


```
import numpy as np
import pandas as pd
```

Loading the Dataset


```
df = pd.read_csv("/content/Iris.csv")
df.head(10)
```



	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa
5	6	5.4	3.9	1.7	0.4	Iris-setosa
6	7	4.6	3.4	1.4	0.3	Iris-setosa
7	8	5.0	3.4	1.5	0.2	Iris-setosa
8	9	4.4	2.9	1.4	0.2	Iris-setosa
9	10	4.9	3.1	1.5	0.1	Iris-setosa

Diving the dataset into target(y) and predictor(x) variables

```
# one-hot encoding
# Target variable = y = dependent variable = output variable
y = pd.get_dummies(df["Species"])
y
```



	Iris-setosa	Iris-versicolor	Iris-virginica
0	1	0	0
1	1	0	0
2	1	0	0
3	1	0	0
4	1	0	0
...
145	0	0	1
146	0	0	1
147	0	0	1
148	0	0	1
149	0	0	1

150 rows × 3 columns

```
# predictor variable = x = independent variable = input variable
x = df.drop(["Id", "Species"], axis=1)
x
```



	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2
...
145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

150 rows × 4 columns

Splitting the data into Training and Test Data

```
# split our dataset
from sklearn.model_selection import train_test_split

x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.3)
```

x_train



	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
115	6.4	3.2	5.3	2.3
63	6.1	2.9	4.7	1.4
28	5.2	3.4	1.4	0.2
55	5.7	2.8	4.5	1.3
30	4.8	3.1	1.6	0.2
...
10	5.4	3.7	1.5	0.2
95	5.7	3.0	4.2	1.2
6	4.6	3.4	1.4	0.3
121	5.6	2.8	4.9	2.0
16	5.4	3.9	1.3	0.4

105 rows × 4 columns

y_train



	Iris-setosa	Iris-versicolor	Iris-virginica
115	0	0	1
63	0	1	0
28	1	0	0
55	0	1	0
30	1	0	0
...
10	1	0	0
95	0	1	0
6	1	0	0
121	0	0	1
16	1	0	0

105 rows × 3 columns

Creating MLP Model

```
# creating our model --> MULTILAYER PERCEPTRON
```

```
from keras.models import Sequential # Sequential - used to initialize a neural network model that
# allows us to add layers one by one
```

```
from keras.layers import Dense # Dense - used to create fully connected layers, where each neuron
#in the layer is connected to every neuron in the previous layer
```

```
model = Sequential() # initializes a Sequential model & The Sequential model is a linear stack
#of layers
```

```
model.add(Dense(6,activation="sigmoid")) # Hidden layer #This adds a dense (fully connected)
#hidden layer with 6 neurons.
#The sigmoid activation function is used. This function
#outputs values between 0 and 1, which is useful for binary classification
```

```
model.add(Dense(3,activation="softmax")) # Output layer # This adds a dense output layer with 3
#neurons (one for each class in the Iris dataset: Iris-setosa, Iris-versicolor, Iris-virginica)
```

```
#The softmax activation function is used in the output layer for multi-class classification.
#It converts the raw output scores into probabilities that sum to 1 across all classes.
```

```
model.compile(loss="categorical_crossentropy",metrics=["accuracy"])
#loss="categorical_crossentropy": This is the loss function used for multi-class classification.
#It measures the difference between the predicted probabilities and the actual one-hot encoded
#class labels.
#metrics=["accuracy"]: This specifies that we want to track the accuracy of the model during
#training and evaluation.
```

```
#Training the Model:
```

```
model.fit(x_train,y_train,epochs= 25 , batch_size= 5)
```

```
#epochs=25: The number of times the entire training dataset is passed through the model during
#training.
```

```
#batch_size=5: The number of samples per gradient update. In other words, the model's weights are
#updated after processing each batch of 5 samples.
```

```
#loss: Displays the loss value for the current epoch.
```

```
#accuracy: Displays the accuracy for the current epoch.
```

```
Epoch 1/25
21/21 [=====] - 3s 3ms/step - loss: 1.7863 - accuracy: 0.3333
Epoch 2/25
21/21 [=====] - 0s 3ms/step - loss: 1.6515 - accuracy: 0.3333
Epoch 3/25
21/21 [=====] - 0s 3ms/step - loss: 1.5448 - accuracy: 0.3333
Epoch 4/25
21/21 [=====] - 0s 3ms/step - loss: 1.4445 - accuracy: 0.3333
Epoch 5/25
21/21 [=====] - 0s 3ms/step - loss: 1.3586 - accuracy: 0.3333
Epoch 6/25
21/21 [=====] - 0s 3ms/step - loss: 1.2842 - accuracy: 0.3333
Epoch 7/25
21/21 [=====] - 0s 3ms/step - loss: 1.2233 - accuracy: 0.3333
Epoch 8/25
21/21 [=====] - 0s 3ms/step - loss: 1.1697 - accuracy: 0.3333
Epoch 9/25
21/21 [=====] - 0s 4ms/step - loss: 1.1290 - accuracy: 0.3333
Epoch 10/25
21/21 [=====] - 0s 3ms/step - loss: 1.0955 - accuracy: 0.4476
Epoch 11/25
21/21 [=====] - 0s 3ms/step - loss: 1.0647 - accuracy: 0.5905
Epoch 12/25
21/21 [=====] - 0s 2ms/step - loss: 1.0399 - accuracy: 0.6476
Epoch 13/25
21/21 [=====] - 0s 3ms/step - loss: 1.0205 - accuracy: 0.6571
Epoch 14/25
21/21 [=====] - 0s 3ms/step - loss: 1.0024 - accuracy: 0.6571
Epoch 15/25
21/21 [=====] - 0s 3ms/step - loss: 0.9861 - accuracy: 0.6571
Epoch 16/25
21/21 [=====] - 0s 3ms/step - loss: 0.9714 - accuracy: 0.6571
Epoch 17/25
21/21 [=====] - 0s 3ms/step - loss: 0.9573 - accuracy: 0.6571
```

```
Epoch 18/25
21/21 [=====] - 0s 2ms/step - loss: 0.9450 - accuracy: 0.6571
Epoch 19/25
21/21 [=====] - 0s 3ms/step - loss: 0.9325 - accuracy: 0.6571
Epoch 20/25
21/21 [=====] - 0s 3ms/step - loss: 0.9195 - accuracy: 0.6571
Epoch 21/25
21/21 [=====] - 0s 3ms/step - loss: 0.9077 - accuracy: 0.6571
Epoch 22/25
21/21 [=====] - 0s 3ms/step - loss: 0.8951 - accuracy: 0.6571
Epoch 23/25
21/21 [=====] - 0s 3ms/step - loss: 0.8836 - accuracy: 0.6571
Epoch 24/25
21/21 [=====] - 0s 3ms/step - loss: 0.8723 - accuracy: 0.6571
Epoch 25/25
21/21 [=====] - 0s 3ms/step - loss: 0.8603 - accuracy: 0.6571
<keras.callbacks.History at 0x7f58705e8d10>
```

#Testing the Model

```
score = model.evaluate(x_test,y_test)
print("ACCURACY : ",score)
```

```
2/2 [=====] - 0s 8ms/step - loss: 0.8300 - accuracy: 0.6889
ACCURACY : [0.8299992680549622, 0.6888889074325562]
```

df.head(5)

```

  Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm  Species
0   1             5.1             3.5             1.4             0.2  Iris-setosa
1   2             4.9             3.0             1.4             0.2  Iris-setosa
2   3             4.7             3.2             1.3             0.2  Iris-setosa
3   4             4.6             3.1             1.5             0.2  Iris-setosa
4   5             5.0             3.6             1.4             0.2  Iris-setosa
```

y.head(5)

```

  Iris-setosa  Iris-versicolor  Iris-virginica
0           1                0                0
1           1                0                0
2           1                0                0
3           1                0                0
4           1                0                0
```

Model Prediction

```
i = model.predict([[5.1,3.5,1.4,0.2]]) #the model is given the values for x (0th row)
i
```

```
array([[0.5674825 , 0.24086782, 0.19164968]], dtype=float32)
```

```
#The model predicts the probability of possible species
#As we can see the probability of the the setosa is 56% and this is correct .
```

Start coding or [generate](#) with AI.

Start coding or [generate](#) with AI.