Neural network approach to Iris dataset

Maimoona Khilji

Data Science,

Institute of management Sciences

Peshawar Pakistan

Maimoon.khilji@gmail.com

Abstract: **— Artificial neural network (ANN) is usually known as neural networks. One of the type of feedforward neural network is Multi-layer Perceptron. In this paper I have worked on classification of flowers using Multi-layer perceptron concept. The dataset that I have used is Iris Dataset that consists of features and labels of iris plant species. The main objective of this model is to recognize the iris species automatically. Tools used in this paper are Pandas, Keras, Seaborn, Matplotlib.pyplot, Numpy, sklearn.processing.**

Index Terms: **— Artificial Neural Network, Multi-Layer perceptron, Classification Technique, Machine Learning, Pandas, Scikit Learn, Python**

INTRODUCTION

Neural Network is a collection of algorithms that is used to recognize the patterns through a process that is similar to the way human brain operates. Neural networks is known as system of neurons. Neural Network consists of two main types of algorithm:

1. Feed Forward Network
2. Feed Backward Network

Multi-Layer Perceptron is one of the type of Feed Forward Network. It is a supervised learning algorithm in which labels are given along the features in training phase.

Six basics steps are used to implement the model with the approach of multi-layer perceptron.

1. Collect Data
2. Split the data into training and testing data batches
3. Select Algorithm
4. Fit the model witch selected algorithm
5. Make prediction on testing data
6. Evaluation of Model

LITERATURE REVIEW

Numerous researchers work on iris-species recognition system using the Multi-Layer Perceptron.

Mokriš I. And Turcaník M. [1]worked on multilayer perceptron using the sigmoidal activation function. Sigmoidal is used for recognition of invariant patterns.

Dutta D., Roy A., Reddy k.[2] improved the performance of ANN by adaptation of weights using PSO (Particle Swarm Optimization).

DATASET

In this paper I have used the Iris Dataset, from Kaggle.[3]

Dataset Information: The Iris dataset consists of 150 records. It has four features that are: Sepal length, Sepal Width, Petal Length, Petal Width. All these features are given as numeric attributes in cm. It has three classes that are Virginica, Setosa and Versicolor.

Iris Setosa

Figure 1 Iris Setosa

Iris Versicolor

Figure 2 Iris Versicolor

Iris Virginica

Figure 3 iris Virginica

METHODOLOGY

In this section, the complete algorithm and implementation of model using MLP approach is explained.

Objective: Train the model in such a way that it recognize the iris plant species using MLP approach.

Solution: In order to achieve this objective, I used Jupyter Notebook. Initially I import required libraries including:

- **Pandas** is used for loading data from csv into table form
- **Seaborn** and **matplotlib.pyplot** is used for visualization
- **Numpy** is used for mathematical operations and linear algebra
- **Normalize** is used for normalization
- **Keras** is used for Neural Network

Import required libraries

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
from sklearn.preprocessing import normalize
import keras
from keras.models import Sequential
from keras.layers import Dense,Activation,Dropout
from tensorflow.keras.layers import BatchNormalization
from keras.utils import np_utils
```

Figure 4

Using pandas, I load the iris dataset.

Input the data

```
data=pd.read_csv("Iris.csv")
```

first 5 records of the dataset

```
data.head(5)
```

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

Figure 5

In above figure it is shown that four columns are of features and one column consists of label or classes. Then used the describe( ) function to know about the statistical analysis of feature columns.

Describing the data

```
data.describe()
```

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

Figure 6

In order to know about the classes of iris- species, unique( ) function can be useful.

Classes of Iris

```
print(data["Species"].unique())
['Iris-setosa' 'Iris-versicolor' 'Iris-virginica']
```

Figure 7

The code in below figure shows the visualization of the dataset in order to understand the data more.

Visualisation of the dataset

```
sns.lmplot(x='SepalLengthCm', y='SepalWidthCm',
           data=data,
           fit_reg=False,
           hue="Species",
           scatter_kws={"marker": "D",
                        "s": 50})
plt.title('SepalLength vs SepalWidth')

sns.lmplot(x='SepalLengthCm', y='PetalWidthCm',
           data=data,
           fit_reg=False,
           hue="Species",
           scatter_kws={"marker": "D",
                        "s": 50})
plt.title('PetalLength vs PetalWidth')

sns.lmplot(x='SepalLengthCm', y='PetalLengthCm',
           data=data,
           fit_reg=False,
           hue="Species",
           scatter_kws={"marker": "D",
                        "s": 50})
plt.title('SepalLength vs PetalLength')

sns.lmplot(x='SepalWidthCm', y='PetalWidthCm',
           data=data,
           fit_reg=False,
           hue="Species",
           scatter_kws={"marker": "D",
                        "s": 50})
plt.title('SepalWidth vs PetalWidth')
plt.show()
```

Figure 8

From the below output, it can be concluded that species are segregated into different regions.

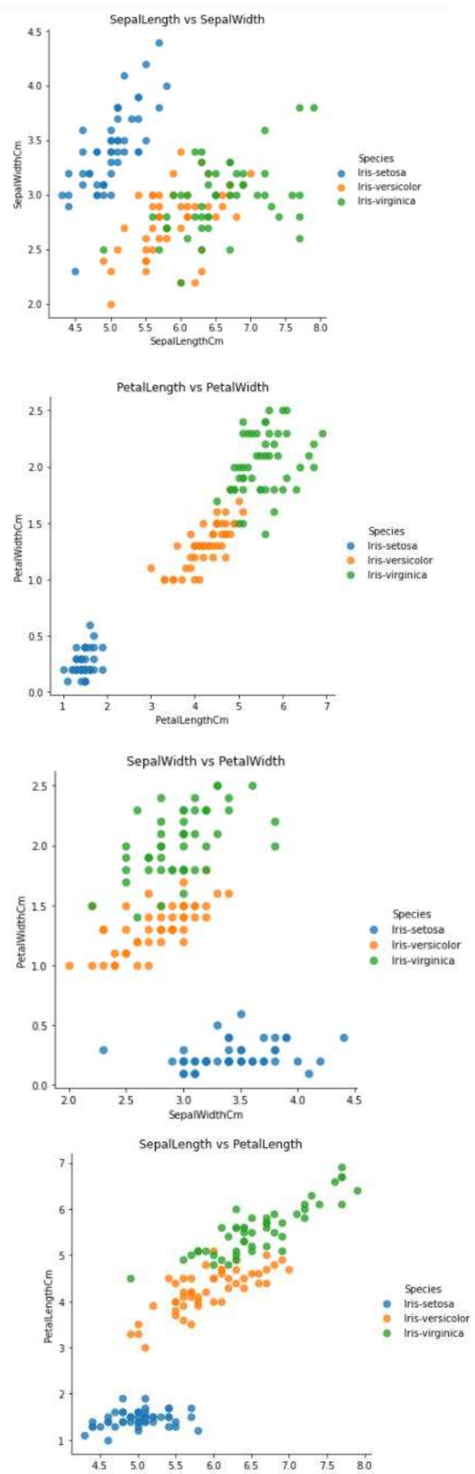

Figure 9

For processing, we generally use numeric values instead of qualitative. So I assigned the quantitative labels into numeric ones.

Convert the alphabetical names of classes into numeric values

```
data.loc[data["Species"]=="Iris-setosa","Species"]=0
data.loc[data["Species"]=="Iris-versicolor","Species"]=1
data.loc[data["Species"]=="Iris-virginica","Species"]=2
```

Figure 10

For confirmation whether the qualitative labels are converted into numeric ones or not, I fetched the data randomly from dataset.

Fetch Data randomly from data

```
data=data.iloc[np.random.permutation(len(data))]
data.head()
```

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
8	4.4	2.9	1.4	0.2	0
52	6.9	3.1	4.9	1.5	1
0	5.1	3.5	1.4	0.2	0
66	5.6	3.0	4.5	1.5	1
125	7.2	3.2	6.0	1.8	2

Figure 11

For processing, we convert the data into array.

Converting data to numpy array in order for processing

```
X=data.iloc[:,0:4].values
y=data.iloc[:,4].values

print("Shape of features column\t",X.shape)
print("Shape of label column\t\t",y.shape)
```

```
Shape of features column        (150, 4)
Shape of label column           (150,)
```

Figure 12

In this dataset, values of feature columns variate like Sepal Length has 7.2cm value, Sepal Width has 3.5cm, Petal Length has 4.9cm and Petal Width has 1.8cm value. That's why the range of the dataset may be different. In order to maintain a good accuracy of the model, the values of feature columns must be normalized to a range of o – 1.

Normalization

```
X_normalized=normalize(X,axis=0)
print("Examples of X_normalised\n",X_normalized[:3])
```

```
Examples of X_normalised
 [[0.06087757 0.07676768 0.02754646 0.01150299]
 [0.0954671  0.082062   0.09641262 0.08627246]
 [0.07056264 0.09265065 0.02754646 0.01150299]]
```

Figure 13

We split dataset into training and testing datasets. Training dataset is used to fit or train the model while testing data is used to test the performance.

Splitting Dataset into Train,test and validation data

80% -- train data
20% -- test data

```
total_length=len(data)
train_length=int(0.8*total_length)
test_length=int(0.2*total_length)

X_train=X_normalized[:train_length]
X_test=X_normalized[train_length:]
y_train=y[:train_length]
y_test=y[train_length:]
```

```
print("Length of train set x:",X_train.shape,"y:",y_train.shape)
print("Length of test set x:",X_test.shape,"y:",y_test.shape)
```

```
Length of train set x: (120, 4) y: (120,)
Length of test set x: (30, 4) y: (30,)
```

Figure 14

As I have converted the qualitative names of classes to quantitative by using 0, 1 and 2. No I will change the labels (classes) to one hot vector

[0]---> [1 0 0]

[1]---> [0 1 0]

[2]---> [0 0 1]

Neural network module

```
y_train=np_utils.to_categorical(y_train,num_classes=3)
y_test=np_utils.to_categorical(y_test,num_classes=3)
y_test
```

```
array([[[[[0., 1., 0.],
          [1., 0., 0.],
          [1., 0., 0.]],

         [[1., 0., 0.],
          [0., 1., 0.],
          [1., 0., 0.]],

         [[0., 1., 0.],
          [1., 0., 0.],
          [1., 0., 0.]]],
```

Figure 15

For neural Network, I used a Sequential ( ).
A Sequential model is appropriate for a plain stack of layers where each layer has exactly one input tensor and one output tensor.[4]

In this network I added four layers with different number of layers, using different activation functions in each layer. The Function Dense shows that the neurons of one layer are completely connected to the neurons of preceding and the coming next layer. The Dropout function is used to prevent the model from overfitting.

Neural Network

```
model=Sequential()
model.add(Dense(1000,input_dim=4,activation='relu'))
model.add(Dense(500,activation='relu'))
model.add(Dense(300,activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(3,activation='softmax'))
model.compile(loss='categorical_crossentropy',optimizer='adam',metrics=['accuracy'])
```

Figure 16

Summary of model can be given using .summary ( ) function.

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 1000)              5000

 dense_1 (Dense)             (None, 500)               500500

 dense_2 (Dense)             (None, 300)               150300

 dropout (Dropout)           (None, 300)               0

 dense_3 (Dense)             (None, 3)                 903

=================================================================
Total params: 656,703
Trainable params: 656,703
Non-trainable params: 0
_________________________________________________________________
```

Figure 17

After creating neural network, using .fit ( ) model we train the functions.

Train the model

```
model.fit(X_train,y_train,validation_data=(X_test,y_test),batch_size=20,epochs=10,verbose=1)
```

```
Epoch 1/10
6/6 [==============================] - 1s 91ms/step - loss: 1.0907 - accuracy: 0.3583 - val_loss: 1.
00
Epoch 2/10
6/6 [==============================] - 0s 23ms/step - loss: 1.0279 - accuracy: 0.6583 - val_loss: 0.
00
Epoch 3/10
6/6 [==============================] - 0s 31ms/step - loss: 0.9073 - accuracy: 0.6583 - val_loss: 0.
00
Epoch 4/10
6/6 [==============================] - 0s 26ms/step - loss: 0.7096 - accuracy: 0.6750 - val_loss: 0.
33
Epoch 5/10
6/6 [==============================] - 0s 22ms/step - loss: 0.5002 - accuracy: 0.7750 - val_loss: 0.
00
Epoch 6/10
6/6 [==============================] - 0s 22ms/step - loss: 0.3639 - accuracy: 0.9333 - val_loss: 0.
33
Epoch 7/10
```

Figure 18

For testing the model, .predict( ) function is used to test the trained model. Here the X_test is the unseen data for the model.

Test the model

```
prediction=model.predict(X_test)
length=len(prediction)
y_label=np.argmax(y_test,axis=1)
predict_label=np.argmax(prediction,axis=1)
```

Figure 19

RESULTS

For evaluation of model, confusion matrix and accuracy of the model are calculated. Result is the important part of the implementation. Result shows whether the model is properly trained or not. If not, then result help us to modify the model.

System Evaluation

```
accuracy=np.sum(y_label==predict_label)/length * 100
print("Accuracy of the dataset",accuracy )
```

```
Accuracy of the dataset 90.0
```

```
from sklearn.metrics import confusion_matrix
cm=confusion_matrix(y_label,predict_label)
cm
```

```
array([[11,  0,  0],
       [ 0,  7,  2],
       [ 0,  1,  9]], dtype=int64)
```

```
from sklearn.metrics import ConfusionMatrixDisplay
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=data["Species"].unique())
disp.plot(cmap=plt.cm.Blues)
plt.show()
```

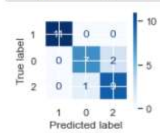

Figure 20

CONCLUSION

In this paper, I trained a model to recognize the iris-plant species using the approach of multi-layer perceptron. The accuracy of this model is 90% and through confusion matrix, it is presented that it 100% accurately identify the class-1 species and make few wrong predictions of class-0 and class-2.

REFERENCES

[1]. Marcek D., „Forecasting of economic quantities using fuzzy autoregressive models and fuzzy neural networks‟, Vol.1, pp.147-155.1

[2]. Dutta D., Roy A., Reddy k.,‟ Training Artificial Neural Network using Particle Swarm Optimization Algorithm‟ IJARCSSE, Vol 3, Issue 3, March 2013, pp. 430-434

[3]. "Iris Species," 2021. [Online]. Available: https://www.kaggle.com/uciml/iris.

[4]. K. Team, "Keras documentation: The Sequential model", *Keras.io*, 2021. [Online]. Available: https://keras.io/guides/sequential_model/. [Accessed: 22- Dec- 2021].