Modelling of Cognitive Processes

Backpropagation

Lesson 10
26/11/2019
Pieter Huycke

Overview

Practical

1. A circular relation: novice
2. Making your own model: journeyman
3. Making your own model: adept
4. Recognizing cats and dogs: expert

Practical

1. A circular relation: novice

A circular dataset

To let you ease into this practical session, we start with a simulated dataset to show you why backpropagation is important.
The dataset was created by us, and you will load in the full dataset later on.
Before we start with the assignment, we will first start with inspecting the dataset.

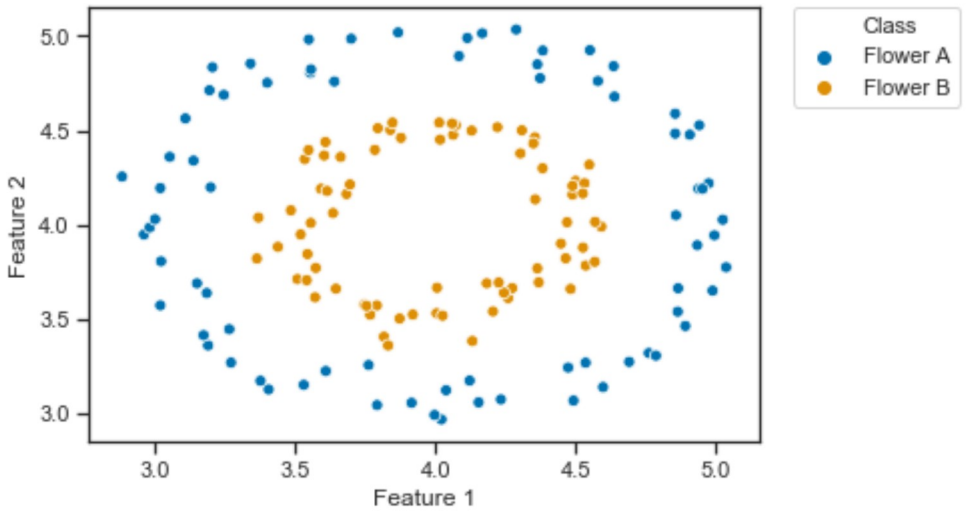
The dataset contains values for 2 different features, and a label for each observation. To stay in line with the previous practical, you can imagine that you have two different types of **flowers**, and that you can classify a flower based on **two different flower features**. To inspect whether we can classify the flowers based on the given features, we look at the plot immediately below.

The plots were made using the Python package `seaborn`.
How the plots were construed is of no importance to you, **only what the plots represent is important**.

In [4]:
```
# plot the data
ax = sns.scatterplot(x       = df['Feature 1'],
                     y       = df['Feature 2'],
                     hue     = df['Class'],
                     markers = ["o", "s"],
                     palette = sns.color_palette('colorblind', 2))

# Put the legend out of the figure
plt.legend(bbox_to_anchor = (1.05, 1),
           loc            = 2,
           borderaxespad  = 0.)
```

Out[4]:
```
<matplotlib.legend.Legend at 0x239c2a7dfc8>
```



The iris dataset: novice

Problem statement

Looking at the plot, we see that flower B is clearly seperated from flower A.
We stored the data in a NumPy array, and saved the data for you to use.

Your task is straightforward: load in the data, and attempt to classify the flowers using both a single-layered Perceptron, and a multi-layered-Perceptron (MLP). Remember that a single-layered perceptron has no hidden layers (like the ones you used in the previous practical session), and an MLP has per definition at least one hidden layer. For now, start with **a single hidden layer** when building your MLP. Manually try a certain amount of hidden units (e.g. 10, 100, 1000).

Try to formulate an answer to the following questions:
Are both models able to classify the flower observations? Is one model clearly superior to the other? Why (not)?
Try to relate your empirical findings back to the course material.

Note: The file "ch5_circular_novice_exercise.py" serves as your starting script, and shows a way to load a downloaded NumPy file.

2. Making your own model: a stroop model

Making your own model: journeyman

Note 1: no start script available
Note 2: mind that the problem statement is two slides long

Problem statement (1/2)

You would like to model a participant that is performing a Stroop task (Stroop, 1935) (https://pure.mpg.de/rest/items/item_2389918/component/file_2389917/content).
Since this is your first time that you build a model on your own, you decide to start with an easy version of this well-known paradigm.

You decide to use two different word stimuli (GREEN and RED), and two different colors (green and red).
Thus, you have the following possible stimuli:

- red
- green
- red
- green

Making your own model: journeyman

Problem statement (2/2)

In line with the classic stroop task, your model should react to either the word dimension or the color dimension.
Only two possible answers are available: LEFT (coded as 1), and RIGHT (coded as -1).

Show your model each possible stimulus 50 times, along with the expected response.
First train your model using a model without hidden units (i.e. a Perceptron), then compare this result with the performance of an MLP with one hidden layer.
Is there a difference?

Finally, try to find **the minimal MLP model**. The minimal MLP model is the MLP that you train 50 times on *different* train data, and where the performance on the test data never drops below 100%.

Try to find an elegant way to find the minimal model (hint: using `break` and `continue` might help you out here).

Making your own model: adept

Making your own model: adept

Problem statement

In the previous exercise, you created a simple Stroop model. However, this model can be extended to fit the experimental findings better. Specifically, the Stroop effect states that human subjects perform better on the word reading than on the color naming. It is hypothesized that this effect occurs because humans are well-trained on reading, but that color naming is a bit 'undertrained'.

Find a way to represent the greater familiarity with reading in your model.

Learning disabilities can also be represented in your computational Stroop model.
How would you represent a subject that has difficulty learning the task?
See whether the model's accuracy decreases with your implementation of a learning disability.

Recognizing cats and dogs: expert

Model building

Similar to the last exercise of the previous practical session, we will work with images to train our model.
The images we will be working with in this case are images of both cats and dogs.
For the sake of parsimony, we will only work with 15 images of each category.

Because image processing is not in our final competences, we did the image processing for you.
Although we don't expect you to come up with a preprocessing strategy yourself, we will shortly explain how we preprocessed our training set.

1. We grayscaled the images
2. We resized the images so that they all have the same dimensions (80×80)
3. We normalize the images by subtracting the mean of each image from each individual image pixel

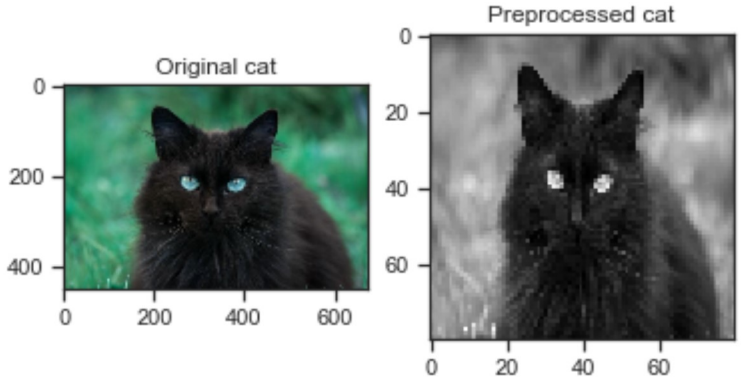
Below, we show an original image of a cat, and its preprocessed equivalent.

In [21]:
```
# define our list of training and test samples
images = os.listdir(TRAIN_DIR)

# original cat
plt.subplot(1, 2, 1)
original_image = cv2.imread(os.path.join(TRAIN_DIR, images[3]))
plt.imshow(original_image)
plt.title('Original cat')

# preprocessed cat
plt.subplot(1, 2, 2)
preprocessed_image = process_image(TRAIN_DIR,
                                   images[3],
                                   DIM_SIZE)
plt.imshow(preprocessed_image,
           cmap = "gray")
plt.title('Preprocessed cat')
```

Out[21]:
```
Text(0.5, 1.0, 'Preprocessed cat')
```



Recognizing cats and dogs: expert

Problem statement

Train a model using the provided images.
The goal of your model building is to create a model that is able to label a new image correctly as a cat or dog.
Use 29 of your images to train the model, and keep the last image to test your model.
Test the following manipulations:

1. Does performance increase or decrease with more hidden layers?
2. How many hidden layers do you need to achieve a classification accuracy of at least 90%?
3. Train a model that has 2+ hidden layers, and test your model on the data it used to train with.
   What happens to your prediction accuracy, and why does this happen?

Use `help(MLPClassifier)` or `print(MLPClassifier.__doc__)` to learn more about the function arguments that can be altered.