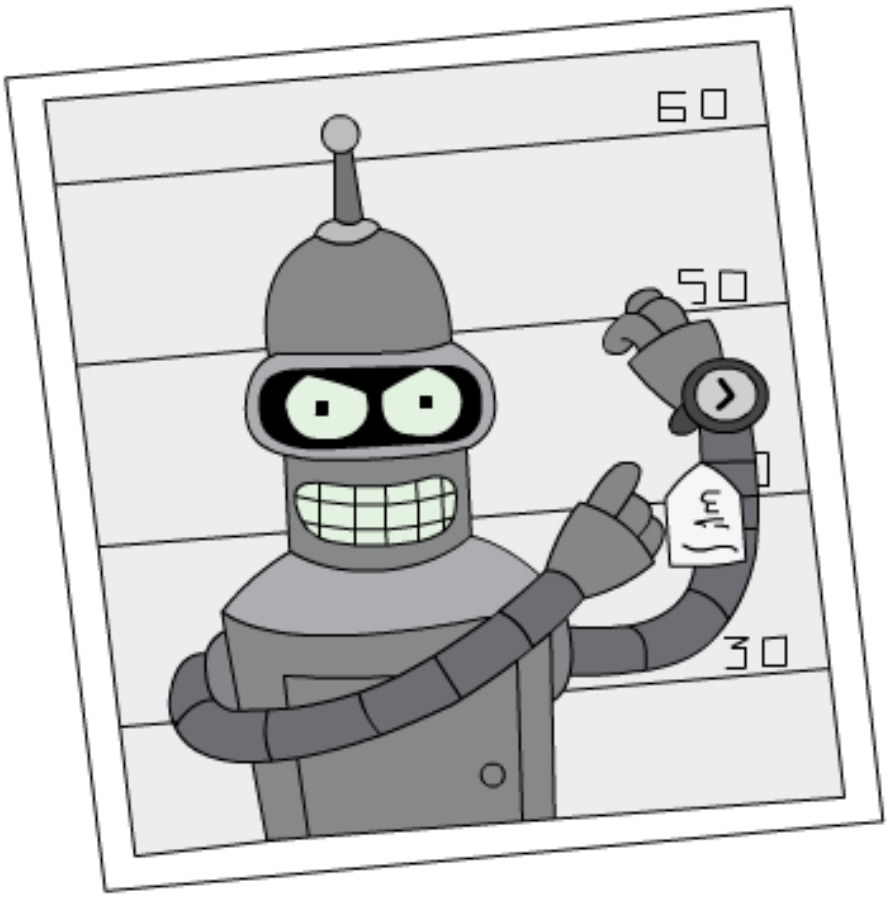
Linear regression

ECE4076/5176 Computer Vision
Lab 4 (Weeks 10,11)



Performance Expectations and Guidelines for Lab Completion

As we progress through the course, we expect you to expand your skills and toolset to tackle more complex and involved problems. This lab has been designed to help you achieve that by providing a comprehensive and engaging learning experience. To ensure that you achieve your best possible results, we recommend that you begin working on the lab well in advance of your lab sessions. Additionally, please note that the computation times required for this lab may vary depending on the efficiency of your code. Sometimes, running your code may take longer than 15 minutes if it is not optimized efficiently. Therefore, we encourage you to put in your best efforts and work diligently to achieve the desired results.

Academic integrity

Every lab submission will be screened for any collusion and/or plagiarism. Breaches of academic integrity will be investigated thoroughly and may result in a zero for the assessment along with interviews with the plagiarism officers at Monash University.

Late submissions

Late submission of the lab will incur a penalty of 10% for each day late. That is, with one day delay, the maximum mark you can get from the assignment is 7.2 out of 8, so if you score 7.5, we will (sadly) give you 7.2. Labs submitted with more than a week delay will not be assessed. Please apply for special consideration for late submission as soon as possible (*e.g.*, documented serious illness).

Lab Instructions and the Use of Generative AI

- You may use NumPy for array handling, and vectorizing your code (reducing the number of for-loops) is encouraged.
- You should use Matplotlib to display images and any intermediate results.
- You may use a generative AI tool or coding assistance. We recommend understanding the concepts and coding as much as possible, as linear regression is a commonly used tool. If you use generative AI tools, please indicate the prompts you used and explain in detail any changes you made to modify the code to complete the assignment.
- Please refrain from using Generative AI in preparing your report, as the purpose of this lab is to assess your understanding of CV concepts.

Lab Grading

Each lab is worth 8%, and there are a number of sections and tasks with their own weighting. A task is only considered complete if you can demonstrate a working program and show an understanding of the underlying concepts. Note that later tasks should reuse code from earlier tasks.

References

- Week 9 Lecture content
- Linear regression
- PSNR

Resources

- *Lab4_student_template.ipynb* - please complete the lab tasks and answer the questions in this template notebook.
- Data task 1: *task1_alc.npy, task1_bmi.npy, task1_gdp.npy, task1_lifeexpt.npy*
- Data task 2: *task2_AU2002_test_alc.npy, task2_AU2002_test_bmi.npy, task2_AU2002_test_gdp.npy, task2_AU2002_test_lifeexpt.npy*
- Data task 3: *train_face_clean.npy, train_face_crpt.npy, test_face_clean.npy, test_face_crpt.npy*
- Data task 4: *train_face_crpt_rdm.npy, test_face_crpt_rdm.npy*

This lab is about linear regression, a simple and widely used method for supervised learning. In particular, linear regression is a useful tool for predicting a quantitative response. The following gives you a short overview of the five tasks you're going to work on:

- Tasks 1 - 2: Perform linear regression on data from a study on life expectancy around the globe. We will use this data to understand how linear regression works. The great thing here is that we use some actual real data instead of synthetic one!
- Tasks 3 - 5: Restore faces from corrupted images. We will use linear regression in order to restore images. You will be amazed at what you can achieve with linear regression, once you've properly modeled a problem.
- Discussion questions: These are at the end of each task within the notebook, and require you to answer them when submitting your python notebook.

Introduction

Linear regression, as the name implies, finds an optimal line (or plane/hyperplane in high-dimensional spaces) describing the relation between the independent and dependent variable (*i.e.*, input data and target output). Recall that we first approached the univariate linear regression in the class and built our way up to multivariate input/output forms. Univariate linear regression describes the situation where only one independent variable is present (*i.e.* $x \in \mathbb{R}$) and the model has to find its linear relationship with the dependent variable (*i.e.* y). In the multivariate form, we have more than one independent variable ($\boldsymbol{x} \in \mathbb{R}^n$) and the model may have multiple outputs to predict ($\boldsymbol{y} \in \mathbb{R}^p$). If you want to have a quick wrap-up, check the following Wikipedia article.

Task 1: Simple Linear Regression

We begin with a simple example. Suppose that we are statistical consultants and want to investigate the factors (i.e. Alcohol, BMI, GDP) affecting life expectancy. We will seek to fit a simple linear regression model, with 'Life expectancy' as the response y and 'Alcohol / BMI, or GDP' as the input x[1].
The linear relationship between x and y can be written as:

$$y \approx w[0] + w[1]x \tag{1}$$

1. From the provided files, use Alcohol data as x to predict life expectancy as y. Write your own **linearRegression_cof()** function, which takes $\boldsymbol{X} \in \mathbb{R}^{m \times n}$ and $\boldsymbol{Y} \in \mathbb{R}^{m \times p}$ to compute the optimal parameter of the model coefficients $\boldsymbol{w} = (w[0], w[1])^\top$. Here m is the number of data samples we have, n is the dimensionality of input $\boldsymbol{x}$ and p is the dimensionality of target $\boldsymbol{y}$. You must **not** use pre-existing linear regression functions (e.g., the one from scikit-learn). Recall that the solution of linear regression can be written as

$$\boldsymbol{W}^* = (\boldsymbol{X}^\top \boldsymbol{X})^{-1} \boldsymbol{X}^\top \boldsymbol{Y} \ . \tag{2}$$

   Here, $\boldsymbol{X}$ is a matrix where data points are stacked in its rows (each row is one data point). Similarly, $\boldsymbol{Y}$ is a matrix where its rows store the desired output of the model. Since we use only Alcohol as x, then $n = 1$. Also, since we are only interested in predicting life expectancy, $p = 1$.

   Hint: You need to include a bias term in your model (i.e., $w[0]$). As discussed in class (see lecture notes), you just need to augment your data by adding a column of ones to the matrix $\boldsymbol{X}$ (add it from the left so the first element of your weight vector shows the bias).

2. Obtain and print the coefficients $w[0], w[1]$ for the "Alcohol" input.

3. Repeat the previous steps, but this time consider "BMI" as input x.

4. Write a function **predict()**, which takes a data point $\boldsymbol{x_q}$ and coefficients $\boldsymbol{w}$, and returns the prediction $\hat{y} = \boldsymbol{w}^\top \boldsymbol{x_q}$.

   Hint: Again you need to be careful about how you use the bias term $w[0]$. One possible solution is to always augment $\boldsymbol{x_q}$ with a constant feature 1 and then to form $\hat{y}_q = \boldsymbol{w}^\top \boldsymbol{x}_{q,\text{aug}}$, where we denote the augmented $\boldsymbol{x_q}$ by $\boldsymbol{x}_{q,\text{aug}}$. Needless to say, you need to put the constant feature 1 at the correct location (so if you augment from left, meaning you add a 1 as the very first element of each training sample $\boldsymbol{x}$, then you need to do the same with $\boldsymbol{x_q}$).

5. Use this **predict()** function to plot y = 'Life expectancy' against x = 'Alcohol' or 'BMI' on your training data. Display the original data as a scatter plot together with your predicted best fit line.

[1] Statisticians use terminology which might be confusing to us, they call inputs "Predictors"! Interestingly, this terminology is used in other disciplines that employ statistical tools (*i.e.* medical studies). So, if you hear predictor, do not assume we are talking about a model that gets an input and predicts something, it is simply the data you use to make predictions!

6. Judging by your visualisations of the data and the regression line, can you explain what the results would mean if you had to use this model to make a prediction? Does the dependency seem reasonable to you, and why / why not? What could be the reason for this dependency in the data?

Task 2: Multivariate Linear Regression

Extend your code and make adjustments so it can fit a multivariate linear regression model. We again use linear regression to fit a model but this time with three features $x[1]$=BMI, $x[2]$=Alcohol, and $x[3]$=GDP. That is,

$$y \approx w[0] + w[1]x[1] + w[2]x[2] + w[3]x[3] \; . \tag{3}$$

1. Print out the regression coefficients $\boldsymbol{w^*} = (w[0], w[1], w[2], w[3])^\top$

Evaluating the performance of Linear Regression model. Up to now, we have fit a regression model on training data. Let's see how good our model is when it comes to data that it hasn't seen during training. Here we test our linear model with data from Australia.

2. Write a function **compute_sse()** which computes the sum of squared errors (SSE):

$$\boldsymbol{L}_{\text{SSE}} = \sum_q ||y_q - \boldsymbol{w}^\top \boldsymbol{x_q}||^2 \tag{4}$$

3. Using the provided test data from year=2002, country = Australia, measure and discuss the following:

   (a) What is the $\boldsymbol{L}_{\text{SSE}}$ if we only use X=BMI to train our model?

   (b) What is the $\boldsymbol{L}_{\text{SSE}}$ if we use $x[1]$=BMI, $x[2]$=Alcohol, and $x[3]$=GDP to train our model?

   (c) Thinking back to Task 1: If you could only choose **one** feature, either X=BMI or X=Alcohol, which **one** would you choose to make a prediction? Explain your reasoning.

   (d) If you wanted to predict 'GDP' from X_1=BMI and X_2=Alcohol, what would you do? Explain your reasoning and the steps to take.

   Hint: There are multiple possibilities, but think about how the variables are related!

Task 3: Image restoration

In the following tasks, you will now use your knowledge of linear regression together with the functions you implemented in the previous tasks to 'complete' or 'reconstruct' images that have been corrupted.
We will work with a subset of the Labeled Faces in the Wild (LFW) dataset that comprises a selection of images depicting the faces of various celebrities. Our selected data set consists of a total number of 273 images, each with height 62px and width 47px. This data is split into 185 training and 88 test images. In this task, you will work on the following four points:

1. Load '*train_face_crpt.npy*' & '*train_face_clean.npy*' as your training data set, and '*test_face_crpt.npy*' & '*test_face_clean.npy*' as your test data set. Display an example image from the corrupted training set ('*train_face_crpt.npy*') and the corrupted test set ('*test_face_crpt.npy*') side-by-side to check the correct data import and to see what you're going to work with.

2. Write a Linear Regression function 'regression_fit()' to fit a model using the data of the training set. Your function should take the corrupted and the uncorrupted images of your training set as arguments (you try to predict the clean image from the corrupted one).

3. To check how well your model works on unseen data, you will now use your regression model to predict the missing pixels of the faces from the unseen test set. Your 'predict()' function will take in the unseen corrupted samples and previously learnt regression coefficients, and return the restored images. For the 9th image of the test set (*i.e.* image[8]), display the uncorrupted original version, the corrupted version and your reconstructed one side-by-side to compare!

4. Repeat step 3 for the training images, *i.e.* perform image completion, and display the results for the 6th image of the training set (*i.e.* image[5]) in the same side-by-side way as before.

5. What do you observe if you compare the image quality of the restored test image to the restored training image? Why does our regression model perform differently for these two sets?

Task 4: Peak Signal-to-Noise Ratio PSNR

PSNR (Reference) is defined as the ratio of the maximum possible power of a signal to the power of corrupting noise that affects representation fidelity. PSNR is most easily defined via the mean squared error (MSE). Given a noise-free $H \times W$ monochrome image I and its noisy approximation K, the MSE is defined as

$$\text{MSE} = \frac{1}{H \times W} \sum_{i=0}^{H-1} \sum_{j=0}^{W-1} [I(i,j) - K(i,j)]^2 \tag{5}$$

$$\text{PSNR} = 20 \cdot log_{10}(\max(I)) - 10 \cdot log_{10}(\text{MSE}) \; . \tag{6}$$

In this task, you will concentrate on the following points:

1. Implement a function 'PSNR()' that takes in two images to measure the Peak Signal to Noise Ratio between them. In our case, we want to measure the PSNR between a reconstructed image and its original uncorrupted version. **Hint:** Make sure to convert the image intensities to integer values ([0,255]) for this computation.

2. We are again considering the 9th image of the test set from Task 3 (i.e., image[8]). Use your implemented function to measure the PSNR of your restored version of this test image. Additionally compute the PSNR of the corrupted version of this image (the 'raw corrupted' test data).

3. Now, load the additionally provided data samples '*train_face_crpt_rdm.npy*' and '*test_face_crpt_rdm.npy*', and visualise the 6th image of this new train set (*i.e.* image[5]) and the 9th image of this new test set (*i.e.* image[8]). If you compare these two images to the ones loaded in Task 3, you will notice that this new data is almost identical to the previously used data (ignoring the uncorrupted 'frame'), but is corrupted with a **random pattern** for each image, whereas the previously used images were all corrupted by the exact **same pattern**. (You can check by inspecting the train and test images you displayed in Task 3.)

4. Repeat the regression fitting, prediction and visualisation from Task 3 on these new data samples.

5. Now again, use your implemented function to measure the PSNR of your restored version of the 9th test image (*i.e.* image[8]) and additionally compute the PSNR of the 'randomly' corrupted version of this image (the new 'raw corrupted' test data).

6. Compare your PSNR and visual results from step 5 with the ones from step 2! Can you explain what might have changed that led to differences between the restored images, and why this happens?

Task 5: Reconstruction from local neighbourhood

We realised in Task 4 that random corruption patterns for each image might be difficult to handle. To further improve the quality of the output, you can restore corrupted pixels, one by one, based on their neighbourhood. For example, you can predict the value of a corrupted pixel by using a window of size 5×11 centered directly above the current pixel – meaning you can scan your image from top to bottom in a raster format, and restore corrupted pixels accordingly. (This is also the reason why our 'randomly corrupted' images have an uncorrupted 'frame' of 5 pixels.)
To accomplish this task, you need:

1. A function 'extract_patch()' to extract patches of a specific patch size directly above a pixel of interest.
2. Use your patch extraction function to create a new training data set as follows:
    a) For each corrupted pixel in the training images, extract a patch from the respective uncorrupted training image directly above the pixel using your 'extract_patch()' function.
    b) Additionally store the 'correct' ground-truth value of the corrupted pixel.
3. Using this new data set, train a linear regression model.
4. Evaluate your model on the unseen randomly corrupted images and compare to your previously obtained results.
5. Display the PSNR values for tasks 3 and 5 for the the 9th test image (*i.e.* image[8])

Hint: To further improve your results, you can apply a low-pass filter (*e.g.*, a simple mean filter). If you are hooked, see Figure 1 for what we have obtained by doing the above (right-most column with filtering).

Discussion Questions

These are discussion questions that need to be answered when submitting your python notebook.

Figure 1: **Restored images** From left to right: Corrupted image taken as input; Restored version using the method of Task3; Restored result using a local window approach; Image restored via local window approach followed by additional filtering