Supervised Learning: Assignment 2

Mahshid Alimi, Antonio Remiro Azócar, MSc Machine Learning

31 December 2016

Exercise 1 (Linear Example)

a) For the primal form, one obtains the expression $\mathbf{X}'\mathbf{X}\mathbf{w}_{learn} + \lambda\mathbf{w}_{learn} = (\mathbf{X}'\mathbf{X} + \lambda\mathbf{I}_{dim})\mathbf{w}_{learn} = \mathbf{X}'\mathbf{y}$, taking the derivative of the regression cost function. Learning on the training data, $\mathbf{X}$ consists of 80 unit variance Gaussian points centered at the origin, with a dimensionality of 10 (80×10 matrix). $\mathbf{w}$ is a column vector of weights of 10 elements (dimensions) we have to learn. 'True' weights w_{true} from a random distribution are used for creating vector $\mathbf{y} = \mathbf{X}\mathbf{w}_{true} + \epsilon$, where ϵ is Gaussian noise with standard deviation $\sigma = 0.1$. λ is set to its correct regularisation parameter $\lambda = \sigma^2 = 0.01$ and $\mathbf{I}_{dim}$, learning on the training data, is the 10×10 identity matrix.

From the above expression, since if $\lambda > 0$, $(\mathbf{X}'\mathbf{X} + \lambda\mathbf{I}_n)$ is always invertible:

$$\mathbf{w}_{learn} = (\mathbf{X}'\mathbf{X} + \lambda\mathbf{I}_n)^{-1}\mathbf{X}'\mathbf{y}. \tag{1}$$

The learned weight vector is computed using (1), solving a linear system with 10 equations and 10 unknowns. One then uses the learned weights to obtain a vector of predicted values $\mathbf{y}_{pred}$ for the test set $\mathbf{X}_{test}$. The following expression is computed for this purpose:

$$\mathbf{y}_{pred} = \mathbf{y}'\mathbf{X}(\mathbf{X}'\mathbf{X} + \lambda\mathbf{I}_n)^{-1}\mathbf{X}_{test} = \mathbf{X}_{test}\mathbf{w}_{learn}. \tag{2}$$

Running our code for the primal form results in a test error, $err_{primal} = 0.0094$. It also results on the following difference between learned and true weight vectors: $|\Delta\mathbf{w}| = |\mathbf{w}_{learn} - \mathbf{w}_{true}| = (0.0036, 0.0021, 0.0059, 0.0079, 0.0183, 0.0039, 0.0098, 0.0037, 0.0230, 0.0015)^T$, with $\frac{1}{n}||\mathbf{w}_{learn} - \mathbf{w}_{true}||_2^2 = 0.000105$.

b) For the dual form, one notes that expression (1) can be reworked as:

$$\mathbf{w}_{learn} = \mathbf{X}'\alpha, \tag{3}$$

with dual variable $\alpha = (\mathbf{G} + \lambda I_n)^{-1}\mathbf{y}$, where $\mathbf{G} = \mathbf{X}\mathbf{X}'$ and $\mathbf{I}_n$ is the 80×80 identity matrix for the training set. The system is trained solving a linear system with 80 equations and 80 unknowns to find the α variables. The α variables are utilised to learn $\mathbf{w}_{learn}$ using (3). Subsequently, we predict $\mathbf{y}_{pred} = \mathbf{X}_{test}\mathbf{w}_{learn}$. The code is run within the same data-generating process as for the primal form. This results in a test error, $err_{dual} = 0.0094$, exactly the same error as that obtained for **(1a)**. Hence, $\Delta(err) = |err_{primal} - err_{dual}| = 0$ and the distance (norm) between the primal and dual predictions is 0. This should come as no surprise since as shown above, both forms of the problem are mathematically equivalent.

Our test errors are not significant, giving accurate predictive values and arising from very small differences between true and learned weights. Potential factors contributing to this include using a training set four times larger than the test set, utilising eight times more training examples than features, and setting the noise ϵ to a relatively low standard deviation (0.1).

Exercise 2 (Non-linear Example)

a) A three-dimensional plot is presented in Figure 1 utilising the following settings: 10,000 points and the random seed set to a value of zero. The axis labelled x_1 represents the points $x_i \in \mathbb{R}^d$ with standard Gaussian entries. The x_2 axis represents the intermediate points $z_i = vec(x_i x_i') \in \mathbb{R}^{d^2}$ The y-axis represents the $y_i =< z_i, w > +\epsilon_i$ observations. Effectively, Figure 1 presents the saddle-shaped hyperbolic paraboloid characteristic of the quadratic kernel function.

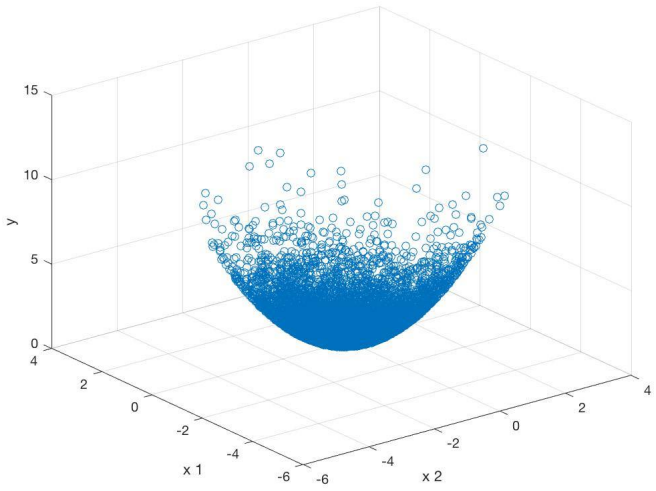


Figure 1: 3D plot for the non-linear dataset generated with the quadratic kernel (hyperbolic paraboloid).

b) The intermediate value $z_i = vec(x_i x_i') \in \mathbb{R}^{d^2}$ has 400 dimensions and takes the non-linear problem to the z-th dimension where it becomes a simple linear regression problem which can be solved following the instructions in **(1b)** - α dual variables are calculated first with $\alpha = (\mathbf{Z}\mathbf{Z}' + \lambda \mathbf{I}_n)^{-1}\mathbf{y}$ to later find the trained weights $\mathbf{w}_{learn} = \mathbf{Z}'\alpha$ and the predicted $\mathbf{y}_{pred} = \mathbf{Z}_{test}\mathbf{w}_{learn}$. Again, we encounter the dual problem, where the solution is given as a linear combination of the training set examples.

Utilising the parameter values provided by the question gave the test error, $err_{test} = 221.67$ for a given run. The same run/data generation process gave an error of 56.86 when reversing proportions and applying a 75/25 train/test split. This exemplifies the negative effect of a low ratio of test to train examples, particularly when the total number of examples is not that large (500); this leads to over-fitting on the training set. The large error may also be triggered by the low number of training examples in comparison to the number of features, and especially, by the very high variance of the noise ($\sigma = 5$).

c) The procedure required for this question can be summarised with the following sequence of steps:

```
for trial in trials
    generate random data
    test/validation/train split
    for each lambda
        train - calculate alpha, weights, y_pred for training set
        compute validation set error
    pick lambda with lowest val. error and store
    compute test set error and store
output mean lambda, mean test error
```

After performing 100 trials, we obtain a mean optimal parameter $\lambda_{min} = 34.91 \pm 33.95$ by computing the error on the validation set. This value is above the theoretically optimal value $\lambda = \sigma^2 = 25$. When using only the training set to train the system, feeding in λ to the test set yields a mean test error of $err_{test} = 238.71 \pm 33.39$. One can optimise performance by using both the original training and validation datasets to compute the test error. This results in a lower mean test error of $err_{test} = 162.63 \pm 66.32$ for the same input dataset. Evidently, performance is increased due to a greater number of training examples (train + val) in the final stage. However, the mean test error remains high for various aforementioned reasons: the high dimensionality of the dataset (more features than training and validation examples put together), a low number of training examples leading to over-fitting on the training set (and a higher value of λ_{min} than the theoretically optimal), and the presence of noise of very high variance. The theme of optimising a parameter using the validation set to later compute the mean test error is recurring throughout the next questions.

Exercise 3 (Error bars and fitting)

a) A three-dimensional plot of the data is presented in Figure 2 for $d = 2$, number of points $n = 500$, kernel parameter $\nu = 0.3$ and the random seed set to a value of 20. The x_1 and x_2 axes represent the values of the X inputs (2 features/dimensions). The y axis represents the values of the observation vector, generated as

$y = Lu$ (zero noise). Effectively, the plot is visually like that characteristic of a Gaussian random field, with combinations of many Gaussian contributions. Kernels are computed using the l^2 norm to find 'distances' between points.

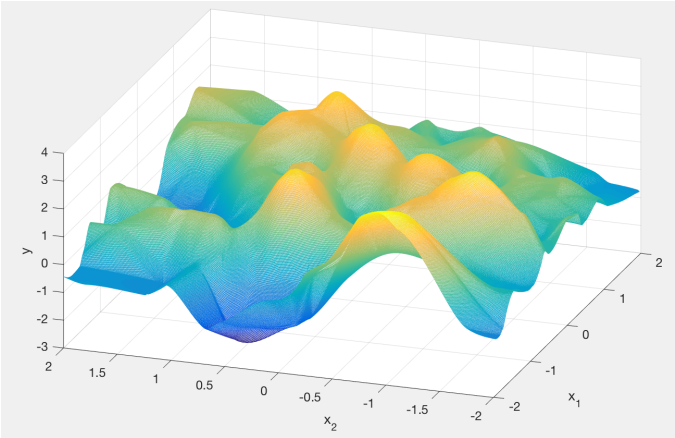
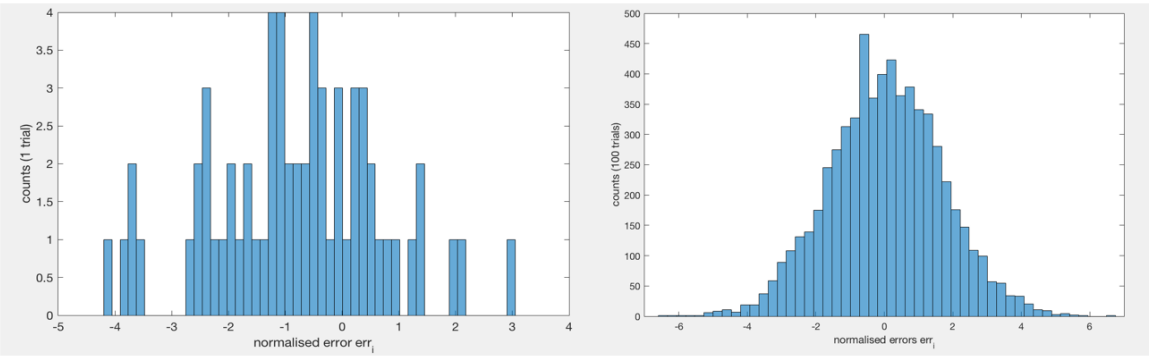


Figure 2: 3D plot for the generated Gaussian field. The function `griddata` was used for smoothing purposes.

b) This exercise is similar to **(2c)** in the sense that it involves optimising a parameter over the validation set. In this case, the parameter is not λ, it is ν; the width of the RBF kernel. Bare in mind the following: the input data is generated with a fixed kernel width, $v_0 = 5$. It does not change when trying to find the optimal ν. The kernel with the purpose of generating the train/validation/test data is labelled K_{true}. The kernel $\mathbf{K}$, an $n_{train} \times n_{train}$ matrix with entries $\mathbf{K}_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$, represents the kernel between training examples we wish to compute; it must be recalculated for each value of ν. So must the vector $\mathbf{k}$, with entries $\mathbf{k}_i = K(\mathbf{x}_i, \mathbf{x})$, representing the vector of covariances between a given test point and all the n_{train} points. We then calculate the dual variables $\alpha = (\mathbf{K} + \lambda \mathbf{I}_n)^{-1}\mathbf{y}$, to compute the predictive mean $\mathbf{y}'_{pred} = \alpha'\mathbf{k}$, the predictive variance $\sigma_i^2 = 1 - \mathbf{k}'(\mathbf{K} + \lambda \mathbf{I}_n)^{-1}\mathbf{k}$ and the normalised errors $err_i = \frac{(y_{true,i} - y_{pred,i})}{\sigma_i}$.

In Figures 3 and 4 one observes histograms of the errors err_i normalised by the number of test points (validation set), with $\nu_0 = 5$, $\nu = 2$. Figure 3, with only 63 test points corresponding to one trial, is not useful as a plotting device. The histogram is used as a mathematical discretisation of a cumulative distribution function. Figure 4, corresponding to 100 trials for $\nu_0 = 5$, $\nu = 2$, is simply presented to show that the errors follow a Gaussian distribution.

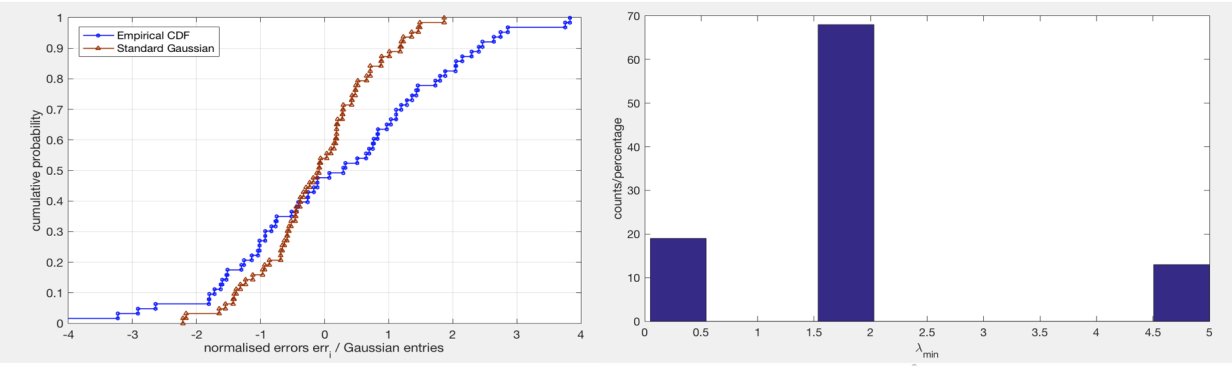


Figures 3 and 4: Histogram normalised error counts for 1 and 100 trials respectively ($\nu_0 = 5$, $\nu = 2$).

Effectively, the cumulative sum of such distributions can be computed to produce the empirical CDF. Figure 5 shows the empirical CDF of a different dataset (generated with the same parameters) plotted against that of a standard Gaussian of 63 points. The distance between both datasets is computed using the Kolgomorov-Smirnov (KS) statistic - used in the KS test to compare a sample distribution with a reference probability distribution, in this case the Gaussian. The KS statistic quantifies the 'distance' between both and is computed using the MATLAB function `kstest`.

Over 100 trials, an optimal parameter $\nu_{min} = 1.61 \pm 1.33$ is obtained, consisting of the mean of all widths minimising the distance between distributions for each trial. Figure 6 presents a histogram of the values of ν selected for every trial. A test error $err_{test} = 2.03 \pm 0.18$ is produced when using only the training set to train the system with ν_{min}. For the same dataset, using both the training and validation sets, one obtains a

slightly lower test error, as expected: $err_{test} = 1.97 + 0.16$. A combined train/validation set provides more training examples but fails to overcome the influence of a moderate level of noise $\sigma = 1.3$, the much larger number of test points (126 against 374) and a considerable number of features (10). Literature suggests that the Kolgomorov-Smirnov statistic only applies to continuous distributions - in that case its performance will be penalised if using a low number of test samples. Additionally, KS tends to be more sensitive near the center of a distribution than at the tails. It may be worth looking into other methods to compute the 'distance' between two distributions. For example, divergence methods related to information theory such as Kullback-Liebler.



Figures 5 (left): Comparison of the noisy empirical cumulative distribution (63 test points, $v = 2$, $v_0 = 5$) - blue, circles - against a that of a standard Gaussian - brown, triangles. Figure 6 (right): Histogram with the counts of the values of v_{min} selected for every trial when varying kernel widths.

Exercise 4 (Evidence based model selection)

a) This exercise shares the essence of **(2c)**, in the sense that our aim is to optimise the value of λ. In this case, the optimisation criterion is different (we are also using a RBF kernel). One needs to pick a value of lambda that **maximises** the evidence formula, otherwise following the steps highlighted in our answer to **(2c)**. The calculated 'evidence' is used to adaptively choose a parametrised kernel. Such kernel function can be observed as a model to be selected. Now, this approach is Bayesian, not frequentist. Hence, we are not expecting a considerable difference in parameter results with different train/test/validation splits. The 'evidence' depends only on the available split we apply the formula to.

Our suspicions are confirmed when running our code over 100 trials. Firstly, we compute the evidence on the training data only. This results in a mean optimal parameter of $\lambda_{max} = 0.261 \pm 0.172$. Very similar means with similar standard deviations are obtained computing the evidence only on the validation set or only on the test set. The small number of examples in some splits does not stop λ_{max} from taking a near theoretically optimal value (almost exactly σ^2) - in a Bayesian framework, σ^2 is set equal to λ (frequentist framework) for normal ridge regression . The mean test error is relatively large, with a value of $err_{test} = 0.522 \pm 0.119$. This is likely attributed to the very small number of training samples, 13. When combining train and validation sets for training, this mean test error decreases to $err_{test} = 0.440 \pm 0.091$ for the same inputs.

b) Whereas in part **a)**, we maximised the evidence to select our model/select parameter λ, this time we need to use the error on the validation set to select a model - like in **(2c)**. This problem is one of comparing a frequentist approach with the Bayesian approach in **(a)**. The frequentist approach gives a mean optimal parameter of $\lambda_{min} = 1.444 \pm 4.701$, when minimising the error over the validation set. This value of λ_{min}, well above its theoretically optimal value, with a very high standard deviation, is likely due to the small number of training examples used. In terms of the mean test error, this approach performs similarly to that in **a)**, with a test error of $err_{test} = 0.480 \pm 0.138$. I must note that for part **(b)** a different dataset was generated than for **(a)**. However we observe that for many runs of the code, the mean test errors for the Bayesian and frequentist approaches are fairly similar (roughly 0.4-0.5), with λ_{min} nearly always greater than 1 for **(b)**.

Consequently, while the Bayesian approach gives similar performance in terms of computing the mean test error, it counts with the benefits of not needing a validation set, allowing models to be compared using training data alone. Also, it avoids over-fitting by marginalising over model parameters, and permits the utilisation of all available data in the train set.