CS5487 Programming Assignment 1
Regression

Antoni Chan
Department of Computer Science
City University of Hong Kong

In this programming assignment you will implement and test some of the regression methods that were introduced in the lecture and in the problem sets. Let $f(x,\theta)$ be a function with input $x \in \mathbb{R}^d$ and parameters $\theta \in \mathbb{R}^D$, and

$$f(x,\theta) = \phi(x)^T\theta, \tag{1}$$

where $\phi(x) : \mathbb{R}^d \to \mathbb{R}^D$ is a feature transformation of x. For example, the K-th order polynomial with input $x \in \mathbb{R}$ can be expressed as

$$f(x,\theta) = \sum_{k=0}^{K} x^k\theta_k = \phi(x)^T\theta, \tag{2}$$

where the feature transformation and parameters are

$$\phi(x) = \begin{bmatrix}1, x, x^2, \cdots, x^K\end{bmatrix}^T \in \mathbb{R}^{K+1}, \qquad \theta = \begin{bmatrix}\theta_0, \cdots, \theta_K\end{bmatrix}^T \in \mathbb{R}^{K+1}. \tag{3}$$

Our goal is to obtain the best estimate of the function given iid samples $\mathcal{D} = \{(x_1,y_1),\ldots,(x_n,y_n)\}$, where y_i are the noisy observations of $f(x_i,\theta)$. We have seen several ways of doing the regression, using different assumed noise models and formulations. For convenience, define the following quantities,

$$y = \begin{bmatrix}y_1, \cdots, y_n\end{bmatrix}^T, \qquad \Phi = \begin{bmatrix}\phi(x_1), \cdots, \phi(x_n)\end{bmatrix}, \qquad X = \begin{bmatrix}x_1, \cdots, x_n\end{bmatrix}. \tag{4}$$

Here is a summary of the various regression algorithms we have seen so far:

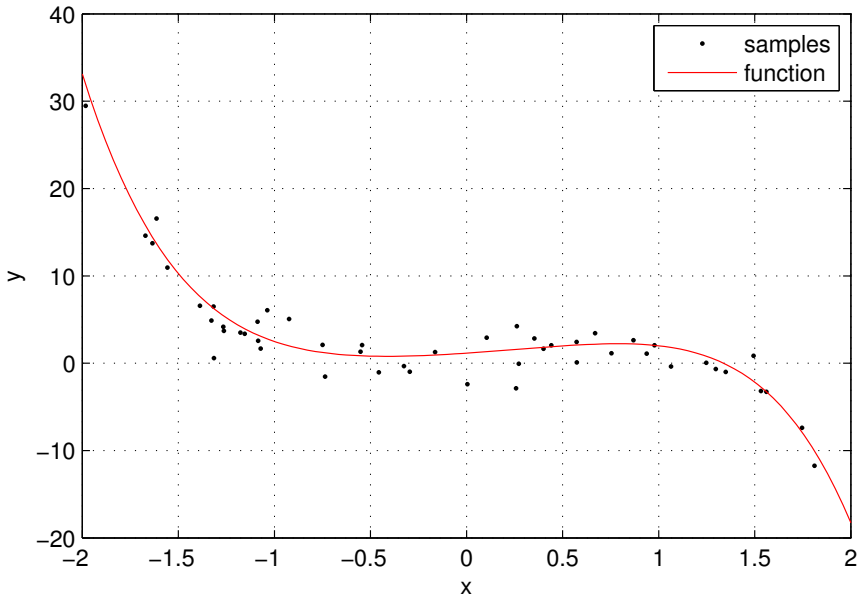
method	objective function	parameter estimate $\hat{\theta}$	prediction f_* for input x_*
least-squares (LS)	$\left\|y - \Phi^T\theta\right\|^2$	$\hat{\theta}_{LS} = (\Phi\Phi^T)^{-1}\Phi y$	$f_* = \phi(x_*)^T\hat{\theta}$
regularized LS (RLS)	$\left\|y - \Phi^T\theta\right\|^2 + \lambda\left\|\theta\right\|^2$	$\hat{\theta}_{RLS} = (\Phi\Phi^T + \lambda I)^{-1}\Phi y$	$f_* = \phi(x_*)^T\hat{\theta}$
L1-regularized LS (LASSO)	$\left\|y - \Phi^T\theta\right\|^2 + \lambda\left\|\theta\right\|_1$	QP solver (see Prob. 3.12)	$f_* = \phi(x_*)^T\hat{\theta}$
robust regression (RR)	$\left\|y - \Phi^T\theta\right\|_1$	LP solver (see Prob. 2.10)	$f_* = \phi(x_*)^T\hat{\theta}$
	distributions	**posterior**	**predictive**
Bayesian regression (BR)	$\theta \sim \mathcal{N}(0, \alpha I)$ $y\|x,\theta \sim \mathcal{N}(f(x,\theta),\sigma^2)$	$\theta\|X,y \sim \mathcal{N}(\hat{\mu}_\theta, \hat{\sigma}^2_\theta)$ $\hat{\mu}_\theta = \frac{1}{\sigma^2}\hat{\Sigma}_\theta\Phi y$ $\hat{\Sigma}_\theta = (\frac{1}{\alpha}I + \frac{1}{\sigma^2}\Phi\Phi^T)^{-1}$	$f_*\|X,y,x_* \sim \mathcal{N}(\hat{\mu}_*, \hat{\sigma}^2_*)$ $\hat{\mu}_* = \phi(x_*)^T\hat{\mu}_\theta$ $\hat{\sigma}^2_* = \phi(x_*)^T\hat{\Sigma}_\theta\phi(x_*)$

Part 1 Polynomial function

In the first problem, you will test these regression methods on a simple dataset, and look at the effects of using different objective functions, regularization terms, and formulations. The MATLAB data file `poly_data.mat` (or `poly_data_*.txt` if you are not using MATLAB) contains a set of samples generated from a 5th order polynomial function, with observation noise $\sigma^2 = 5$. The file contains the following variables:

- `sampx` - sample input values (x_i), each entry is an input value.
- `sampy` - sample output values (y_i), each entry is an output.
- `polyx` - input values for the true function (also these are the test inputs x_*).
- `polyy` - output values for the true function.
- `thtrue` - the true value of the θ used to generate the data.

A plot of the samples (`sampx`,`sampy`) and the true polynomial function (`polyx`, `polyy`) are shown below.



The goal is to try to estimate the polynomial function from the samples.

(a) Implement the above 5 regression algorithms for the K-th order polynomial given in (2). In the next problem, you will use these regression methods with a different feature transformation $\phi(x)$. Hence, it would be better to separate the regression algorithm and the feature transformation in your implementation.

(b) For each regression method, use the sample data (`sampx`, `sampy`) to estimate the parameters of a 5th order polynomial function. Make a plot of the estimated function using `polyx` as inputs, along with the sample data. For BR, also plot the standard deviation around the mean. What is the mean-squared error between the learned function outputs and the true function outputs (`polyy`), averaged over all input values in `polyx`? For algorithms with hyperparameters, select some values that tend to work well.

(c) Repeat (b), but reduce the amount of training data available, by selecting a subset of the samples (e.g., 10%, 25%, 50%, 75%). Plot the estimated functions. Which models are more robust with less data, and which tend to overfit? Make a plot of error versus training size, and comment on any important trends and findings. (You should run multiple trials with different random subsets, and take the average error).

(d) Add some outliers output values (e.g., add large numbers to a few values in `sampy`), and repeat (b). Which methods are robust to the presence of outliers, and which are most sensitive? Why?

(e) Repeat (b) but estimate a higher-order polynomial (e.g., 10th order). Which models tend to overfit the data when learning a more complex model, and which models do not? Verify your observations by examining the estimated parameter values.

.....

Part 2 A real world regression problem – counting people

Now we will consider a real world regression problem – estimating the number of people in an image. Below is an example image of a sidewalk with people.

From each image, we extract a feature vector $x_i \in \mathbb{R}^9$, which is based on some properties of the crowd (e.g., the area covered by the crowd, the perimeter of the crowd, etc). We also have the number of people in the image, $y_i \in \mathbb{R}$. The goal is to learn a regression function between this feature vector and the number of people in the image. (Let's ignore the fact that the count is actually a non-negative integer, and just apply our standard regression methods).

The data can be found in the `count_data.mat` MATLAB file (or `count_data_*.txt` if you are not using MATLAB). The mat file contains the following variables:

- `trainx` – training inputs. Each column is a 9-dimensional feature vector.
- `trainy` – training outputs (mean-subtracted counts for each input vector).
- `testx` – test inputs.
- `testy` – true outputs for the test inputs.

Note that the output values are actually the mean-subtracted counts, so you will see some negative outputs.

(a) Let's first look at using the features directly, i.e., set $\phi(x) = x$. Use the training set (`trainx`, `trainy`), estimate a function with some of the regression algorithms above. Use the test set inputs `testx` to predict the output, and compare the predictions to the true output `testy`. (You can round the function outputs so they are counting numbers). Calculate the mean-absolute error and mean-squared error. Which method works the best? Plot the test predictions and the true counts, and discuss any interesting findings.

(b) Now try some other feature transformations. For example, you can create a simple 2nd order polynomial as $\phi(x) = \left[x_1, \cdots, x_9, x_1^2, \cdots, x_9^2\right]^T$. This could be expanded to include more cross-terms $x_i x_j$. Try some other non-linear transformation of the features. Can you improve your results from (a)?

.....

Part 3 Estimating hyperparameters (optional)

So far you probably have been setting the hyperparameters of each method (e.g., λ) by hand to get the best performance on the test set. This is not a fair way to do an experiment, since it amounts to optimizing to the test set, and doesn't reflect whether the model can generalize well to unseen data. In a real scenario, you wouldn't have the true outputs for the test set! Here are two common ways to set the hyperparameters using only the training set:

1. **N-fold cross-validation**: With this method, the original training set is split into a new training set and a validation set. The regression function is estimated from the new (smaller) training set, and tested on the validation set, for different hyperparameter values. This is repeated for different partitions of the original training set. Formally, separate the training set $\mathcal{D}$ into N equal partitions, $\{\mathcal{D}_1, \cdots, \mathcal{D}_N\}$. For each partition i and candidate hyperparameter value λ_j: 1) train a regression function using λ_j on $\mathcal{D} \backslash \mathcal{D}_i$ (all the data except for $\mathcal{D}_i$); 2) test the function on $\mathcal{D}_i$; 3) record the error $e_{i,j}$. Finally, sum the error over the N partitions, $E_j = \sum_{i=1}^{N} e_{i,j}$. Select the λ_j with the lowest overall error, and train the final regression function on the full training set $\mathcal{D}$. (usually people use between $N = 3$ to $N = 10$ folds, depending on the size of the training set).

2. **maximum marginal likelihood**: For Bayesian regression, another way to select the hyperparameters is to maximize the *marginal likelihood* of the training data (see Problem 3.11 for details),

$$\hat{\alpha}, \hat{\sigma}^2 = \underset{\alpha, \sigma^2}{\operatorname{argmax}}\, p(y|X, \sigma^2, \alpha) = \underset{\alpha, \sigma^2}{\operatorname{argmax}}\, \log p(y|X, \sigma^2, \alpha) \tag{5}$$

where the marginal log-likelihood is

$$\log p(y|X, \sigma^2, \alpha) = \log \int p(y|\theta, X, \sigma^2) p(\theta|\alpha) d\theta \tag{6}$$

$$= -\frac{D}{2} \log \alpha - \frac{n}{2} \log \sigma^2 - \frac{1}{2\sigma^2} \left\| y - \Phi^T \hat{\mu}_\theta \right\|^2 - \frac{1}{2\alpha} \left\| \hat{\mu}_\theta \right\|^2 - \frac{1}{2} \log \left| \hat{\Sigma}_\theta^{-1} \right| - \frac{n}{2} \log(2\pi). \tag{7}$$

The procedure is as follows: for each candidate hyperparameter pair $\{\alpha_j, \sigma_j^2\}$, calculate the marginal likelihood in (7). Select the hyperparameter pair that results in the largest marginal likelihood.

(a) For Problem 2, use cross-validation or maximum marginal likelihood to select the hyperparameters using only the training data. Does the accuracy increase or decrease? How close were the automatically selected parameters to your hand-selected ones?

.....

- **What to hand in** – You need to turn in the following things:

  1. Answers, plots, analysis, discussion, etc. for the above problems.
  2. Source code files.

  You must submit your programming assignment using the BlackBoard website. Go to "Assignments" ⇒ "Assignment submissions" ⇒ and create a journal entry.

- **Plagiarism** – You should implement each regression algorithm using your own code. Do not use someone else's implementation of the regression algorithms (including MATLAB's implementation). It is okay to use common library components and functions, e.g. standard matrix operations (e.g., `inv`, `eig`), general purpose optimization toolboxes (e.g., `quadprog`, `linprog`), etc. If you use any special toolboxes or libraries, make sure to note them in your report.

- **Grading** – The marks for this assignment will be distributed as follows:

  - 10% – Implementation of the regression algorithms (1a).
  - 40% – Results testing the regression functions (1b, 1c, 1d, 1e).
  - 20% – Results on counting people (2a).
  - 10% – Trying out other features for counting people (2b).
  - 20% – Quality of the written report. More points given to insightful observations and analysis.

  Note: if you really cannot implement the algorithms correctly, it is okay to use 3rd party software. You will not get marks for implementation, but you can still get marks for presenting the results.

- **Optional Problems** – The optional problem 3 will not be graded.