ISTA 421 / INFO 521 – Homework 2

Due: Friday, Oct 6, 8pm

20 points total Undergraduate / 25 points total Graduate

Kargi Chauhan

STUDENT NAME

Undergraduate / Graduate

Instructions

The purpose of this set of exercises is to implement the generalized matrix form of the normal equations for linear regression, implement and use cross-validation, and gain experience interpreting simple regression results.

In this assignment you are required to write 2 scripts in Python. They will be submitted as 2 separate files, although you are free to copy any parts of code from one of the provided scripts to the other as desired. The names for the script files are specified in problems 1 and 3 below.

Included in the Homework 2 release are two sample scripts:
`fitpoly.py` and `cv.py`
and two data files:
`womens100.csv` and `synthdata2023.csv`

The sample scripts are provided for your convenience — you may use any part of the code in those scripts for your submissions. Note that neither will run as-is—you must fill in the calculation of $\mathbf{w}$. The data files are provided in CSV (comma separated values) format. The script `fitpoly.py` contains the function `read_data` which shows how to load the data files into a numpy array.

All exercises after Exercise 1 require that you provide some **written** answers. In some of these, you must also include figures. **Always** label any figure axes and include an informative figure caption with each figure. You will submit a PDF of your written answers. You can use LaTeX or any other system (including handwritten). Plots must be program-generated. The final version must be in PDF format and any handwritten answers MUST be legible or we will not grade it.

The final submission will include, at the minimum:

- the two scripts
- a PDF version of your written part of the assignment. The PDF should satisfy the following requirements.
  - It must contain programmatically-generated plots.
  - It must be named `hw2-answers.pdf`.

Both the scripts and the PDF must be added, committed, and pushed to your Github Classroom repository before the due date/time.

NOTE: Problems 4 and 5 are required for Graduate students only; Undergraduates may complete them for extra credit equal to the point value.

FCML refers to the course text: Rogers and Girolami (2016), *A First Course in Machine Learning, Second Edition.*

For general notes on using LaTeX to typeset math, see: `http://en.wikibooks.org/wiki/LaTeX/Mathematics`

1. [5 points] Adapted from Exercise 1.2 of FCML p.35:

In this exercise you will complete the implementation of the function `fitpoly` in `code/fitpoly.py` to ensure that it can compute the best-fit parameters **w**, (a vector of parameters), for a linear model with one feature per observation. After you have implemented `fitpoly`, complete the implementation of the function `exercise_1` to fit to the Women's 100 meter dash data.

- `fitpoly` takes as input an array, `x`, representing all of the feature observations, and the array `t` representing the corresponding response values. The elements at index 0 of `x` and `t` together represent the first feature/response "training" pair, index 1 of `x` and `t` represents the second "training" pair, etc.
- `fitpoly` is used in `exercise_1` to only fit a simple line to the data (i.e., you only need to fit parameters w_0 and w_1). However, in the later exercises (such as Exercise 2) you will need to fit higher-order polynomial models (e.g., $t = w_0 + w_1 x + w_2 x^2 + ...$). For this reason, you must make your implementation of `fitpoly` generalized to handle higher-order polynomials.
- The parameter `model_order` is used in `fitpoly` to represent the non-negative integer, n, that in turn represents the highest-order polynomial exponent of the polynomial model:

$$t = w_0 x^0 + w_1 x^1 + w_2 x^2 + ... + w_n x^n$$

- The implementation of `fitpoly` starts with a construction of the *design matrix*, **X**: the representation of input features as a matrix, where each row in the matrix corresponds to an individual observation and the columns represent the features for each individual. See the provided comments in `fitpoly` for an explanation of how the design matrix **X** is represented.

Your first task in this exercise is to implement the matrix normal equations to compute the parameter vector **w** (3$).

Note:

- The `fitpoly.py` file includes three helper functions (Utilities) to read data, plot data, and plot the model (once you've determined the weight vector **w**), but the function for computing linear least-squares fit is *incomplete*. `fitpoly` takes as input the (one-dimensional) data vector **x**, the target values vector **t**, and a non-negative integer `model_order` that represents the highest polynomial order term of the model; `fitpoly` is intended to return the **w** vector (as a numpy array).
- The Appendix to HW 1 has a brief tutorial on working with numpy arrays.
- The objective of this exercise is for you to implement the linear least squares fit solution (i.e., the normal equation) in its general linear algebra form. **DO NOT** use existing least squares solvers, such as `numpy.linalg.lstsq`, or scikit learn's `sklearn.linear_model.LogisticRegression` as your implemented solution; however, it is certainly fine to use those functions to help you *verify* your implementation's output (but don't submit those as your solution).

Test your implementation of `fitpoly` by completing the implementation of the `exercise_1` function.

- Table 1.3 (p.13) of FCML lists the women's 100m gold medal Olympic results. This data is provided in the file `womens100.csv` in the `data` folder. Use the provided function `read_data_fit_plot` to find the 1st-order polynomial model (i.e., a line with parameters w_0 and w_1) that minimizes the squared loss of this data.
- Note that the call of `exercise_1` by the TOP LEVEL SCRIPT at the bottom of the file will already provide the path to the data file; you need to fill in the rest of the parameters to `read_data_fit_plot`.

Solution 1:

$$t = 40.924 - 0.015x$$

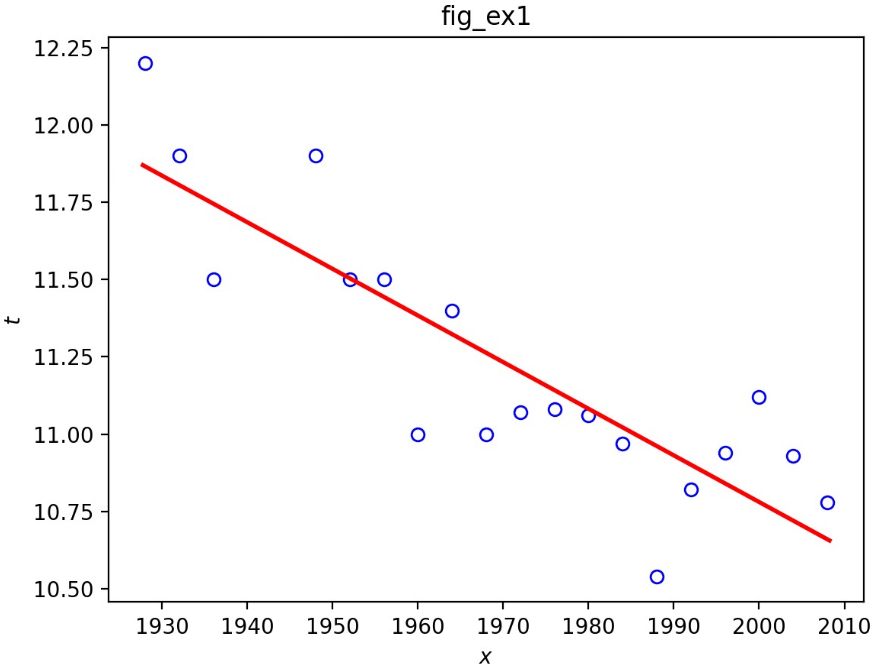


Fig 1: Solution for Ex1 (1st order polynomial mode)

The equation '$t = 40.924 - 0.15x$' represents a linear model which states the relationship between the independent variable 'x' and the predicted time 't' for the data of the women's 100-meter dash data

40.924 is the intercept of the models.

However, if x represents the year of olympic events then this equation suggests that the winning time has been seen decreasing by 0.15 sec per year on average.

- Choose parameters to `read_data_fit_plot` to save your plot figure in the `figures` directory.
- Report the model here as an equation (e.g., $t = 2 + 4x \quad \leftarrow$ this is just an example); you only need to report parameter values up to 3 decimal places.
- Plot the data with your best-first model and include the plot in your answer (label axes and include an informative caption!) (2$).

Solution.

2. [3 points] Adapted from Exercise 1.9 of FCML p.36:

Now fill in `exercise_2` similar to `exercise_1`. This time you will load the data stored in the file `synthdata2023.csv` (in the data folder) and fit a 3rd order polynomial function – $f(x; \mathbf{w}) = w_0 + w_1 x + w_2 x^2 + w_3 x^3$ – to this data. There are 30 observations in this data. Report the best-fit model parameters as an equation. Plot the data and your model and include the plot in your answer (be sure to include an informative caption to your plot).

Solution.

3. [12 points]

The script `code/cv.py` is an *almost complete* implementation of the demonstration in Chapter 1 of FCML, pp.31-32, of cross-validation (CV). The "top-level" function for the demonstration is called `run_demo`. In the demonstration, the provided function `generate_synthetic_data` is used to generate synthetic data, and then CV is used to assess the best fit of polynomial models across different polynomial model orders given only 30 total observations. In the demo, the code also contrasts the cross-validation results with results from an *independent* set of data (sampled from the *same* data generating process in `generate_synthetic_data`) that is significantly larger (in this case, 1000 observations).

The purpose of the demonstration is both to show an example of how cross-validation for model search is implemented, and to show that even though cross-validation might work with a very small data set (in this case, 30 observations), its results are similar to evaluating with much more data (1000 observations). Read the description in FCML Ch. 1 and make sure you understand how this has been implemented in the code.

The core of the implementation is in the functions `run_cv` and `run_demo`. The global variable $\widehat{\text{R}}$UN_DEMO is a boolean that controls whether the demo is run (by default this is set to `True`, on line 28 of `code/cv.py`).

I say that this is an *almost complete* implementation because it is missing one step: the calculation of the best-fit values for $\mathbf{w}$. You will need to fill this in yourself at the indicated location in `run_cv` ($). The calculation here should be the same solution you used for calculating `w` in `fitpoly` in Exercise 1. Before you implement this, when you run the script, it will generate one plot (of the generated data) and print the following two lines to the terminal:

```
run_demo(): best_polynomial model order: None
run_demo(): min_mean_log_cv_loss: None
```

Once you have filled in the calculation of `w`, then when you run the demo, it will generate two figures, and the best polynomial model order and the minimum log cv loss (explained below) will no longer be printed to the terminal as `None`.

Solution 2

$$f(x;w) = 3.86 + 51.36x + 19.71x^2 - 8.05x^3$$

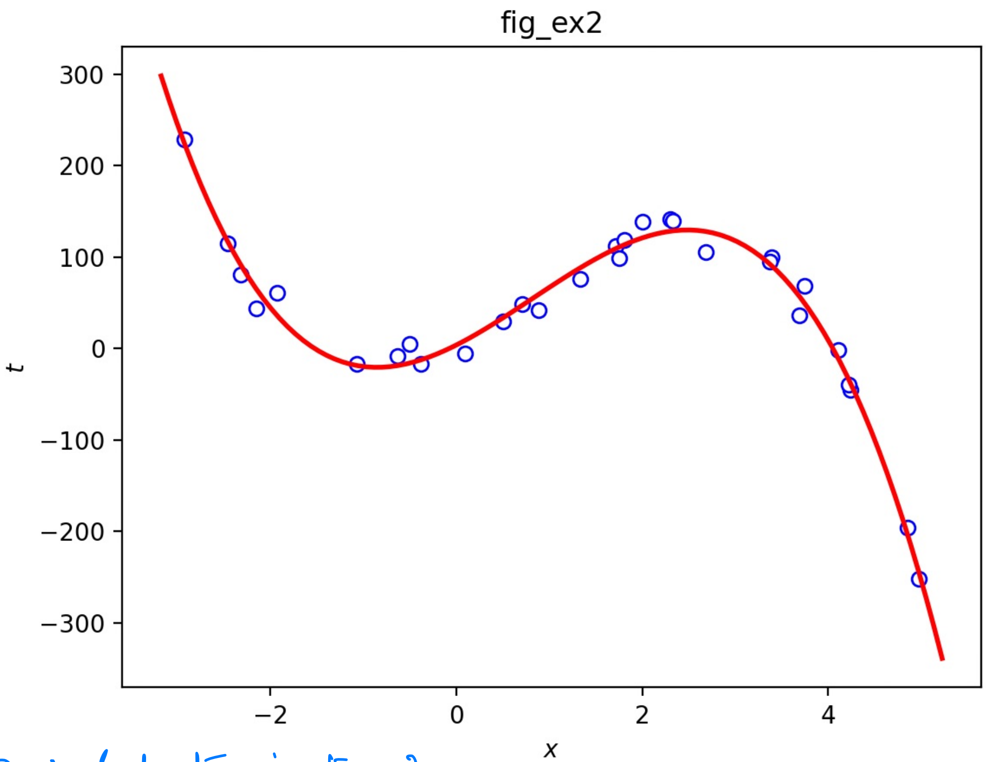


Fig 2: Solution Ex2
(3rd order polynomial model)

$f(x;w)$ represents the predicted value of output variable. '3.86' is the baseline value representing value of output variable without affecting input variable. '51.36' is the linear component of the model. '$19.71x^2$' is quadratic of model '$-8.05x^3$' cubic component of model. However, this 3rd order polynomial model helps in understanding relationship.

The first figure generated by the demo shows a plot of the synthetically-generated 30 data points (blue circles) used for cross-validation; the green line represents the function (the mean of the "generating process") from which these were sampled (the $*$ on the t indicates that these are response values of t from the "true" function):

$$t^* = x + 5x^2 + 2x^3$$

The random samples (blue circles) consist of x values uniformly randomly sampled between -5 and 5, and t sampled from a Gaussian (Normal) distribution with mean t^* and standard deviation `sigma`: $\mathcal{N}(t^*, \texttt{sigma})$.

The second figure shows three plots of the log mean squared error (MSE) of the loss as the polynomial order of the model varies from 0 to 7. The first plot is the log MSE loss of the models on the training data. As this is cross-validation, this is the *mean* loss across each of the fold training data. In `run_demo` you can see the number of folds, `K`, is set to 10, so this is 10-fold cross-validation. Because we have a total of 30 data points, and $K = 10$, then the training set of each fold will have 27 data points, and the CV Train Loss will be assessed for fit on the 27 data points in each fold training set; there are 10 such training loss calculations (one for each fold), so the reported mean in the first plot is then the (log) mean of those 10 losses based on the CV training sets. The second plot shows the log MSE loss for each model order where the CV Test Loss is now computed based on the 3 held-out data points within each fold, and there are 10 such folds, so the mean is across those 10 losses. Finally, the third plot shows the log MSE loss on the completely independent test dataset (still generated from the same "true" function) consisting of 1000 data points, so the mean loss (for each model order) is based on the model fit to the 1000 points in the independent test set. As can be seen, the Independent Test and CV Test loss are both similar U-shaped curves, and there is a trend toward a minimum loss around model order 3 (which happens to be the order of the polynomial used to generate both the CV and independent test set data). The CV Train Loss, however, keeps getting smaller as the model order increases, because the model is getting to fit to exactly the same data used to report the loss, and in general as the model order increases, it will fit better to that data due to increased flexibility of the model (given its higher order polynomial flexibility).

Note that the first step in the top-level `run_demo` function is to set the random seed (as you did in Homework 1), and the call to `run_demo` in the TOP LEVEL SCRIPT by default sets this to 29. This makes it so that each time you execute the script, you get the same results, even though you are sampling from the random number generator. If you change the random seed, you will get different behavior, which is expected. Give it a try! You'll see that some times the CV and Independent Test Losses don't always have a minimum at model order 3. And some times the CV Test and Independent Test loss across model orders can go up and then down again.

Now to the main task. In this Exercise, you are tasked with implementing K-fold cross-validation to perform model selection, in this case to search for the model of polynomial order (between orders 0 and 7) with the best predictive error for the data in `data/synthdata2023.csv`. You can use and adapt any part of implementation of the demo. A key difference is that unlike the demo, you are working with the given data in `data/synthdata2023.csv`, so **you will *not* be generating your own synthetic data, and there will *not* be an independent test set of data** – you will just focus on implementing cross-validation (with its train and test data splits within each fold), without an *additional* independent test set to compare against (as was done in the demo).

Run your implementation under two conditions:

1. 5-fold cross-validation, to be performed in the function `exercise_3_5fold`
2. Leave-One-Out cross-validation (LOOCV), to be performed in the function `exercise_3_LOOCV`

It is recommended, but not necessary, that you create a single function that can run your cross-validation experiment and return the needed values, so that you don't have to repeat a lot of code

Solution 3:

$$f(x;w) = 3.86 + 51.36x^2 + 19.17x^3 - 8.05x^4$$

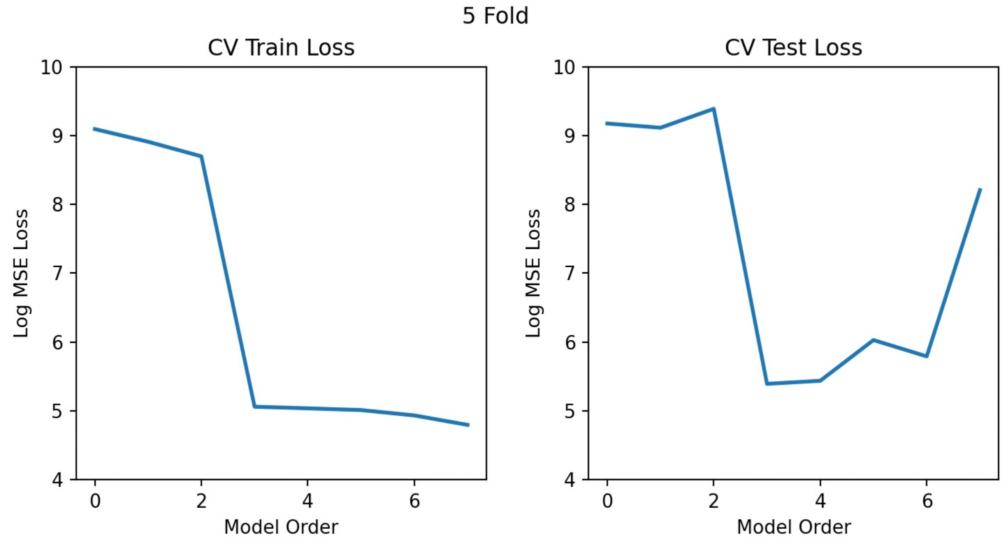


Figure 3: Solution ex3_5 fold cross-Validation best-fit model

5-fold CV

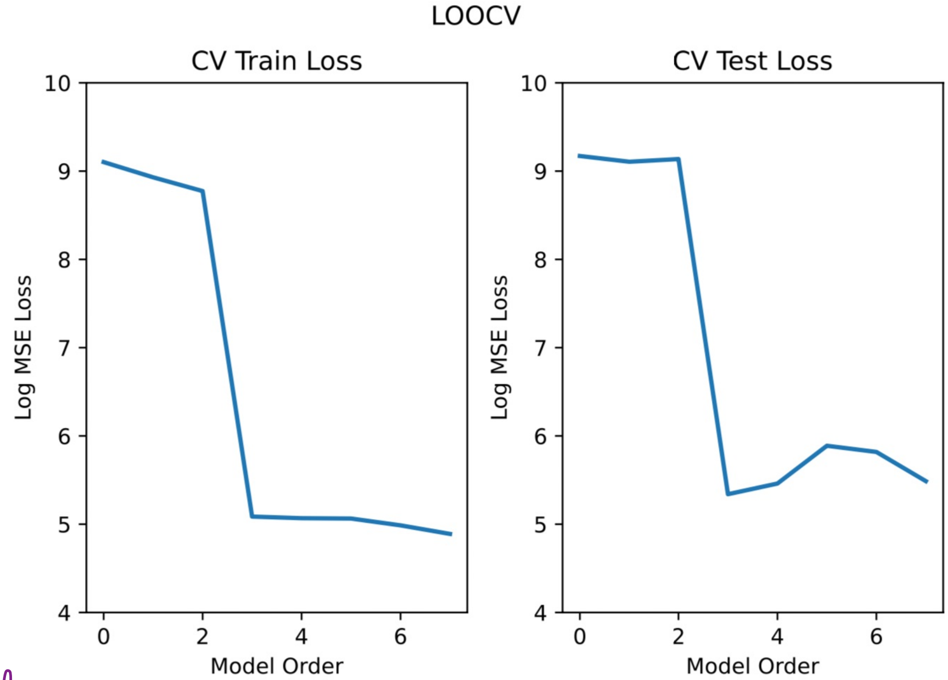
best model order: 3

best model log MSE CV test loss: 5.39128807670057

best model parameter:

[3.863364 51.36047167 19.71377298 -8.0536747]

This figure shows 5 fold K data of cross-Validation for Synthdata 2023. The loss plot shows that MSE loss get smaller as the model order increases until it reaches 3 then it stables. Even the CV test loss plot shows same features as the CV test loss plot goes ahead to reach the log MSE loss it starts to increase showing how the increasing affects the model if it goes over.



best model order: 3

log_mean_train_loss:

[9.09490804 8.91105138 8.69898656 5.0590038 5.035474
5.0106384 4.93234048 4.79491944]

log_mean_cv_loss:

[9.17617108 9.11471333 9.38980011 5.39128808 5.43592685
6.02794582 5.79100666 8.20780955]

minimum mean_log_cv_loss of 5.391 for order 3

LOOCV result

best model order: 3

best model log MSE CV test loss: 5.391288076700057

best model parameter:

[3.863364 51.36047167 19.71377298 -8.05367471]

This figure demonstrates the cross-Validation of synthdata 2023. The given CV train plot shows that log MSE loss get smaller as the model order increases same as the 5 fold when it reaches 3 it gets stable. Even the CV loss plot does the same showing that MSE loss starts to increase at 4 showing how increasing the model order beyond 3 causes the risks of overfitting in the model. However, the CV plot and LOOCV shows that which model of both helps in understanding model with fitting data providing it analyses. Although, LOOCV is more maintence because of the large datasets.

between `exercise_3_5fold` and `exercise_3_LOOCV`.

For both cases (again, possibly by a call to a single function), you are asked to *randomize* the order of your data. Although the demo does not do this by default, the implementation does include code to show you how you can randomize the order of the data before you then perform your cross-validation. The reason for asking you to do this here is that as long as you are not working with data that has a natural order dependency (such as time series), it is generally good practice to randomize, especially if you are not sure that the order of your data has already been randomized. In this case, the data is independent data, so it is good to randomize. If the order is not randomized, then the estimate of the generalization error being made by cross-validation could be biased. Because we still want to allow for reproducibility (this helps automating grading), set the random seed to 29 before you proceed with your randomization.

In both `exercise_3_5fold` and `exercise_3_LOOCV`, based on your cross-validation results, find and report the following in your written solution:

1. The polynomial model order that was found to fit best overall to the synthetic data,
2. The log mean squared error (MSE) for the CV Test data of that best-fit model order (i.e., for that best-fit model, calculating the loss on each of the held-out test data in each fold, and reporting the mean across the folds).
3. The model parameters for the best fit model of the best-fit model order

A pytest unit test is provided to test the results of Exercise 3 (for both `exercise_3_5fold` and `exercise_3_LOOCV`) against the results that I get in my reference implementation. HOWEVER, note that because there are many different ways that you might end up calling the random number generator, and thus affecting the outcome, you may have a correct implementation and yet these tests still fail. The tests are provided as a guide to help you know when you have a definite solution, but if they are not passing, that does not mean you do not have a viable solution. I wish there was a way to provide a more "relaxed" guide to tell you when you have achieved *some* viable solution, but unfortunately that is not possible to fully automate.

Finally, for `exercise_3_5fold` and `exercise_3_LOOCV`, you must also generate plots of the cross-validation results, similar to what is done in the demo, showing the CV Training log MSE loss, and the CV Test log MSE loss, across the 8 different polynomial model orders: 0..7. In total, this means you will provide the following: (1) 5-fold CV Training with (2) related Test loss, and (3) LOOCV Training, and (4) Test loss. NOTE: You can use the provided `plot_cv_results` function to plot the just the CV Training and CV Test loss (whether 5-fold or LOOCV) as a pair of plots – in this case, if you pass the value `None` to the argument for `ind_loss`, the function will skip rendering the third "independent test" plot (this was used in the demo). In hw2, you must save these figures to the '`figures`' directory, and also include them in your PDF submission (this means you must add, commit and push them in your final submission).

If you use `plot_cv_results`, save your figures as follows:

- For 5-fold CV, save the combined plots in '`figures/synthetic2023-5fold-CV.pdf`'
- For LOOCV, save the combined plots in '`figures/synthetic2023-LOOCV.pdf`'

There is again a unit test to test whether these plots have been generated. If you instead generated the plots under a different name, the unit tests will fail, but as long as you create the plots and include then in your written PDF submission, you will get full credit.

Solution.

4. **[2 points – Required only for Graduates]**
   Exercise 1.10 from FCML p.36

   Derive the optimal least squares parameter value, $\hat{\mathbf{w}}$, for the total training loss:

   $$\mathcal{L} = \sum_{n=1}^{N} \left(t_n - \mathbf{w}^\top \mathbf{x}_n\right)^2$$

   How does the expression compare with that derived from the average (mean) loss? (Hint: Express this loss in the **full** matrix form and derive the normal equation.)

   Solution.

5. **[3 points – Required only for Graduates]**
   Exercise 1.11 from FCML p.36

   The following expression is known as the *weighted* average loss:

   $$\mathcal{L} = \frac{1}{N} \sum_{n=1}^{N} \alpha_n \left(t_n - \mathbf{w}^\top \mathbf{x}_n\right)^2$$

   where the influence of each data point is controlled by its associated parameter. Assuming that each α_n is fixed, derive the optimal least squares parameter value $\hat{\mathbf{w}}$. (Hint: When expressing in the full matrix form, the *alpha*'s become a matrix...)

   Solution.

Solution 4 :

The total training loss :

$$L = \sum_{n=1}^{N} \left(t_n - w^T x_n \right)^2$$

In matrix form :

$$L = (t - Xw)^T (t - Xw)$$

$$L = (t^T t - 2w^T X^T t + w^T X^T X w)$$

Differentiate

$$\frac{\partial L}{\partial w} = -2X^T t + 2X^T X w$$

$$-2X^T t + 2X^T X w = 0$$

$$-\cancel{2}X^T X w = \cancel{2}X^T t$$

$$\hat{w} = \frac{X^T t}{X^T X} \quad \text{or}$$

$$\boxed{\hat{w} = (X^T X)^{-1} X^T t}$$

Solution 5:

The weighted Average loss:

$$L = \frac{1}{N} \sum_{n=1}^{N} \alpha_n \left(t_n - w^T x_n\right)^2$$

if

$$A = \begin{bmatrix} \alpha_1 & 0 & \cdots & 0 \\ 0 & \alpha_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \\ 0 & 0 & \cdots & \alpha_n \end{bmatrix}$$

The loss Vector Form:

$$L = \frac{1}{N} \sum_{n=1}^{N} (t - Xw)^T A (t - Xw)$$

After multiplying:

$$\frac{1}{N}\left(t^T A t - 2w^T X^T A t + w^T X^T A X w\right)$$

differentiating:

$$\frac{\partial L}{\partial W} = -\frac{2}{N} X^T A t + \frac{2}{N} X^T A X w$$

$$= -\cancel{\frac{2}{N}}\left(-X^T A T + X^T A X w\right) = 0$$

$$X^T A X W = X^T A t$$

$$\boxed{\hat{W} = \frac{X^T A t}{X^T A X}} \quad \text{or} \quad \boxed{\hat{W} = (X^T A X)^{-1} X A t}$$