Image Interpretation – Assignment 3

This assignment on **Regression** covers the labs from the 11th and 18th of October. The exercises on this sheet are graded by a maximum of 10(+4) points. You will be asked to implement several functions and write a brief report.

In order to submit the results, send a ZIP file with your implemented code (functions prefixed with `ii_` and any "helper" functions you wrote) to

`<mikhail.usvyatsov@geod.baug.ethz.ch>`

with subject

`Image Interpretation 2018 Assignment 3`

no later than on the

`25th of October, 2018.`

Your functions should work when called by the provided test code (functions prefixed with `test_`) which **must not** be modified. When run, they should produce a plausible output, no warnings, and no unnecessary output (remember to end your statements with semicolons). Eventual example output is given in the `ref_`-images. The functions that you write take images as arguments, not image filenames (i.e. do not use `imread` inside the functions). Your functions should not generate figures/plots themselves, the plots are generated by the test scripts.

In addition to the functions, include a brief report (max. 4 pages PDF) explaining the structure of the code and the MATLAB functions used. This includes the reasons for choosing particular functions as well as a short justification of their parameter setting. For the more complicated tasks, the choice of the underlying data structures and algorithms should be explained too. We encourage you to add also diagrams, illustrations, and figures into the report when appropriate, but it is not necessary to copy the related theory from the lecture slides. The report should not contain any code snippets (but the code should contain comments if appropriate).

Team work is not allowed. Everybody implements his/her own code and writes his/her own report. Discussing issues with others is fine, sharing code and/or report with others is **not**. If you use any code fragments found on the Internet, make sure you reference them properly.

The general purpose of this assignment is to do regression manually, without using MATLAB built-in regression functions like `fit`, `regress`, or `polyfit`, so unless specifically stated in the exercise, don't use those functions.

In this assignment, Y stands for a column vector of "target" values, that is the i-th row of Y contains the desired output for the i-th data point. X is a matrix containing the input, where the i-th row contains the input values for the i-th data point. Often we want to add a constant term in our regression methods. For simplicity, this is done by adding a column of ones to our input data, in MATLAB e.g. by doing:

```
X_ext = [ones(size(X,1),1) X];
```

The core of many regression methods is least-squares fitting. If we have a linear model

$$Y = X_{\text{ext}}\theta$$

it is usually over-determined, which means we can **not** find a θ that fulfills this equation exactly. Instead, we are looking for a θ that minimizes the squared error

$$\theta = \arg\min_{\theta'} \|Y - X_{\text{ext}}\theta'\|^2$$

which is done in MATLAB by:

```
theta = X_ext \ Y;
```

(see documentation of MATLAB function `mldivide`).

If we want to use this θ to estimate the values at new input positions, we extend the new input matrix with ones and multiply by θ:

```
X2_ext = [ones(size(X2,1),1) X2];
Y2 = X2_ext * theta;
```

The file `lineardemo.m` in the handout performs this workflow on some synthetically generated data and shows the values of each variable that is changed, so you can watch what happens during evaluation.

Exercise 1. 3 P.

In this exercise you are given a timeseries (just some observations) that was artificially generated (see file *timeseries.csv*). It was obtained combining some known to me and unknown to you polynomial (e.g. $y(t) = t^2 + 2$) plus some random noise. Your task is to find the parameters of the polynomial model that was used.

Use `csvread('./materials/timeseries.csv')` to load the data. In order to visualize data use `plot(timeseries)`.

Polynomial fitting is a traditional application of linear regression. Implement function `ii_fit_poly` that fits a polynomial to the input data. The function should take three arguments: the input data (as a column vector), a column vector of target values, and the desired degree of the polynomial (a non-negative integer). Return your estimated model (theta).

Also, implement function `ii_apply_poly` that takes two arguments: a model (as returned by your function `ii_fit_poly`) and a column vector of points on which to evaluate the model. Return the estimated values as a column vector.

A helper function that extends the feature vector with the polynomial terms (computes X_{ext} from X for a given degree) might be useful, as polynomial fits are also going to be used in later exercises. You can use script `test_poly` to test your implementation.

You have to research which degree works better (in a range [1,10]) and report it.

Exercise 2. 2 P.

Regularization is important for avoiding overfitting on a noisy data input. Implement function `ii_fit_poly_ridge` that fits a polynomial to the input data using ridge regression. It should take the same arguments as `ii_fit_poly`, except for an additional argument which is the regularization strength λ. You can perform ridge regression by replacing the expression `X\Y` (which performs the least-squares estimation) by `ridge(Y,X,lambda,0)`. However, MATLAB function `ridge` which performs ridge regression adds the constant column of ones to the input data itself (so don't do it yourself). The return value should be compatible with `ii_apply_poly`. You can use script `test_poly_ridge` to test your implementation.

Try to regularize your model for the Exercise 1 problem. Experiment how λ influences the quality of your model (to do that you can try to use cross-validation that is not compulsory, but strongly recommended).

Provide the parameters of your best model in the report.

Exercise 3. 3 P.

Here we are going to reconstruct right part of human face from the left part of a human face. We will use linear regression to model the right part of the human face as a function of the corresponding left part. Implement function `ii_predict_face`

that takes two arguments: a training dataset and a query image with no right part of the face. Training dataset consists of full (left and right part are both there) images of size (231×395). From those images you have to extract labels (right parts) and construct a dataset in the way that will allow you to apply squared error minimization method. The images in the provided dataset are stacked along the third axis (if you imagine what is the shape of the data - the answer would be [image length, image width, number of images]). You can visualize an image with function `imshow(image, [])`. The function should return predicted right part with the same shape as the query has (see below).

Using the data from the training dataset, learn a linear regression model with the right part of the image as the output value and the left part of the image as input value. You can use `C1=dataset(:,1:198,:)` to extract the left part of the images in the dataset. In order to get the shape of the data use function `size(data)`.

Your output vector in the linear regression (Y in the intro) should be a $N \times length * WIDTH/2$ matrix, your input matrix (X_{ext}) should be $N \times length * WIDTH/2 + 1$, with ones in the last column and raveled images in the columns left.

After you determined the model θ, estimate the right part of the face using your model. You can use `reshape(v, [rows cols num_examples])` to convert a column vector representation back into the dataset of images.

Estimate the right part of the query by using the previously learned model. Reshape it again to the form of image. You can use script `test_predict_face` to test your implementation.

Exercise 4. 2 P.

Regularization is important for avoiding over-fitting on a noisy data input. For the case of images, the model has too much weights and so, regularization might help to improve results significantly. Implement function `ii_predict_face_reg` that solves the problem from Exercise 3 using ridge regression. It should take the same arguments as `ii_predict_face`, except for an additional argument which is the regularization strength λ. You can perform ridge regression by replacing the expression `X\Y` (which performs the least-squares estimation) by `(`X^T `* X + N * lambda * eye(d)) \ (`X^T `* Y);`. You can use script `test_predict_face_reg` to test your implementation. Research, how the choice of λ contributes to the results. Report your findings.

Exercise 5. (+4 P.)

Solve problems from Exercise 3 and 4 for dataset of bigger images. For this problem, create copy of `ii_predict_face_reg` add to its names postfix *_big*.

The problem here is the size. Analytical method requires expensive operation (matrix inversion, see (Normal Equation)). After the introduction of regularizer, the total size of the problem becomes too big. Estimate the size of matrices for closed form solution of this problem (report it) and you will see that inversion of this matrix is too expensive (it requires $O(n^3)$ operations). Iterative methods provided by matlab are also too slow due to high complexity of the problem (they usually trying to perform matrix decomposition).

What could be done: iterative methods (see gradient descent (GD) and stochastic average gradient (SAG)) or use extremely powerful PC :).

To test your solution use `test_predict_face_reg_big`.