Predicting Iris Species

Jean-Sebastien Roger

December 9, 2020

Abstract

In this experiment we were tasked with constructing a procedure which trains a model to predict the species of iris flower based on sepal and petal dimensions. Amongst other factors, we would expect the model we train to pick up on distinguishing factors such as flower size. Not only were we able to train a model which predicts the species of iris, we were also able to test our results against an out of sample test set showing that our model was accurate.

1 Introduction

We begin this lab with an introduction to our data set and a review of our initial problem. Our data set consists of 150 entries, each entry has a unique combination of id, sepal length, sepal width, petal length, petal width, and species. These observations are evenly distributed between the three species, with each species having 50 observations. With this data set, we see the need to be able to predict the species of iris flower based on 4 dimensions (Sepal and Petal width and length).

In Figure 1, we see clear distinctions between iris species when it comes to petal and sepal width and length. Along the diagonal, we see a typical bar graph showing the uni-variate counts for each variable. The rest of the grid plot in Figure 1 shows bi-variate scatter plots which aim to depict the relationship between variables (i.e. petal length plotted against petal width). These scatter plots make it very easy to visualize clear differences between species. The code for this graph is provided in the Appendix.

2 Theory

Once we have trained a model that can predict iris species, we can then use this model to make inferences on new data sets with the proper variables. This gives us the power to take a random set of iris dimensions and conclude whether we are dealing with a setosa, versicolor, or virginica. An important aspect of model training is also being able to speak to the accuracy of the model, this is why it was important for us to also design a way to test

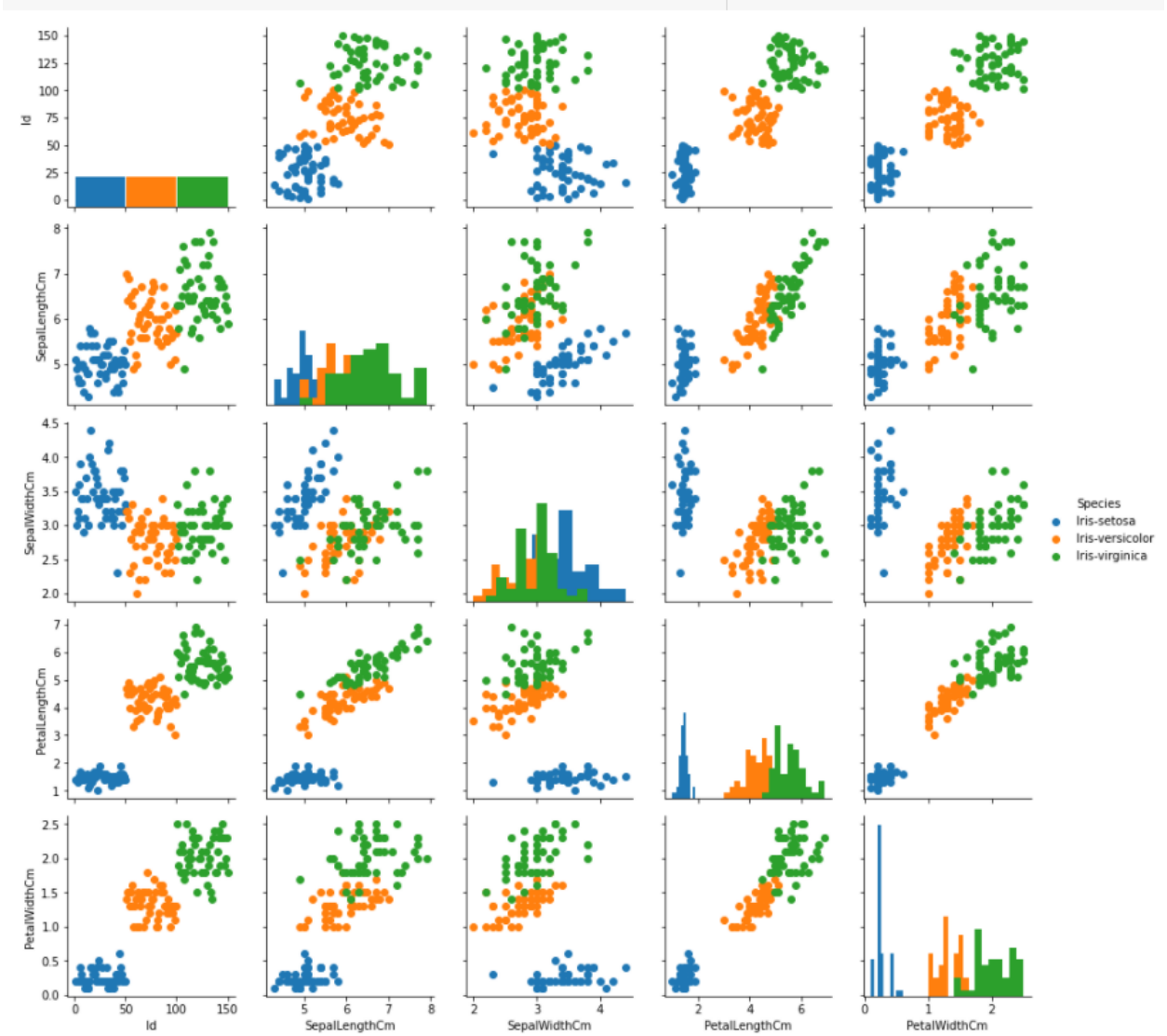

Figure 1: This is a grid of plots comparing each of the groups of species. We can easily see here how the setosa species (blue) seems to have smaller petals sizes on average.

our final model using an out of sample test set. In this experiment, we are setting 20% of our 150 observations aside for testing the accuracy of the model.

In order to properly model the species and form a prediction, we needed to encode our dependent variable, species. We performed this encoding by assigning a number to each unique species, in this case we have setosa being equal to 0, versicolor equal to 1, and virginica equal to 2. From here, all that is left to do is randomize and begin to train the model.

3 Procedures

We begin by reading in the iris data set and loading the mpcr package to ensure we have the functions needed for modeling and plotting.

```
!pip install git+https://github.com/williamedwardhahn/mpcr
from mpcr import *

## Read in iris dataset stored in Data1 folder
dataset = pd.read_csv('/content/drive/My Drive/Data1/Iris.csv')
```

Once we have loaded the data and gotten a general understanding of the structure and contents of the data set, we then proceeded to encode the Species variable using the following code

```
target_data = pd.get_dummies(target_data.Species)

_, target_data = np.where(target_data==1)
```

With our target variable created, we now can proceed with randomizing our data set and splitting into a training set and a testing set. We first randomize in order to remove any bias, then we split into a test and train set by taking the first 80% of records that were randomly reordered.

```
r = np.random.permutation(input_data.shape[0])

cut = int(0.8*len(r))

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

4 Analysis

With our training set prepared, we can now run through our modeling procedure. This procedure uses a model which takes in our a random sample of observations from our training set and a set of normalized random weights and then outputs a prediction on our target. Within the training process we attempt to minimize the cross entropy between our predictions and the true species. During this whole training process, we are plotting the results of our training and saving them so they can be viewed later.

```
wb.init(project="Iris");
c = wb.config

c.h = 0.05
c.b = 20
```

```
c.layers = 3
c.epochs = 2500

c.f_n = [4,16,16,3]

w = [GPU(randn_trunc((c.f_n[i],c.f_n[i+1]))) for i in range(c.layers)]

for i in range(c.epochs):

    x,y = get_batch('train')

    loss = cross_entropy(softmax(model(x,w)),y)

    loss.backward()

    gradient_step(w)

    if (i+1) % 1 == 0:

        make_plots()
```

Now that we have trained a model by adjusting the weights matrix we have created, we can check the accuracy of our modelś predictions first on the training set and then on the testing set.

```
acc(model(X,w),Y)
0.95
acc(model(X_test,w),Y_test)
0.9666666667
```

5 Conclusions

In this lab, we were introduced to our first neural network model with 3 layers. Our goal was to be able to predict the species of an iris flower by using just the dimensions. We also had to put the idea of encoding from the previous lab, into work here in this lab with our Species variable, in order to make sense of the variable and perform mathematical calculations on it, we must to convert it from a typical categorical variable into a numeric variable with a uniform distribution between the three levels. From this experiement, we saw the power of a neural network, and how with very little training we can create a model which makes fairly accurate predictions based on just 4 dimensions of measurement.

The next steps for this lab would be to test this model on another dataset. This gives us the opportunity to calculate additional evaluation metrics through the use of a confusion matrix and other visual methods of analysis.

6 Appendix

```
## Code to create \ref{fig:gridScatter}
g = sns.PairGrid(dataset, hue="Species")
g = g.map_diag(plt.hist)
g = g.map_offdiag(plt.scatter)
g = g.add_legend()

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

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

## randn trunc
def randn_trunc(s):
    mu = 0
    sigma = 0.1
    R = stats.truncnorm((-2*sigma - mu)/sigma,
                        (2*sigma - mu)/sigma,
                        loc=mu,
                        scale=sigma)
    return R.rvs(s)

## acc
def acc(out,y):
    with torch.no_grad():
        return (torch.sum(torch.max(out,1)[1] == y).item())/y.shape[0]

## GPU
def GPU(data):
    return torch.tensor(data,
                        requires_grad=True,
                        dtype=torch.float,
                        device=torch.device('cuda'))

## GPU data
def GPU_data(data):
    return torch.tensor(data,
```

```
                        requires_grad=False,
                        dtype=torch.float,
                        device=torch.device('cuda'))

## get batch
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

## gradient step
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_()

## make plots
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})

## relu
def relu(x):
    return x * (x > 0)

## model
def model(x,w):

    for j in range(len(w)):
        x = relu(matmul(x,w[j]))

    return x
```