EXAM 29/06/2020

EXERCISE N°1

POINT1

The binary classification problem is a supervised learning problem with :

- Domain set *X* which is the set of all possible objects to make predictions about, where a domain point $\vec{x} \in X$ is called instance and is usually represented by a vector of features
- Label set $Y = \{-1, +1\}$, that defines the set of all possible labels in this case the labels are only 2

Given a training set $S = \big((x_1, y_1) \dots (x_m, y_m)\big)$ with $x_i \in X, y_i \in Y \; \forall i = 1, .., m$ we need to choose an hypothesis class H, which defines the possible models or classification rules, from which we can pick our model to make predictions, and a loss function $l : HxZ \to R^+$ where $Z = XxY$ that given an hypothesis provides a measure of how much we lose by predicting the value $h(\vec{x})$ for $\vec{x}$ instead of the correct value y.
The goal is to find an hypothesis $\hat{h} \in H$ with low generalization error : $L_d(\hat{h}) = E_{z\sim D}[l(\hat{h}, z)]$ where :

- D is the unknown probability distribution over Z from which $(x_i, y_i) \in S$ have been drawn (as independent samples).

EXERCISE N°2

POINT1

Given a binary classification problem where :

- Domain set *X* which is the set of all possible objects to make predictions about, where a domain point $\vec{x} \in X$ is called instance and is usually represented by a vector of features
- Label set $Y = \{-1, +1\}$, that defines the set of all possible labels in this case the labels are only 2
- set $S = \big((x_1, y_1) \dots (x_m, y_m)\big)$ with $x_i \in X, y_i \in Y \; \forall i = 1, .., m$ is the so called training set

A separating hyperplane is an hyperplane that separates all the points of the training set S into two parts, namely into two halfspaces which contains all and only the points that have either class -1 or +1.
In case in which the training set S is linearly separable, then there exists more than one such hyperplane and in particular we can define the optimal as the one that maximizes the margin i.e. the distance between the hyperplane and the closest points in S to him.

The optimal separating hyperplane introduced above is exactly the solution computed by the SVM when data are linearly separable. In particular if we define :

- hyperplane as : $L = \{v : < \vec{w}, \vec{v} > \; + b = 0\}$
- distance between a point x and L : $d(x, L) = \min\{\big|\big|\vec{x} - \vec{v}\big|\big| : v \in L\}$

Then the optimal separating hyperplane can be computed as :

$$argmax_{(\vec{w},b):\, ||\vec{w}||=1 \; i\in\{1,..,m\}} \min |< \vec{w}, \vec{x_i} > \; + b| \;\; subject\ to : \; \forall\, i \; y_i(< \vec{w}, \vec{x_i} > + b)$$

Which corresponds to the formulation of the Hard-SVM, namely the formulation of the SVM when data are linearly separable

POINT2

Soft-SVM is similar to Hard-SVM, but can be used also when training data is not linearly separable (i.e., the training data cannot be perfectly classified with a linear model). Soft-SVM finds a model of large margin while allowing the training set to be inside the margin or wrongly classified. This is obtained by adding slack variables ξ_i to the constraints of hard SVM. The constraint for soft-SVM are then :
- $\xi_i \geq 0 \; for \; each \; i = 1, \dots, m$
- $y_i(< \vec{w}, \vec{x_i} > \; + b) \geq 1 - \; \xi_i \;\; for \; each \; (\vec{x_i}, y_i) \; i = 1, \dots, m$ in the training set where $\vec{w}$ and b defines the model.

The interpretation of ξ_i is the following :
- if $\xi_i = 0$ then $\vec{x_i}$ is correctly classified and outside the margin;
- if $0 < \xi_i < 1$ then $\vec{x_i}$ is correctly classified and inside the margin;
- if $\xi_i \geq 1$ then $\vec{x_i}$ is wrongly classified.

The function optimized by soft-SVM consider both the margin and the "violation" of the hard-SVM given by ξ_i.

Therefore the optimization problem for soft-SVM can be described as follows :
Input : $S = \left((\vec{x_1}, y_1) \dots (\vec{x_m}, y_m)\right)$ and parameter $\lambda > 0$
Goal : $min_{\vec{w}, b, \vec{\xi}} \; \lambda ||\vec{w}||^2 + \frac{1}{m} \sum_{i=1}^{m} \xi_i$
Output : $\vec{w}, b, \vec{\xi}$

The parameter lambda controls the tradeoff between an high margin with several points misclassified and a small margin with most of the points correctly classified in particular, if $\lambda \approx 0$ then, soft-SVM will produce a model that tries to be as similar as possible as the one produced by hard-SVM namely, it tries to classified as many as possible points correctly. On the other hand, if $\lambda \gg 0$ then, soft-SVM cares almost to maximize the margin therefore, it produces a model that has most of the points misclassified but it has an high margin.

EXERCISE N°3

POINT1

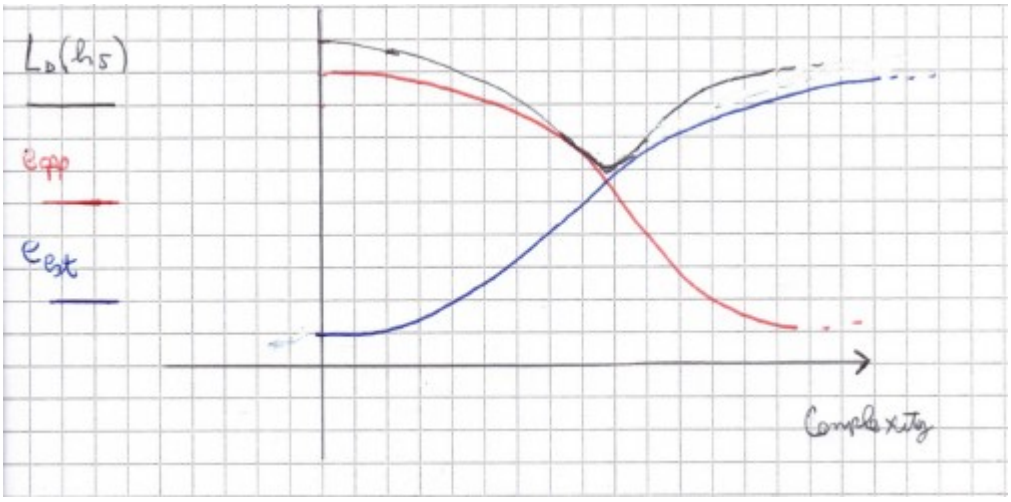
The generalization error $L_D(h_S)$ can be decomposed into $L_D(h_S) = \; \epsilon_{app} + \epsilon_{est}$ where $\epsilon_{app} = \; min_{h \in H} L_D(h)$ and $\epsilon_{est} = L_D(h_S) - \; min_{h \in H} L_D(h)$.
In words :

- ϵ_{app} = approximation error : is the minimum risk achivable by a predictior in the hypotesis class.
- ϵ_{est} = estimation error : is the difference between the approximation error an the error schive by the ERM predictor.

Therefore if we plot in a graph those two error and the $L_D(h_S)$ as functions of the complexity we obtain that :

- ϵ_{app} : will decrese if the complexity decreases if complexity increases
- ϵ_{est} : wil increase if the complexity increases
- $L_D(h_S)$: has a "bell" behaviour

The graph is the following



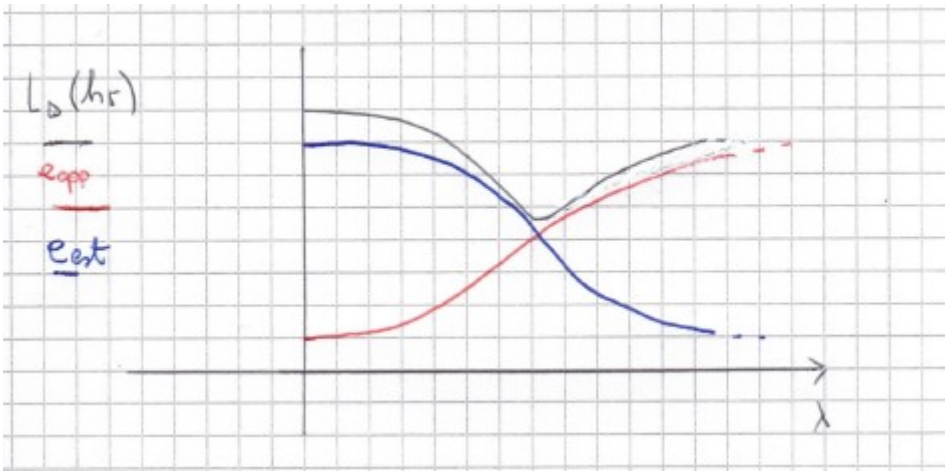
Notice that a low complexity hypothesis class corresponds to a underfitting situation while a high complexity class corresponds to a overfitting situation.

POINT2

Tikhonov regularization is the Regularized loss minimization paradigm with regularization function $R(\vec{w}) = \lambda||\vec{w}||^2$ where $\lambda > 0$ and $||\vec{w}||^2$ is the l_2 norm of the vector $\vec{w}$.
Therefore the function to be optimized, i.e. minimized is the following : $argmin_{\vec{w}}(L_S(\vec{w}) + \lambda\left|\left|\vec{w}\right|\right|^2)$ where $L_S(\vec{w})$ is the training error and S is the training set.
Notice that $||\vec{w}||^2$ measures the complexity of the hypothesis $\vec{w}$, while λ is the parameter used to tradeoff the empirical risk and the complexity of the model.
Now, the behavior of the ϵ_{app}, ϵ_{est} and $L_D(h_S)$ in terms of the parameter λ is the following :

- ϵ_{app} will increase if λ increases because with a low λ we have a complex hypothesis class
- ϵ_{est} will decrese if λ increases because with an high λ we have a low-complexity hypothesis class
- $L_D(h_S)$: has a “bell” behaviour

The graph is the following



EXERCISE N°4

POINT1

K-means is a cost minimization clustering problem. Let $X' \subseteq X$ be the set of points to be clustered with $X = \{\overrightarrow{x_1}, \dots, \overrightarrow{x_m}\}$ while X' is the space of possible points that we assume to be in R^d (i.e $X' = R^d$).
Let $k \in N^+$ be the number of clusters, that is, with X, the input of the problem.
Let $d(.)$ be a distance function : $d(\vec{x}, \overrightarrow{x'}) = ||\vec{x} - \overrightarrow{x'}||$.
The goal is to find :

- A partition $C = (C_1, .., C_k)$ of X
- Centroids $\overrightarrow{\mu_1}, .., \overrightarrow{\mu_k}$ of $C_1, .., C_k$ respectively

That minimizes the k-means cost function : $\sum_{i=1}^{k} \sum_{\vec{x} \in C_i} d(\vec{x}, \overrightarrow{\mu_i})^2$

To find the solution of this problem we can use an heuristic algorithm called Lloyd's Algorithm. Such algorithm works as follows :

Input: data points $X = \{\overrightarrow{x_1}, \dots, \overrightarrow{x_m}\}$ and $k \in N^+$
Output: $C = (C_1, .., C_k)$ of X and centroids $\overrightarrow{\mu_1}, .., \overrightarrow{\mu_k}$ of $C_1, .., C_k$ respectively

To produce such things the algorithm works as follows :

- Randomly initialize the centroids $\overrightarrow{\mu_1}, .., \overrightarrow{\mu_k}$
- Until the convergence is not reached iteratively repeats the following steps :
    1) Compute each cluster C_i by associating the points to him that closest to its centroids compared to others centroids.
    2) Compute the new centroids for the next iterations.
    3) If the convergence is reached then the output described above will be returned.

The pseudocode for such algorithm is the following:

Randomly choose $\vec{\mu}_1^0, .., \vec{\mu}_k^0$
for t <- 0,… do
  for i = 1,…,k do $C_i \leftarrow \{\vec{x} \in X : i\ argmin_j d(\vec{x}, \vec{\mu}_j^t)\}$
  for i = 1,…,k do $\vec{\mu}_i^{t+1} = \frac{1}{|C_i|} \sum_{\vec{x} \in C_i} \vec{x}$
  if convergence reached then
    return $C = (C_1, .., C_k)$ $\vec{\mu}_1^{t+1}, .., \vec{\mu}_k^{t+1}$

The convergence criteria that can be used for such algorithm are the following :

- The k-means objective function is no lower than the k-means objective function at the next iteration
- $\sum_{i=1}^{k} d(\vec{\mu}_i^{t+1}, \vec{\mu}_i^t) \leq \varepsilon$
- $max_{1 \leq i \leq k} d(\vec{\mu}_i^{t+1}, \vec{\mu}_i^t) \leq \varepsilon$

Note that if the first criteria is used then, the algorithm will always converge.

The time complexity is $O(tkmd)$ where $O(kmd)$ is due to the assignments of the points, while $O(md)$ is due to the computation of the centroids. Therefore the complexity depends on t which is the number of iterations of the algorithm

POINT2