IEMS 308 Lab 2.2

Running scripts from command line

Suraj Yerramilli

January 18th, 2019

argparse Library

The argparse module is a convenient way to write user-friendly command-line interfaces. argparse deals with

- Parsing arguments.
- Generating help and usage messages
- Error handling when invalid arguments are provided

There are broadly two types of arguments: positional and optional

Simple example I

File in repository: argparse_simple.py

```
1  # Simple example which returns the square of the given integer
2  import argparse
3
4  # Define the argument parser
5  parser = argparse.ArgumentParser(description="Returns the square of
       the given integer")
6  # Define arguments
7  parser.add_argument("value",help="The integer whose square is
       needed",type=int)
8  args = parser.parse_args()
9
10 # Print to stdout
11 print(args.value**2)
```

Simple example II

What do the following commands return?

```
python argparse_simple.py --help
```

```
python argparse_simple.py 10
```

```
python argparse_simple.py 10.1
```

Two positional arguments

File in repository: argparse_twoargs.py

```
1  import argparse
2
3  # Define the argument parser
4  parser = argparse.ArgumentParser(description="Returns the given
       power of the given integer")
5  # Define arguments
6  parser.add_argument("value",help="The integer",type=int)
7  parser.add_argument("power",help="The power",type=int)
8  args = parser.parse_args()
9
10 # Print to stdout
11 print(args.value**args.power)
```

Exercise

Complete the script np_summary.py which reads the given file and print either the mean or standard deviation across columns, as specified by the user. To do this, define the argument parser with two positional arguments:

- filepath: a string containing the path to the file
- function: a string from ["mean","sd"] (use the argument choice when defining this argument)

Test your script on the iris dataset located in "../data/iris.csv"

Optional arguments

- name in `add_argument` begins with `--` to indicate that the argument is optional
- Unlike positional arguments, optional arguments can be specified in any order.
- By default, if the optional argument is not used, `None` is stored in the corresponding variable. You can change it by specifying `default=<default_value>`. when adding the argument.
- **Short options**: You can also add a shorter version for the argument using `-<letter>`. For example:

```
parser.add_argument("-s","--slow",...)
```

Optional arguments - Example 1

File in repository: argparse_verbose.py

```
1  import argparse
2
3  # Define the argument parser
4  parser = argparse.ArgumentParser(description="Returns the given
       power of the given integer")
5  # Define arguments
6  parser.add_argument("value",help="The integer",type=int)
7  parser.add_argument("power",help="The power",type=int)
8  # arguments beginining with -- indicate optional arguments
9  parser.add_argument("-v","--verbose",default=0,type=int,
10   help="increase verbosity")
11 args = parser.parse_args()
12
13 # Print to stdout
14 out = args.value**args.power
15 if args.verbose>=1:
16   print("{}^{} equals {}".format(args.value,args.power,out))
17 else:
18   print(out)
```

Optional arguments - Example 2 (flags)

File in repository: argparse_verbose2.py

```
1  import argparse
2
3  # Define the argument parser
4  parser = argparse.ArgumentParser(description="Returns the given
       power of the given integer")
5  # Define arguments
6  parser.add_argument("value",help="The integer",type=int)
7  parser.add_argument("power",help="The power",type=int)
8  # arguments beginining with -- indicate optional arguments
9  parser.add_argument("-v","--verbose",action="store_true",
10   help="verbose output")
11 args = parser.parse_args()
12
13 # Print to stdout
14 out = args.value**args.power
15 if args.verbose:
16   print("{}^{} equals {}".format(args.value,args.power,out))
17 else:
18   print(out)
```

Putting it all together exercise

Complete the script classify.py which trains a classifer on a given dataset and returns the training score. It should take the following arguments:

- filepath: a string containing the path to the file
- header: binary variable ([0,1]) indicating whether the csv file has a header
- classifier: a string from ["logreg","svm","rf"] - type of classifier
- -n,–normalize: flag to normalize the features

Test it on the spambase and banana datasets. Sample commands (from script directory):

```
python classify.py --normalize ../data/spambase.data 0 logreg
python classify.py ../data/banana.csv 1 svm
```