MTAT.03.227 Machine Learning
Spring 2015 / Exercise session IX
Nominal score: 10p
Maximum score: 15p
Deadline: 14th of April 16:15 EET

1. Sometimes outliers can seriously hinder the overall performance of regression methods based on minimisation of mean square error. The following exercise explains how to detect that there is a problem with outliers and how to solve it using confidence intervals for predictions.

   File `regression-benchmark.Rdata` contains data that roughly follows quadratic dependency $y = x^2$. However, there are some data points that seem to have arbitrary values

   (a) Fit a model $y \sim \alpha x^2 + \beta$ on the data and draw a plot which contains the data points, prediction line. How good is the fitted model? Do you see a systematic bias? Also, use qqpplot to test whether the residuals follow normal distribution, i.e., whether the assumptions of means square minimisation procedure are satisfied. (**1p**)

   (b) Although the error distribution is far from normal distribution you can still try to find standard deviation of residuals σ and approximate error distribution with $\mathcal{N}(0, \sigma)$. Compute 95% confidence intervals for the error, i.e., solve the equations:

   $$\Pr\left[\varepsilon \leftarrow \mathcal{N}(0,\sigma) : \varepsilon \leq q_0\right] = 2.5\%$$
   $$\Pr\left[\varepsilon \leftarrow \mathcal{N}(0,\sigma) : \varepsilon \leq q_1\right] = 97.5\%$$

   for q_0 and q_1. Draw two additional dashed lines $y = \hat{y}(x) + q_0$ and $y = \hat{y}(x) + q_1$ where $\hat{y}(x)$ is the prediction line obtained in the first part. We can label all points outside the region bounded by dashed lines as outliers. Write a function that detects all outliers. (**1p**)

   (c) Next, try to fit a model $y \sim \alpha x^2 + \beta$ only for the set of normal data points. Again, compute standard deviation for the residuals and corresponding 95% confidence intervals for the error and visualise 95% confidence intervals for the prediction, i.e., draw lines $y = \hat{y}(x) + q_0$ and $y = \hat{y}(x) + q_1$ for the prediction line $\hat{y}(x)$.
   Compare the first and the second plot. Does this technique improve the quality of predictions. Are the assumptions of means square minimisation now satisfied? (**1p**)

   (d) Write a function `robust.lm` that repeats outlier detection several times to fit a model. Describe the convergence criterion and demonstrate that it works also for multivariate linear regression. (**2p**)

2. Most regression methods assume that individual measurement errors are uncorrelated and come from the normal distribution, i.e., they are designed

for white Gaussian noise. Sometimes error terms are strongly correlated and we have to fit the model in the presence of coloured Gaussian noise. The latter often occurs when the response vector $\boldsymbol{y}$ is indirectly computed form other measurements, e.g. the location of the robot is inferred form the video signal. Another common cause of coloured Gaussian noise is measurement errors in the input $\boldsymbol{x}$ data, e.g., it might be impossible to completely control the experimental conditions. The following exercise illustrates how coloured Gaussian noise can emerge when the output vector $\boldsymbol{y}$ depends linearly on the input $\boldsymbol{x}$, i.e., $\boldsymbol{y} = A\boldsymbol{x} + \boldsymbol{b}$.

(a) Conduct an experiment to see whether you can detect that the noise in the data is coloured. For that generate data using the model

$$\begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} 1 & 8 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} x_1 + \varepsilon_1 \\ x_2 + \varepsilon_2 \end{pmatrix}$$

where $\varepsilon_1, \varepsilon_1 \sim \mathcal{N}(0, 0.1)$. Generate 100 samples from this distribution by sampling x_1, x_2 uniformly form the range $[0, 1]$ the and estimate coefficients of linear regression with `lm`. Visualise residuals with `dataEllipse` form the `car` library. Next, draw prediction errors for the first and second component on the same graph. Interpret the phenomenon occurring in the both graphs (**1p**)

(b) To analyse the issue more thoroughly, assume that $\boldsymbol{y} = A\boldsymbol{x} + \boldsymbol{b}$ and that measurements errors corrupt both the input and output data:

$$\begin{aligned} \hat{\boldsymbol{x}} &= \boldsymbol{x} + \boldsymbol{\varepsilon}_x, \qquad & \boldsymbol{\varepsilon}_x &\leftarrow \mathcal{N}(0, \sigma_x I) \ , \\ \hat{\boldsymbol{y}} &= \boldsymbol{y} + \boldsymbol{\varepsilon}_y, \qquad & \boldsymbol{\varepsilon}_y &\leftarrow \mathcal{N}(0, \sigma_y I) \ . \end{aligned}$$

This notation means that the error component for the input x_i is drawn independently form the univariate normal distribution $\mathcal{N}(0, \sigma_x)$ and for the output y_i is drawn independently form $\mathcal{N}(0, \sigma_y)$. Now if we try to fit the model on the observations, we obtain

$$\hat{\boldsymbol{y}} = A(\hat{\boldsymbol{x}} - \boldsymbol{\varepsilon}_x) + \boldsymbol{\varepsilon}_y = A\hat{\boldsymbol{x}} + (A\boldsymbol{\varepsilon}_x + \boldsymbol{\varepsilon}_y)$$

where the noise term $A\boldsymbol{\varepsilon}_x + \boldsymbol{\varepsilon}_y$ is a coloured Gaussian. Compute covariance and mean of the resulting coloured Gaussian. (**1p**)
Hint: For that you can use formulae from Matrix Cookbook.

3. In the lecture, we argued that the problem of outliers can be solved using Laplace distribution as an error distribution in the linear regression. The latter leads to the minimisation of sum of absolute deviations (LAD). The latter is implemented as `rq` function in `quantreg` package. The package implements quantile regression. However, the quantile regression with parameter $\tau = 0.5$ and method `br` is equivalent to the LAD.

(a) Use the data in the file `regression-benchmark.Rdata` as benchmark. Fit a model $y \sim \alpha x^2 + \beta$ on the data and draw a plot which

contains the data points, prediction line. (**1p**). If you implement your own fitting procedure based on stohhastic gradient decent or some other minimisation method, you get extra point (**1p**)
Hint: Note that $\frac{d}{dx}|x| = sign(x)$ and thus it is straightforward to express necessary derivatives for the gradient descent:

$$\frac{\partial}{\partial b}\sum_{i=1}^{n}\left|y_i - ax_i^2 - b\right| = -\sum_{i=1}^{n} sign(y_i - ax_i^2 - b) \ ,$$
$$\frac{\partial}{\partial a}\sum_{i=1}^{n}\left|y_i - ax_i^2 - b\right| = -\sum_{i=1}^{n} sign(y_i - ax_i^2 - b)x_i^2 \ .$$

Moreover, the first derivative is zero if and only if b is the median of $y_i - ax_i^2$. Hence, you can always choose the optimal b value.

(b) To plot 95% confidence intervals, we need to estimate the μ and β parameter from the residuals and then compute 95% confidence intervals for the Laplace distribution. For that note that maximum likelihood estimates for these parameters are:

$$\mu = \mathsf{median}(\varepsilon_1, \ldots, \varepsilon_n)$$
$$\beta = \frac{1}{n} \cdot \sum_{i=1}^{n} |\varepsilon_i - \mu| \ .$$

Use these parameters and the relation between Laplace and exponential distribution to compute confidence intervals. Plot them as dashed lines. Does the residues follow Laplacian distribution? Compare the result with the first exercise. Interpret the result. (**1p**)
Hint: Recall that Laplacian distribution is closely related to exponential distribution. If you know the correspondence, it is straightforward to use `qexp` in order to compute necessary quantiles.

4. The next exercise shows basic similarities and differences between regularised linear regression methods. For all experiments, generate a data set consisting of 20 inputs $x_1, \ldots, x_{20}$ sampled uniformly and independently form the range $[0, 1]$ and y generated by a linear combination of first five inputs $x_1, \ldots, x_{20}$ together with additional noise $\mathcal{N}(0, \sigma)$.

   (a) Let us consider the simplest correspondence $y = x_1$ and study how well the standard linear regression algorithm performs in terms of predicting coefficients and predicting the output. For that sample different responses to the same input data and compare predicted coefficients for different levels of noise $\sigma \in \{0, 0.1, 1, 10\}$. Draw corresponding prediction graph for inputs $x_1 \in [0, 1]$ and $x_i = 0$. Make at least 10 experiments for each noise level. Interpret results. (**1p**)

   (b) Repeat the same experiment with ridge regression (`lm.ridge`) where the regularisation parameter $\lambda = 10$. Experiment what will happen if λ value is set to 10^6. Interpret results. (**1p**).

(c) Regularisation with ℓ_1 penalty term is known to produce sparse models with clear interpretation. Test this fact in practice. Fix a model $y = a_1x_1 + \ldots + a_5x_5$ for some different integer coefficients. Use `lars` function from the `lars` package to perform linear regression with ℓ_1-penalty term (the right option is `lasso` but you can use `par` parameter as well). Study how big can the noise level be so that first 5 nonzero coefficients discovered are $x_1, \ldots, x_5$ in 95% of cases. (**2p**).

5. When the number of hidden neurons is too big for the problem instance, training algorithms without regularisation commonly produce sub-optimal results, since many hidden layer neurons represent the same concept. Regularisation helps to alleviate this problem. The following exercise shows what is the main difference between ℓ_1 and ℓ_2 regularisation.

   (a) Generate data by choosing 200 samples $x_1, \ldots, x_{200}$ uniformly from the range $[-2, 2]$. Let the target function $f(x)$ be a square hump

   $$f(x) = \begin{cases} 1, & \text{if } |x| \leq 1 \ , \\ 0, & \text{if } |x| > 1 \ . \end{cases}$$

   Compute the response $y_i = f(x_i) + \varepsilon_i$ where $\varepsilon_i \sim \mathcal{N}(0, 0.1)$. Use `nnet` package to train a neural network with 20 hidden neurons. Make sure that you train the network to predict and not to classify, i.e., set `linout=TRUE`. Calculate the output of the hidden layer on the data samples. You should get 200×20 matrix H where the columns contain the outputs of hidden neurons on inputs $x_1, \ldots, x_{200}$. (**1p**)

   Hint: You can and should reuse the code from Exercise Session VI where we considered neural networks. In particular, you should use functions `as.neuralnetwork` and `CalculateNeuronOutputs`. The latter allows you to inspect the outputs of the hidden layer on different inputs. Alternatively you could use `compute` function from the package `neuralnet`.

   (b) Use `lm` and `lm.ridge` to find the weights of the output neuron. Recall that for prediction the output neuron is linear and thus we can determine the weights by linear regression of type $y \sim H + 1$ when the hidden layer neurons are fixed. Try different values for the regularisation coefficient λ. Start with near-zero and increase it till the regularisation completely destroys the shape of the prediction function. Choose 5 values for λ between these two extremes (equally spaced on logarithmic scale) and visualise the effect. For that draw two graphs for each value of λ. The first graph should contain the target function, training data and the prediction function. The second graph should just output the values of regression coefficients for different neurons. Is the coefficient pattern stable when we alter λ, i.e., same coefficients are large compared to the others? (**2p**)

   Hint: Unfortunately, `predict` function does not work with the model

produced by `lm.ridge`. Hence, you have to use `coef` function to extract model coefficients and then matrix algebra to evaluate prediction on the data matrix.

(c) Use the `lars` package for ℓ_1-regularisation. The function `lars` outputs values of regression coefficient on critical points. All other values can be reconstructed from these points via linear interpolation—it can be shown that the coefficient vector is a piecewise linear function of regularisation parameter. In practice, you should consider coefficient values only in critical points as potential weights for the output neuron. The `lars` function implements several sparse regression algorithms and only `lasso` implements ℓ_1-regularisation.

Use `lars` package to plot predictions on first five critical points together with coefficient values as in the previous part of the exercise. Explain how you could use ℓ_1-regression to prune redundant neurons in the hidden layer. Confer the results with ℓ_2-regularisation (**2p**)

Hint: Unfortunately, the package is seriously under-documented. Fortunately, functions `predict.lars` and `coef.lars` allow you to make predictions and extract coefficients for each critical value.

($\star$) Implement a neural network pruning strategy that uses ℓ_1-regularisation to remove unnecessary neurons from the hidden layer. The algorithm should work as an add-on for `nnet`. Test is on standard target functions (square hump, saw-tooth and staircase) and few more complex one-dimensional functions. (**2p**)