Ludwig-Maximilians-Universitaet Muenchen
Institute for Informatics
Prof. Dr. Volker Tresp
Rajat Koner
Andreas Lohrer
Sebastian Schmoll

Machine Learning

Summer 2021

Exercise Sheet 2

Exercise 2-1 The ADALINE learning rule

The *ada*ptive *lin*ear *e*lement (ADALINE) model uses the *mean squared error* cost function

$$\text{cost} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \,,$$

for N training set elements, where y_i is the true and $\hat{y}_i$ is the predicted value for the i-th sample. In contrast to the simple perceptron, classification is not realized by the signum-function. (As a reminder: M is the number of input features of patterns $x_i \in \mathbb{R}^M$ and the dimensionality of the weight vector $w \in \mathbb{R}^M$; remember that we can append $x_0 = 1$ to be constant and corresponds to the bias or offset.)

a) Deduce the gradient descent-based learning rule (or: adaption rule) for the ADALINE process (analoguously to the perceptron learning rule).

b) Specify the corresponding pattern-based (SGD) learning rule.

c) What advantages do pattern-based learning rules have?

d) Name the most distinctive characteristics between the ADALINE model and the perceptron model.

Possible Solution

a) First we take the derivative of the cost function with respect to each w_j. Therefore, we first define the cost function:

$$\text{cost} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 = \frac{1}{N}\sum_{i=1}^{N}\left(y_i - \sum_{j=0}^{M-1} w_j x_{i,j}\right)^2$$

Next, we can take the derivative of each w_j. For the sake of simplicity, we denote them as w_{j^*}

$$\begin{aligned}
\frac{\partial \text{cost}}{\partial w_{j^*}} &= \frac{1}{N}\sum_{i=1}^{N} 2 \cdot \left(y_i - \sum_{j=0}^{M-1} w_j x_{i,j}\right) \cdot \frac{\partial}{\partial w_{j^*}}\left(y_i - \sum_{j=0}^{M-1} w_j x_{i,j}\right) = \\
&= \frac{2}{N}\sum_{i=1}^{N}\left(y_i - \sum_{j=0}^{M-1} w_j x_{i,j}\right) \cdot (-x_{i,j^*}) = \\
&= \frac{2}{N}\sum_{i=1}^{N} x_{i,j^*} \cdot \left(\left(\sum_{j=0}^{M-1} w_j x_{i,j}\right) - y_i\right) = \frac{2}{N}\sum_{i=1}^{N} x_{i,j^*}\,(\hat{y}_i - y_i)
\end{aligned}$$

Now, we can define the learning rule for the gradient descent as:

$$w_j \leftarrow w_j - \eta \frac{\delta \text{cost}}{\delta w_j} = w_j + \eta \frac{2}{N} \sum_{i=1}^{N} x_{i,j} \left(y_i - \hat{y}_i\right) \ , \tag{1}$$

or vectorized:

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \frac{\delta cost}{\delta \mathbf{w}} \tag{2}$$

b) A sample-based learning rule for a sample t can be defined as:

$$w_j \leftarrow w_j + \eta x_j(t) \left(y(t) - \hat{y}(t)\right) = w_j + \eta x_{t,j} \left(y_t - \hat{y}_t\right) \ , \tag{3}$$

which is also called *Delta rule*.

c) Sample-based learning rules can be learned on-the-fly: If an existing model is extended by a new sample, it is not necessary to recompute the whole model. This is essential for large datasets which cannot be fit into memory.

d) Striking difference: objective function. Consequences:

- Perceptron generates binary output while ADALINE outputs and optimizes vector of reals.
- If the data is separable, Perceptron converges faster than ADALINE.
- ADALINE is used to approximate the separating hyperplane more than it is used to classify (this could more easily be achieved using the Perceptron)
- the Perceptron uses class labels to learn model coefficients
- ADALINE uses continuous predicted values to learn model coefficients which is 'more' powerful since it tells us by 'how much' we were right or wrong.

Exercise 2-2 Regularization / Overfitting

(a) What is *overfitting* and how does it occur?

(b) How can a model be identified as "overfitted"?

(c) How can overfitting be avoided?

Possible Solution

(a) In general 'Overfitting': Over-adaptation of a model to a given dataset. We will look at the overfitting problem from three different point of views:

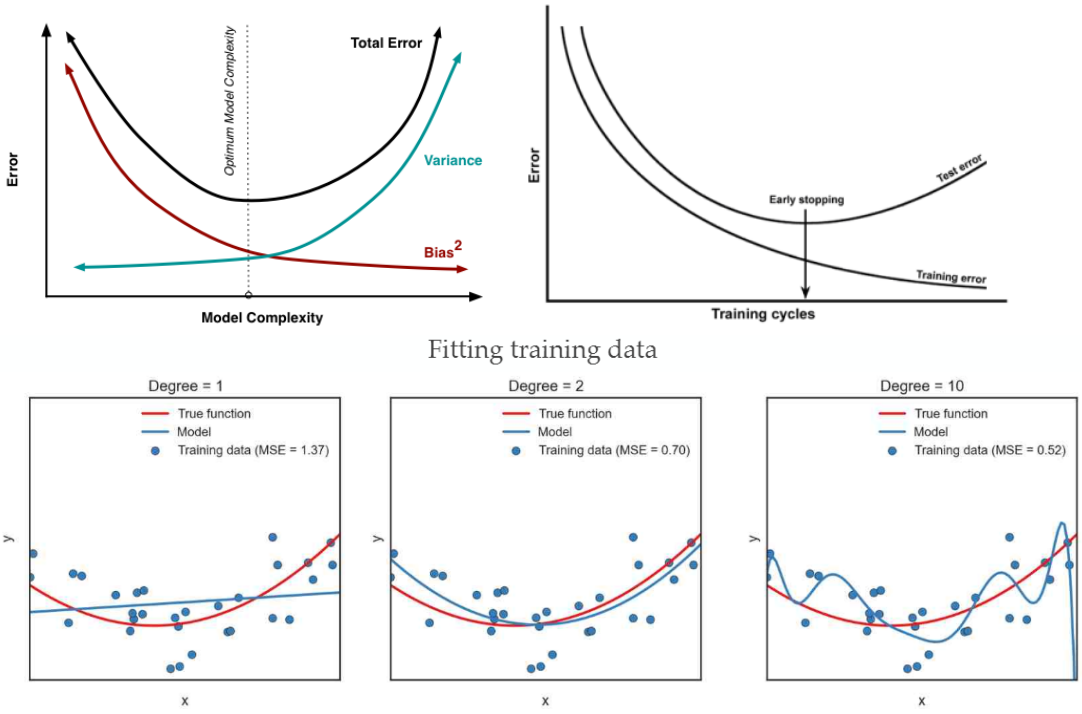
- **Model's complexity.** Complex models - such as DNNs - can detect subtle patterns in the training data. If the dataset is noisy, or if it is too small (introducing sampling noise) then the model is likely to detect patterns in the noise itself. Trivially, these patterns will not generalize to new instances.
- **# of Training epochs.** With increasing number of epochs, the algorithm learns and is its prediction error on the training set naturally goes down, and so does the prediction error on the validation set. However, we can observe that the validation error stops decreasing and start to go back up again. This indicates that the model has started to overfit the training data. *(Hint: early stopping*

- **Bias-Variance Trade-off.** One can also approach the problem via the bias-variance trade-off (not discussed in detail in the lecture, therefore one can omit this: A model's generalization error can be expressed as the sum of three different errors:
  Bias. This error is due to wrong assumptions, such as assuming that the data is linear when it is actually quadratic, respectively, constraining the complexity of the model too much. A high-bias model is most likely to underfit the training data.
  Variance. This error is due to the model's excessive sensitivity to small variations in the training data, i.e, a model with many degrees of freedom (such sa a high-degree polynomial model) is likely to have a high variance and thus is most likely to overfit the training data.
  Irreducible error. This error is due to the noisiness of the data itself. The only way to tackle this problem is to do some pre-processing and clean up the data (remove null values, etc.) In summary: Increasing a model's complexity will typically increase its variance and reduce its bias. Conversely, reducing the model's complexity increasese its bias and reduces its variance. This trade-off is widely known as the *Bias-Variance trade-off)*

(b)
- M: Number of model variables, N: Number of observations (samples); $M \approx N$ instead of $M << N$, i.e. too many explaining variables/regressors are included into the model.
- Unstable estimator $\hat{w}_{LS}$, i.e. small perturbations in the data result in relatively large changes in the components of the estimator. Changes could be different or additional observations.

(c)
- On the error-function, a penalty/regularization-term is added, which penalizes the number of model parameters. $J_N^{PEN}(w) = \sum_{i=1}^{N}(y_i - f(x_i, w))^2 \Rightarrow \sum_{i=1}^{N}(y_i - f(x_i, w))^2 + \underbrace{\lambda \sum_{i=0}^{M-1} w_i^2}_{\text{Choice of } \lambda \text{ depends on domain-knowledge/experience}}$
- $N >> M$

Images to a)



Exercise 2-3 Regression w/ Regularization

Considering a regression problem and we are given the following cost function:

$$J(\boldsymbol{w}) = \frac{1}{m}\sum_{i=1}^{m}(f_{\boldsymbol{w}}(x_i) - y_i)^2$$

The regularization term $R(\boldsymbol{w}) = \lambda \cdot ||\boldsymbol{w}||^2$ is added to the model's cost function. Given this modification, derive its update rule.

Possible Solution

Adding the regularization term to the cost function, we get

$$J(\boldsymbol{w}) = \frac{1}{m}\sum_{i=1}^{m}(f_{\boldsymbol{w}}(x_i) - y_i)^2 + \lambda||\boldsymbol{w}||^2$$

The derivative w.r.t w_j is:

$$\frac{\partial J}{\partial w_j} = \frac{2}{m}\sum_{i=1}^{m}(f_w(x_i) - y_i)x_{i,j} + 2\lambda w_j$$

Therefore, the update rule is as follows:

$$w_j \leftarrow= w_j - \eta\Big(\frac{2}{m}\sum_{i=1}^{m}(f_w(x_i) - y_i)x_{i,j} + 2\lambda w_j\Big)$$

Exercise 2-4 PyTorch Basics + Linear Regression (optional)

On the course website you will find a jupyter notebook leading you through some basic tasks in pytorch and we will start with a simple implementation of a linear regression problem. Try to make yourself familiar with the basic learning procedure in PyTorch.