Machine Learning Applications for Biological Data

Niranga Udumulla

December 6, 2020

Abstract

In this report consist the 50 samples from each of three species of Iris (Iris setosa, Iris virginica and Iris versicolor). Four features were measured from each sample: the length and the width of the sepals and petals, in centimeters.Overall,simple neural network model were applied to predict the species types using their features.

1 Introduction

This is perhaps the best known database to be found in the pattern recognition literature. Fisher's paper is a classic in the field and is referenced frequently to this day. (See Duda Hart, for example.) The data set contains 3 classes of 50 instances each, where each class refers to a type of iris plant. One class is linearly separable from the other 2; the latter are NOT linearly separable from each other.

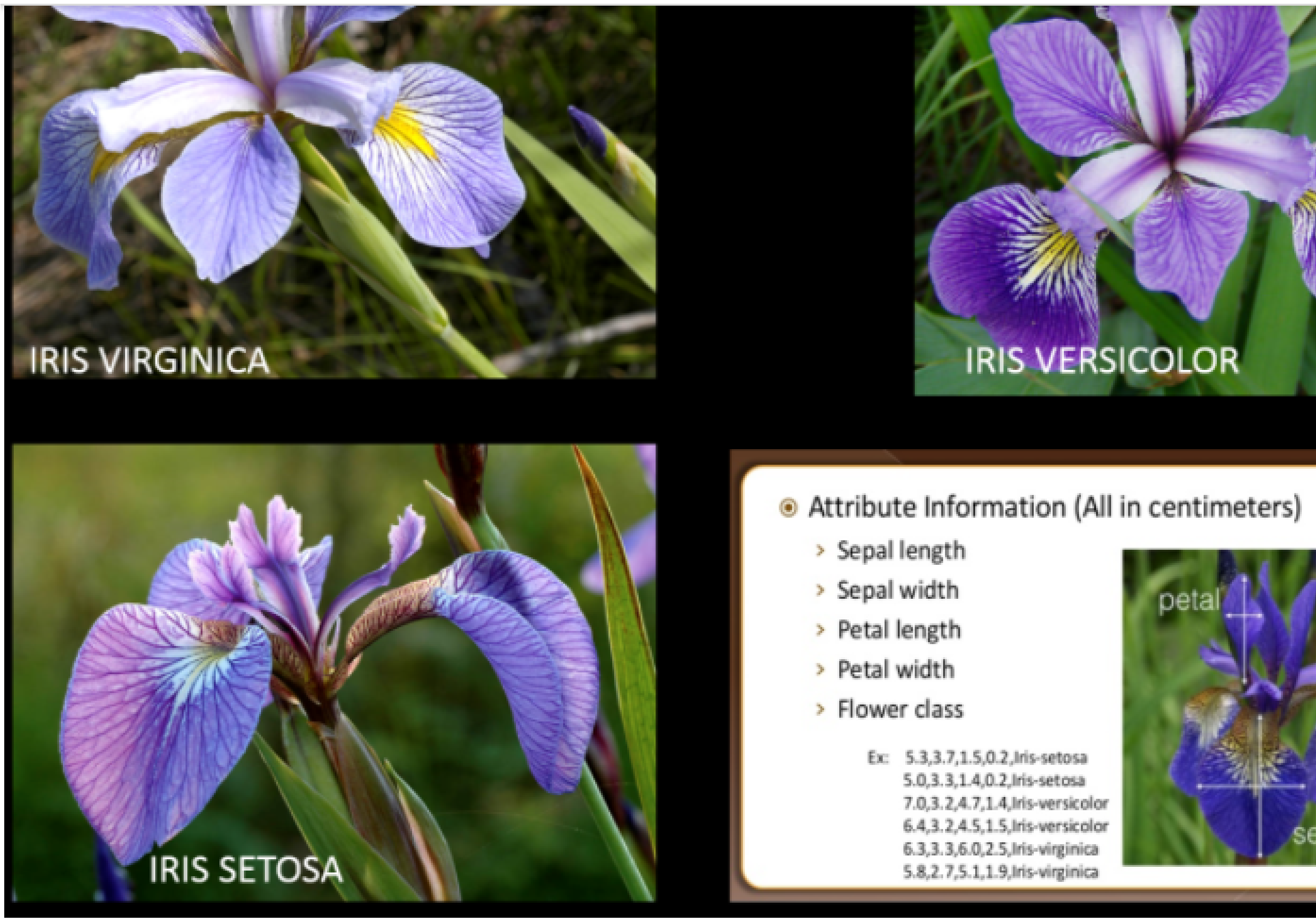
Predicted attribute: class of iris plant.

This is an exceedingly simple domain.

This data differs from the data presented in Fishers article (identified by Steve Chadwick, spchadwick '@' espeedaz.net). The 35th sample should be: 4.9,3.1,1.5,0.2,"Iris-setosa" where the error is in the fourth feature. The 38th sample: 4.9,3.6,1.4,0.1,"Iris-setosa" where the errors are in the second and third features.

2 Theory

In this lab demonstrate how to use machine learning based technique to identify the category of the biological data(flowers) using their features.

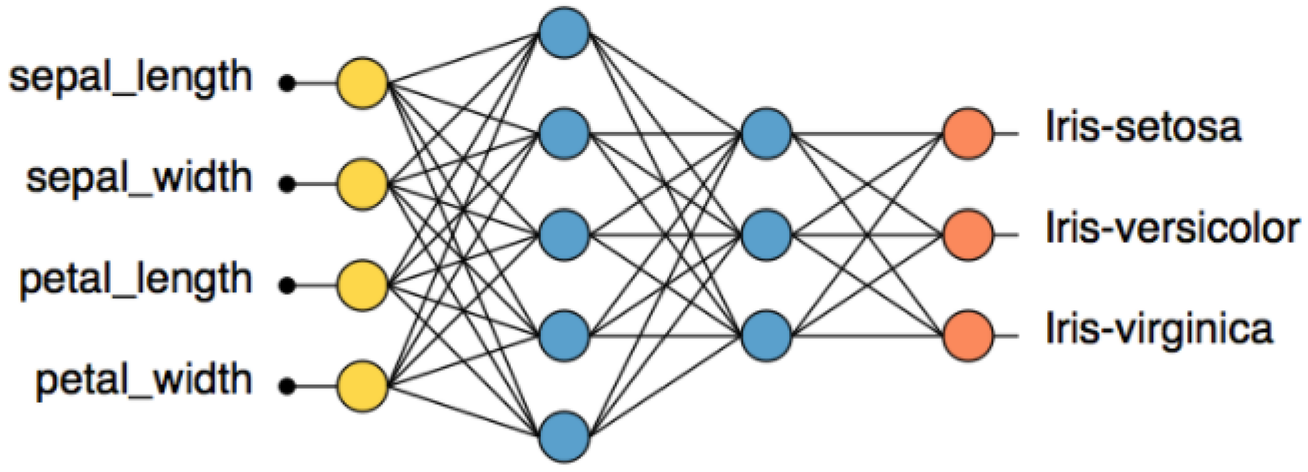


- Attribute Information (All in centimeters)
  - Sepal length
  - Sepal width
  - Petal length
  - Petal width
  - Flower class

```
Ex:   5.3,3.7,1.5,0.2,Iris-setosa
      5.0,3.3,1.4,0.2,Iris-setosa
      7.0,3.2,4.7,1.4,Iris-versicolor
      6.4,3.2,4.5,1.5,Iris-versicolor
      6.3,3.3,6.0,2.5,Iris-virginica
      5.8,2.7,5.1,1.9,Iris-virginica
```

3 Procedure

The neural network model were utilized to predict the category.



3.1 Load Data

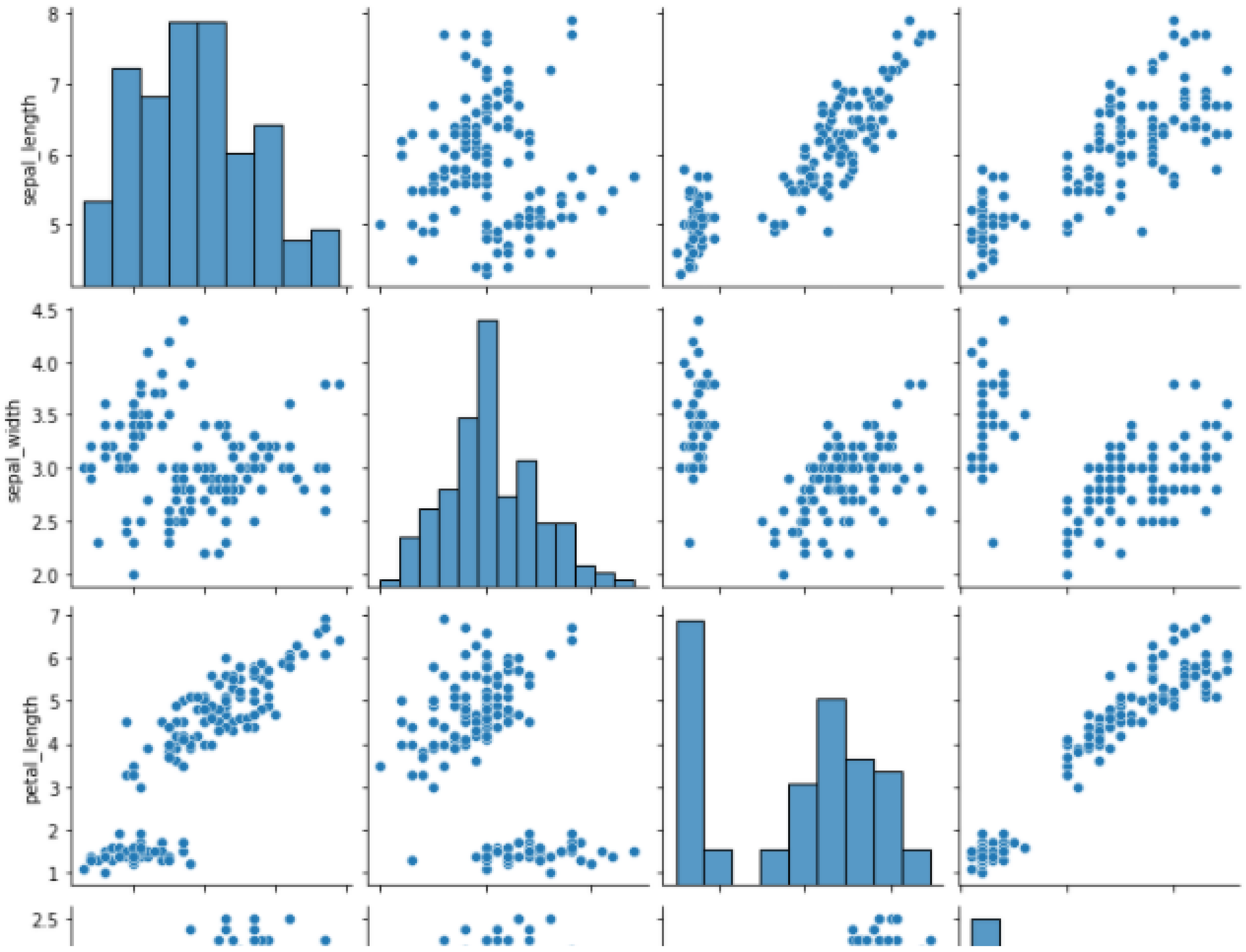
We download the iris data set and load our google colab as a data frame.It has 5 attributes and 4 of them are inputs and species name act as the output.

```
dataset = pd.read_csv('/content/drive/My Drive/Iris_data/iris.csv')
```

4 Analysis

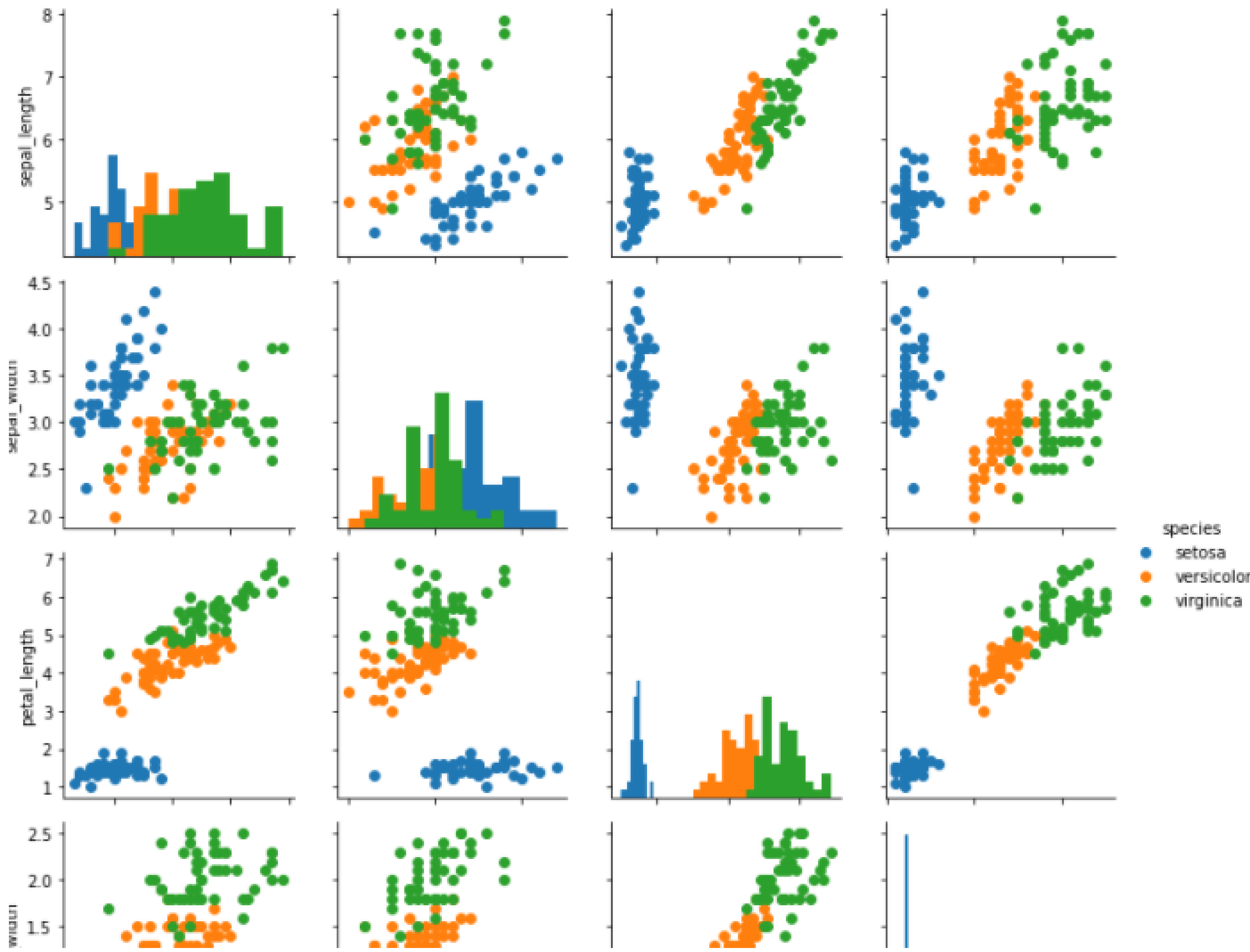
This Part will how to find the species category using the external measurements of the species.First,we Find the pairwise relationship of variables

```
g = sns.pairplot(dataset) # pairplot for each variables
```



We can use these pairwise plot to identify the relationships between variables.After that we execute the same plot for different species.

```
g = sns.PairGrid(dataset, hue="species") # pareplot for each 2 pair of varoab
g = g.map_diag(plt.hist)
g = g.map_offdiag(plt.scatter)
g = g.add_legend()
```



4.1 Inputs and outputs

In this part we separate our inputs and outputs and convert into a array.

```
target_data = dataset[['species']]              # all the data values without sp
input_data = dataset.drop(['species'], axis=1)
input_data = np.array(input_data)# convert list to an array
target_data = pd.get_dummies(target_data.species)# outputs
_, target_data = np.where(target_data==1)
```

```
X = input_data[r[:cut],:]
X_test = input_data[r[cut:],:]
Y = target_data[r[:cut]]
```

```
Y_test = target_data[r[cut:]]

def softmax(x):
    s1 = torch.exp(x - torch.max(x,1)[0][:,None])
    s = s1 / s1.sum(1)[:,None]
    return s

def cross_entropy(outputs, labels):
    return -torch.sum(softmax(outputs).log()[range(outputs.size()[0]), labels

def randn_trunc(s): #Truncated Normal Random Numbers
    mu = 0
    sigma = 0.1
    R = stats.truncnorm((-2*sigma - mu) / sigma, (2*sigma - mu) / sigma, loc=
    return R.rvs(s)
def acc(out,y):
    with torch.no_grad():
        return (torch.sum(torch.max(out,1)[1] == y).item())/y.shape[0]
def GPU(data):
    return torch.tensor(data, requires_grad=True, dtype=torch.float, device=t

def GPU_data(data):
    return torch.tensor(data, requires_grad=False, dtype=torch.float, device=
def get_batch(mode):
    b = c.b
    if mode == "train":
        r = np.random.randint(X.shape[0]-b)
        x = X[r:r+b,:]
        y = Y[r:r+b]
    elif mode == "test":
        r = np.random.randint(X_test.shape[0]-b)
        x = X_test[r:r+b,:]
        y = Y_test[r:r+b]
    return x,y
def gradient_step(w):

    for j in range(len(w)):

            w[j].data = w[j].data - c.h*w[j].grad.data

            w[j].grad.data.zero_()

def make_plots():

    acc_train = acc(model(x,w),y)
```

```
        xt,yt = get_batch('test')

        acc_test = acc(model(xt,w),yt)

        wb.log({"acc_train": acc_train, "acc_test": acc_test})

X = GPU_data(X)
Y = GPU_data(Y)
X_test = GPU_data(X_test)
Y_test = GPU_data(Y_test)
```