W02_05_Itertools_and_Factorial_Design_of_Experiments

September 12, 2022

1 itertools and Factorial Design of Experiments

2 Itertools

`itertools` is one of my favourite modules in the Python standard library. It provides a lot of functions for creating iterators, especially for things like permutations, combinations, and Cartesian products. See https://docs.python.org/3/library/itertools.html for the full documentation including great examples.

2.0.1 Example: Factorial Design of Experiments

Let's pretend we are working with neural networks. Let's pretend we have programmed a neural network training function `NN(alpha, beta)` where the two parameters are numbers that control how it behaves. Suppose it returns one number representing performance (higher is better).

[1]:
```python
import random
def NN(alpha, beta):
    # warning, this is just a placeholder.
    return alpha * random.random() + (
        1 - beta) * random.randrange(2)
```

We want to decide what are the best values for these parameters, i.e. what values gives the highest performance. Suppose we have a few likely values for each, already. We'll use nested for loops as follows.

[2]:
```python
alphas = [0.0, 0.1, 0.2]
betas = [0, 1]
for alpha in alphas:
    for beta in betas:
        print(alpha, beta, NN(alpha, beta))
```

```
0.0 0 0.0
0.0 1 0.0
0.1 0 1.0638748038774313
0.1 1 0.09007449459279822
0.2 0 1.1301372412978836
0.2 1 0.015563778529370187
```

This is called a *factorial design of experiments*. We had two *factors* and we tried every possible value and we can judge which is the best.

However, the program design is bad in one way. What if our colleagues suggest to add another hyperparameter gamma?

[3]:
```
def NN(alpha, beta, gamma):
    # warning, this is just a placeholder.
    return (alpha * random.random() +
            (1 - beta) *
            random.randrange(2) / gamma)
```

[ ]:
```
alphas = [0.0, 0.1, 0.2]
betas = [0, 1]
gammas = [0.9, 0.99, 0.999, 0.9999]
for alpha in alphas:
    for beta in betas:
        for gamma in gammas:
            print(alpha, beta, gamma,
                  NN(alpha, beta, gamma))
```

The addition of the extra hyperparameter means we have to change the structure, adding a for-loop. If we have a lot of hyperparameters, our code will look awful and will be a pain to work with.

[4]:
```
import itertools

alphas = [0.0, 0.1, 0.2]
betas = [0, 1]
gammas = [0.9, 0.99, 0.999, 0.9999]

# "Cartesian product", ie a grid over
# alphas x betas x gammas
for alpha, beta, gamma in itertools.product(
    alphas, betas, gammas):
    print(alpha, beta, gamma,
          NN(alpha, beta, gamma))
```

```
0.0 0 0.9 0.0
0.0 0 0.99 1.0101010101010102
0.0 0 0.999 0.0
0.0 0 0.9999 1.000100010001
0.0 1 0.9 0.0
0.0 1 0.99 0.0
0.0 1 0.999 0.0
0.0 1 0.9999 0.0
0.1 0 0.9 1.1558396000819633
0.1 0 0.99 0.05966784325368675
0.1 0 0.999 1.0717814645906805
0.1 0 0.9999 0.08348786761663249
```

```
0.1 1 0.9 0.026967774875449504
0.1 1 0.99 0.03109578891139805
0.1 1 0.999 0.033408642342887165
0.1 1 0.9999 0.014099061754399622
0.2 0 0.9 0.013180929834195077
0.2 0 0.99 1.0932624835597278
0.2 0 0.999 0.010007774240233958
0.2 0 0.9999 0.154025250487095
0.2 1 0.9 0.009814100695902961
0.2 1 0.99 0.13053908579954676
0.2 1 0.999 0.08986472883750807
0.2 1 0.9999 0.10043362111024019
```

`itertools.product` gives us an iterator over all the possible hyperparameter settings. Highly neat!

2.0.2 Further reading

- The `itertools` docs are good: https://docs.python.org/3/library/itertools.html
- There are some great recipes too: https://docs.python.org/3/library/itertools.html#itertools-recipes
- A study in algorithmic thinking and clear, simple Python from Peter Norvig, examples of comprehensions and generators, and more `itertools`: http://nbviewer.jupyter.org/url/norvig.com/ipython/Golomb-Puzzle.ipynb (By the way, you don't need to understand the parts about coloured rectangles and animations.)