A(E)3M33UI — Exercise 3: Basis expansion for linear and logistic regression

Petr Pošík

March 11, 2014

The goal of this exercise is to show

- how to make linear models non-linear using basis expansion,
- how to build a simple custom operator in `scikit-learn`,
- how to construct pipelines in `scikit-learn`, and
- to demonstrate how more flexible models achieve lower error rates on the training data.

The program code for this exercise is organized in several Python modules:

- `ex3-1.py`, which contains the main script for the first part of the exercise dealing with regression models,
- `ex3-2.py`, containing the main script for the second part of the exercise dealing with classification models,
- `mpg.py`, which contains the helper functions for loading the data,
- `plotting.py`, which contains helper functions for graphical display of the data and models,
- `mapping.py`, which contains functions for basis expansion,
- `model_evaluation.py`, which contains functions for computing model errors.

After completion, zip all the above files and hand in the archive via the Upload system.
If you will not manage to complete the exercise in the lab, finish it as a homework!

1 Building non-linear models using linear regression and basis expansion

As in previous exercise, we shall work with the `auto-mpg.csv` dataset, and study the relation of horse power and displacement, i.e. you shall build the model $\widehat{hp} = h(disp)$.

Run the `ex3-1.py` script. It shell plot the data and ends up in an error.

Task 1: In module `plotting.py`, fill in the missing code in `plot_1D_regr_model()`, so that the example script will show up the linear predictions.

Hints: Assume that the `model` argument is a `sklearn` predictive model, i.e. that it has a `predict()` method. There is nothing new in this task, you have done this already in the last exercise.

Now, you shall see the predictions of the linear model in the figure. We shall now compute the error of this model, as measured by the mean squared error on the training data.

Task 2: In `model_evaluation.py`, fill in the function `compute_model_error(model,X,y,err_func)`, which takes a trained `model`, the training data `X`, `y`, and the error function `err_func`, and produces the error of the model.

Hints:

- Look at the function `compute_err_MSE()` in `model_evaluation.py`. You should be familiar with it, we implemented it last week.
- Your task is to make the function `compute_model_error` universal in such a way that it can compute the error of any predictive model, as long as a suitable `err_func` is provided by the user.
- In `ex3-1.py`, we supply the `compute_err_MSE` to `compute_model_error` as the `err_func`.

1.1 Basis expansion: polynomials

In this part of the exercise, we shall implement the basis expansion as a transformation usable in the `scikit-learn` pipeline, i.e. it must implement relevant APIs. Namely, it must implement methods `fit()` and `transform()`.

Task 3: In `mapping.py`, implement the class `PolynomialMapping`:

- `fit()` method shall be empty, but shall return `self`, i.e. the current instance of `PolynomialMapping` class, and
- `transform()` method shall take a $[m \times D]$ matrix X, and shall return a matrix of size $[m \times \texttt{max}_\texttt{d}\texttt{eg} \cdot D]$, where the first block of D columns will be just a copy of `X`, the second block shall be X^2 (meaning a matrix of squares of items), etc.

Hints:

- You should be already able to concatenate Numpy arrays using `numpy.hstack()`.
- You can test your solution in the Python shell. By issuing the commands:

  ```
  >>> pm = PolynomialMapping(2)
  >>> pm.transform(X)
  ```

  you should get a matrix with 2 columns, where the values in the second one are squares of the values in the first one.

Now, you shall learn something about `scikit-learn` pipelines. They allow us to chain a series of preprocessing steps (transformations) with a final predictive model, making the efficiently just one larger model, which can be used as a whole. In our case, we would like to build a pipeline of `PolynomialMapping` and `LinearRegression`.

Task 4: In `ex3-1.py`, fill in the missing code to

1. create an instance of `PolynomialMapping` class with degree 2,
2. create an instance of a pipe containing the polynomial mapping and the linear regresion model (which alredy exists in the workspace),
3. fit the pipe,
4. plot the pipe predictions in the graph, and
5. compute the error of the pipe (quadratic model).

2 Summary

Complete the exercise as a homework, ask questions on the forum, and upload the solution via Upload system!