MSc in Computer Science
at University of Milan

Statistical Methods for Machine Learning

course held by **Nicoló Cesa-Bianchi**

Email:
leonardo.acquaroli@studenti.unimi.it

Created by:
Leonardo Acquaroli

Academic year 2022/2023

23 possible bonus questions

The following answers and the selection of the question have to be intended only as personal notes. The number in the square brackets like **[1]** indicate the chapter number as for the '22/'23 course.

1. **[1] - Absolute loss**
   $l(y, \hat{y} = |y - \hat{y}|$

2. **[2] - What happens when there are more than k nearest neighbours or there is a tie in the label count?**
   Just like in the case of 1-NN, there could be training points at the same distance from x such that more than k points are closest to x. In this case we proceed by ranking the training points based on their distance from x and then taking the k' closest points where k' is the smallest integer bigger or equal to k such that the (k'+1)-th point in the ranking has distance from x strictly larger that the k'-th point. If no such k' exists, then we take all the points in the training set. If k' is strictly bigger than k, even, and there is an equal number of closest points with positive and negative labels, then the algorithm predicts a default value in -1, 1.

3. **[2] - Bound on the training error of a predictor generated by kNN**
   $l_S(h_{kNN}) < 50\%$ of $|S|$ since kNN uses the majority criteria to assign labels.

4. **[3] - Can an unbounded tree achieve 0 training error? If yes, why is it to avoid?**
   A tree which is allowed to grow unbounded can always achieve zero training error but this will make the predictor incur in overfitting.

5. **[4] - Of which order of magnitude should m be in order to balance overfitting and underfitting in the ERM?**
   Given the bound on the variance of ERM, m must be sufficiently larger than $\ln |H|$

6. **[5] - ERM risk bound for unbounded trees**
   In this case the number of nodes $N = 2^{d+1} - 1$, $|H| = 2^d$ and the bound is:

$$l_D(h_S) \leq l_D(h^*) + \sqrt{\frac{2}{m}\big(2^d \ln(2) + \ln(\frac{2}{\delta})\big)}$$

7. **[5] - Explain the Occam's razor principle and how it is used for choosing the best tree predictor.**
   The Occam's razor principle states that among multiple solutions of the same problem the one which relies on the fewest steps or information is to be preferred. Thus, when it comes to choose a tree predictor the best one, given the same training error across multiple predictors, is the one with the smaller number of splits or more formally

$$\hat{h} = \arg\min_{h \in H} |\sigma(h)|$$

8. **[6] - Explain the Train-Validation-Test approach**
   To perform hyperparameters tuning the approach of Training-Validation-Test divides the dataset in S_{train}, $S_{validation}$ and S_{test} and then proceeds with four phases:

(a) Pass a list of suitable hyperparameter values (θ)

(b) Train the algorithm on S_{train} and validate on $S_{validation}$

(c) Take the predictor with the values of θ which minimize the validation error and train on $S = S_{train} + S_{validation}$

(d) Test on S_{test} and compute the test error.

9. **[7] - Explain why a nonparametric algorithm may fail to be consistent**
A nonparametric algorithm may fail to be consistent when the number fo features d enhance what is called "The curse of dimensionality", i.e. when there are two many features and too few training examples to extract a general relation between the features and produce a risk-less predictor.

10. **[7] - Rate of convergency towards for a non consistent parametric algorithm without any assumption on $\eta(x)$**
The risk of a non consistent algorithm would converge faster than the risk of a consistent algorithm, indeed the rate of convergency would be $m^{-\frac{1}{2}}$ but it would not tend the Bayes risk.

11. **[7] - Consistency for greedy algorithm for building trees**
In order for the greedy algorithm for building trees to be consistent the diameter of the leaf region tends to zero and the number of observations routed to each leaf tends to ∞ as $m \xrightarrow{\infty}$. That means that the tree must grow unboundedly but not too fast.

12. **[9] - Why do I need to add a constant feature to the training set when training linear predictors?**
Because it is computationally more efficient to learn only a vector $\vec{v} = (\vec{w}, c)$ instead of a vector $\vec{w}$ and a scalar c. in particular, for $x' = (\vec{x}, 1)$, the prediction becomes:

$$w^\top x + c = v^\top x'$$

13. **[9] - Write the mathematical definition of margin of an hyperplane on a training set**
Let $S = \{(x_1, y_1), \dots, (x_m, y_m)\}$ be the training set and u be the hyperplane generated by a linear predictor, the margin of u on S is defined as:

$$\gamma(u) = \min_{t=1,\dots,m} y_t u^\top x_t > 0$$

14. **[9] - Write the closed formula (i.e. not the argmin definition of the linear regression predictor)**
The linear regression is the ERM for the square loss and the vector of weights which it produces is obtained as:

$$w = \left(S^\top S\right)^{-1} S^\top y$$

15. **[10] - Define the sequential risk for Online learning algorithms**
Since the online learning procedures imply a local update on each epoch $t = 1, \dots, T$, the sequential risk is the average loss of a predictor over T epochs. In formula:

$$\frac{1}{T} \sum_{t=1}^{T} l\big(h_t(x_t), y_t\big)$$

16. **[11] - Explain what is the kernel trick**
The kernel trick is used to compute non-linear feature expansions that can reduce the bias of an algorithm. Since calculations with expanded features are computationally intensive a Kernel function is used in order to reduce the number of calculations and obtain the same result. For example the Kernel Perceptron predictions are made as sgn $\left(\sum_{s \in S} y_S K(x_s, x_t)\right)$ where $K(x_s, x_t)$ is the kernel function which computes the inner product of the non-linear maps $\phi(x)^\top \phi(x')$

17. **[12] - Write the update line of Pegasos run with Kernel SVM for non linear separable training sets**
The solution w^* to the SVM problem can be written as

$$w^* = \sum_{s \in S} y_s \alpha_s x_s$$

where $\alpha_s > 0$ and $S \equiv \{t = 1, \dots, m : h_t(w^*) > 0\}$. Thus we can lift w^* to a RKHS H_K, where the objective function F becomes

$$F_K(g) = \frac{1}{m} \sum_{t=1}^{m} h_t(g) + \frac{\lambda}{2} \|g\|_K^2, \quad g \in H_K$$

with $h_t(g) = \begin{cases} 1 - y_t g(x_t) & \text{if } h_t(g) > 0, \\ 0 & \text{otherwise.} \end{cases}$
In H_K, the SVM solution can therefore be written as $\sum_{s \in S} y_s \alpha_s K(x_s, \cdot)$ which is clearly an element of the RKHS. As we did for the Perceptron, we can run Pegasos in H_K. The gradient update in kernel Pegasos on some training example (x_{st}, y_{st}) can be written as

$$g_{t+1} = \left(1 - \frac{1}{t}\right) g_t + \frac{y_{st}}{\lambda_t} \mathbf{I}_{\{h_{st}(g_t) > 0\}} K(x_{st}, \cdot),$$

.

18. **[13] - What does it mean for an algorithm to be $\gamma - ERM$?**

$$l_S(h_S) \inf_{h \in H} l_S(h) + \gamma$$

19. **[14] - Write the gradient update of the weights of a feedforward neural network with error backpropagation**

$$\delta_j = \frac{\partial \ell_t(W)}{\partial s_j} = \begin{cases} \ell'(v_0)\sigma'(s_0) & \text{if } j = 0 \\ \sigma'(s_j) \sum_{k:(j,k) \in E} \delta_k w_{j,k} & \text{otherwise} \end{cases}$$

$$\frac{\partial \ell_t(W)}{\partial w_{i,j}} = \frac{\partial \ell_t(W)}{\partial s_j} \frac{\partial s_j}{\partial w_{i,j}} = \delta_j v_i$$

20. **[15] - Describe the double descent phenomenon**
Double descent is a still studied phenomenon which occurs when the size of the class $|H|$ of predictors of a neural network increases over what is called the interpolating threshold and make both the training and the test error diminish. The situation is also referred to as over-parametrized neural networks that means that the number of parameters is bigger than the number of training examples.

21. **[15] - Write the formula of the boosting loss function**

$$l(y, \hat{y}) = \exp -y\hat{y}$$

22. **[15] - Write the formula for Bayes optimal predictor and Bayes risk for the hinge loss**
The Bayes optimal predictor for the hinge loss is

$$g^*(x) = \operatorname*{arg\,min}_{\hat{y} \in \mathbb{R}} \left(\eta(x)[1 - \hat{y}]_+ + (1 - \eta(x))[1 + \hat{y}]_+\right)$$

$$g^*(x) = \begin{cases} -1 & \eta(x) \le 0.5 \\ +1 & \eta(x) > 0.5 \end{cases}$$

Therefore the Bayes risk is:

$$l_D(g*) = \mathbb{E}\big[\max\{\eta(x), 1-\eta(x)\}\big]$$

23. **[15] - Name two consistent surrogate loss functions**
Boosting: $l(y, \hat{y}) = e^{-y\hat{y}}$
Logistic: $l(y, \hat{y}) = \log_2(1 + e^{-y\hat{y}}$
Square loss: $l(y, \hat{y}) = (y - \hat{y})^2$
Hinge loss: $l(y, \hat{y}) = \max\{0, 1 - y_t\hat{y}\}$
Squared hinge loss: $l(y, \hat{y}) = (\max\{0, 1 - y_t\hat{y}\})^2$