test_LinRegPython

January 16, 2024

1 Testing `LinReg` functionality

This notebook contains a small showcase on how to use the functions inside the `LinReg.py` file.

1.1 Testing

[ ]:
```
import numpy as np
import pandas as pd
import LinReg
```

Read the data:

[ ]:
```
data = pd.read_csv("dataset.txt", header=None)
data
```

[ ]:

	0	1	2	3	4	5	6	7	8	9	…	92	\
0	8.0	1.0	0.19	0.33	0.02	0.90	0.12	0.17	0.34	0.47	…	0.12	
1	53.0	1.0	0.00	0.16	0.12	0.74	0.45	0.07	0.26	0.59	…	0.21	
2	24.0	1.0	0.00	0.42	0.49	0.56	0.17	0.04	0.39	0.47	…	0.14	
3	34.0	1.0	0.04	0.77	1.00	0.08	0.12	0.10	0.51	0.50	…	0.19	
4	42.0	1.0	0.01	0.55	0.02	0.95	0.09	0.05	0.38	0.38	…	0.11	
…	…	…	…	…	…	…	…	…	…	…	…	…	
1989	12.0	10.0	0.01	0.40	0.10	0.87	0.12	0.16	0.43	0.51	…	0.22	
1990	6.0	10.0	0.05	0.96	0.46	0.28	0.83	0.32	0.69	0.86	…	0.53	
1991	9.0	10.0	0.16	0.37	0.25	0.69	0.04	0.25	0.35	0.50	…	0.25	
1992	25.0	10.0	0.08	0.51	0.06	0.87	0.22	0.10	0.58	0.74	…	0.45	
1993	6.0	10.0	0.20	0.78	0.14	0.46	0.24	0.77	0.50	0.62	…	0.68	

	93	94	95	96	97	98	99	100	101
0	0.42	0.50	0.51	0.64	0.12	0.26	0.20	0.32	0.20
1	0.50	0.34	0.60	0.52	0.02	0.12	0.45	0.00	0.67
2	0.49	0.54	0.67	0.56	0.01	0.21	0.02	0.00	0.43
3	0.30	0.73	0.64	0.65	0.02	0.39	0.28	0.00	0.12
4	0.72	0.64	0.61	0.53	0.04	0.09	0.02	0.00	0.03
…	…	…	…	…	…	…	…	…	…
1989	0.28	0.34	0.48	0.39	0.01	0.28	0.05	0.00	0.09
1990	0.25	0.17	0.10	0.00	0.02	0.37	0.20	0.00	0.45
1991	0.68	0.61	0.79	0.76	0.08	0.32	0.18	0.91	0.23

```
1992  0.64  0.54  0.59  0.52  0.03  0.38  0.33  0.22  0.19
1993  0.50  0.34  0.35  0.68  0.11  0.30  0.05  1.00  0.48

[1994 rows x 102 columns]
```

Implement the regressor:

[ ]:
```
regressor = LinReg.LinReg()
```

Now implement a random number generator, and generate a dummy binary array:

[ ]:
```
myRNG = np.random.default_rng()
```

[ ]:
```
rand_ind = myRNG.integers(0, 1, size=data.shape[1], endpoint=True)
rand_ind
```

[ ]:
```
array([1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0,
       0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1,
       0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0,
       1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1,
       0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1])
```

We can use the `get_columns` method of the regressor to get the columns marked as 1 from the `data` and save it in a matrix X

[ ]:
```
X = regressor.get_columns(data.values, rand_ind)
```

We finally use the `get_fitness` method to train, test, and calculate the root mean square error of our prediction using:

- Observations are taken from X: all rows, and all columns except the last one
- Targets are taken from the last column of X

[ ]:
```
regressor.get_fitness(X[:,:-1], X[:,-1])
```

[ ]:
```
0.14500601104190275
```

1.2 Documentation

All methods are well documented via docstrings, which can be understood both by humans and Python. For example, we can use the `help` function:

[ ]:
```
help(regressor.get_fitness)
```

```
Help on method get_fitness in module LinReg:

get_fitness(x, y, rng=None) method of LinReg.LinReg instance
    Return the error of the trained model

    Parameters
```

```
----------
x : an `n x m` matrix of
    Data that should be used for training the model
y : a vector of length `n`
    Regression values of observarions
rng : int, optional
    Random seed, by default None

Returns
-------
float
    The square root of the MSE of the model
```