- ***Describe the supervised learning technique denominated Support Vector Machines for classification problems.***

  A Support Vector Machine (SVM) performs classification by finding the hyperplane that maximizes the margin between two classes. The vectors (cases) that define the hyperplane are the support vectors.

  We define the hyperplane as $\mathbf{w}^T\mathbf{x} = 0$, it is possible to prove that, by observing that $\mathbf{w} \perp \text{hyperplane}$ and by adding the normalization constraint $|\mathbf{w}^T\mathbf{x}_n| = 1$ ( where $\mathbf{x}_n$ is the nearest point to the hyperplane) the distance from the hyperplane to $\mathbf{x}_n$ is:

  $$\text{distance} = \frac{1}{||\mathbf{w}||}$$

  We want to maximize this quantity (which is half the margin).

  Our optimization problem can be written as

  $$\underset{w}{\text{argmin}}\, \frac{1}{2}\mathbf{w}^T\mathbf{w}$$
  $$y_n(\mathbf{w}^T\mathbf{x}_n + b) \geq 1 \quad \text{for } n = 1, 2, \ldots, N$$

  Where $y_n$ is the target value associated with sample $n$.

  It is possible to formulate this problem as a constraint optimization problem with inequality constraints, we have to derive the *Lagrangian* and apply the *KKT* (Karush–Kuhn–Tucker) conditions. After that, we can solve the problem by means of *quadratic programming*.

  Finally we end up with the following equation for classifying *new points*:

  $$\hat{y}(\mathbf{x}) = sign\left(\sum_{n=1}^{N} \alpha_n y_n k(\mathbf{x}, \mathbf{x}_n) + b\right)$$

  where $\alpha_n$ are the *Lagrangian Multipliers*.

  The method described until here is called *hard-margin SVM* since the margin has to be satisfied strictly, it can happens that the points are not *linearly separable* in *any* way, or we just want to handle *noisy data* to avoid overfitting, so now we're going to briefly define another version of it, which is called *soft-margin SVM* that allows for few errors and penalizes for them. We introduce *slack variables* $\xi_i$ , in this way we allow to *violate* the margin constraint but we add a *penalty*.

  *Primal*:

  $$\text{Minimize } \; ||\mathbf{w}||_2^2 + C\sum_i \xi_i$$
  $$\text{s.t.}$$
  $$y_i(\mathbf{w}^T x_i + b) \geq 1 - \xi_i \;, \quad \forall i$$
  $$\xi_i \geq 0 \;, \quad \forall i$$

  *Dual:*

  $$\text{Maximize } \quad \mathcal{L}(\alpha) = \sum_{n=1}^{N} \alpha_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{M} y_n y_m \alpha_n \alpha_m k(\mathbf{x}_n \mathbf{x}_m)$$
  $$\text{s.t.}$$
  $$0 \leq \alpha_n \leq C \quad \forall i$$
  $$\sum_{n=1}^{N} \alpha_n t_n = 0$$

  The *dual formulation* allows us to use *kernels* $k(\mathbf{x}_n \mathbf{x}_m)$

Support vectors are points associated with $\alpha_n > 0$

if $\alpha_n < C$ the points lies *on the margin*

if $\alpha_n = C$ the point lies *inside the margin*, and it can be either *correctly classified* ($\xi_i \leq 1$) or *misclassified* ($\xi_i > 1$)

When $C$ is large, larger slacks penalize the objective function of SVM's more than when $C$ is small. As $C$ approaches infinity, this means that having any slack variable set to non-zero would have infinite penalty. Consequently, as $C$ approaches infinity, all slack variables are set to $0$ and we end up with a hard-margin SVM classifier.

- ***Which algorithm can we use to train an SVM? Provide an upper bound to the generalization error of an SVM.***

  Sometimes for computational reasons, when we solve a problem characterized by a huge dataset, it is not possible to compute *all* the support vectors with generic quadratic programming solvers (the number of constraints depends on the number of samples), hence, specialized optimization algorithms are often used. One example is *Sequential Minimal Optimization (SMO)*:

  Remember our formulation for the *soft-margin SVM*:

$$\mathcal{L}(\alpha) = \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{M} y_n y_m \alpha_n \alpha_m k(\mathbf{x}_n \mathbf{x}_m)$$
$$s.t.$$
$$0 \leq \alpha_i \leq C \quad \text{for } i = 1, 2, \ldots, n$$
$$\sum_{i=1}^{n} y_i \alpha_i = 0$$

  *SMO* breaks this problem into a series of smallest possible sub-problems, which are then solved analytically. Because of the linear equality constraint involving the Lagrange multipliers $\alpha_i$ , the smallest possible problem involves two such multipliers. Then, for any two multipliers $\alpha_1$ and $\alpha_2$ the constraints are reduced to:

$$0 \leq \alpha_1, \alpha_2 \leq C$$
$$y_1 \alpha_1 + y_2 \alpha_2 = k$$

  and this reduced problem can be solved analytically: one needs to find a minimum of a one-dimensional quadratic function. $k$ is the negative of the sum over the rest of terms in the equality constraint, which is fixed in each iteration ( we do this because we want that $\sum_{i=1}^{n} y_i \alpha_i = 0$ ).

  The algorithm proceeds as follows:

  - Find a Lagrange multiplier $\alpha_1$ that violates the *KKT* conditions for the optimization problem.
  - Pick a second multiplier $\alpha_2$ and optimize the pair ($\alpha_1$,$\alpha_2$).
  - Repeat steps $1$ and $2$ until convergence.

  When all the Lagrange multipliers satisfy the KKT conditions (within a user-defined tolerance), the problem has been solved. Although this algorithm is guaranteed to converge, heuristics are used to choose the pair of multipliers so as to accelerate the rate of convergence. This is critical for large data sets since there are $\frac{n(n-1)}{2}$ possible choices for $\alpha_i$ and $\alpha_j$ .

  As *Vapnik* said: "In the support-vectors learning algorithm the complexity of the construction does not depend on the dimensionality of the feature space, but on the number of support vectors." So it's reasonable to define an upper bound of the error as:

$$L_h \leq \frac{\mathbb{E}[\text{number of support vectors}]}{N}$$

This is called *Leave-One-Out Bound*. The good thing is that it can be easily computed and we don't need to run SVM multiple times.

The other kind of bound is called *Margin bound*: a bound on the VC dimension which decreases with the margin. The larger the margin, the less the variance and so, the less the VC dimension. Unfortunately the bound is quite pessimistic .

- ***Define the VC dimension and describe the importance and usefulness of VC dimension in machine learning. What is the VC dimension of a linear classifier?***

  A *dichotomy* is a hypothesis $h$ that maps an input from the *sample size* to a result:

  $$h : \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\} \to \{-1, +1\}$$

  The *growth function* is a function that counts the *most* dichotomies on any $N$ points.

  $$m_{\mathcal{H}}(N) = \max_{\mathbf{x}_1, \ldots, \mathbf{x}_N \in \mathcal{X}} |\mathcal{H}(\mathbf{x}_1, \ldots, \mathbf{x}_N)|$$

  This translates into choosing any $N$ points and laying them out in *any* fashion in the input space. Determining $m$ is equivalent to looking for such a layout of the $N$ points that yields the *most* dichotomies.

  The growth function satisfies:

  $$m_{\mathcal{H}}(N) \leq 2^N$$

  The *VC ( Vapnik-Chervonenkis ) dimension* of a hypothesis set $\mathcal{H}$ , denoted by $d_{VC}(\mathcal{H})$ is the largest value of $N$ for which $m_{\mathcal{H}}(N) = 2^N$ , in other words is "*the most points $\mathcal{H}$ can shatter* "

  We can say that the *VC* dimension is one of many measures that characterize the expressive power, or capacity, of a hypothesis class.

  You can think of the VC dimension as "how many points can this model class memorize/shatter?" (a ton? $\to$ BAD! not so many? $\to$ GOOD!).

  With respect to learning, the effect of the VC dimension is that if the VC dimension is finite, then the hypothesis will generalize:

  $$d_{vc}(\mathcal{H}) \implies g \in \mathcal{H} \text{ will generalize}$$

  The key observation here is that this statement is not dependent on:

  - The learning algorithm
  - The input distribution
  - The target function

  The only things that factor into this are the training examples, the hypothesis set, and the final hypothesis.

  The VC dimension for a linear classifier (i.e. a *line* in 2D, a *plane* in 3D etc...) is $d + 1$ (a line can shatter at most $2 + 1 = 3$ points, a plane can shatter at most $3 + 1 = 4$ points etc...)