re

from keras.layers.convolutional import MaxPooling1D

from keras.optimizers import Adam

from keras.layers import Dense, Dropout, Reshape, Flatten, concatenate, Input, Conv1D, GlobalMaxPooling1D, Embedding

import matplotlib

import numpy as np

import pandas as pd

from keras.preprocessing import sequence

from keras.models import Sequential

from keras.layers import Dense, Embedding

from keras.layers import LSTM

from keras.utils import to\_categorical

from keras.preprocessing.text import Tokenizer

from keras.preprocessing.sequence import pad\_sequences

import matplotlib.pyplot as plt

df\_train = pd.read\_csv('train.tsv', sep='\t')

df\_test = pd.read\_csv('test.tsv', sep='\t')

##Data preprocessing

#we make text lower case and leave only letters from a-z and digits

df\_train['Phrase'] = df\_train['Phrase'].str.lower()

df\_train['Phrase'] = df\_train['Phrase'].apply((lambda x: re.sub('[^a-zA-z0-9\s]','',x)))

df\_test['Phrase'] = df\_test['Phrase'].str.lower()

df\_test['Phrase'] = df\_test['Phrase'].apply((lambda x: re.sub('[^a-zA-z0-9\s]','',x)))

X\_train = df\_train.Phrase

y\_train = df\_train.Sentiment

max\_fatures = 2000

tokenize = Tokenizer(num\_words=max\_fatures, split=' ')

tokenize.fit\_on\_texts(X\_train.values)

X\_test = df\_test.Phrase

X\_train = tokenize.texts\_to\_sequences(X\_train)

X\_test = tokenize.texts\_to\_sequences(X\_test)

max\_lenght = max([len(s.split()) for s in df\_train['Phrase']])

X\_train = pad\_sequences(X\_train, max\_lenght)

X\_test = pad\_sequences(X\_test, max\_lenght)

print(X\_train.shape)

print(X\_test.shape)

#The Model building part

model=Sequential()

model.add(Embedding(max\_fatures, output\_dim=100,input\_length=48))

model.add(LSTM(128,dropout=0.5, recurrent\_dropout=0.5,return\_sequences=True))

model.add(LSTM(64,dropout=0.5, recurrent\_dropout=0.5,return\_sequences=False))

model.add(Dense(100,activation='relu'))

model.add(Dropout(0.5))

model.add(Dense(5,activation='softmax'))

model.compile(loss='sparse\_categorical\_crossentropy',optimizer=Adam(lr=0.004),metrics=['accuracy'])

model.summary()

history = model.fit(X\_train, y\_train, epochs=10, verbose=True, batch\_size=1024)

**Program-7**

from keras.layers import Input, Dense

from keras.models import Model

encoding\_dim = 32

input\_img = Input(shape=(784,))

encoded = Dense(encoding\_dim, activation='relu')(input\_img)

decoded = Dense(784, activation='sigmoid')(encoded)

autoencoder = Model(input\_img, decoded)

autoencoder.compile(optimizer='adadelta', loss='binary\_crossentropy')

from keras.datasets import mnist, fashion\_mnist

import numpy as np

(x\_train, y\_train), (x\_test, y\_test) = fashion\_mnist.load\_data()

x\_train = x\_train.astype('float32') / 255.

x\_test = x\_test.astype('float32') / 255.

x\_train = x\_train.reshape((len(x\_train), np.prod(x\_train.shape[1:])))

x\_test = x\_test.reshape((len(x\_test), np.prod(x\_test.shape[1:])))

autoencoder.fit(x\_train, x\_train,

                epochs=5,

                batch\_size=256,

                shuffle=True,

                validation\_data=(x\_test, x\_test))

from keras.layers import Input, Dense

from keras.models import Model

encoding\_dim = 32

input\_img = Input(shape=(784,))

encoded = Dense(units=128, activation='relu')(input\_img)

encoded = Dense(units=32, activation='relu')(encoded)

decoded = Dense(units=128, activation='relu')(encoded)

decoded = Dense(units=784, activation='sigmoid')(decoded)

autoencoder = Model(input\_img, decoded)

encoder = Model(input\_img, encoded)

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

from keras.datasets import mnist, fashion\_mnist

import numpy as np

(x\_train, y\_train), (x\_test, y\_test) = fashion\_mnist.load\_data()

x\_train = x\_train.astype('float32') / 255.

x\_test = x\_test.astype('float32') / 255.

x\_train = x\_train.reshape((len(x\_train), np.prod(x\_train.shape[1:])))

x\_test = x\_test.reshape((len(x\_test), np.prod(x\_test.shape[1:])))

history = autoencoder.fit(x\_train, x\_train,

                epochs=5,

                batch\_size=256,

                shuffle=True,

                validation\_data=(x\_test, x\_test))

encoded\_imgs = encoder.predict(x\_test)

predicted = autoencoder.predict(x\_test)

from matplotlib import pyplot as plt

plt.figure(figsize=(40, 4))

for i in range(10):

    ax = plt.subplot(3, 20, i + 1)

    plt.imshow(x\_test[i].reshape(28, 28))

    plt.gray()

    ax.get\_xaxis().set\_visible(False)

    ax.get\_yaxis().set\_visible(False)

    ax = plt.subplot(3, 20, i + 1 + 20)

    plt.imshow(encoded\_imgs[i].reshape(8,4))

    plt.gray()

    ax.get\_xaxis().set\_visible(False)

    ax.get\_yaxis().set\_visible(False)

    ax = plt.subplot(3, 20, 2\*20 +i+ 1)

    plt.imshow(predicted[i].reshape(28, 28))

    plt.gray()

    ax.get\_xaxis().set\_visible(False)

    ax.get\_yaxis().set\_visible(False)

from matplotlib import pyplot as plt

plt.plot(history.history['acc'])

plt.plot(history.history['val\_acc'])

plt.plot(history.history['loss'])

plt.plot(history.history['val\_loss'])

plt.title('model accuracy')

plt.ylabel('accuracy')

plt.xlabel('epoch')

plt.legend(['train\_acc','val\_acc', 'train\_loss','val\_loss'], loc='upper left')

plt.show()