**2. Implement a Multi-layer perceptron for classification using any dataset of your choice.**

**SOURCE CODE:**

import numpy as np

from keras.datasets import mnist

from keras.models import Sequential

from keras.layers import Dense, Dropout

from keras.optimizers import RMSprop

from keras.utils import to\_categorical

**# Load the MNIST dataset and preprocess the data**

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

x\_train = x\_train.reshape(60000, 784)

x\_test = x\_test.reshape(10000, 784)

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

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

x\_train /= 255

x\_test /= 255

y\_train = to\_categorical(y\_train, 10)

y\_test = to\_categorical(y\_test, 10)

**# Define the architecture of the MLP**

model = Sequential()

model.add(Dense(512, activation='relu', input\_shape=(784,)))

model.add(Dropout(0.2))

model.add(Dense(512, activation='relu'))

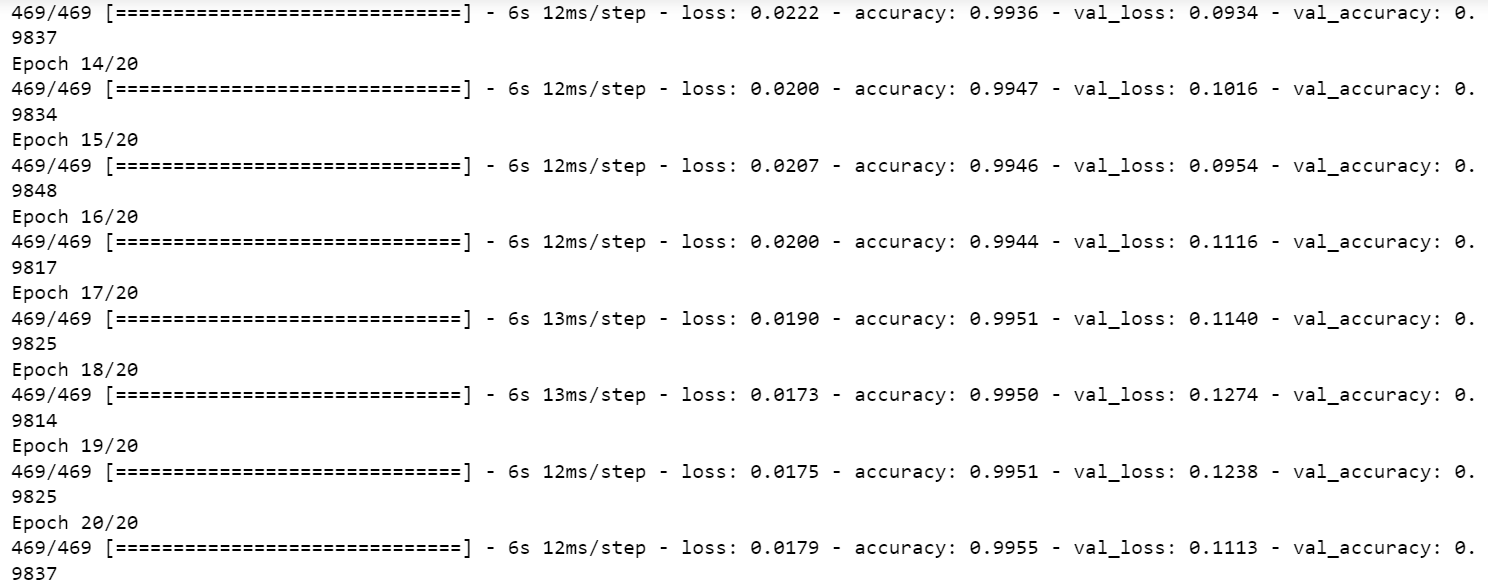
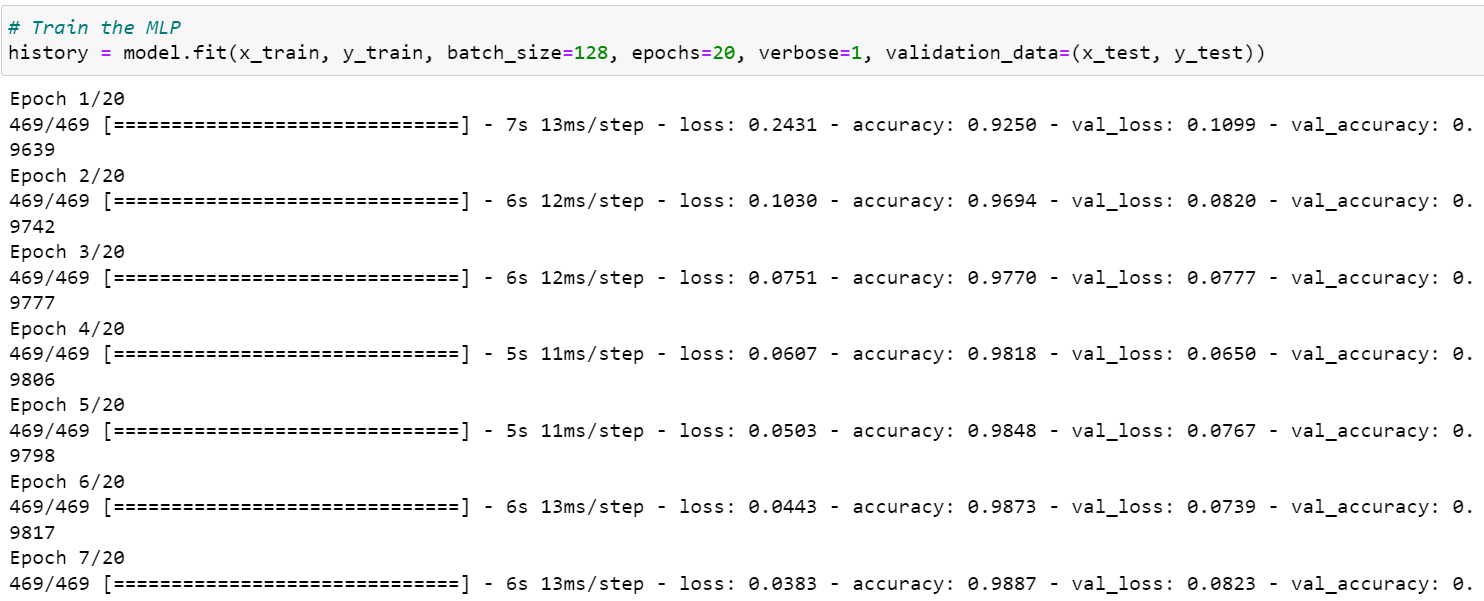
model.add(Dropout(0.2))

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

**# Compile the model and define the loss function and optimizer**

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

**# Train the MLP**

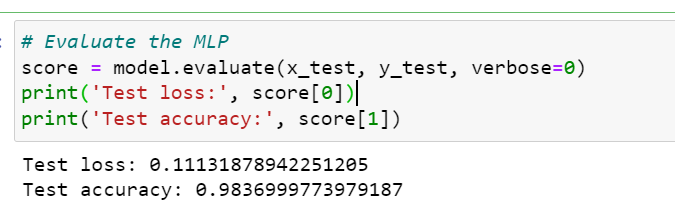
history = model.fit(x\_train, y\_train, batch\_size=128, epochs=20, verbose=1, validation\_data=(x\_test, y\_test)) 

**# Evaluate the MLP**

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

print('Test loss:', score[0])

print('Test accuracy:', score[1])



**OBSERVATION:**

* The **test loss of 0.11131878942251205** means that on average, the MLP made an error of 0.1113 when predicting the class labels of the test data. Since the loss function used in the code is categorical cross-entropy, a lower value of the loss function indicates better performance.
* The **test accuracy of 0.9837** means that the MLP correctly classified **98.37%** of the test images. This is a **very good accuracy**, indicating that the MLP is able to recognize the handwritten digits with a high degree of accuracy.
* The results suggest that the MLP with two hidden layers of 512 neurons each is able to learn a good representation of the MNIST dataset, and is able to classify the images with high accuracy.

**3.Design a Large CNN model for the classification of any Dataset in Keras**.

**SOURCE CODE:**

import keras as K

from keras.datasets import cifar10

from keras.models import Sequential

from keras.layers import Dense, Convolution2D, MaxPooling2D, Dropout, Flatten

from keras.optimizers import SGD

from keras.utils import to\_categorical

model = Sequential()

model.add(Convolution2D(32, (3, 3), padding="same", activation='relu', input\_shape=(32, 32, 3)))

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

model.add(Dropout(.2))

model.add(Convolution2D(64, (3, 3), padding="same", activation='relu'))

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

model.add(Dropout(.2))

model.add(Convolution2D(128, (3, 3), padding="same", activation='relu'))

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

model.add(Dropout(.2))

model.add(Flatten())

model.add(Dense(1024, activation="relu"))

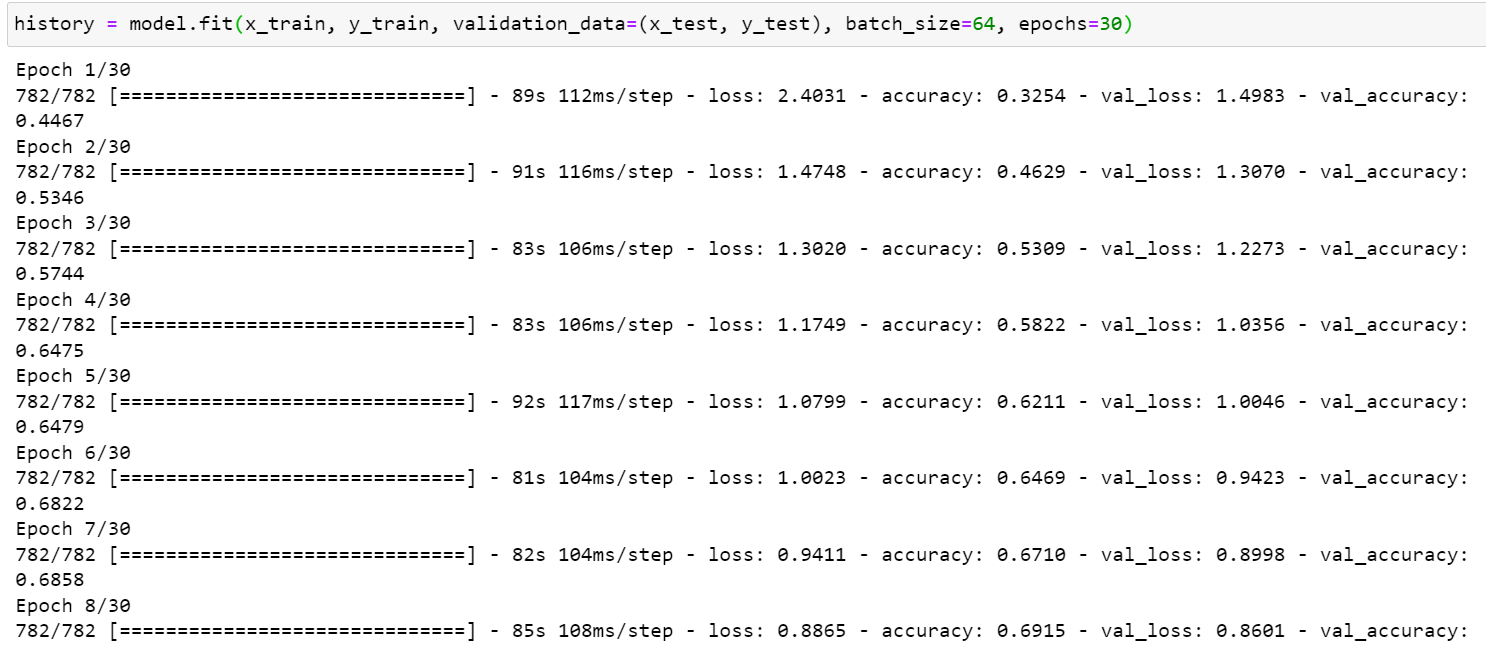
model.add(Dense(512, activation="relu"))

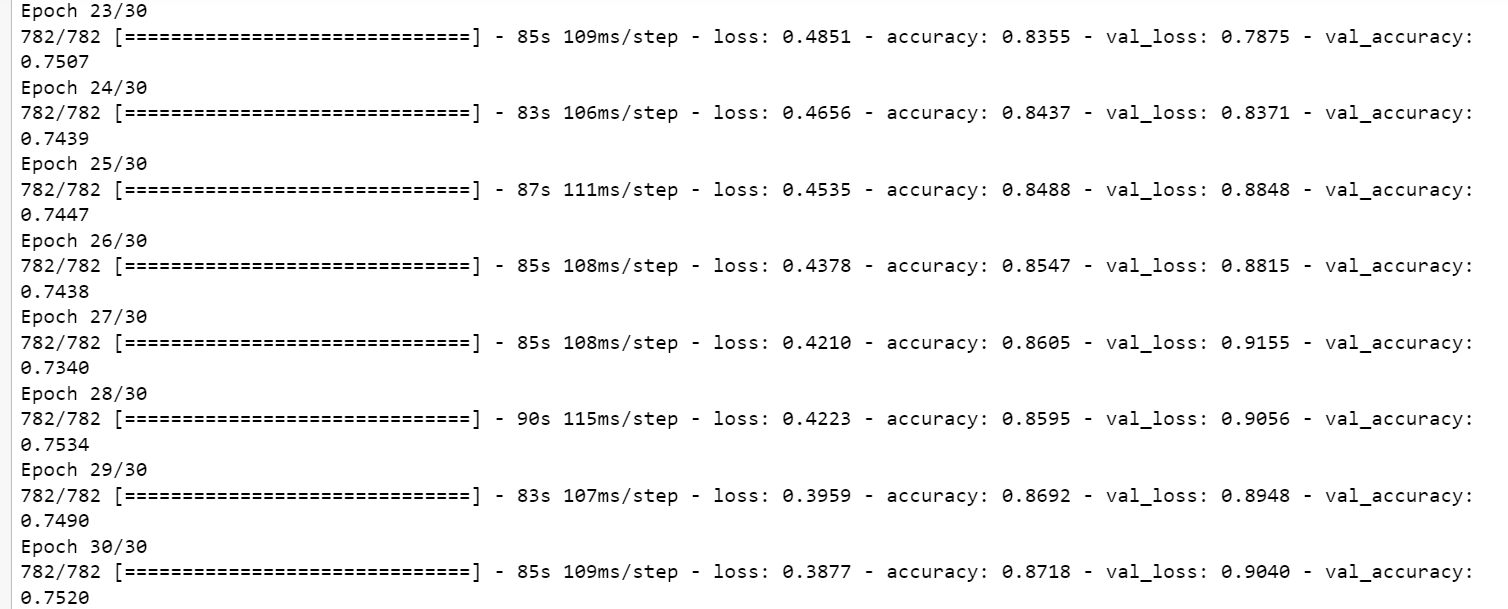
model.add(Dense(10, activation="softmax"))

sgd = SGD(learning\_rate=0.001, momentum=.9)

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

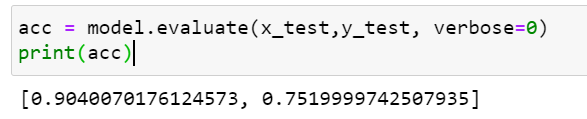
history = model.fit(x\_train, y\_train, validation\_data=(x\_test, y\_test), batch\_size=64, epochs=30)





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

print(acc)



**OBSERVATION:**

* The first layer added to the model is a Convolution2D layer with 32 filters, a kernel size of 3x3, and ReLU activation function. The padding="same" argument ensures that the output has the same shape as the input. The input\_shape=(32, 32, 3) argument defines the shape of the input images.
* The MaxPooling2D layer is then added with a pool size of 2x2 to reduce the spatial size of the output. The Dropout layer is added to randomly drop out 20% of the neurons to reduce overfitting.
* Two more layers of Convolution2D, MaxPooling2D, and Dropout are added with 64 and 128 filters respectively.
* The Flatten() layer flattens the output from the convolutional layers into a 1D array which can be passed as input to the fully connected layers.
* The Dense layer with 1024 neurons and ReLU activation function is added to provide additional capacity to the model for learning more advanced features. Another Dense layer with 512 neurons and ReLU activation function is added to further refine the learned features.
* Finally, a Dense layer with 10 neurons and softmax activation function is added as the output layer to predict the probability of each class. The softmax activation function is used to ensure that the sum of probabilities across all classes is equal to 1, making it suitable for multi-class classification problems.
* The **loss value of 0.8913** indicates that the model's predictions on the test set are on average, 89.13% off the actual values.
* The **accuracy value of 0.7524** indicates that the model's predictions are correct for 75.24% of the samples in the test set.

**4.Implement the concept of linear regression using any dataset of your choice.**

**SOURCE CODE:**

**# Simple Linear Regression**

**# Importing the libraries**

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

**# Importing the dataset**

dataset = pd.read\_csv('C:\\Users\\GAYATHRI\\Documents\\SRET-I YR- MSC\\TERM III\\Machine Learning\\Salary\_Data.csv')

X = dataset.iloc[:, :-1].values

y = dataset.iloc[:, -1].values

**# Splitting the dataset into the Training set and Test set**

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

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 1/3, random\_state = 0)

**# Training the Simple Linear Regression model on the Training set**

from sklearn.linear\_model import LinearRegression

regressor = LinearRegression()

regressor.fit(X\_train, y\_train)

**# Visualising the Training set results**

plt.scatter(X\_train, y\_train, color = 'red')

plt.plot(X\_train, regressor.predict(X\_train), color = 'blue')

plt.title('Salary vs Experience (Training set)')

plt.xlabel('Years of Experience')

plt.ylabel('Salary')

plt.show()



**# Visualising the Test set results**

plt.scatter(X\_test, y\_test, color = 'red')

plt.plot(X\_train, regressor.predict(X\_train), color = 'blue')

plt.title('Salary vs Experience (Test set)')

plt.xlabel('Years of Experience')

plt.ylabel('Salary')

plt.show()



**5. A hospital wants to develop a model to predict the severity of a patient's condition upon admission. The hospital has data on patients' demographics, medical history, and vital signs at admission. The hospital would like to use this information to predict whether the patient's condition is mild, moderate, or severe.**

**SOURCE CODE:**

import pandas as pd

import numpy as np

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

from sklearn.preprocessing import LabelEncoder

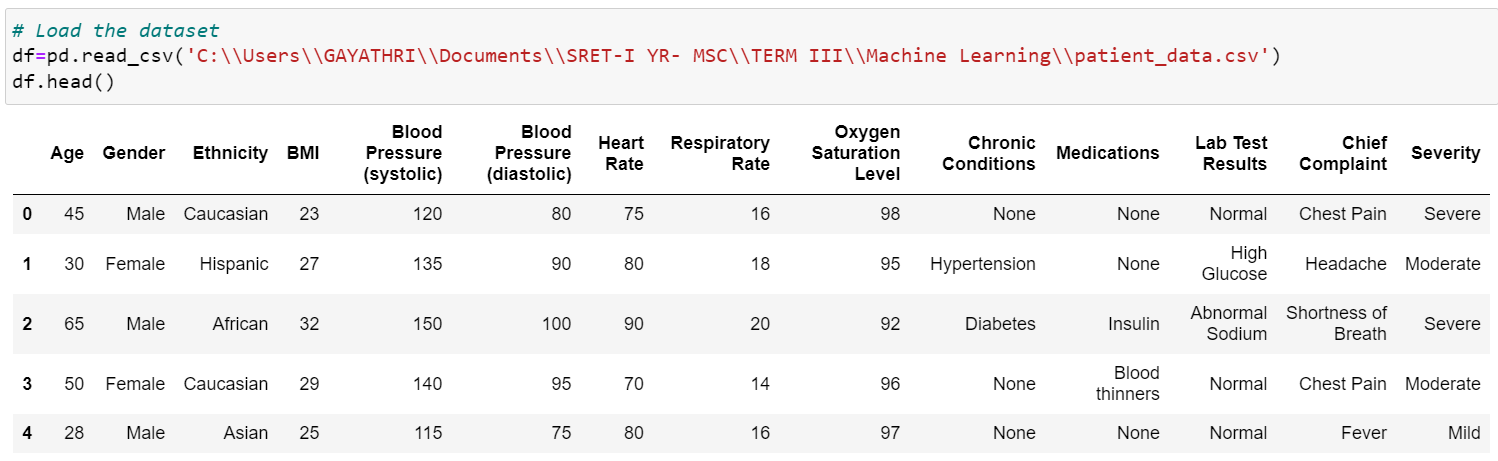
from sklearn.metrics import accuracy\_score

from sklearn.tree import DecisionTreeClassifier

**# Load the dataset**

df=pd.read\_csv('C:\\Users\\GAYATHRI\\Documents\\SRET-I YR- MSC\\TERM III\\Machine Learning\\patient\_data.csv')

df.head()



**# Convert categorical variables to numerical using LabelEncoder**

le = LabelEncoder()

df['Gender'] = le.fit\_transform(df['Gender'])

df['Ethnicity'] = le.fit\_transform(df['Ethnicity'])

df['Age'] = le.fit\_transform(df['Age'])

df['Severity'] = le.fit\_transform(df['Severity'])

df['Chronic Conditions'] = le.fit\_transform(df['Chronic Conditions'])

df['Medications'] = le.fit\_transform(df['Medications'])

df['Oxygen Saturation Level'] = le.fit\_transform(df['Oxygen Saturation Level'])

**# Split the dataset into training and testing sets**

X = df.drop(['Severity','Lab Test Results','Chief Complaint'],axis=1)

y = df['Severity']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

**Decision Tree Classification Model**

**# Train a decision tree classifier model**

clf = DecisionTreeClassifier(random\_state=42)

clf.fit(X\_train, y\_train)

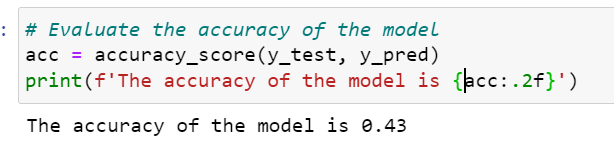
**# Make predictions on the testing set**

y\_pred = clf.predict(X\_test)

**# Evaluate the accuracy of the model**

acc = accuracy\_score(y\_test, y\_pred)

print(f'The accuracy of the model is {acc:.2f}')



**SUPPORT VECTOR MACHINE -SVM**

from sklearn.svm import SVC

from sklearn.metrics import accuracy\_score

**# Train a SVM classifier model**

clf = SVC(kernel='linear', random\_state=42) #train an SVM classifier model with a linear kernel

clf.fit(X\_train, y\_train)

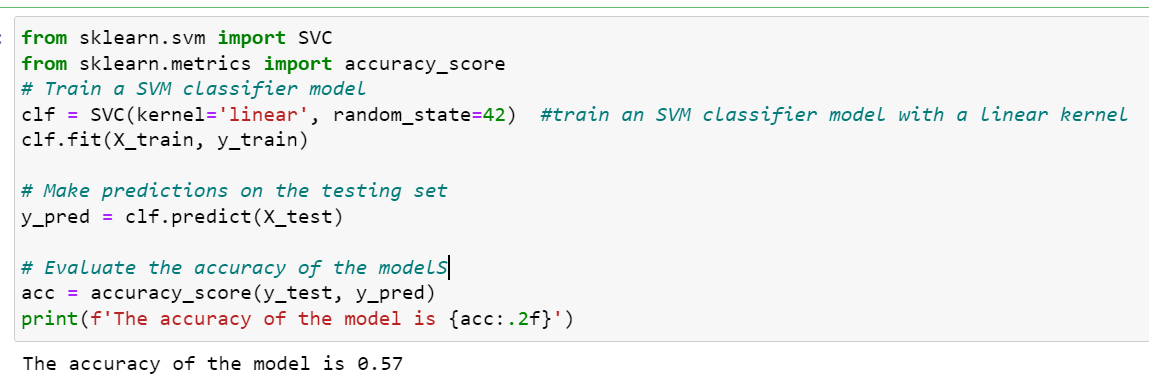
**# Make predictions on the testing set**

y\_pred = clf.predict(X\_test)

**# Evaluate the accuracy of the model**

acc = accuracy\_score(y\_test, y\_pred)

print(f'The accuracy of the model is {acc:.2f}')



**KNN:**

**#Fitting K-NN classifier to the training set**

from sklearn.neighbors import KNeighborsClassifier

classifier= KNeighborsClassifier(n\_neighbors=5, p=2 )

classifier.fit(X\_train, y\_train)

**#Predicting the test set result**

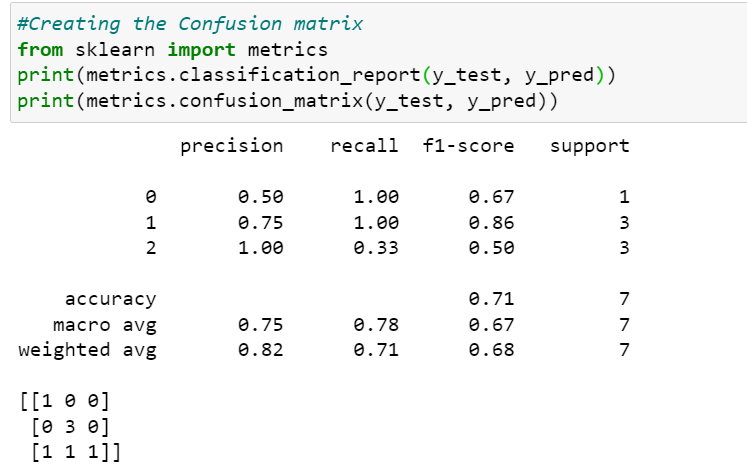
y\_pred= classifier.predict(X\_test)

**#Creating the Confusion matrix**

from sklearn import metrics

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

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



**OBSERVATION:**

* Load the dataset
* Use LabelEncoder to convert categorical variables to numerical.
* Split the dataset into training and testing sets, and train a decision tree classifier model,SVM and KNN.
* Finally, we make predictions on the testing set and evaluate the accuracy of the model.
* **K-Nearest Neighbour** provides **best accuracy score for severity prediction model ---> 71%**

**6. Use a linearly non separable dataset directly from the Scikit-learn library to implement SVM**

**SOURCE CODE:**

import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import make\_moons

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

from sklearn.svm import SVC

from sklearn.metrics import accuracy\_score

**# Generate the make\_moons dataset and split the data into training and testing sets**

X, y = make\_moons(noise=0.2, random\_state=42)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

**# Train an SVM model on the training data**

model = SVC(kernel='rbf', C=1.0, gamma=0.1)

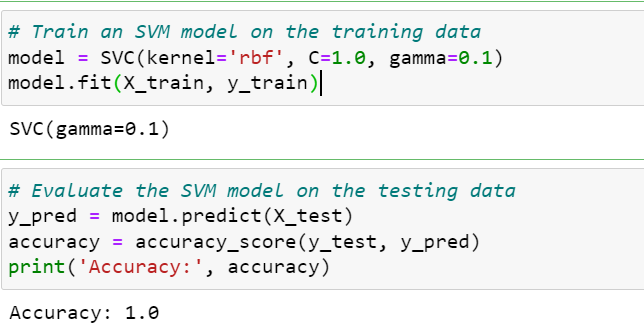
model.fit(X\_train, y\_train)

**# Evaluate the SVM model on the testing data**

y\_pred = model.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_pred)

print('Accuracy:', accuracy)



**# Plot the decision boundary**

x\_min, x\_max = X[:, 0].min() - 0.5, X[:, 0].max() + 0.5

y\_min, y\_max = X[:, 1].min() - 0.5, X[:, 1].max() + 0.5

xx, yy = np.meshgrid(np.arange(x\_min, x\_max, 0.02), np.arange(y\_min, y\_max, 0.02))

Z = model.predict(np.c\_[xx.ravel(), yy.ravel()])

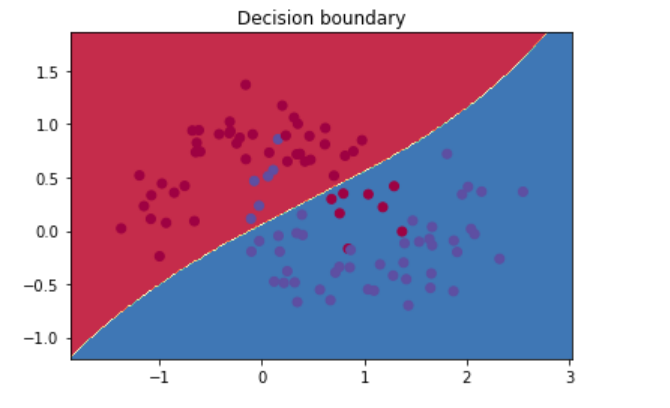
Z = Z.reshape(xx.shape)

plt.contourf(xx, yy, Z, cmap=plt.cm.Spectral)

plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Spectral)

plt.title('Decision boundary')

plt.show()



**Observation**

* We uses the scikit-learn library to train a Support Vector Machine (SVM) model using the **make\_moons dataset**.
* Firstly, the make\_moons function generates a two-class classification dataset with two features and some added noise. This dataset is then split into training and testing sets using a test size of 0.2 and a random state of 42.
* Next, an SVM model is created with a radial basis function (RBF) kernel and a regularization parameter of C=1.0 and a gamma value of 0.1.
* The SVM model is then trained using the training set and used to make predictions on the testing set.
* The accuracy of the model is then calculated using the accuracy\_score function and printed to the console, which in this case has achieved a **perfect accuracy score of 1.0**

**7. Design a Support vector machine for the classification using any dataset of your choice**

**SOURCE CODE:**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import sklearn

from sklearn import preprocessing

from sklearn.svm import SVC

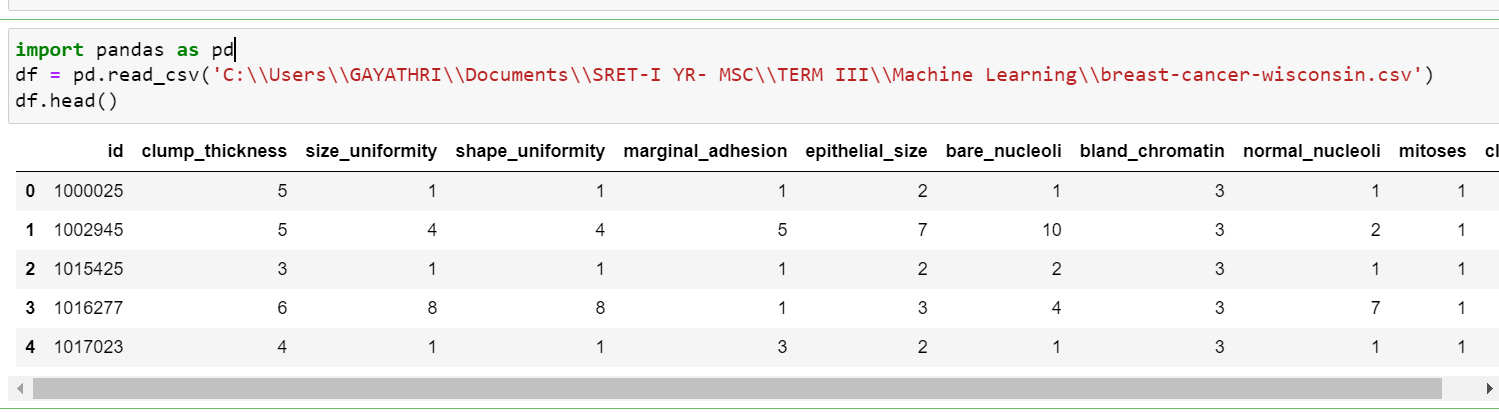
from sklearn import model\_selection

from sklearn.metrics import classification\_report,accuracy\_score

import pandas as pd

df = pd.read\_csv('C:\\Users\\GAYATHRI\\Documents\\SRET-I YR- MSC\\TERM III\\Machine Learning\\breast-cancer-wisconsin.csv')

df.head()



df.drop('id',axis = 1,inplace = True) # 'axis = 1' denotes column and 'inplace = True' denotes changes are saved in 'df'.

df.drop('bare\_nucleoli',axis = 1,inplace = True) #The ‘bare nuclei’ column is dropped due to format issues

X = df.drop('class',1) # X is input

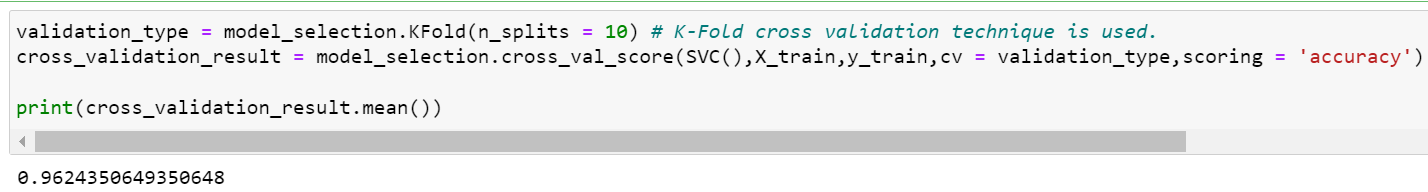
y = df['class'] # y is output

X\_train,X\_test,y\_train,y\_test = model\_selection.train\_test\_split(X,y,test\_size=0.2) # Spitting into 'Train\_set' and 'Test\_set'.

validation\_type = model\_selection.KFold(n\_splits = 10) # K-Fold cross validation technique is used.

cross\_validation\_result = model\_selection.cross\_val\_score(SVC(),X\_train,y\_train,cv = validation\_type,scoring = 'accuracy') # Cross validation score of SVC model.

print(cross\_validation\_result.mean())

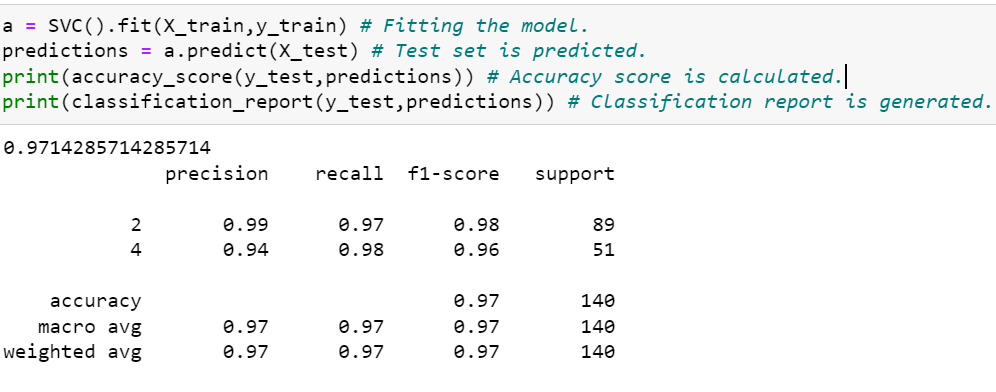


a = SVC().fit(X\_train,y\_train) # Fitting the model.

predictions = a.predict(X\_test) # Test set is predicted.

print(accuracy\_score(y\_test,predictions)) # Accuracy score is calculated.

print(classification\_report(y\_test,predictions)) # Classification report is generated.



prediction = a.predict(np.array([[1,2,2,5,3,6,4,8]]))

print(prediction)



**OBSERVATION:**

* The trained SVC model is used to predict a particular case :- ‘clump thickness’ = 1, ‘uniformity of cell size’ = 2, ‘uniformity of cell shape’ = 2, ‘marginal adhesion’ = 5 , ‘single epithelial cell size’ = 3 , ‘bland chromatin’ = 6, ‘normal nucleoli’ = 4, ‘mitosis’ = 8.
* The predicted value of ‘class’ is 4 which suggests it is a malignant tumor.
* **Accuracy score of SVC model = 0.971**
* Accuracy and F1 score of **SVC model is better**

**8. Design a LSTM with dropout for sequence classification of any dataset in keras**

**SOURCE CODE:**

**# LSTM with Dropout for sequence classification in the IMDB dataset**

import tensorflow as tf

from tensorflow.keras.datasets import imdb

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

from tensorflow.keras.layers import LSTM

from tensorflow.keras.layers import Dropout

from tensorflow.keras.layers import Embedding

from tensorflow.keras.preprocessing import sequence

**# fix random seed for reproducibility**

tf.random.set\_seed(7)

**# load the dataset but only keep the top n words, zero the rest**

top\_words = 5000

(X\_train, y\_train), (X\_test, y\_test) = imdb.load\_data(num\_words=top\_words)

**# truncate and pad input sequences**

max\_review\_length = 500

X\_train = sequence.pad\_sequences(X\_train, maxlen=max\_review\_length)

X\_test = sequence.pad\_sequences(X\_test, maxlen=max\_review\_length)

**# create the model**

embedding\_vecor\_length = 32

model = Sequential()

model.add(Embedding(top\_words, embedding\_vecor\_length, input\_length=max\_review\_length))

model.add(Dropout(0.2))

model.add(LSTM(100))

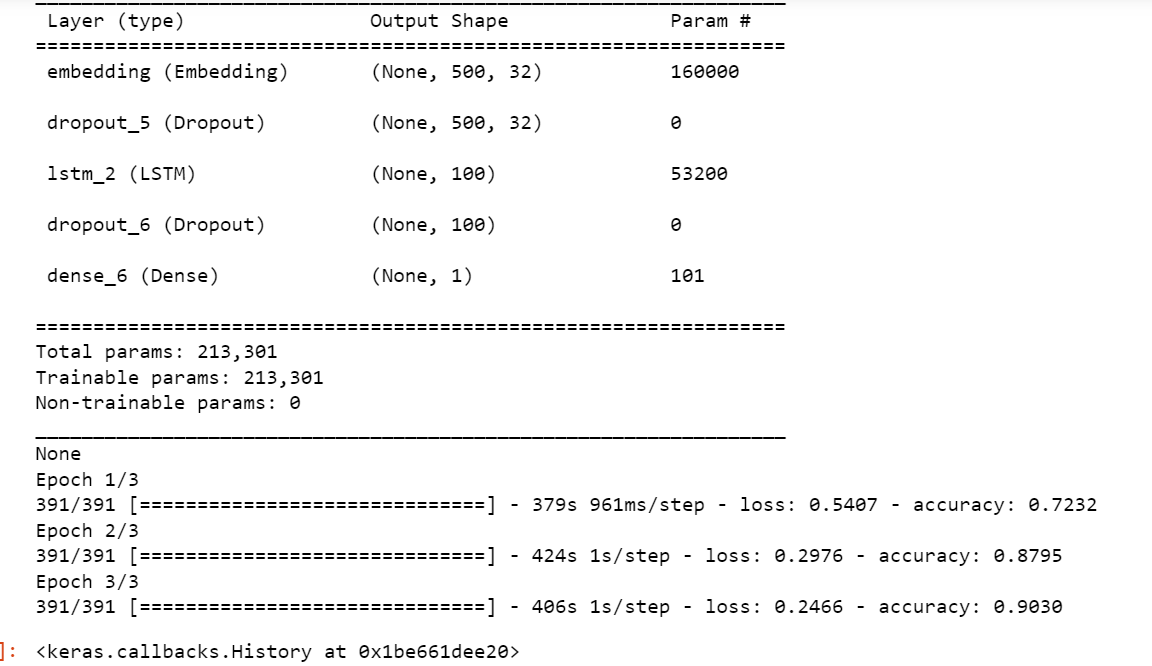
model.add(Dropout(0.2))

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

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

print(model.summary())

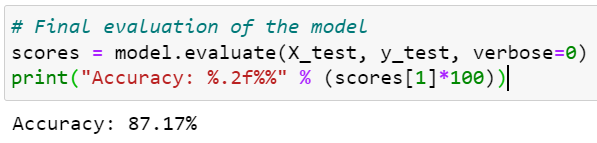
model.fit(X\_train, y\_train, epochs=3, batch\_size=64)



**# Final evaluation of the model**

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

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



**OBSERVATION:**

* Recurrent neural networks like LSTM generally have the problem of overfitting.
* Dropout can be applied between layers using the Dropout Keras layer. You can do this easily by adding new Dropout layers between the Embedding and LSTM layers and the LSTM and Dense output layers
* **Accuracy: 87.17%**