ISTA 421/521 – Homework 2

Due: Tuesday, September 30, 10pm

12 points total Undergraduate / 16 points total Graduate

Emanuel Carlos de Alcantara Valente

Undergraduate

Instructions

In this assignment you are required to write 3 scripts in python. They will be submitted as 3 separate files, although you are free to copy chunks of code from one script to the next as desired. The names for the script files are specified in problems 1, 5 and 8, below.

Included in the homework 2 release are two sample scripts (`fitpoly_incomplete.py` and `cv_demo_incomplete.py`) and two data files (`womens100.csv`, and `synthdata2014.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 just as provided: you must fill in the calculation of $\mathbf{w}$. The data files are provided in `.csv` (comma separated values) format. The script `fitpoly_incomplete.py` shows how to load the files.

All problems except problem 1 require that you provide some "written" answer (in some cases also figures), so you will also submit a .pdf of your written answers. (You can use LaTeX or any other system (including handwritten; plots, of course, must be program-generated) as long as the final version is in PDF.)

The final submission will include (minimally) the three scripts and a PDF version of your written part of the assignment. You are required to create either a .zip or tarball (.tar.gz / .tgz) archive of all of the files for your submission and submit the archive to the d2l dropbox by the date/time deadline above.

NOTE: Problems 6 and 7 are required for Graduate students only; Undergraduates may complete them for extra credit equal to the point value.

(FCMA refers to the course text: Rogers and Girolami (2012), *A First Course in Machine Learning*. For general notes on using LaTeX to typeset math, see: `http://en.wikibooks.org/wiki/LaTeX/Mathematics`)

1. [2 points] Adapted from **Exercise 1.2** of FCMA p.35:

   Write a Python script that can find the parameters $\mathbf{w}$ for an arbitrary dataset of x_n, t_n pairs. You will use this script to answer problem 2, which only requires fitting a simple line to the data (i.e., you only need to fit parameters w_0 and w_1); however, in problems 5 and 8 you will need to fit higher-order polynomial models (e.g., $t = w_0 + w_1 x + w_2 x^2 + ...$), so it is recommended that your script is generalized to handle higher-order polynomials. The script `fitpoly_incomplete.py` is provided to help get you started. `fitpoly_incomplete.py` provides helper functions to read data, plot data, and plot the model (once you've determined the weight vector $\mathbf{w}$), but the function for computing linear least-squares fit, fitpoly (starting on line 36), is incomplete – you need to fill this in – it takes as input the data vector $\mathbf{x}$, the target values vector $\mathbf{t}$, and the scalar `model_order`, which is an integer representing the polynomial order of the model; fitpoly is intended to return the $\mathbf{w}$ (column) vector (e.g., as a numpy matrix such as the following:

```
>>> m = numpy.matrix([1,2,3]) # this creates a matrix with one row
matrix([[1, 2, 3]])
>>> m.transpose()   # transposition makes returns a column vector
matrix([[1],
        [2],
        [3]])
>>> numpy.matrix([[1],[2],[3]]) # or we could specify the column directly
matrix([[1],
        [2],
        [3]])
```

   The above isn't literally the code you'd use, but shows how you can create and transpose matrices, with row and column vectors simply being limiting cases of matrices with one row or 3 rows with 1 value each, respectively). You will want to look at the numpy package, including the linear algebra package `numpy.linalg`, for linear algebra operators. Also, you don't have to use `fitpoly_incomplete.py`; you can write your own script from scratch. Whichever you choose, your script must have the ability to plot the data with the fitted model.

   Just to state the obvious: the objective of this exercise is for you to implement the linear least squares fit solution (i.e., the normal equations) – in their general matrix form. **DO NOT** use existing least squares solvers, such as `numpy.linalg.lstsq`, or scikit learn's `sklearn.linear_model.LogisticRegression`; however, it is certainly fine to use these to help *verify* your implementation's output.

   You will submit your script as a stand-alone file called `fitpoly.py`.

   Solution: The code is attached: 'fitpoly.py'

2. [1 point] Adapted from **Exercise 1.6** of FCMA p.35:

   Table 1.3 (p.13) of FCMA lists the women's 100m gold medal Olympic results – this data is provided in the file `womens100.csv` in the data folder. Using your script from problem 1, 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. Report the model here as a linear equation and also include a figure that plots your model with the data.

   Solution: According to the output of our program from exercise 1:

```
[emanuel@localhost submit]$ python fitpoly.py
Identified model parameters:
[[  4.09241546e+01]
 [ -1.50718122e-02]]
```

   The $w_0 = 40.924$ and $w_1 = -0.015$, so the 1st-order polynomial model is going to be:

$$t = 40.924 - 0.015x.$$

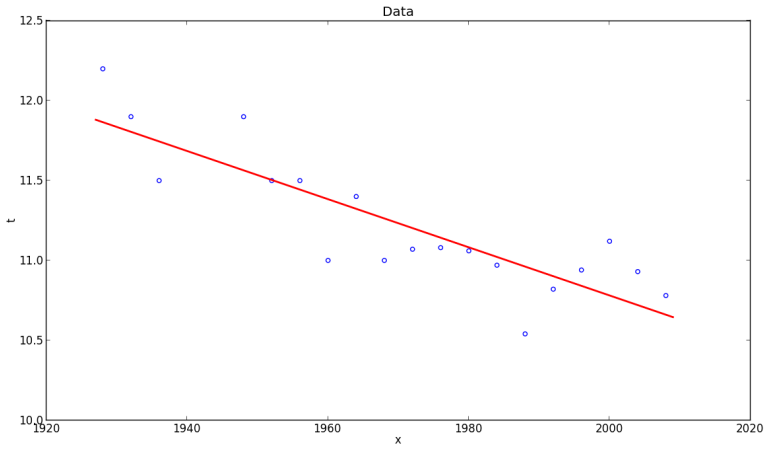

Figure 1: Solution: 1st-order polynomial model

3. [1 point] Adapted from **Exercise 1.7** of FCMA p.35:

   Using the model obtained in the previous exercise, predict the women's winning time at the 2012 and 2016 Olympic games; report the values here. What is the squared error of your model's prediction for Shelly-Ann Fraser-Price's gold medal time of 10.75 seconds in the 2012 Olympics women's 100m race?

   Solution: Substituting $x = 2012$ in our model equation, we will have:

   $t_{2012} = 40.924 - 0.015 * 2012 = 10.74$

   $t_{2016} = 40.924 - 0.015 * 2016 = 10.68$

   $\epsilon_{2012} = (10.75 - 10.74)^2 = 0.01^2 = 10^{-4}$

4. [1 point] Adapted from **Exercise 1.9** of FCMA p.36:

   Use your python script from problem 1 to load the data stored in the file `synthdata2014.csv` (in the data folder). 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 (if you extended the fitpoly_incomplete.py script, then `model_order`= 3). Present and describe the

parameters you obtain fitting to this data. Also, plot the data and your linear-fit model and include the plot in your answer.

Solution: According to the output of our model, we have:

```
[emanuel@localhost scripts]$ python exer4.py
Identified model parameters:
[[  9.02190177]
 [  4.54356124]
 [ 49.43723008]
 [  5.00595904]]
```

So, the parameters are:

$w_0 = 9.02190177, w_1 = 4.54356124, w_2 = 49.43723008$, and $w_3 = 5.00595904$

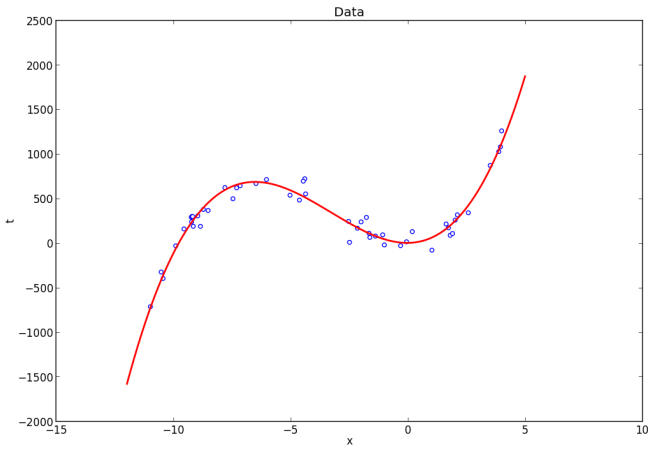

Figure 2: Solution: 3rd-order polynomial model fitting data

5. [3 points] Write a script that implements K-fold cross-validation to choose the polynomial order (between orders 0 and 7) with the best predictive error for the `synthdata2014.csv`. The provided script `cv_demo_incomplete.py` implements the synthetic data experiment described in Ch 1 (pp.31-32) of the book; you are welcome to use and adapt any part of this code you like; keep in mind that this script won't successfully execute until you add the general (matrix form) normal equation calculation on line 81. Note also that in the synthetic data experiment in `cv_demo_incomplete.py`, 1000 test data points are generated in addition to the 100 data points used for 10-fold cross-validation; for problem 5, you won't have this independent test set, only the data from `synthdata.csv` on which to perform K-fold cross-validation.

   Run your script with 10-fold cross-validation and Leave-One-Out-CV (LOOCV). Which model order do the two cross-validation methods predict as the best order for predictive accuracy? Do the two different cross-validation runs agree?

   Report the best-fit model parameters for the best model order according to 10-fold CV, and plot this model with the data. Include a plot of the CV-loss and training loss for the 8 different (0..7) polynomial model orders for both the 10-fold cross-validation and for LOOCV.

   You will submit your script as a stand-alone file called `cv.py`

Solution: According to the model developed, both cross-validation methods runs agree and the best model order for both was 3.

The best fit model parameters (model order 3) are:

$w_0 = 9.13981397, w_1 = 4.46592473, w_2 = 49.41560256$, and $w_3 = 5.00477336$

and the data will be fit as following:

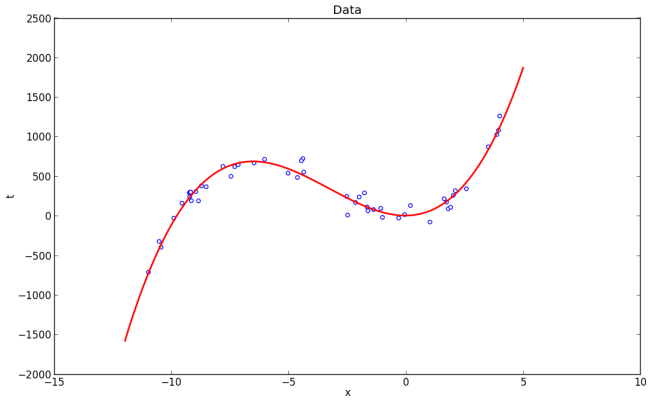

Figure 3: Solution: Best fit model parameters (model order = 3)

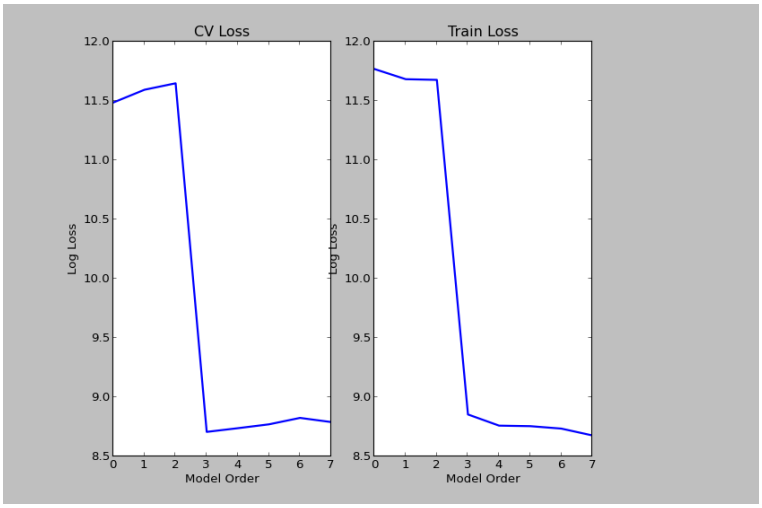

Figure 4: Solution: CV-loss and training loss for the 8 different (0..7) polynomial model orders - CV, K = 10

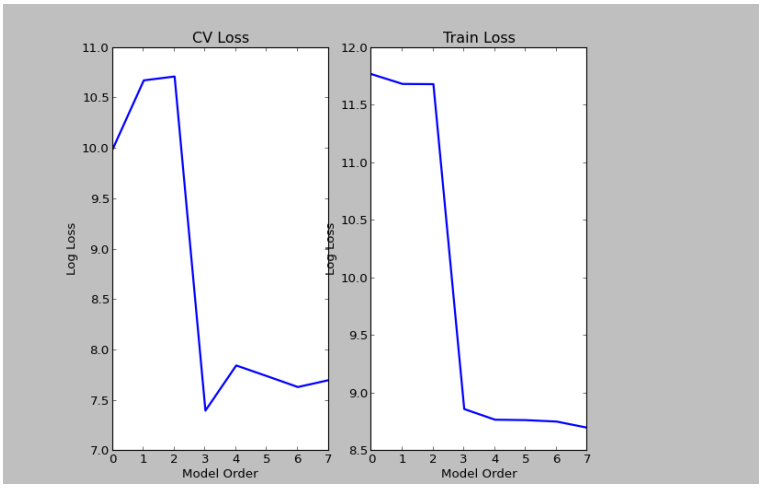

Figure 5: Solution: CV-loss and training loss for the 8 different (0..7) polynomial model orders - LOOCV, K = 50

6. [2 points – **Required only for Graduates**] **Exercise 1.10** from FCMA 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: Let us express $\mathcal{L}$ in the full matrix form: Suppose we have one target (one column) vector $\mathbf{t}$, one design matrix (2 column) $\mathbf{X}$, and one parameter vector $\mathbf{w}$. So, we have:

$$\mathbf{t} - \mathbf{X}\mathbf{w} = \begin{bmatrix} t_1 - w_0 - w_1 x_1 \\ t_2 - w_0 - w_1 x_2 \\ \vdots \\ t_n - w_0 - w_1 x_n \end{bmatrix}$$

   Now, we will use a single multiplication and transpose to neatly perform the squaring and summation and obtain our original loss funcion $\mathcal{L}$:

   $(\mathbf{t} - \mathbf{X}\mathbf{w})^\top (\mathbf{t} - \mathbf{X}\mathbf{w}) = (w_0 + w_1 x_1 - t_1)^2 + (w_0 + w_1 x_2 - t_2)^2 + \ldots + (w_0 + w_1 x_n - t_n)^2 =$
   $= \sum_{n=1}^{N} (w_0 + w_1 x_n - t_n)^2 = \sum_{n=1}^{N} (t_n - f(x_n; w_0, w_1))^2$

   Thus, our loss function in full matrix notation will be:

   $\mathcal{L} = (\mathbf{t} - \mathbf{X}\mathbf{w})^\top (\mathbf{t} - \mathbf{X}\mathbf{w})$

   Applying the distributing and transpose property in $\mathcal{L}$:

   $\mathcal{L} = ((\mathbf{t}^\top - \mathbf{w}^\top \mathbf{X}^\top)(\mathbf{t} - \mathbf{X}\mathbf{w})) =$
   $= (\mathbf{t}^\top \mathbf{t} - \mathbf{t}^\top \mathbf{X}\mathbf{w} - \mathbf{w}^\top \mathbf{X}^\top \mathbf{t} + \mathbf{w}\mathbf{X}^\mathbf{t}\mathbf{X}\mathbf{w})$

The two terms $-\mathbf{t}^\top\mathbf{X}\mathbf{w}$ and $\mathbf{w}^\top\mathbf{X}^\top\mathbf{t}$ are the transpose of one another and also scalars. Therefore, they can be combined, so:

$\mathcal{L} = (\mathbf{t}^\top\mathbf{t} - \mathbf{2t}^\top\mathbf{X}\mathbf{w} + \mathbf{w}\mathbf{X}^\mathbf{t}\mathbf{X}\mathbf{w})$

Differentiating $\mathcal{L}$ and setting to zero:

$\frac{\partial\mathcal{L}}{\partial\mathbf{w}} = -\mathbf{2}\mathbf{X}^\top\mathbf{t} + \mathbf{2}\mathbf{X}^\top\mathbf{X}\mathbf{w} = 0 =>$
$=> \mathbf{X}^\top\mathbf{X}\mathbf{w} = \mathbf{X}^\top\mathbf{t} => (\mathbf{X}^\top\mathbf{X})^{-\mathbf{1}}\mathbf{X}^\top\mathbf{X}\mathbf{w} = (\mathbf{X}^\top\mathbf{X})^{-\mathbf{1}}\mathbf{X}^\top\mathbf{t}$

However, we know that $(\mathbf{X}^\top\mathbf{X})^{-\mathbf{1}}\mathbf{X}^\top\mathbf{X} = \mathbb{I}$. Therefore:

$\mathbb{I}\mathbf{w} = (\mathbf{X}^\top\mathbf{X})^{-\mathbf{1}}\mathbf{X}^\top\mathbf{t}$

Therefore, $\hat{\mathbf{w}} = (\mathbf{X}^\top\mathbf{X})^{-\mathbf{1}}\mathbf{X}^\top\mathbf{t}$

7. [2 points – **Required only for Graduates**] **Exercise 1.11** from FCMA 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: Before writing $\mathcal{L}$ in the full matrix form, we will define the matrix α as A as follows:

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

Using the same process we took in the previous exercise, we can write $\mathcal{L}$ as follows:

$\mathcal{L} = \frac{1}{N}((\mathbf{t} - \mathbf{X}\mathbf{w})^\top A(\mathbf{t} - \mathbf{X}\mathbf{w}))$

Doing this way, we can guarantee that we will have one α_n for each data point. Now, we will do the same process we used in the previous exercise. We will apply the distributing property in $\mathcal{L}$:

$\mathcal{L} = \frac{1}{N}((\mathbf{t} - \mathbf{X}\mathbf{w})^\top(\mathbf{A}\mathbf{t} - \mathbf{A}\mathbf{X}\mathbf{w})) =$
$= \frac{1}{N}((\mathbf{t}^\top - (\mathbf{X}\mathbf{w})^\top)(\mathbf{A}\mathbf{t} - \mathbf{A}\mathbf{X}\mathbf{w})) =$
$= \frac{1}{N}(\mathbf{t}^\top\mathbf{A}\mathbf{t} - \mathbf{t}^\top\mathbf{A}\mathbf{X}\mathbf{w} - (\mathbf{X}\mathbf{w})^\top\mathbf{A}\mathbf{t} + (\mathbf{X}\mathbf{w})^\top\mathbf{A}\mathbf{X}\mathbf{w})$

The two terms $\mathbf{t}^\top\mathbf{A}\mathbf{X}\mathbf{w}$ and $(\mathbf{X}\mathbf{w})^\top\mathbf{A}\mathbf{t}$ are the transpose of one another and also scalars. Therefore, they can be combined, so:

$\mathcal{L} = \frac{1}{N}(\mathbf{t}^\top\mathbf{A}\mathbf{t} - \mathbf{2}(\mathbf{X}\mathbf{w})^\top\mathbf{A}\mathbf{t} + (\mathbf{X}\mathbf{w})^\top\mathbf{A}\mathbf{X}\mathbf{w}) =$
$\frac{1}{N}(\mathbf{t}^\top\mathbf{A}\mathbf{t} - \mathbf{2}\mathbf{w}^\top\mathbf{X}^\top\mathbf{A}\mathbf{t} + (\mathbf{X}\mathbf{w})^\top\mathbf{A}\mathbf{X}\mathbf{w})$

Differentiating $\mathcal{L}$ and setting to zero:

$\frac{\partial\mathcal{L}}{\partial\mathbf{w}} = -\mathbf{2}\mathbf{X}^\top\mathbf{A}\mathbf{t} + \mathbf{2}\mathbf{X}^\top\mathbf{A}\mathbf{X}\mathbf{w} = 0 =>$
$\mathbf{X}^\top\mathbf{A}\mathbf{X}\mathbf{w} = \mathbf{X}^\top\mathbf{A}\mathbf{t} => (\mathbf{X}^\top\mathbf{A}\mathbf{X})^{-\mathbf{1}}\mathbf{X}^\top\mathbf{A}\mathbf{X}\mathbf{w} = (\mathbf{X}^\top\mathbf{A}\mathbf{X})^{-\mathbf{1}}\mathbf{X}^\top\mathbf{A}\mathbf{t} =>$

$\mathbb{I}\mathbf{w} = (\mathbf{X}^\top \mathbf{A}\mathbf{X})^{-\mathbf{1}}\mathbf{X}^\top \mathbf{A}\mathbf{t}$

Therefore, $\hat{\mathbf{w}} = (\mathbf{X}^\top \mathbf{A}\mathbf{X})^{-\mathbf{1}}\mathbf{X}^\top \mathbf{A}\mathbf{t}$

8. [4 points] Variant of **Exercise 1.12** from FCMA p.36

   Write a new Python script that uses K-fold cross-validation to find the value of λ that gives the (approximate) best predictive performance on the synthetic data (`synthdata.csv`) using **regularized** least squares with a **7th order polynomial** model. Report here the lambda you identified, the best fit linear model using that lambda (as a linear equation) and include a plot of the MSE loss as a function of λ that shows the loss curve with a minimum, and a plot of the best-fit model with the data.

   You will submit the script as a stand-alone file called `regularize.py`.

   Solution: The best lambda identified was 0.1. According to the output of our model:

```
emanuel@localhost submit]$ python regularize.py
Best parameters considering best lambda =  0.1 are:
[  7.80554395e+00   4.00116996e+00   4.57982000e+01   1.13186782e+00
  -1.12223146e-01   2.33469379e-01   3.61837459e-02   1.51821005e-03]
```

   The parameters are:

$$\hat{\mathbf{w}} = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_7 \end{bmatrix} = \begin{bmatrix} 7.80554395e+00 \\ 4.00116996e+00 \\ 4.57982000e+01 \\ 1.13186782e+00 \\ -1.12223146e-01 \\ 2.33469379e-01 \\ 3.61837459e-02 \\ 1.51821005e-03 \end{bmatrix}$$

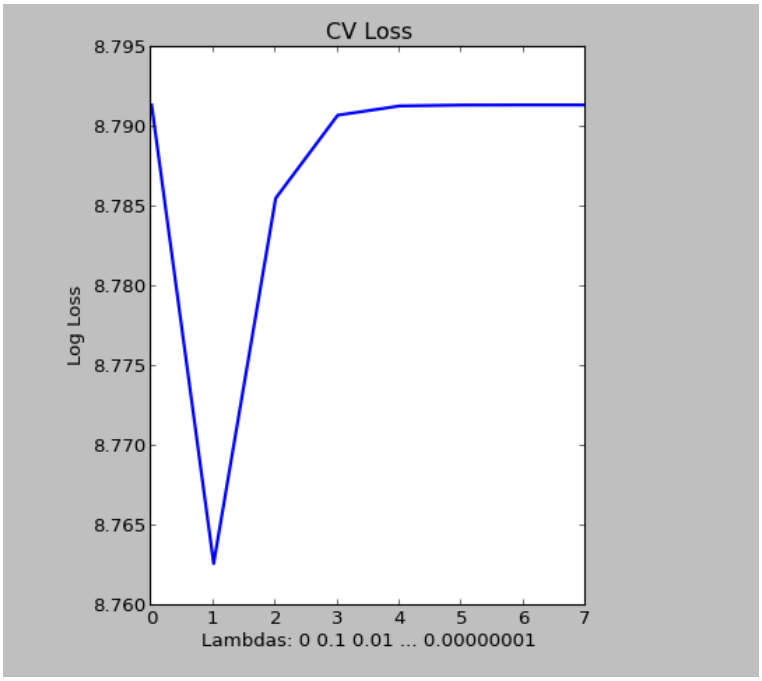

Figure 6: MSE loss as a function of λ. Each integer in X-axis is the index of lambda array. The lambada array is [0, 0.1, 0.01, 0.001, 0.0001, 0.00001, 0.000001, 0.0000001]

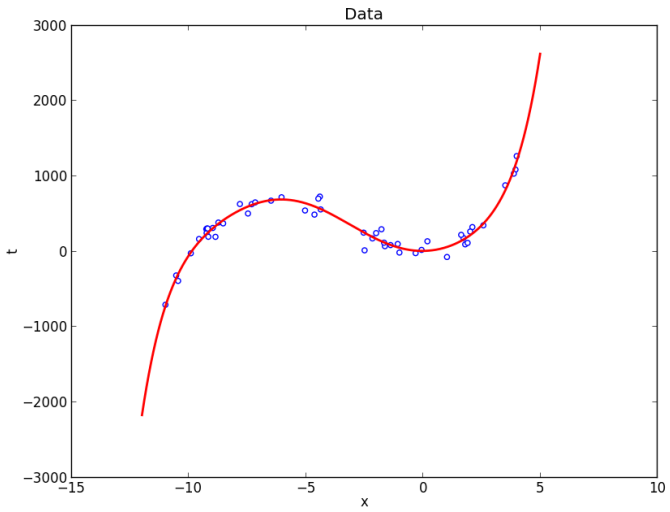

Figure 7: Best fit model: lambda = 0.1, model order = 7, CV k=10