Functional

Iris-Lab 3

by Kerry-Ann Bartley Donaldson (Florida Atlantic University)
on November 25, 2020

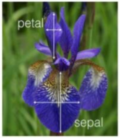
» Introduction

Turning Biology into Mathematics

In this lab demonstration we will access the Iris data set. It consists of 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.

- 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



» Dataset

Data Set Information:
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.

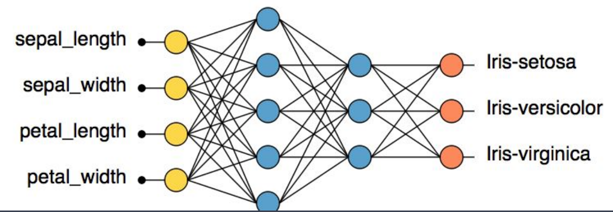
	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

» Predicted Class

Prediction — given new data points, how accurately can the model predict their classes (species)?
Predicted attribute (y): class of iris plant.
Independent (x) variables: sepal length, sepal width, petal length and petal width

» Predicted Class cont'd

Attribute Information:
1. sepal length in cm 2. sepal width in cm 3. petal length in cm 4. petal width in cm 5. class: – Iris Setosa – Iris Versicolour – Iris Virginica
Fig 1.Image of network classification



» Analysis of data using Python Codes

» Analysis of data using Python Codes

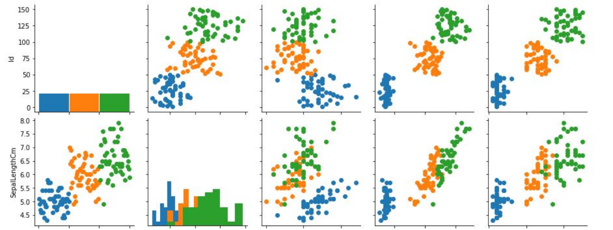
```
# installing pip from instructor's github
from mpcr import
!pip install git+https://github.com
/williamedwardhahn/mpcr
```

» Analysis of data using Python Codes

```
g = sns.pairplot(dataset)

g = sns.PairGrid(dataset, hue="Species")

g = g.map_diag(plt.hist)

g = g.map_offdiag(plt.scatter)

g = g.add_legend()
```

» Analysis of data using Python Codes

Running plot of pair of variables to spot the correlation between variables. Plotting by species, differentiating by colors



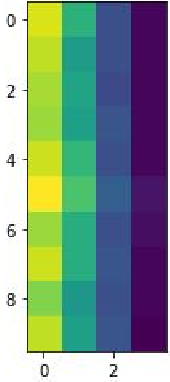
» Analysis of data using Python Codes

```
target_data = dataset[['Species']]

#Defining new datasets,
#target_data is species only from the dataset

input_data = dataset.drop(['Id','Species'],axis=1)
#Defining new datasets,
#input_data is every other column except, ID and sp

input_data = np.array(input_data)
#Using numbers to represent input_data datset
input_data
#show input_data
```

```
plt.imshow(input_data[0:10,:])
```

#show the image of the plot of input data using scale 0 to 10

<matplotlib.image.AxesImage at 0x7fa0853eb630>



```
target_data = pd.get_dummies(target_data.Species)
#defining dataset to classify species
using dummy class 0 and 1

r = np.random.permutation(input_data.shape[0])
# from input_dataset choose 1st column, name it r

cut = int(0.8*len(r))
# Defining cut = integer 80% of the length of r

input_data.shape  #print out the size of input_data

X = input_data[r[:cut],:]
#Defining X and Y so that a comparison of the test

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

» Analysis of data using Python cont'd

```
def softmax(x): #Using pytorch to run the analysis, 
bring in all the programs needed
    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.long()])
    /outputs.size()[0]

def randn_trunc(s): #Truncated Normal Random Number
    mu = 0
    sigma = 0.1
    R = stats.truncnorm((-2*sigma - mu) / sigma,
    (2*sigma - mu) / sigma, loc=mu, scale=sigma)
    return R.rvs(s)
```

» Analysis of data using Python cont'd

```
def GPU_data(data):
    return torch.tensor(data, requires_grad=False,
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
```

» Analysis of data using Python cont'd

```
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)
```

» Analysis of data using Python cont'd

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

def relu(x):
    return x * (x > 0)
def model(x,w):

    for j in range(len(w)):

        x = relu(matmul(x,w[j]))

    return x

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

» Analysis of data using Python cont'd

```
#smoothing out the loss
c = wb.config

c.h = 0.05
c.b = 20
c.layers = 3
c.epochs = 2500

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

» Analysis of data using Python cont'd

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

» Analysis of data using Python cont'd

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

#comparing test model and actual model

X[0]
model(X[0],w)
torch.argmax(model(X[0],w))
Y[0]
w

for i in range(len(w)):
#for all values of w, show the plot
```

THE END