Outline: Supervised learning

1. **Linear machines**
   - Gradient descent
   - Linear Classification
   - Linear Regression
   - Logistic regression
   - Polynomial regression

2. **Learning theory**
   - Error measurement
   - Cross-validation
   - PAC learning theory and VC dimension
   - Feature space

3. **Neural networks**
   - Feature space
   - Multi-layer perceptron
   - Radial-basis function network

4. **Support Vector Machines**
   - Principle of a maximal margin classifier
   - Lagrange method for optimization
   - Soft-margin classifier
   - Working in a feature space: the kernel trick
   - Learning procedures in a SVM
Rationale for support-vector machines

- MLP and RBF networks do not care about the "quality" of their classification: once they find an hyperplane that correctly classifies the training data, they stop.

- The number of hidden neurons in MLP and RBF is free: it has to be found through cross-validation.

- If the number of hidden neurons is big, the training error becomes small, but the generalization error increases and the computational complexity of both learning and usage becomes untractable.

- If one could work in a virtually infinite feature space but with keeping a finite VC dimension, the training and generalization errors would both be kept small.

\[
\epsilon(h) \leq \hat{\epsilon}_S(h) + \sqrt{\frac{VC_{\text{dim}}(\mathcal{H}) \cdot (1 + \log(\frac{2 \cdot N}{VC_{\text{dim}}(\mathcal{H})}))}{N}} - \log\left(\frac{\delta}{4}\right)
\]

- This is the main idea of support-vector machines, created by Vladimir Vapnik in the 60's, but developed and used in the 90's when coupled with the kernel trick.
Principle of a maximal margin classifier
Why is the margin important?

Classical classifiers (linear, MLP, RBF...) do not care about their generalization error: they only concentrate on the training set.
Why is the margin important?

A successful classifier with a very small margin is an acceptable solution.
Why is the margin important?

However, classifiers with small margins are more likely to make mistakes on new examples (noise in the data).
Why is the margin important?

A classifier with a fat margin has the same success on the training set...
Why is the margin important?

... but is much less likely to make a mistake on new examples: its generalization error is bigger.
Effect of the margin on the VC dimension

- The VC dimension is linked to the number of possible dichotomies learnable by a classifier on the training set.
Effect of the margin on the VC dimension

- Forcing a fat margin for a classifier reduces the number of possible dichotomies, therefore reduces its VC dimension.

- A lower VC dimension means better generalization:

\[ \epsilon(h) \leq \hat{\epsilon}_S(h) + \sqrt{\frac{V\text{C}_{\text{dim}}(\mathcal{H}) \cdot (1 + \log\left(\frac{2 \cdot N}{V\text{C}_{\text{dim}}(\mathcal{H})}\right)) - \log\left(\frac{\delta}{4}\right)}{N}} \]
Maximal margin classifier for linearly separable data

- Functional margin of an example:
  \[ \hat{\gamma}_i = y_i \cdot (\langle w, x_i \rangle + b) \]

- Geometric margin of an example:
  \[ \gamma_i = y_i \cdot \left( \frac{\langle w, x \rangle + b}{\|w\|} \right) \]

- Functional margin of a training set:
  \[ \hat{\gamma} = \min_i \hat{\gamma}_i \]

- Geometric margin of a training set:
  \[ \gamma = \min_i \gamma_i \]
The goal of a maximal margin classifier is to find the hyperplane $F(x) = \langle w, x \rangle + b$ which has the maximal geometric margin while correctly classifying every example:

$$\text{maximize}_{w,b} \quad \gamma = \min \gamma_i = \min_i y_i \cdot \left( \frac{\langle w, x \rangle + b}{\|w\|} \right) = \frac{\hat{\gamma}}{\|w\|}$$

s.t. $\hat{\gamma}_i = y_i \cdot (\langle w, x_i \rangle + b) \geq \hat{\gamma}$ $\forall i$. 

Maximal margin classifier for linearly separable data
Maximal margin classifier for linearly separable data

- Problem: the geometric margin is a non-convex function of $\mathbf{w}$, therefore very hard to optimize (local maxima).

- The decision function $F(x) = \langle \mathbf{w}, \mathbf{x} \rangle + b$ is not influenced by the norm of the vector, only the functional margin is.
Maximal margin classifier for linearly separable data

- We can reformulate the problem by forcing the functional margin to 1:

$$\maximize_{w, b} \quad \gamma = \frac{1}{\|w\|}$$

$s.t. \quad \gamma_i = y_i \cdot (\langle w. x_i \rangle + b) \geq 1 \quad \forall i.$

- This is only a convention to ease mathematical analysis, but it does not change the result of the classification:

$$\langle w. x \rangle + b = 0$$
Maximal margin classifier for linearly separable data

- The optimization problem is now:

$$\text{maximize}_{w,b} \quad \gamma = \frac{1}{\|w\|}$$

$$\text{s.t.} \quad y_i \cdot (\langle w, x_i \rangle + b) \geq 1 \quad \forall i.$$  

- Or alternatively:

$$\text{minimize}_{w,b} \quad \frac{1}{2} \cdot \|w\|^2 = \frac{1}{2} \cdot \langle w, w \rangle$$

$$\text{s.t.} \quad y_i \cdot (\langle w, x_i \rangle + b) \geq 1 \quad \forall i.$$  

- The maximal margin classifier tries to minimize the norm of the weight vector while ensuring that every example has a functional margin of at least 1.

- The optimization problem is now a quadratic optimization problem, having therefore only one solution and being very easy to compute.

- The linear inequality constraints restrict the correct values for the weight vector (a zero weight vector would have the minimal norm, but would not satisfy the constraints).
Lagrange method for optimization
Optimization under constraints

- Global optimization:

\[
\text{minimize}_x \quad f(x)
\]

We search its minimum \( x^* \) through:

\[
\frac{\partial f}{\partial x}(x^*) = 0
\]
Optimization under constraints

- Optimization under linear equality constraints:

  \[
  \text{minimize}_x \quad f(x) \quad \text{s.t.} \quad g(x) = 0
  \]

We define the Lagrange function for the constrained optimization problem, parameterized by \( \lambda \), called a Lagrange multiplier:

\[
\mathcal{L}(x, \lambda) = f(x) + \lambda \cdot g(x)
\]

and search for its minimum \((x^*, \lambda^*)\):

\[
\text{minimize}_{x, \lambda} \quad \mathcal{L}(x, \lambda) \iff \frac{\partial \mathcal{L}}{\partial x}(x^*, \lambda^*) = 0 \quad ; \quad \frac{\partial \mathcal{L}}{\partial \lambda}(x^*, \lambda^*) = 0
\]

\( \rightarrow x^* \) is the minimum of \( f \) on the subspace defined by \( g(x) = 0 \).
Optimization under constraints

- **Optimization under linear inequality constraints:**

\[
\text{minimize}_x \quad f(x) \quad \text{s.t.} \quad h(x) \leq 0
\]

We define the generalized Lagrange function for the constrained optimization problem, with a parameter \( \alpha \geq 0 \) called the **Karush-Kuhn-Tucker** (KKT) multiplier:

\[
\mathcal{L}(x, \alpha) = f(x) + \alpha \cdot h(x)
\]

We then search \((x^*, \alpha^*)\) respecting the **Karush-Kuhn-Tucker** conditions:

\[
\begin{cases}
\frac{\partial \mathcal{L}}{\partial x}(x^*, \alpha^*) = 0 \\
\alpha^* \cdot \frac{\partial \mathcal{L}}{\partial \alpha}(x^*, \alpha^*) = 0 \iff \alpha^* \cdot h(x^*) = 0 \\
\alpha^* \geq 0 \\
h(x^*) \leq 0
\end{cases}
\]

\(\rightarrow x^*\) is the minimum of \(f\) on the subspace defined by \(h(x) \leq 0\)
Optimization under constraints: example 1

- Goal: minimize the quadratic function in 2D

\[
\min_{x,y} \ f(x, y) = x^2 + y^2
\]

- We only need to find where the partial derivatives of this function are zero:

\[
\begin{align*}
\frac{\partial f}{\partial x}(x^*, y^*) &= 2 \cdot x^* = 0 \\
\frac{\partial f}{\partial y}(x^*, y^*) &= 2 \cdot y^* = 0
\end{align*}
\]

- The minimum of this function is therefore obtained for \((x^*, y^*) = (0, 0)\).
Optimization under constraints: example 2

- Goal: minimize the quadratic function in 2D under linear equality constraint:

\[
\begin{align*}
\text{minimize}_{x,y} \quad & f(x, y) = x^2 + y^2 \\
\text{s.t.} \quad & g(x, y) = x + 2y + 1 = 0
\end{align*}
\]
Optimization under constraints: example 2

\[
\begin{align*}
\text{minimize}_{x,y} & \quad f(x, y) = x^2 + y^2 \\
\text{s.t.} & \quad g(x, y) = x + 2y + 1 = 0
\end{align*}
\]

We define the Lagrangian of this optimization problem:

\[
L(x, y, \lambda) = f(x, y) + \lambda \cdot g(x, y) = x^2 + y^2 + \lambda \cdot (x + 2y + 1)
\]

and we search for its minimum \((x^*, y^*, \lambda^*)\):

\[
\begin{align*}
\frac{\partial L}{\partial x}(x^*, y^*, \lambda^*) &= 2 \cdot x^* + \lambda^* = 0 \\
\frac{\partial L}{\partial y}(x^*, y^*, \lambda^*) &= 2 \cdot y^* + 2\lambda^* = 0 \\
\frac{\partial L}{\partial \lambda}(x^*, y^*, \lambda^*) &= x^* + 2y^* + 1 = 0
\end{align*}
\]

We obtain a system of 3 linear equations with 3 variables. We obtain the only solution \((x^*, y^*, \lambda^*) = (-\frac{1}{5}, -\frac{2}{5}, \frac{2}{5})\). This solution verifies \(x^* + 2y^* + 1 = 0\) and we have:

\[
f(x^*, y^*) = \frac{1}{5}
\]
Optimization under constraints: example 3

- Goal: minimize the quadratic function in 2D under linear inequality constraint:

\[
\begin{align*}
\text{minimize}_{x,y} & \quad f(x, y) = x^2 + y^2 \\
\text{s.t.} & \quad h(x, y) = x + 2y + 1 \leq 0
\end{align*}
\]
Optimization under constraints : example 3

$$\begin{align*}
\text{minimize}_{x,y} & \quad f(x, y) = x^2 + y^2 \\
\text{s.t.} & \quad h(x, y) = x + 2y + 1 \leq 0
\end{align*}$$

We define the generalized Lagrangian of this optimization problem:

$$\mathcal{L}(x, y, \alpha) = f(x, y) + \alpha \cdot h(x, y) = x^2 + y^2 + \alpha \cdot (x + 2y + 1)$$

and we search for a tuple \((x^*, y^*, \alpha^*)\) satisfying the Karush-Kuhn-Tucker conditions:

$$\begin{cases}
\frac{\partial \mathcal{L}}{\partial x}(x^*, y^*, \alpha^*) = 2 \cdot x^* + \alpha^* = 0 \\
\frac{\partial \mathcal{L}}{\partial y}(x^*, y^*, \alpha^*) = 2 \cdot y^* + 2\alpha^* = 0 \\
\alpha^* \cdot h(x^*, y^*) = \alpha^* \cdot (x^* + 2y^* + 1) = 0 \\
\alpha^* \geq 0 \\
h(x^*, y^*) = x^* + 2y^* + 1 \leq 0
\end{cases}$$
Optimization under constraints: example 3

\[ \begin{array}{l}
\text{minimize}_{x,y} \quad f(x, y) = x^2 + y^2 \\
\text{s.t.} \quad h(x, y) = x + 2y + 1 \leq 0
\end{array} \]

The third KKT condition admits two cases:

\[ \alpha^* \cdot h(x^*, y^*) = \alpha^* \cdot (x^* + 2y^* + 1) = 0 \]

- \( \alpha^* = 0 \): it implies \((x^* + 2y^* + 1)\) can have any value. The two first conditions become:
  \[ \begin{align*}
  2 \cdot x^* + \alpha^* &= 2 \cdot x^* = 0 \\
  2 \cdot y^* + 2\alpha^* &= 2 \cdot y^* = 0
  \end{align*} \]

which have the solution \((x^*, y^*) = (0, 0)\) but which is in conflict with the last KKT condition \(x^* + 2y^* + 1 \leq 0\). The system is then not solvable, and it is not possible that \(\alpha^* = 0\).
Optimization under constraints: example 3

\[
\begin{align*}
\text{minimize} \quad & f(x, y) = x^2 + y^2 \\
\text{s.t.} \quad & h(x, y) = x + 2y + 1 \leq 0
\end{align*}
\]

The third KKT condition admits two cases:

\[
\alpha^* \cdot h(x^*, y^*) = \alpha^* \cdot (x^* + 2y^* + 1) = 0
\]

- \( \alpha^* > 0 \): it implies \( x^* + 2y^* + 1 = 0 \). We now have the system:
  \[
  \begin{align*}
  2 \cdot x^* + \alpha^* &= 0 \\
  2 \cdot y^* + 2\alpha^* &= 0 \\
  x^* + 2y^* + 1 &= 0
  \end{align*}
  \]
  which has the solution \((x^*, y^*, \alpha^*) = (-\frac{1}{5}, -\frac{2}{5}, \frac{2}{5})\). This solution verifies \( x^* + 2y^* + 1 = 0 \leq 0 \).

When \( \alpha \) is strictly positive, it means the solution is on the border of the inequality: the constraint is said "saturated".
Optimization under constraints: example 4

- Goal: minimize the quadratic function in 2D under linear inequality constraint:

\[
\min_{x, y} \quad f(x, y) = x^2 + y^2 \\
\text{s.t.} \quad h(x, y) = -x - 2y - 1 \leq 0
\]
Optimization under constraints: example 4

\[
\begin{align*}
\text{minimize}_{x,y} & \quad f(x, y) = x^2 + y^2 \\
\text{s.t.} & \quad h(x, y) = -x - 2y - 1 \leq 0
\end{align*}
\]

We define the generalized Lagrangian of this optimization problem:

\[
\mathcal{L}(x, y, \alpha) = f(x, y) + \alpha \cdot h(x, y) = x^2 + y^2 + \alpha \cdot (-x - 2y - 1)
\]

and we search for a tuple \((x^*, y^*, \alpha^*)\) satisfying the Karush-Kuhn-Tucker conditions:

\[
\begin{aligned}
\frac{\partial \mathcal{L}}{\partial x}(x^*, y^*, \alpha^*) &= 2 \cdot x^* + \alpha^* = 0 \\
\frac{\partial \mathcal{L}}{\partial y}(x^*, y^*, \alpha^*) &= 2 \cdot y^* + 2\alpha^* = 0 \\
\alpha^* \cdot h(x^*, y^*) &= \alpha^* \cdot (-x^* - 2y^* - 1) = 0 \\
\alpha^* &\geq 0 \\
h(x^*, y^*) &= -x^* - 2y^* - 1 \leq 0
\end{aligned}
\]
Optimization under constraints: example 4

minimize \( x, y \) \( f(x, y) = x^2 + y^2 \)

s.t. \( h(x, y) = -x - 2y - 1 \leq 0 \)

The third KKT condition admits two cases:

\[
\alpha^* \cdot h(x^*, y^*) = \alpha^* \cdot (-x^* - 2y^* - 1) = 0
\]

- \( \alpha^* = 0 \): it implies \((x^* + 2y^* + 1)\) can have any value. The two first conditions become:

\[
2 \cdot x^* + \alpha^* = 2 \cdot x^* = 0
\]
\[
2 \cdot y^* + 2\alpha^* = 2 \cdot y^* = 0
\]

which have the solution \((x^*, y^*) = (0, 0)\). This solution now fits with the last KKT condition \(-x^* - 2y^* - 1 \leq 0\). This is an admissible solution.
Optimization under constraints : example 4

\[ \begin{align*}
\text{minimize}_{x,y} & \quad f(x, y) = x^2 + y^2 \\
\text{s.t.} & \quad h(x, y) = -x - 2y - 1 \leq 0
\end{align*} \]

The third KKT condition admits two cases:

\[ \alpha^* \cdot h(x^*, y^*) = \alpha^* \cdot (-x^* - 2y^* - 1) = 0 \]

- \( \alpha^* > 0 \) : it implies \(-x^* - 2y^* - 1 = 0 \). We now have the system:

\[ \begin{align*}
2 \cdot x^* - \alpha^* &= 0 \\
2 \cdot y^* - 2\alpha^* &= 0 \\
x^* + 2y^* + 1 &= 0
\end{align*} \]

which has the solution \((x^*, y^*, \alpha^*) = (-\frac{1}{5}, -\frac{2}{5}, -\frac{2}{5})\). This solution is in conflict with the KKT condition \( \alpha^* > 0 \). It is therefore not valid.

When \( \alpha \) is zero, it means the solution is largely within the inequality constraint: the constraint is said "free".
Generalized Lagrange Method

\[
\begin{align*}
\text{minimize}_{\mathbf{x}} & \quad f(\mathbf{x}) \\
\text{s.t.} & \quad g_i(\mathbf{x}) = 0 \quad \forall i \in [1, l] \\
\text{s.t.} & \quad h_j(\mathbf{x}) \leq 0 \quad \forall j \in [1, k]
\end{align*}
\]

We write the Lagrangian:

\[
\mathcal{L}(\mathbf{x}, \lambda_1, \ldots, \lambda_l, \alpha_1, \ldots, \alpha_k) = f(\mathbf{x}) + \sum_{i=1}^{l} \lambda_i \cdot g_i(\mathbf{x}) + \sum_{j=1}^{k} \alpha_j \cdot h_j(\mathbf{x})
\]

and find the variables \((\mathbf{x}^*, \lambda_1^*, \ldots, \lambda_l^*, \alpha_1^*, \ldots, \alpha_k^*)\) which satisfy the KKT conditions:

\[
\begin{align*}
\frac{\partial \mathcal{L}}{\partial \mathbf{x}}(\mathbf{x}^*, \lambda_1^*, \ldots, \lambda_l^*, \alpha_1^*, \ldots, \alpha_k^*) &= 0 \\
\frac{\partial \mathcal{L}}{\partial \lambda_i}(\mathbf{x}^*, \lambda_1^*, \ldots, \lambda_l^*, \alpha_1^*, \ldots, \alpha_k^*) &= 0 \quad \forall i \in [1, l] \\
\alpha_j^* \cdot \frac{\partial \mathcal{L}}{\partial \alpha_j}(\mathbf{x}^*, \lambda_1^*, \ldots, \lambda_l^*, \alpha_1^*, \ldots, \alpha_k^*) &= 0 \quad \forall j \in [1, k]
\end{align*}
\]

\[
\begin{align*}
\alpha_j^* &\geq 0 \quad \forall j \in [1, k] \\
h_j(\mathbf{x}^*) &\leq 0 \quad \forall j \in [1, k]
\end{align*}
\]
Maximal margin classifier
Primal form of the maximal margin classifier

Definition of the maximal margin classifier

Given a linearly separable training set:

\[ S = \{(x_1, y_1), \ldots, (x_N, y_N)\} \]

the optimal hyperplane \((w^*, b^*)\) that solves the optimization problem:

\[
\begin{align*}
\text{minimize}_{w, b} & \quad \frac{1}{2} \cdot \|w\|^2 = \frac{1}{2} \cdot \langle w, w \rangle \\
\text{s.t.} & \quad 1 - y_i \cdot (\langle w, x_i \rangle + b) \leq 0 \quad \forall i \in [1, N]
\end{align*}
\]

realizes the maximal margin hyperplane with geometric margin:

\[
\gamma = \frac{1}{\|w\|}
\]
Lagrangian of the maximal margin classifier

\[ L(w, b, \alpha_1, \ldots, \alpha_N) = \frac{1}{2} \cdot \langle w, w \rangle + \sum_{i=1}^{N} \alpha_i \cdot (1 - y_i \cdot (\langle w, x_i \rangle + b)) \]

The optimal solution \((w^*, b^*, \alpha_1^*, \ldots, \alpha_N^*)\) satisfies the Karush-Kuhn-Tucker conditions:

\[
\frac{\partial L}{\partial w}(w^*, b^*, \alpha_1^*, \ldots, \alpha_N^*) = w^* - \sum_{i=1}^{N} \alpha_i^* \cdot y_i \cdot x_i = 0
\]

\[
\frac{\partial L}{\partial b}(w^*, b^*, \alpha_1^*, \ldots, \alpha_N^*) = - \sum_{i=1}^{N} \alpha_i^* \cdot y_i = 0
\]

\[
\alpha_i^* \cdot (1 - y_i \cdot (\langle w^*, x_i \rangle + b^*)) = 0 \quad \forall i \in [1, N]
\]

\[
\alpha_i^* \geq 0 \quad \forall i \in [1, N]
\]

\[
(1 - y_i \cdot (\langle w^*, x_i \rangle + b^*)) \leq 0 \quad \forall i \in [1, N]
\]
The weight vector depends on the data

The first KKT condition tells us that the optimal weight vector will be a linear combination of the training data:

\[
\mathbf{w}^* = \sum_{i=1}^{N} \alpha_i^* \cdot y_i \cdot \mathbf{x}_i
\]

The intercept term \(b^*\) disappeared from the conditions, but can be found through:

\[
b^* = -\frac{\max_{i \in C^-} \langle \mathbf{w}^*, \mathbf{x}_i \rangle + \min_{i \in C^+} \langle \mathbf{w}^*, \mathbf{x}_i \rangle}{2}
\]
Dual form of the Perceptron algorithm

- Primal form of the learning rule: $w \leftarrow w + \eta \cdot (y_i - f_w(x_i)) \cdot x_i$

- If $w(0) = 0$, the weight vector converges towards a linear combination of the examples:

$$w = \sum_{j=1}^{N} \alpha_j \cdot y_j \cdot x_j$$

- $\alpha_j$ is proportional to the number of times when the example was misclassified: embedding strength.

- The hypothesis can be rewritten in the dual form:

$$f_w(x) = \text{sign}(\langle w \cdot x \rangle)$$

$$= \text{sign}(\langle \sum_{j=1}^{N} \alpha_j \cdot y_j \cdot x_j \cdot x \rangle)$$

$$= \text{sign}(\sum_{j=1}^{N} \alpha_j \cdot y_j \cdot \langle x_j \cdot x \rangle)$$
Dual form of the Perceptron algorithm

\[ \alpha \leftarrow 0^N \]

\textbf{while} \( \hat{\gamma} < 0 \) :

\textbf{forall} examples \((x_i, y_i)\):

\[ \hat{\gamma}_i = y_i \cdot \sum_{j=1}^{N} \alpha_j \cdot y_j \cdot \langle x_j \cdot x_i \rangle \]

\textbf{if} \( \hat{\gamma}_i < 0 \):

\[ \alpha_i = \alpha_i + 1 \]

\[ \hat{\gamma} = \min_i \hat{\gamma}_i \]

The dual form of an algorithm only relies on the Gram matrix of the training examples:

\[ G = X^T \cdot X = (\langle x_i \cdot x_j \rangle)_{i,j=1..N} \]
Dual form of the maximal margin classifier

If we plug back the optimal value of $\mathbf{w}^* = \sum_{i=1}^{N} \alpha_i^* \cdot y_i \cdot \mathbf{x}_i$ into the Lagrangian, we obtain:

$$
\mathcal{L}(\mathbf{w}, b, \alpha_1, \ldots, \alpha_N) = \frac{1}{2} \cdot \langle \mathbf{w} \cdot \mathbf{w} \rangle + \sum_{i=1}^{N} \alpha_i \cdot (1 - y_i \cdot (\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b))
$$

$$
= \frac{1}{2} \cdot \langle \left( \sum_{i=1}^{N} \alpha_i \cdot y_i \cdot \mathbf{x}_i \right) \cdot \left( \sum_{j=1}^{N} \alpha_j \cdot y_j \cdot \mathbf{x}_j \right) \rangle
$$

$$
+ \sum_{i=1}^{N} \alpha_i \cdot (1 - y_i \cdot (\langle \sum_{j=1}^{N} \alpha_j \cdot y_j \cdot \mathbf{x}_j \rangle \cdot \mathbf{x}_i \rangle + b))
$$

$$
= \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \cdot \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \cdot \alpha_j \cdot y_i \cdot y_j \cdot \langle \mathbf{x}_i \cdot \mathbf{x}_j \rangle
$$
Dual form of the maximal margin classifier

Given a linearly separable training set:

\[ S = \{(x_1, y_1), \ldots, (x_N, y_N)\} \]

the parameters \((\alpha_1, \ldots, \alpha_N)\) that solve the optimization problem:

\[
\text{maximize}_{\alpha_1, \ldots, \alpha_N} \quad Q(\alpha_1, \ldots, \alpha_N) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \cdot \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \cdot \alpha_j \cdot y_i \cdot y_j \cdot \langle x_i, x_j \rangle
\]

s.t. \[ \sum_{i=1}^{N} \alpha_i \cdot y_i = 0 \]

s.t. \[ \alpha_i \geq 0 \quad \forall i \in [1, N] \]

define a decision rule \(\text{sign}(F(x))\) where:

\[
F(x) = \sum_{i=1}^{N} \alpha_i \cdot y_i \cdot \langle x_i, x \rangle + b
\]

that is equivalent to the maximal margin hyperplane with the geometric margin:

\[
\gamma = \frac{1}{\sqrt{\sum_{i=1}^{N} \alpha_i}}
\]
The (theoretical) weight vector of the hyperplane is entirely defined by the $\alpha_i$ and the training data:

$$\mathbf{w}^* = \sum_{i=1}^{N} \alpha_i^* \cdot y_i \cdot \mathbf{x}_i$$

However, only the training examples whose constraint is saturated ($y_i \cdot (\langle \mathbf{w}^* \cdot \mathbf{x}_i \rangle + b^*) = 1$) will have a KKT multiplier different from 0.

$\rightarrow$ These examples are called support vectors.

The $N_{SV} < N$ support vectors and their KKT multipliers are the only relevant information after learning:
\[ w^* = \sum_{i=1}^{N_{SV}} \alpha_i^* \cdot y_i \cdot x_i \]

\[ b^* = 1 - \sum_{i=1}^{N_{SV}} \alpha_i \cdot y_i \cdot \langle x_i, x_{SV}^+ \rangle \quad \text{where} \quad x_{SV}^+ \quad \text{is a positive support vector.} \]

\[ F(x) = \sum_{i=1}^{N_{SV}} \alpha_i \cdot y_i \cdot \langle x_i, x \rangle + b^* \]
Maximal-margin classifier: the linear support-vector machine

\[ F(x) = \sum_{i=1}^{N_{SV}} \alpha_i \cdot y_i \cdot \langle x_i \cdot x \rangle + b^* \]

\[ \sum_{i=1}^{N_{SV}} \alpha_i \cdot y_i = 0 \]

\[ \alpha_i > 0 \quad \forall i \in [1, N_{SV}] \]

- The decision rule does not depend on the weights, but only on the support vectors and their multipliers.

- The complexity of the classifier is therefore only dependent on the number of support vectors, i.e. the complexity of the data.

- The training examples which are not support vectors are useless after learning: they can be suppressed without changing the result.
Maximal-margin classifier: the linear support-vector machine

\[ F(x) = \sum_{i=1}^{N_{SV}} \alpha_i \cdot y_i \cdot \langle x_i, x \rangle + b^* \]

\[ \sum_{i=1}^{N_{SV}} \alpha_i \cdot y_i = 0 \]

\[ \alpha_i > 0 \quad \forall i \in [1, N_{SV}] \]

- Generalization error with probability \((1-\delta)\):

\[ \epsilon(F') \leq \frac{1}{N - N_{SV}} \cdot (N_{SV} \cdot (1 - \log \frac{N_{SV}}{N}) + \log \frac{N}{\delta}) \]

- Expected generalization error:

\[ \epsilon = \frac{N_{SV}}{N} \]

- Occam's Razor: the fewer the number of support vectors (hence the simpler the model), the better is the generalization.
Maximal-margin classifier: the linear support-vector machine

\[ F(x) = \sum_{i=1}^{N_{SV}} \alpha_i \cdot y_i \cdot \langle x_i \cdot x \rangle + b^* \]

\[ \sum_{i=1}^{N_{SV}} \alpha_i \cdot y_i = 0 \]

\[ \alpha_i > 0 \quad \forall i \in [1, N_{SV}] \]

- Bound on the VC dimension:

\[ VC_{\text{dim}}(\text{SVM}) = \min \left( \left\lceil \frac{\max_{(i,j)\in[1,N_{SV}]}^2 \|x_i - x_j\|^2}{2 \cdot \gamma} \right\rceil, d \right) + 1 \]

- By normalizing the input and controlling the geometric margin \( \gamma = \frac{1}{\sqrt{\sum_{i=1}^{N} \alpha_i}} \), one can obtain a classifier of smaller VC dimension that the equivalent linear classifier.

→ important result when the input is transformed into a feature space of high dimension.
Soft-margin classifier
Soft-margin classifier
Soft-margin classifiers
The soft-margin classifier adds some flexibility to the optimization algorithm by allowing some points to violate the functional margin condition.

\[ y_i \cdot (\langle w \cdot x_i \rangle + b) \geq 1 - \xi_i \quad \forall i \in [1, N] \]

The parameter \( \xi_i \geq 0 \) is called the *slack variable* of the example: it tells how much the example violates the classifier.

\( \xi_i = 0 \): no problem; \( 0 < \xi_i < 1 \): good classification but inside the margin; \( \xi_i > 1 \): misclassification.

The limit on the functional margin is hard:

\[ y_i \cdot (\langle w \cdot x_i \rangle + b) \geq 1 \]

This limit is violated when the data is noisy (the classification is still correct, but the geometric margin is small) or when there exist outliers (the classification can not be correct as the data is not linearly separable).
The goal of a soft-margin classifier is to maximize the geometric margin (or minimize the norm of the weight vector) while minimizing the slack variables. This trade-off between two conflicting objectives is called regularization.

Primal form of the 1-norm soft-margin optimization

\[
\begin{align*}
\text{minimize}_{w,b} & \quad \frac{1}{2} \langle w, w \rangle + C \sum_{i=1}^{N} \xi_i \\
\text{s.t.} & \quad (1 - \xi_i) - y_i \cdot (\langle w, x_i \rangle + b) \leq 0 \quad \forall i \in [1, N] \\
& \text{s.t.} \quad \xi_i \geq 0 \quad \forall i \in [1, N]
\end{align*}
\]

The user-defined parameter \( C \) controls the trade-off between the complexity of the machine (its number of support vectors) and the number on non-separable examples.
Soft-margin classifier

- The goal of a soft-margin classifier is to maximize the geometric margin (or minimize the norm of the weight vector) while minimizing the slack variables.
- This trade-off between two conflicting objectives is called *regularization*.

Dual form of the 1-norm soft-margin optimization

\[
\begin{align*}
\text{maximize}_{\alpha_1, \ldots, \alpha_N} & \quad Q(\alpha_1, \ldots, \alpha_N) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \cdot \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \cdot \alpha_j \cdot y_i \cdot y_j \cdot \langle x_i, x_j \rangle \\
\text{s.t.} & \quad \sum_{i=1}^{N} \alpha_i \cdot y_i = 0 \\
\text{s.t.} & \quad 0 \leq \alpha_i \leq C \quad \forall i \in [1, N]
\end{align*}
\]
1-norm soft-margin classifier

Dual form of the 1-norm soft-margin optimization

$$Q(\alpha_1, \ldots, \alpha_N) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \cdot \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \cdot \alpha_j \cdot y_i \cdot y_j \cdot \langle x_i \cdot x_j \rangle$$

s.t. \ \sum_{i=1}^{N} \alpha_i \cdot y_i = 0

s.t. \ 0 \leq \alpha_i \leq C \ \forall i \in [1, N]

- This is exactly the same optimization problem as the maximal-margin classifier, except the KKT multipliers $\alpha_i$ are bounded by C.
- The examples where $\alpha_i = C$ lie within the margin of separation, while the examples where $0 < \alpha_i < C$ are on the maximal margin.
- The support vectors are characterized by $0 < \alpha_i < C$, and the decision function is computed in the same way as before.
- This regularization procedure reduces the number of support vectors (by tolerating some errors), hence decreasing the complexity of the model and increasing the geometric margin.
Working in a feature space: the kernel trick
Use of a feature space

- The methods seen before only work for (almost) linearly separable problems.

- For non-linear problems, as with RBF networks, one can project the input data into a higher dimensional feature space through a given function $\varphi$, and hope that the problem becomes there linearly separable.

$$F(x) = \sum_{i=1}^{N_{SV}} \alpha_i \cdot y_i \cdot \langle \varphi(x_i), \varphi(x) \rangle + b^*$$

$$= \sum_{i=1}^{N_{SV}} \alpha_i \cdot y_i \cdot K(x_i, x) + b^*$$

- The function $K(x, z) = \langle \varphi(x), \varphi(z) \rangle$ is called a kernel function.

- A soft-margin classifier equipped with a good kernel is called a support vector machine.

- The question is to choose the correct kernel that will allow a SVM to learn correctly.
Example of the polynomial kernel

Let's consider the quadratic kernel in $\mathbb{R}^3$:

$$\forall (x, z) \in \mathbb{R}^3 \times \mathbb{R}^3$$

$$K(x, z) = (\langle x, z \rangle)^2$$

$$= (\sum_{i=1}^{3} x_i \cdot z_i) \cdot (\sum_{j=1}^{3} x_j \cdot z_j)$$

$$= \sum_{i=1}^{3} \sum_{j=1}^{3} (x_i \cdot x_j) \cdot (z_i \cdot z_j)$$

$$= \langle \varphi(x), \varphi(z) \rangle$$

with:

$$\varphi(x) = \begin{bmatrix} x_1 \cdot x_1 \\ x_1 \cdot x_2 \\ x_1 \cdot x_3 \\ x_2 \cdot x_1 \\ x_2 \cdot x_2 \\ x_2 \cdot x_3 \\ x_3 \cdot x_1 \\ x_3 \cdot x_2 \\ x_3 \cdot x_3 \end{bmatrix}$$

The quadratic kernel implicitly transforms an input space with three dimensions into a feature space of 9 dimensions.
**Example of the polynomial kernel**

- More generally, the polynomial kernel in $\mathbb{R}^d$ of degree $p$:
  $$\forall (x, z) \in \mathbb{R}^d \times \mathbb{R}^d \quad K(x, z) = (\langle x, z \rangle)^p = \langle \varphi(x), \varphi(z) \rangle$$

transforms the input from a space with $d$ dimensions into a feature space of $d^p$ dimensions.

- While the inner product in the feature space would require $O(d^p)$ operations, the calculation of the kernel directly in the input space only requires $O(d)$ operations.

- This is called the **kernel trick**: when a linear algorithm only relies on the inner product between input vectors, it can be safely projected into a higher dimensional feature space through a kernel function, without increasing too much its computational complexity, and without ever computing the values in the feature space.

- For example, the dual version of the perceptron algorithm can be used in the feature space, creating the *kernel perceptron* algorithm.

- RBF networks can be seen as kernel algorithms if the centers of the hidden neurons are the support vectors.
The support vector machine

For a valid kernel function $K$ and a training set $S = \{(x_1, y_1), \ldots, (x_N, y_N)\}$, the KKT multipliers that solve the 1-norm optimization problem:

$$
\text{maximize}_{\alpha_1, \ldots, \alpha_N} \quad Q(\alpha_1, \ldots, \alpha_N) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \cdot \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \cdot \alpha_j \cdot y_i \cdot y_j \cdot K(x_i, x_j)
$$

s.t. $\sum_{i=1}^{N} \alpha_i \cdot y_i = 0$

s.t. $0 \leq \alpha_i \leq C \quad \forall i \in [1, N]$

realize a decision function that can correctly classify the training set

$$
F(x) = \sum_{i=1}^{N_{sv}} \alpha_i \cdot y_i \cdot K(x_i, x) + b^*
$$
What is a valid kernel?

Mercer’s theorem (1909)

Let $K : \mathcal{R}^d \times \mathcal{R}^d \rightarrow \mathcal{R}$ be a function and $\{x_1, \ldots, x_N\}$ a set of points in $\mathcal{R}^d$. $K$ is a valid kernel if the Kernel matrix $\mathcal{K}$ defined by:

$$
\mathcal{K} = \{K(x_i, x_j)_{(i,j)\in[1,N]^2}\}
$$

is symmetric positive semi-definite.

$$
\forall x \in \mathcal{R}^d, \quad x^T \cdot \mathcal{K} \cdot x \leq 0
$$

- The validity of a kernel depends on the feature mapping itself, but also on the data set. The kernel matrix can be tested on the training data before learning in order to know if the SVM will converge or not.

- Any valid kernel can be used in a SVM to classify the training data. Only the generalization error will depend on the choice of the Kernel function.
Examples of kernels used in SVMs

- Polynomial kernel: dimension of the feature space = $d^p$.
  \[ K(x, z) = (\langle x \cdot z \rangle)^p \]

- Gaussian kernel: dimension of the feature space = $\infty$.
  \[ K(x, z) = \exp\left(-\frac{\|x - z\|^2}{2\sigma^2}\right) \]

- Hyperbolic tangent kernel: dimension of the feature space = $\infty$
  \[ k(x, z) = \tanh(\langle \kappa x \cdot z \rangle + c) \]

- In practice, the choice of the kernel family depends more on the nature of data (text, image...) than on the complexity of the learning problem.
Why SVMs are a good idea

- Gaussian kernel: dimension of the feature space = $+\infty \rightarrow$ small training error.

- VC dimension:

\[
\text{VC\_dim}(\text{SVM}) = \min \left( \frac{\max_{(i,j)\in[1, N_{SV}]} \|x_i - x_j\|^2}{2 \cdot \gamma}, d \right) + 1
\]

\[
= \left[ \frac{\max_{(i,j)\in[1, N_{SV}]} \|x_i - x_j\|^2}{2 \cdot \gamma} \right] + 1
\]

with $\gamma = \frac{1}{\sqrt{\sum_{i=1}^{N} \alpha_i}}$ being the geometric margin.

- The soft-margin parameter $C$ can control the number and strength of support vectors: $\gamma$ can be increased by tolerating some errors.

- SVMs can work in a feature space with infinite dimensions, while keeping the VC dimension finite!

- Both training error and generalization errors can be kept small, depending on the value of $C$. 
Learning procedures in a SVM
Optimization procedure

\[
\begin{align*}
\text{maximize}_{\alpha_1, \ldots, \alpha_N} \quad & Q(\alpha_1, \ldots, \alpha_N) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \cdot \alpha_j \cdot y_i \cdot y_j \cdot K(x_i, x_j) \\
\text{s.t.} \quad & \sum_{i=1}^{N} \alpha_i \cdot y_i = 0 \\
\text{s.t.} \quad & 0 \leq \alpha_i \leq C \quad \forall i \in [1, N]
\end{align*}
\]

The functional \( Q(\alpha) \) is convex, so learning should always converge efficiently if the kernel is valid.
Naive stochastic gradient ascent

\[
Q(\alpha_1, \ldots, \alpha_N) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \cdot \alpha_j \cdot y_i \cdot y_j \cdot K(x_i, x_j)
\]

\[
\Delta \alpha = \eta \cdot \frac{\partial Q(\alpha)}{\partial \alpha}
\]

\[
\alpha \leftarrow 0^N
\]

while not converged:

for all examples \((x_i, y_i)\)

\[
\alpha_i \leftarrow \alpha_i + \eta_i \cdot (1 - y_i \cdot \sum_{j=1}^{N} \alpha_j \cdot y_j \cdot K(x_i, x_j))
\]

if \(\alpha_i < 0\) : \(\alpha_i \leftarrow 0\)

else if \(\alpha_i > C\) : \(\alpha_i \leftarrow C\)
Naive stochastic gradient ascent

- This algorithm does not care about the constraint:

\[
\sum_{i=1}^{N} \alpha_i \cdot y_i = 0
\]

as it would be impossible to modify only one \( \alpha_i \) at a time. The value of \( b^* \) is therefore unknown, and must be found by another algorithm.

- The stopping criterion can be computed by checking all the KKT conditions. When they are all achieved within a certain tolerance criterion, the algorithm stops.

- The drawback of this algorithm is that it needs to store the Kernel Matrix in memory: can become a problem when the number of examples increases.

- A good choice for \( \eta_i \) is:

\[
\eta_i = \frac{1}{K(x_i, x_i)}
\]

- It is better to sort the training examples during each epoch in the descending order of their \( \alpha_i \).
Chunking algorithm for the 1-norm margin

The idea is to leave out as soon as possible the examples which will never acquire a positive $\alpha$, in order to save memory space.

$\alpha \leftarrow 0^N$

Select a working subset $\hat{S}$ of $S$

**while** not converged :

- Solve optimization problem on $\hat{S}$
- Leave out all the non-support vectors from $\hat{S}$
- Test the found classifier on the rest of $S$
- Add the $M$ points which violates the most the KKT conditions to $\hat{S}$

This heuristic works better in practice than the online version and needs less memory.
Sequential Minimal Optimization (SMO)

Stochastic gradient ascent can not consider the KKT constraint:

\[
\sum_{i=1}^{N} \alpha_i \cdot y_i = 0
\]

because it updates the \( \alpha_i \) values one after another. Each \( \alpha_i \) is constrained by the other values:

\[
\alpha_j = -y_j \cdot \sum_{i \neq j} \alpha_i \cdot y_i
\]

The idea of SMO is to select two examples from the dataset using an adequate heuristic (e.g. the two examples that would modify the most the functional \( Q(\alpha) \)) and update the \( \alpha \) values only for these two examples, the others being fixed.

\[
y_1 \cdot \alpha_1 + y_2 \cdot \alpha_2 = -\sum_{i \geq 3} \alpha_i \cdot y_i = \xi
\]

\[
\alpha_2 = y_2 \cdot (\xi - y_1 \cdot \alpha_1)
\]
Sequential Minimal Optimization (SMO)

\[
\text{maximize}_{\alpha_1, \alpha_2} \quad Q(\alpha_1, \alpha_2) = a \cdot \alpha_1^2 + b \cdot \alpha_2^2 + c \cdot \alpha_1 \cdot \alpha_2 + d \cdot \alpha_1 + e \cdot \alpha_2 + f \\
\text{s.t.} \quad y_1 \cdot \alpha_1 + y_2 \cdot \alpha_2 = \xi \\
\text{s.t.} \quad 0 \leq \alpha_1 \leq C \\
\text{s.t.} \quad 0 \leq \alpha_2 \leq C
\]
Sequential Minimal Optimization (SMO)

\[ \alpha \leftarrow 0^N \]

while not converged:

Select two examples \((\alpha_1, \alpha_2)\) according to a heuristic.

Optimize the functional \(Q(\alpha_1, \alpha_2)\) only regarding \((\alpha_1, \alpha_2)\), the other \(\alpha_i\) being fixed while ensuring that:

\[ y_1 \cdot \alpha_1 + y_2 \cdot \alpha_2 = \xi \]

\[ 0 \leq \alpha_1 \leq C \]

\[ 0 \leq \alpha_2 \leq C \]
Comparison of SVM with other algorithms

- SVM always find a global and efficient maxima, unlike neural networks (NN).
- The solution found by a SVM is sparse (small number of support vectors).
- The complexity of a SVM does not depend on the dimension of the input space: typical choice for high-dimensional data.
- SVM are less prone to overfitting than NN.
- There exists a version of SVM for regression, which is slightly different.
- The mathematical foundation of SVM is strong, but rather complicated. However, already-implemented algorithms with various optimizations are freely available. Don't code your own SVM! Example: SVMlight, libsvm...