

Machine Learning, Spring 2019: Exercise Sheet 6 – Solutions

Problem 1. Consider the polynomial curve fitting example from Section 7.2.1 in the LNs. In the LNs I say that polynomial curve fitting can be done by calling a ready-made function *polyfit* that Matlab offers you. In fact, *polyfit* is just a special case of linear regression. Find out for yourself how *polyfit* can be programmed (without peeking at the Matlab documentation). More specifically, consider the following optimization problem. Given: a sample $(x_i, y_i)_{i=1, \dots, N}$ of argument-value training points, where x_i, y_i are real numbers. Also given: a maximal polynomial degree m . Design an algorithm that finds the optimal weight vector $W = (w_0, \dots, w_m)'$ defined by the loss function

$$L(W) = \frac{1}{N} \sum_{i=1}^N (w_0 + w_1 x_i + \dots + w_m x_i^m - y_i)^2$$

All you have to do is to find a way to translate this problem into an instance of the linear regression problem specified in Equation (18) in the LNs.

Solution. This works by assigning to each data point x_i a feature vector $\mathbf{f}_i = (1, x_i, x_i^2, \dots, x_i^m)'$, setting $z_i = y_i$, then applying Equation (18) mechanically.

Problem 2. Let $\gamma[\mu, \sigma^2]$ denote the pdf of a normal distribution with expectation μ and variance σ^2 . A real-valued random variable X has a *mixture of Gaussians* (MoG) pdf $p(x) = \sum_{i=1, \dots, k} \alpha_i \gamma[\mu_i, \sigma_i^2](x)$, where the k *mixture coefficients* α_i are non-negative and sum to 1. What is $E[X]$? – *Note:* Mixture of Gaussians representations of distributions are very popular in ML because they combine simplicity with flexibility. The three graphics in Figure 23 in the LN actually show a MoG example where σ_i^2 is shrinking from the left to the right panel and all α_i are equal to 1/6.

Solution. Let for each i , X_i be a RV distributed with pdf $\gamma[\mu_i, \sigma_i^2]$. Then $X = \sum_{i=1, \dots, k} \alpha_i X_i$ and

$$E[X] = E\left[\sum_{i=1, \dots, k} \alpha_i X_i\right] = \sum_{i=1, \dots, k} E[\alpha_i X_i] = \sum_{i=1, \dots, k} \alpha_i E[X_i] = \sum_{i=1, \dots, k} \alpha_i \mu_i.$$

Problem 3. Given a size- N training sample $(x_i, y_i)_{i=1, \dots, N}$ of argument-value training points, setting up an L -fold cross validation scheme for model optimization requires one to decide on the number L of folds in the first place. The two extreme cases are $L = 2$ (smallest possible number of folds) and $L = N$ (largest possible number of folds, "leave-one-out cross-validation"). Following the routines of the cross-validation procedure, in each of these cases one will determine an optimal model capacity. Here is a claim:

"When one uses 2-fold cross-validation, the optimal model capacity found is likely to be too low, that is, one will settle on a model capacity $m_{opt \text{ 2-fold}}$ which is smaller than the true optimal model capacity m_{opt} and hence will give underfitting models in the end; whereas if one uses leave-one-out cross-validation, one will determine a model capacity $m_{opt \text{ N-fold}}$ which is greater than m_{opt} , hence ultimately leading to overfitting."

Is this claim true? What is, in effect, the best choice for L ? Give your reasoning in plain English.

Solution. The claim is half true. When one uses a small number of folds, the number of data points that are used to optimize the training error is smaller than with a larger number of folds. With fewer training data points, models of smaller capacity can already fit well (or overfit) the training data, so the validation leads to stronger regularization. Thus 2-fold cross-validation will significantly tend to underfitting. However, even leave-one-out cross-validation will lead to underfitting (though negligible in practice) – so the second part of the claim is incorrect. The best one can do is thus leave-one-out cross-validation ($L = N$), which unfortunately is also the most expensive option. If computational costs permit, I personally typically go for 5- or 10-fold cross-validation.