A(E)3M33UI — Exercise 2: Linear regression

Petr Pošík

March 3, 2014

The goal of this exercise is

- to make you more familiar with the general process of model fitting, to
- to help you understand the linear regression model in particular, and to
- to introduce you to the `numpy` package and its features.

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

- `ex2.py`, which contains the main script for the exercise, and
- `linreg.py`, which contains the helper functions and is the place where you shall fill in the majority of your code.

After completion, zip these 2 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 Problem description

In this exercise, we will use the `auto-mpg.csv` dataset, which contains examples of cars and their various features: miles per gallon, number of cylinders, displacement, horse power, weight, acceleration, manufacturing year, origin (USA, EUR, JAP), and the car name.

For the simple regression exercise, you shall study the relation of horse power and displacement, i.e. you shall build the model $\widehat{hp} = h(disp)$.

Task 1: Run the `ex2.py` script. It will end up in an error, but it shall at least plot the data. Try to estimate the parameters w_0 and w_1 of the linear model by hand. Fill them into `ex2.py`.

2 Computing predictions of a linear model given w_0, w_1

From the lecture, you should know that the predictions

$$\widehat{\boldsymbol{y}} = \boldsymbol{X}\boldsymbol{w}^T,$$

where $\boldsymbol{X}$ is the $[|T| \times (D+1)]$ matrix of inputs in homogeneous coordinates, $\boldsymbol{w} = (w_0, w_1)$ is the $(D+1)$-vector of parameters of the linear model, and $\widehat{\boldsymbol{y}}$ is the $|T|$-vector of predictions.

Task 2: In `linreg.py`, create function `homogenize(X)` which

- takes a $[|T| \times D]$ matrix of inputs, and
- produces a $[|T| \times (D+1)]$ matrix of inputs in homogeneous coordinates, i.e. it prepends a column vector of 1s to X.

Hints: Have a look at:

1. The `numpy.ndarray.shape` property. How do you learn the number of rows and columns?
2. The `numpy.ones()` function. How do you create a $[|T| \times 1]$ array of ones?
3. The `numpy.hstack()` function. How do you add the column of ones to the data X?

Task 3: In `linreg.py`, create function `pred_regr_lin(w,X)` which

- takes
  1. a $(D+1)$-vector of linear model parameters $\boldsymbol{w}$ and
  2. a $[|T| \times D]$ matrix of inputs $\boldsymbol{X}$, and
- produces a $|T|$-vector of estimates $\widehat{\boldsymbol{y}}$.

Hints:

1. Do not forget to `homogenize` the inputs.
2. Have a look at the `numpy.ndarray.T` property. How do you get the transposed vector $\boldsymbol{w}^T$? Does the transpose have any effect for vectors?
3. Have a look at the `numpy.ndarray.dot()` method. How do you compute the matrix product $\boldsymbol{X}\boldsymbol{w}^T$?

Now, you should see the predictions of the linear model in the figure. What is the error that your hand-crafted model makes on the data?

3 Computing the error of the linear model

The `linreg.py` module contains function `compute_cost_regr_lin(w,X,y)`, i.e. it implements the $J(\boldsymbol{w}, T)$ function which describes how well the linear model with parameters $\boldsymbol{w}$ fits the data T. It first computes the predictions of the model, and then calls function `compute_err_MSE(y,yhat)` which should return the mean squared error (MSE). This function, however, is not implemented yet.

Task 4: In `linreg.py`, create function `compute_err_MSE(y,yhat)` which

- takes
  1. a $|T|$-vector of true values of y and
  2. a $|T|$-vector of the corresponding predictions $\widehat{y}$, and
- computes the mean squared error (MSE), i.e. the score used by the *least squares method*.

Hints:

- you can compute the sum of squares either
  1. by squaring the differences (`diffs**2`) and then suming them (`numpy.sum()`), or
  2. by using the dot product of the vector of differences with itself (`diffs.dot(diffs)`.
- Do not forget to divide the sum of squares by the number of training examples.

Task 5: Now, when you have an objective measure of the model fit, try a few times and find by hand better values of w_0 and w_1, i.e. values which result in a lower error.

4 Model fitting by minimization of $J(\boldsymbol{w}, T)$

You have just sought for the best values of w_0 and w_1 by hand. We can make the computer to do this operation for us. We shall use the `scipy.optimize` package.

Task 6: In `linreg.py`, create function `fit_regr_lin_by_minimization(X,y)` which

- takes the training data T as input (`X,y`) and
- returns the (hopefully) optimal values of $\boldsymbol{w}$ as a Numpy array.

Hints:

1. Take a look at the `scipy.optimize.minimize()` function. For us, it will be sufficient to use it in the form

   ```
   result = minimize(f, w_init, method='bfgs')
   ```

   where `f` is the function (of a single argument) to be minimized, `w_init` is the initial guess of the weights, and `method='bfgs'` chooses the BFGS optimization method.

2. The resulting values of $\boldsymbol{w}$ can be found in `result.x`.

3. Function `compute_cost_regr_lin(w,X,y)` is the implementation of $J(\boldsymbol{w}, T)$, which we would like to minimize with respect to $\boldsymbol{w}$. However, we cannot pass this function to `minimize` directly, since the call

   ```
   v = f(w_init)
   ```

   must pass without any error. There are two possibilities:

- either define a local function inside the `fit_regr_lin_by_minimization` like this:

```
def fit_regr_lin_by_minimization(w,X,y):
    # Possibly some code
    def f(w):
        return compute_cost_regr_lin(w,X,y)
    # Some other code
```

- or define `f(w)` using the `lambda` notation:

```
def fit_regr_lin_by_minimization(w,X,y):
    # Possibly some code
    f = lambda w: compute_cost_regr_lin(w,X,y)
    # Some other code
```

Then you can use function `f(w)` as an argument to `minimize`.

4. As the initial guess `w_init`, you can use your hand-crafted $\boldsymbol{w}$, or e.g. $(0, 0)$, or $(100, 1)$; it does not matter much in this case.

5 Model fitting using the normal equation

Task 7: In `linreg.py`, create function `fit_regr_lin_by_normal_equation(X,y)` which

- takes the training data T as input (`X,y`), and
- produces optimal values of $\boldsymbol{w}$ as a Numpy array by implementing the equation

$$\boldsymbol{w}^* = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}.$$

Hints:

1. Do not forget to `homogenize` X.
2. You should know how to do matrix-matrix and matrix-vector multiplication using the `numpy.ndarray.dot()` method.
3. Have a look at `numpy.linalg.inv()` function for inverting a matrix.

Now, you should see the predictions of the optimal linear model in the figure. It is probably not very different from your hand-tuned one.

6 Linear model using `scikit.learn` package

In the above tasks, you learned how to fit a linear regression model manually for educational purposes. Of course, in practice you shall take advantage of a ready-to-use package. In Python, one possibility is to use the `scikit.learn` package. It is a collection of functions and classes for machine learning and modeling. Since we shall use it in the forthcoming exercises, let's try the same thing using the `scikit.learn` API.

Task 8: In the respective section of `ex2.py`, fill in the code to train a linear model and get its predictions using `scikit.learn` API.

Hints:

1. Take a look at `sklearn.linear_models.LinearRegression`.
2. The model fitting is done by the `sklearn.linear_models.LinearRegression.fit()` method.
3. The prediction is done by the `sklearn.linear_models.LinearRegression.predict()` method.
4. The parameters of the linear model can be found in `LinearRegression.intercept_` property (w_0), and in `LinearRegression.coef_` property ($w_1, \ldots, w_D$).

7 Multivariate linear regression

So far we have worked with the univariate case, $\boldsymbol{x} = x, D = 1$. Let's try similar thing as above, but for the multivariate case, i.e. try to predict the horse power on the basis of miles-per-gallon, number of cylinders, displacement, weight and acceleration: $\widehat{hp} = h(mpg, cyl, disp, wgt, acc)$.

Task 9: In the respective section of `ex2.py`, fill in the code to load the data for multivariate regression, train the multivariate model and compute its error.

Hints: Do you really need any?

The resulting error shall be lower than in the case of simple regression, since the model uses more knowledge about the cars, and can thus account for more variability in the data.

8 Summary

In this exercise, we have used 3 ways of learning the linear regression model: the minimization of cost function J, the normal equation, and the `scikit-learn` ready-to-use implementation.

The approach based on error minimization is very general and can, in principle, be used with any kind of model. In the next exercises, we shall mostly use the `scikit-learn` methods.

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