Human-Oriented Robotics

Supervised Learning

Part 2/3

Kai Arras
Social Robotics Lab, University of Freiburg
Non-Probabilistic Discriminant Functions

- So far, we have considered **probabilistic classifiers** that compute a posterior probability distribution $p(w|x)$ over the world state, for example, a discrete distribution over different class labels.

- We can also learn the **discriminant function** $y = f(x)$ **directly** (even more “directly” than a probabilistic discriminant classifier). For instance, in a two-class problem, $f(.)$ might be binary-valued such that $f(x) = 0$ represents class $C_1$ and $f(x) = 1$ represents class $C_2$.

- Inference and decision stages are **combined**.

- Choosing a model for $f(.)$ and using training data to learn $y = f(x)$ corresponds to **learning the decision boundary directly**.

- This is unlike probabilistic classifiers where the decision boundary **followed indirectly** from our choices for the involved models.
Non-Probabilistic Discriminant Functions

• Let us consider **linear discriminant functions** \( y = f(x) \). This choice implies the assumption that our data are **linearly separable**

• Let us again consider a **binary classification** problem, \( y \in \{-1, +1\} \)

• The representation of a linear function is

\[
y = f(x) = w^T x + b
\]

where \( w \) is the normal to the hyperplane (sometimes called weight vector) and \( b \) is called bias

• The hyperplane itself is described by \( w^T x + b = 0 \)

• The perpendicular distance from the plane to the origin is \( \frac{b}{\|w\|} \)

• (Notice the change in notation: in this section, we adopt the standard notion \( w \) to denote the **normal** to the hyperplane, not the world state)
Non-Probabilistic Discriminant Functions

- Figure shows geometry of \( f(x) = w^T x + b \) in 2 dimensions
- Consider two points \( x_A, x_B \) that both lie on the plane
  \[
  f(x_A) = w^T x_A + b = 0 \\
  f(x_B) = w^T x_B + b = 0 \\
  w^T x_A + b = w^T x_B + b \\
  w^T (x_A - x_B) = 0
  \]
- Thus, vector \( w \) is orthogonal to every vector lying within the hyperplane, and so \( w \) determines the orientation of the plane
Non-Probabilistic Discriminant Functions

- Consider a point $x$ and its orthogonal projection onto the plane $x_\perp$. Then

$$x = x_\perp + r \frac{w}{\|w\|}$$

- Let us solve for $r$, the signed perpendicular distance from $x$ to the plane. Multiplying both sides by $w^T$ and adding $b$

$$w^T x + b = w^T x_\perp + w^T r \frac{w}{\|w\|} + b = w^T x_\perp + b + r \frac{w^T w}{\|w\|} = r \frac{w^T w}{\|w\|}$$

$$f(x) = r \frac{\|w\|^2}{\|w\|} = r \|w\| \quad \Leftrightarrow \quad r = \frac{f(x)}{\|w\|}$$

- Note that distance $r$ is **signed**

- For $x = 0$, the perpendicular distance from the plane to the origin is $\frac{b}{\|w\|}$
Non-Probabilistic Discriminant Functions

- This can also be seen from the definition of the dot product

\[ \mathbf{w}^T \mathbf{x} = \| \mathbf{w} \| \| \mathbf{x} \| \cos \theta = \| \mathbf{w} \| \mathbf{x}_\mathbf{w} \]

\[ f(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b = \| \mathbf{w} \| \mathbf{x}_\mathbf{w} + b \]

\[ r = \frac{f(\mathbf{x})}{\| \mathbf{w} \|} = \mathbf{x}_\mathbf{w} + \frac{b}{\| \mathbf{w} \|} \]

\[ r \begin{cases} > 0 & \text{if } \mathbf{x}_\mathbf{w} > -\frac{b}{\| \mathbf{w} \|} \\ = 0 & \text{if } \mathbf{x}_\mathbf{w} = -\frac{b}{\| \mathbf{w} \|} \\ < 0 & \text{if } \mathbf{x}_\mathbf{w} < -\frac{b}{\| \mathbf{w} \|} \end{cases} \]
Supervised Learning

Non-Probabilistic Discriminant Functions

- Consider a linearly separable classification problem with two classes and outputs $y \in \{-1, +1\}$

$y_1 = +1$
$y_2 = -1$
$y_3 = +1$
$\vdots$
$y_N = -1$

- How to separate the classes?
Non-Probabilistic Discriminant Functions

- Consider a linearly separable classification problem with two classes and outputs $y \in \{-1, +1\}$

- There is an infinite number of decision boundaries that perfectly separate the classes in the training set

- Which one to choose?
Non-Probabilistic Discriminant Functions

- The one with the **smallest generalization error!**

- This is what **Support Vector Machines** (SVM) do. The approach to minimize the generalization error is to **maximize the margin**
Margin and Support Vectors

- The margin is defined as the **perpendicular distance between the decision boundary and the closest data points**

- The closest data points are called **support vectors**

- The aim of Support Vector Machines is to orientate a hyperplane in such a way as to be as far as possible from the support vectors of **both classes**
Margin and Support Vectors

- This amounts to the estimation of the normal vector \( \mathbf{w} \) and the bias \( b \).
- We have seen that \( \mathbf{w} \) determines the **orientation** of the hyperplane and the ratio \( \frac{b}{||\mathbf{w}||} \) its **position from the origin**.
- Thus, in addition to the direction of \( \mathbf{w} \) and the value for \( b \), there is one more degree of freedom, namely the **magnitude** of the normal vector \( ||\mathbf{w}|| \).
- We can thus define \( ||\mathbf{w}|| \) in a way that, without loss of generality, for support vectors \( |f(\mathbf{x})| = |y| = 1 \) holds.
Margin and Support Vectors

- We then define two planes $H_1, H_2$ through the support vectors. They are described by

$$H_1 : \mathbf{w}^T \mathbf{x} + b = +1$$
$$H_2 : \mathbf{w}^T \mathbf{x} + b = -1$$

- Our training data $(\mathbf{x}_i, y_i)$ for all $i$ can thus be described by

$$\mathbf{w}^T \mathbf{x}_i + b \geq +1 \quad \text{for } y_i = +1$$
$$\mathbf{w}^T \mathbf{x}_i + b \leq -1 \quad \text{for } y_i = -1$$

which can be combined to

$$y_i (\mathbf{w}^T \mathbf{x}_i + b) - 1 \geq 0$$
Margin and Support Vectors

- Let us look at this expression

\[ y_i (\mathbf{w}^T \mathbf{x}_i + b) - 1 \geq 0 \]

- It is a set of \( N \) constraints on \( \mathbf{w} \) and \( b \) to be satisfied during the learning phase.

- However, the constraints alone do not maximize the margin.

- From our choice of \( \|\mathbf{w}\| \) it follows that the margin is

\[ r = \frac{f(\mathbf{x})}{\|\mathbf{w}\|} = \frac{1}{\|\mathbf{w}\|} \]

- Thus, maximizing the margin is equivalent to minimizing \( \|\mathbf{w}\| \)
Learning

- **SVM learning** consists in minimizing $\|w\|$ subject to the constraints

  $$y_i (w^T x_i + b) - 1 \geq 0$$

- Instead of minimizing $\|w\|$ we can also minimize $\frac{1}{2} \|w\|^2$ which leads to the formulation

  $$\arg\min_{w,b} \frac{1}{2} \|w\|^2 \quad \text{s.t.} \quad y_i (w^T x_i + b) - 1 \geq 0$$

- This is a **quadratic programming** problem in which we are trying to minimize a quadratic function subject to a set of linear inequality constraints

- In order to solve this constrained optimization problem, we will need to introduce **Lagrange multipliers**
Lagrange Multipliers

- The **method of Lagrange multipliers** is a strategy for finding the local maxima and minima of a function subject to equality constraints.
- Consider, for instance, the constraint optimization problem

\[
\text{maximize } f(x, y) \\
\text{subject to } g(x, y) = c
\]

- Let us visualize contours of \( f \) given by

\[
f(x, y) = d
\]

for various values of \( f \) and the contour of \( g \) given by \( g(x, y) = c \)
Lagrange Multipliers

- Following the contour lines of $g = c$, we want to find the point on it with the largest value of $f$. Then, $f$ will be stationary as we move along $g = c$.

- In general, contour lines of $g = c$ will cross/intersect the contour lines of $f$. This is equivalent to saying that the value of $f$ varies while moving along $g = c$.

- Only when the line for $g = c$ meets contour lines of $f$ tangentially, that is, the lines touch but do not cross, the value of $f$ does not change along $g = c$. 

Source [6]
Lagrange Multipliers

- Contour lines touch when their tangent vectors are parallel. This is the same as saying that the gradients are parallel, because the gradient is always perpendicular to the contour.
- This can be formally expressed as

\[ \nabla_{x,y} f = -\lambda \nabla_{x,y} g \]

with

\[
\nabla_{x,y} f = \left( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right) \quad \nabla_{x,y} g = \left( \frac{\partial g}{\partial x}, \frac{\partial g}{\partial y} \right)
\]

- In general

\[ \nabla_x f(x) = -\lambda \nabla_x g(x) \]
Lagrange Multipliers

- The constant $\lambda$ is required because magnitudes and directions of the gradient vectors are generally not equal.
- Rearranging $\nabla_{x,y} f = -\lambda \nabla_{x,y} g$ gives
  $$\nabla_{x,y} f + \lambda \nabla_{x,y} g = 0$$
- If we were to define the function
  $$L(x, y, \lambda) = f(x, y) + \lambda \cdot (g(x, y) - c)$$
  we could write the above condition compactly as
  $$\nabla_{x,y,\lambda} L(x, y, \lambda) = 0$$
- This is the **method of Lagrange multipliers**
Lagrange Multipliers

- The constant $\lambda$ is required because magnitudes and directions of the gradient vectors are generally not equal.
- Rearranging $\nabla_{x,y} f = -\lambda \nabla_{x,y} g$ gives
  $$\nabla_{x,y} f + \lambda \nabla_{x,y} g = 0$$
- If we were to define the function
  $$L(x, y, \lambda) = f(x, y) + \lambda \cdot (g(x, y) - c)$$
  we could write the above condition compactly as
  $$\nabla_{x,y,\lambda} L(x, y, \lambda) = 0$$
- This is the **method of Lagrange multipliers**
Lagrange Multipliers

- The partial derivatives w.r.t. $x$, $y$ recover the parallel-gradient equation, while the partial derivative w.r.t. $\lambda$ recovers the constraint.

- Solving the Lagrange function for its **unconstrained stationary points** generates exactly the same stationary points as solving for the stationary points of $f$ under the constraint $g$.

- We are looking for **stationary points** of the Lagrange function.
  - Recall, **stationary points** are points of a differentiable function where the derivative is zero (i.e. where the function stops increasing or decreasing, hence the name).

- However, **not all stationary points** yield a solution of the original optimization problem.

- Thus, the method of Lagrange multipliers yields only **necessary conditions** for optimality and we have to **evaluate** $f$ at the stationary points to find our solution.
Lagrange Multipliers

- Let us make an example:

  $\text{maximize } f(x, y) = x^2 y$
  $\text{subject to } g(x, y) = x^2 + y^2 = 3$

  i.e. maximize $f$ with the constraint that the $x$ and $y$ coordinates lie on the circle around the origin with radius $\sqrt{3}$

- The Lagrangian is

  $$L(x, y, \lambda) = f(x, y) + \lambda \cdot (g(x, y) - c) = x^2 y + \lambda \cdot (x^2 + y^2 - 3)$$

- Let us partially derivate $L$ with respect to $x, y$ and $\lambda$

- Note that, as mentioned above, $\nabla_{\lambda} L(x, y, \lambda) = 0$ gives the original constraint $g(x, y) = c$
Lagrange Multipliers

- The partial derivatives are

\[
\frac{\partial L}{\partial x} = 2xy + 2\lambda x = 0 \quad (1)
\]

\[
\frac{\partial L}{\partial y} = x^2 + 2\lambda y = 0 \quad (2)
\]

\[
\frac{\partial L}{\partial \lambda} = x^2 + y^2 - 3 = 0 \quad (3)
\]

- Eq. (1) implies either \( x = 0 \) or \( \lambda = -y \). In the former case, it follows by eq. (3) that \( y = \pm \sqrt{3} \) and \( \lambda = 0 \)

- In the case \( \lambda = -y \), it follows \( x^2 = 2y^2 \) by eq. (2). Substitution into (3) yields \( y = \pm 1 \) and \( x = \pm \sqrt{2} \)

- Thus, there are **six stationary points** of the Lagrangian

\[(\sqrt{2}, 1), \quad (-\sqrt{2}, 1), \quad (\sqrt{2}, -1), \quad (-\sqrt{2}, -1), \quad (0, \sqrt{3}), \quad (0, -\sqrt{3})\]
Support Vector Machines

Lagrange Multipliers

- Evaluation of $f$ at the **stationary points**

$(\sqrt{2}, 1), \ (-\sqrt{2}, 1), \ (\sqrt{2}, -1), \ (-\sqrt{2}, -1), \ (0, \sqrt{3}), \ (0, -\sqrt{3})$ yields

\[
\begin{align*}
    f(\pm \sqrt{2}, 1) & = 2, \\
    f(\pm \sqrt{2}, -1) & = -2, \\
    f(0, \pm \sqrt{3}) & = 0
\end{align*}
\]

- Therefore, the objective function attains the **global maximum**, subject to the constraint, at $(\pm \sqrt{2}, 1)$
Lagrange Multipliers

- In the case of multiple constraints, the same reasoning applies.
- Let us recap: in the presence of a constraint, $\nabla_x f(x)$ does not have to be zero at $x$, but it has to be entirely contained in the (1-dimensional) subspace spanned by $\nabla_x g(x)$.

\[
\nabla_x f(x) = -\lambda \nabla_x g(x)
\]

- This generalizes to multiple constraints: for $N$ constraints $g_i(x) = 0$ we have

\[
\nabla_x f(x) = -\sum_{i=1}^{N} \lambda_i \nabla_x g_i(x)
\]

- The subspace is now a linear combination of the gradients $\nabla_x g_i(x)$ with weights $\lambda_i$. 

Source [6]
Lagrange Multipliers

- Thus, the Lagrangian for multiple constraints is

\[
L(x, \lambda) = f(x) + \sum_{i=1}^{N} \lambda_i g_i(x)
\]

where \( \lambda = \{\lambda_1, \lambda_2, \ldots, \lambda_N\} \)

- Again, we partially derivate the Lagrangian

\[
\nabla_{x,\lambda} L(x, \lambda) = 0
\]

and solve for its stationary points, evaluate \( f \) at those points

- Again, the partial derivatives w.r.t. \( x \) recover the parallel-gradient equation, while the partial derivative w.r.t. \( \lambda \) recovers the constraint
Karush–Kuhn–Tucker Conditions

- Now, assume we also have **inequality constraints**
- The constraint optimization problem is then

  maximize \( f(x) \)  
  subject to \( g_i(x) = 0 \) for \( i \in \{1, \ldots, N\} \)  
  and to \( h_i(x) \leq 0 \) for \( i \in \{1, \ldots, M\} \)

- The problem can be solved via the **general Lagrangian**

\[
L(x, \lambda, \mu) = f(x) + \sum_{i=1}^{N} \lambda_i g_i(x) + \sum_{i=1}^{M} \mu_i h_i(x)
\]

- The **stationary points** of the general Lagrangian are again the same than the constraint stationary points of \( f \)
Karush–Kuhn–Tucker Conditions

• However, inequality constraints are different than equality constraints and our previously made considerations are not sufficient anymore

• We require a set of additional conditions (or constraints) to guarantee optimality of solutions

• The combined set of constraints is called Karush–Kuhn–Tucker (KKT) conditions

• Allowing inequality constraints, the KKT approach generalizes the method of Lagrange multipliers, which allows only equality constraints

• We will not go deeper at this point, but will return to SVM learning
Support Vector Machines

Learning

- In the case of SVM learning, we have

$$\text{arg min}_{w,b} \frac{1}{2} \|w\|^2 \quad \text{s.t.} \quad y_i (w^T x_i + b) - 1 \geq 0$$

with a set of $N$ inequality constraints

- Thus, the Karush–Kuhn–Tucker (KKT) conditions apply

- We allocate Lagrange multipliers $\lambda = \{\lambda_1, \lambda_2, \ldots, \lambda_N\}$

$$L(w, b, \lambda) = \frac{1}{2} \|w\|^2 - \sum_{i=1}^{N} \lambda_i (y_i (w^T x_i + b) - 1)$$

where now $\lambda_i \geq 0 \ \forall i$ (which is one of the KKT conditions)

- The minus sign comes from the KKT problem statement $h_i(x) \leq 0$

- $L$ is minimized if we minimize it w.r.t. $w, b$ and maximize it w.r.t. $\lambda$
Learning

- Note that the Lagrangian is a function of \( w, b \) (and this is the general "x" from the Lagrange subsection)
- Derivation of \( L \) with respect to \( w, b \) gives

\[
\frac{\partial L}{\partial w} = 0 \iff w = \sum_{i=1}^{N} \lambda_i y_i x_i \quad \frac{\partial L}{\partial b} = 0 \iff \sum_{i=1}^{N} \lambda_i y_i = 0
\]

- Instead of solving for the stationary points of \( L \) directly, let us substitute these expressions back into the Lagrange function (eliminating \( w, b \))

\[
L(w, b, \lambda) = \frac{1}{2} \|w\|^2 - \sum_{i=1}^{N} \lambda_i (y_i (w^T x_i + b) - 1)
\]

\[
= \frac{1}{2} \|w\|^2 - \sum_{i=1}^{N} \lambda_i y_i (w^T x_i + b) + \sum_{i=1}^{N} \lambda_i
\]
Support Vector Machines

Learning

- Working the new expression for the normal...

\[ w^T = \sum_{i=1}^{N} \lambda_i y_i x_i^T \]

\[ \|w\|^2 = w^T w = \left( \sum_{i=1}^{N} \lambda_i y_i x_i^T \right) \left( \sum_{j=1}^{N} \lambda_j y_j x_j \right) = \sum_{i,j} \lambda_i \lambda_j y_i y_j x_i^T x_j \]

- Substitution into

\[ L(w, b, \lambda) = \frac{1}{2} \|w\|^2 - \sum_{i=1}^{N} \lambda_i y_i (w^T x_i + b) + \sum_{i=1}^{N} \lambda_i \]

yields

\[ L(w, b, \lambda) = -\frac{1}{2} \sum_{i,j=1}^{N} \lambda_i \lambda_j y_i y_j x_i^T x_j + \sum_{i=1}^{N} \lambda_i - b \sum_{i=1}^{N} \lambda_i y_i \]
Learning

- This gives the **dual form** $L(\lambda)$ of the **primary** $L(w, b, \lambda)$

\[
L(\lambda) = \sum_{i=1}^{N} \lambda_i - \frac{1}{2} \sum_{i,j=1}^{N} \lambda_i \lambda_j y_i y_j x_i^T x_j
\]

- We came here by minimizing the original Lagrangian w.r.t. $w, b$. What remains to do is to maximize it w.r.t. $\lambda$. This leads to the following **dual optimization problem**

maximize $L(\lambda)$
subject to $\lambda_i \geq 0 \ \forall i$ and $\sum_i \lambda_i y_i = 0$

- We can solve the **dual** optimization problem in lieu of the **primal** problem

- Note that the dual form requires only the **inner product of each input vector** to be calculated. This will be important for the **kernel trick**
Learning

- The dual optimization problem takes the form of a **quadratic programming problem** in which we optimize a quadratic function of the $\lambda_i$'s subject to a set of inequality constraints.
- There are many **QP solvers** for this purpose (such as Matlab's `quadprog`).
- We then obtain the Lagrange multipliers $\lambda$ and can compute:

  $$
  w = \sum_{i=1}^{N} \lambda_i y_i x_i
  $$

- Substitution into the discriminative function model yields the **dual version** of the classifier:

  $$
  f(x) = \sum_{i=1}^{N} \lambda_i y_i x_i^T x + b
  $$

  Primal version

  $$
  f(x) = w^T x + b
  $$

  Dual version
Support Vector Machines

Learning

- For the computation of the normal or the dual version of the classifier, we do not need to sum over all \( N \) training pairs. It follows from the KKT conditions that only support vectors have non-zero \( \lambda_i \)'s.
- This is how we can find the support vectors among the training samples.
- This is noteworthy and the reason why SVM are also called sparse kernel machines. The learned classifier only depends sparsely on the training set.

- What remains to do is to calculate the bias \( b \).
- Remember our \( N \) inequality constraints

\[
y_i \left( w^T x_i + b \right) - 1 \geq 0
\]

- We have defined the normal in a way that for support vectors

\[
y_s \left( w^T x_s + b \right) = 1 \quad s \in S
\]

Set of indices of the support vectors
Learning

- Substituting the dual version of the classifier leads to
  \[ y_{si} \left( \sum_{s_j \in S} \lambda_{sj} y_{sj} \mathbf{x}_{sj}^T \mathbf{x}_{si} + b \right) = 1 \]

- Multiplication with the label on both sides gives
  \[ y_{si}^2 \left( \sum_{s_j \in S} \lambda_{sj} y_{sj} \mathbf{x}_{sj}^T \mathbf{x}_{si} + b \right) = y_{si} \]

- Using \( y_{si}^2 = 1 \) and solving for \( b \)
  \[ b = y_{si} - \sum_{s_j \in S} \lambda_{sj} y_{sj} \mathbf{x}_{sj}^T \mathbf{x}_{si} \]
  
  \[ b = \frac{1}{NS} \sum_{s_i \in S} \left( y_{si} - \sum_{s_j \in S} \lambda_{sj} y_{sj} \mathbf{x}_{sj}^T \mathbf{x}_{si} \right) \]

- Although we can solve this equation for \( b \) using an arbitrary support vector \( s_i \), it is numerically more stable to take an average over all support vectors
Support Vector Machines

Inference and Decision

- We now have the variables $\mathbf{w}$ and $b$ that define our separating hyperplane’s optimal orientation and hence our Support Vector Machine.

- For classification, each new input $\mathbf{x}'$ is predicted by

$$y' = \text{sign}(\mathbf{w}^T \mathbf{x'} + b)$$

- Note the resemblance of the dual version to the k-NN classifier

$$y' = \text{sign}(\sum_{i=1}^{N} \lambda_i y_i \mathbf{x}_i^T \mathbf{x'} + b)$$
Support Vector Machines

Soft-Margin SVM

- So far, we have assumed that the training data points are linearly separable in feature space. But often, the class-conditional distributions overlap in which case exact separation is not possible.

- We now modify the approach so that data points are allowed to be “on the wrong side” of the decision boundary.

- We introduce a penalty that increases with the distance from that boundary. The penalty is a linear function of this distance.

- To this end, we introduce a slack variable $\xi_i \geq 0 \ \forall i \in \{1, \ldots, N\}$ for each training sample ($\xi$ or “xi” is pronounced zī like “high”).

- They are defined to be zero for data points on or inside the “right side” of the boundary, and $\xi_i = |y_i - f(x_i)|$ for other points.
Soft-Margin SVM

- Let us visualize $\xi_i$
- The relationship $\xi_i = |y_i - f(x_i)|$ implies that points on the boundary have $\xi_i = 1$
- Misclassified points receive $\xi_i > 1$
- The set of $N$ constraints that describe our training data $(x_i, y_i)$ is now
  \[
  w^T x_i + b \geq +1 - \xi_i \quad \text{for } y_i = +1
  \]
  \[
  w^T x_i + b \leq -1 + \xi_i \quad \text{for } y_i = -1
  \]
- Points with $\xi_i > 0$ that violate the margin are called **non-margin support vectors**. They are also considered support vectors
Soft-Margin SVM

- They can be combined into
  \[ y_i (\mathbf{w}^T \mathbf{x}_i + b) - 1 + \xi_i \geq 0 \quad \xi_i \geq 0 \quad \forall i \]

- Notice the set of new constraints on the slack variables

- While before, in the non-overlapping case, the optimization objective was
  \[
  \arg \min_{\mathbf{w}, b} \frac{1}{2} \| \mathbf{w} \|^2 \quad \text{s.t.} \quad y_i (\mathbf{w}^T \mathbf{x}_i + b) - 1 \geq 0
  \]
  our goal is now to also reduce the number of misclassified data points

- This is done – in addition to the maximization of the margin – by softly penalizing data points on the wrong side of the decision boundary
  \[
  \arg \min_{\mathbf{w}, b} \frac{1}{2} \| \mathbf{w} \|^2 + C \sum_{i=1}^{N} \xi_i \quad \text{s.t.} \quad y_i (\mathbf{w}^T \mathbf{x}_i + b) - 1 + \xi_i \geq 0 \quad \forall i \\
  \xi_i \geq 0 \quad \forall i
  \]
Soft-Margin SVM

- Parameter $C > 0$ is called **stiffness parameter** and controls the trade-off between slack variable penalty and the size of the margin
- The method tries to splits the training data as cleanly as possible, while still maximizing the distance to the nearest cleanly split samples
- The corresponding Lagrangian is

$$L(w, b, \lambda, \mu) = \frac{1}{2} \|w\|^2 + C \sum_{i=1}^{N} \xi_i - \sum_{i=1}^{N} \lambda_i \left( y_i (w^T x_i + b) - 1 + \xi_i \right) - \sum_{i=1}^{N} \mu_i \xi_i$$

where $\lambda_i \geq 0$, $\mu_i \geq 0 \ \forall i$ (KKT conditions) are the Lagrange multipliers

- The corresponding **extended set** of KKT conditions collects all constraints
- We need to minimize $L$ w.r.t. $w$, $b$ and $\xi_i$ and maximize it w.r.t. $\lambda$ and $\mu$
- We proceed as before...
Soft-Margin SVM

- Differentiating w.r.t. $w$, $b$ and $\xi_i$ and setting the derivatives to zero

\[
\frac{\partial L}{\partial w} = 0 \iff w = \sum_{i=1}^{N} \lambda_i y_i x_i
\]

\[
\frac{\partial L}{\partial b} = 0 \iff \sum_{i=1}^{N} \lambda_i y_i = 0
\]

\[
\frac{\partial L}{\partial \xi_i} = 0 \iff C = \lambda_i + \mu_i
\]

- Substitution into the Lagrangian eliminates $w$, $b$ and $\xi_i$ from $L$ and we obtain the dual form – which is identical to the non-overlapping case

\[
L(\lambda) = \sum_{i=1}^{N} \lambda_i - \frac{1}{2} \sum_{i,j=1}^{N} \lambda_i \lambda_j y_i y_j x_i^T x_j
\]
Soft-Margin SVM

- However, the constraints are **different**. From $C = \lambda_i + \mu_i$ and $\mu_i \geq 0 \ \forall i$ follows $\lambda_i \leq C$

- The **dual optimization problem** is then:

  maximize $L(\lambda)$

  subject to $0 \leq \lambda_i \leq C \ \forall i$ and $\sum_i \lambda_i y_i = 0$

- Again, we can use standard **QP solvers** for this optimization task

- **Support vectors** are now found via the condition $0 < \lambda_i \leq C$

- What remains to do is to **calculate the bias** $b$. This is done in the same way as before using an average over all support vectors

- Class prediction (**inference** and **decision**) is then made by:

  $$y' = \text{sign}(w^T x' + b)$$
Soft-Margin SVM

- Increasing $C'$ places more weight on the slack variables $\xi_i$ leading to a stricter separation of the classes and a smaller margin. **Reducing** $C'$ leads to a larger margin and more misclassified points.
**Non-Linear SVM**

- So far, we looked at classification problems with linearly separable class distributions (up to some extent of overlapping)
- When data are not linearly separable, we have a **non-linear classification problem**
- How can we solve such problems using Support Vector Machines
- **Idea:** make the data linearly separable by mapping them into a higher dimensional space

\[ \mathbf{x} \rightarrow \phi(\mathbf{x}) \quad \mathbb{R}^m \rightarrow \mathbb{R}^d \]
Non-Linear SVM

- Consider the following mapping

$$\phi : \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \rightarrow \begin{pmatrix} x_1^2 \\ x_2^2 \\ \sqrt{2} x_1 x_2 \end{pmatrix} \quad \mathbb{R}^2 \rightarrow \mathbb{R}^3$$
Non-Linear SVM

- Consider the following mapping

\[
\phi : \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \rightarrow \begin{pmatrix} x_1^2 \\ x_2^2 \\ \sqrt{2} x_1 x_2 \end{pmatrix} \quad \mathbb{R}^2 \rightarrow \mathbb{R}^3
\]
Non-Linear SVM

- Consider the following mapping

$$\phi : \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \rightarrow \begin{pmatrix} x_1^2 \\ x_2^2 \\ \sqrt{2} x_1 x_2 \end{pmatrix} \quad \mathbb{R}^2 \rightarrow \mathbb{R}^3$$

Linearly separable!
Non-Linear SVM

- Data may be **linearly separable** in the high dimensional space although in the original feature space they are not linearly separable.

- This phenomenon is actually fairly general: if data are mapped into a space of **sufficiently high dimension**, then they will **almost always be linearly separable**.

- For example, **four** dimensions suffice for linearly separating a **circle** anywhere in the plane (not just at the origin), and **five** dimensions suffice to linearly separate any **ellipse**.

- In general (up to some exceptions), when we have $N$ data points then they will always be **separable in spaces of $N-1$ dimensions** or more.

- In order to frame the non-linear problem as a linear classification problem in the $\phi$-space, we go over our learning and inference algorithms and replace $x$ **everywhere by** $\phi(x)$:
Non-Linear SVM

- In our Lagrange function in dual form

\[
L(\lambda) = \sum_{i=1}^{N} \lambda_i - \frac{1}{2} \sum_{i,j=1}^{N} \lambda_i \lambda_j y_i y_j x_i^T x_j
\]

in the expression for the bias \( b \)

\[
b = \frac{1}{N_S} \sum_{s_i \in S} \left( y_{s_i} - \sum_{s_j \in S} \lambda_{s_j} y_{s_j} x_{s_j}^T x_{s_i} \right)
\]

and in the dual version of the classifier

\[
y' = \text{sign}\left( \sum_{i=1}^{N} \lambda_i y_i x_i^T x' + b \right)
\]
Non-Linear SVM

- In our **Lagrange function** in dual form

\[
L(\lambda) = \sum_{i=1}^{N} \lambda_i - \frac{1}{2} \sum_{i,j=1}^{N} \lambda_i \lambda_j y_i y_j \phi(x_i)^T \phi(x_j)
\]

in the expression for the **bias** \( b \)

\[
b = \frac{1}{NS} \sum_{s_i \in S} (y_{s_i} - \sum_{s_j \in S} \lambda_{s_j} y_{s_j} \phi(x_{s_j})^T \phi(x_{s_i}))
\]

and in the **dual version** of the **classifier**

\[
y' = \text{sign} \left( \sum_{i=1}^{N} \lambda_i y_i \phi(x_i)^T \phi(x') + b \right)
\]
Non-Linear SVM

- In our **Lagrange function** in dual form

\[ L(\lambda) = \sum_{i=1}^{N} \lambda_i - \frac{1}{2} \sum_{i,j=1}^{N} \lambda_i \lambda_j y_i y_j \phi(x_i)^T \phi(x_j) \]

in the expression for the **bias** \( b \)

\[ b = \frac{1}{N_S} \sum_{s_i \in S} (y_{s_i} - \sum_{s_j \in S} \lambda_{s_j} y_{s_j} \phi(x_{s_j})^T \phi(x_{s_i})) \]

and in the **dual version** of the **classifier**

\[ y' = \text{sign}(\sum_{i=1}^{N} \lambda_i y_i \phi(x_i)^T \phi(x') + b) \]

- Vectors \( x \) or \( \phi(x) \) enter only in the form of **inner products**!
Non-Linear SVM

- The fact that we can express our algorithm in terms of these inner products is key for the **kernel trick**

- A **kernel** is defined as

  \[ k(x_i, x_j) = \phi(x_i)^T \phi(x_j) \]

- Given \( \phi(x) \), we could easily compute \( k(x_i, x_j) \) by finding \( \phi(x_i)^T \) and \( \phi(x_j) \) and taking their inner product

- But dimension \( d \) may be extremely **large**. When the transformed space is high-dimensional, it may be **very costly** to compute the vectors \( \phi(x) \) explicitly and then compute the inner product

- Interestingly, computing \( k(x_i, x_j) \) may be **very inexpensive to calculate**, even though \( \phi(x) \) itself may be very expensive to calculate
Non-Linear SVM

• Thus, with an efficient way to calculate \( k(x_i, x_j) \), we can get SVMs to learn in the high dimensional feature space given by \( \phi \), but **without ever having to explicitly find or represent** vectors \( \phi(x) \)

• Let us exemplify this

\[
\phi: \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \rightarrow \begin{pmatrix} x_1^2 \\ x_2^2 \\ \sqrt{2} x_1 x_2 \end{pmatrix}
\]

\[
\phi(x)^T \phi(z) = \begin{pmatrix} x_1^2 & x_2^2 & \sqrt{2} x_1 x_2 \end{pmatrix} \begin{pmatrix} z_1^2 \\ z_2^2 \\ \sqrt{2} z_1 z_2 \end{pmatrix}
\]

\[
= x_1^2 z_1^2 + x_2^2 z_2^2 + 2 x_1 x_2 z_1 z_2
\]

\[
= (x_1 z_1 + x_2 z_2)^2
\]

\[
= (x^T z)^2
\]
Non-Linear SVM

- Thus, we could have used the kernel \( k(x, z) = (x^T z)^2 \) without explicitly computing \( \phi(x) \)

- Let us look at this more systematically with a feature mapping involving all monomials of the form \( x_i x_j \) (the previous one had visualization purposes). Assume again \( m = 2 \)

\[
\phi : \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \rightarrow \begin{pmatrix} x_1 x_1 \\ x_1 x_2 \\ x_2 x_1 \\ x_2 x_2 \end{pmatrix} \quad \mathbb{R}^2 \rightarrow \mathbb{R}^4
\]

- The **cost of computing** the high-dimensional \( \phi(x) \) is \( O(m^2) \)
Non-Linear SVM

• The inner product $\phi(x)^T \phi(z)$ leads to the same kernel

$$
\phi(x)^T \phi(z) = (x_1 x_1 \ x_1 x_2 \ x_2 x_1 \ x_2 x_2) \begin{pmatrix} z_1 z_1 \\ z_1 z_2 \\ z_2 z_1 \\ z_2 z_2 \end{pmatrix} \\
= x_1^2 z_1^2 + x_1 x_2 z_1 z_2 + x_2 x_1 z_2 z_1 + x_2^2 z_2^2 \\
= (x_1 z_1 + x_2 z_2)^2 \\
= (x^T z)^2
$$

• The cost of computing the kernel is only $O(m)$
Non-Linear SVM

- Let us convince ourselves that this kernel can be written as the inner product $\phi(x)^T \phi(z)$ for general input vector dimensions $m$

\[
k(x, z) = (x^T z)^2
\]
\[
= \left( \sum_{i=1}^{m} x_i z_i \right) \left( \sum_{j=1}^{m} x_j z_j \right)
\]
\[
= \sum_{i=1}^{m} \sum_{j=1}^{m} x_i x_j z_i z_j
\]
\[
= \sum_{i,j=1}^{m} (x_i x_j)(z_i z_j)
\]
- This is indeed $\phi(x)^T \phi(z)$ with $\phi(x)$ as defined above
Non-Linear SVM

- Consider now the kernel $k(x, z) = (x^T z + c)^2$
- It happens to contain an inner product of $x$ and $z$ but this is not a requirement. Kernels are **general functions** of $x$ and $z$ (or $x_i$ and $x_j$)
- It can be shown that

$$k(x, z) = (x^T z + c)^2$$

$$= \sum_{i,j=1}^{m} (x_i x_j)(z_i z_j) + \sum_{i=1}^{m} (\sqrt{2}c x_i)(\sqrt{2}c z_i) + c^2$$

and that this result corresponds to $\phi(x)^T \phi(z)$ with the feature mapping $\phi(x)$ shown on the right for $m = 3$

- Note the cost difference: $O(m)$ for kernel vs. $O(m^2)$ for $\phi(x)$
Non-Linear SVM

- Kernels do not transform the input data into the \( \phi \)-space and then take an inner product. Kernels are regular functions of \( x \)
- However, as shown in the examples, kernels correspond to a transformation to some \( \phi \)-space and taking an inner product there without ever explicitly computing feature vectors in this high-dimensional space
- This is called the kernel trick
- Not every function has this property. Given some candidate kernel \( k(x, z) \), how do we know if it corresponds to a scalar product in some space?
- A kernel is a valid kernel if the following holds (Mercer kernels)
  - Symmetry: \( k(x_i, x_j) = k(x_j, x_i) \)
  - Positive semi-definiteness: let \( K \) be the \( N \times N \) Kernel matrix \( K_{ij} = k(x_i, x_j) \), then \( K \) has to be positive semi-definite, i.e. \( v^T K v \geq 0 \ \forall v \in \mathbb{R}^N \)
  - For example, \( k(x, z) = x^T z \) is a valid kernel, \( k(x, z) = x - x^T z \) is not
Non-Linear SVM

- Popular examples of valid kernels include the **linear** kernel
  \[ k(x_i, x_j) = x_i^T x_j \]
- The degree \( p \) **polynomial kernel**, \( p > 0 \)
  \[ k(x_i, x_j) = (x_i^T x_j + 1)^p \]
- The **Radial Basis function (RBF)** or **Gaussian kernel**, \( \sigma > 0 \)
  \[ k(x_i, x_j) = \exp \left( -\frac{(x_i - x_j)^T (x_i - x_j)}{2\sigma^2} \right) \]
- The Gaussian kernel induces an **infinite dimensional** feature space (decomposition into \( x_i \)'s and \( x_j \)'s is done in a Taylor expansion of \( \exp \))
Non-Linear SVM

- The idea of kernels has significantly broader applicability than SVMs and is used in many learning algorithm that can be written in terms of only inner products
  - Examples include: perceptrons, kernel-PCA, kernel logistic regression, etc.
- There are many kernel functions, including ones that act upon symbolic inputs (as opposed to real-valued) and are defined over graphs, sets, strings or text documents
- Unless domain knowledge suggest the use of a specific kernel, the Gaussian kernel is a good generic choice for many practical classification tasks
- The concepts of SVMs using kernels (kernelized SVMs) and soft-margin SVMs can be readily combined
Example Classifications

- **Gaussian** (RBF) kernel, $\sigma = 3.2$
Example Classifications

- Circular class distributions

- **Gaussian** (RBF) kernel, $\sigma = 3.2$

- Kernel type, kernel parameters and stiffness parameter are usually determined by **cross validation** (later in this course)
Algorithm Summary

- **Learning**
  1. Find the **Lagrange multipliers** so that
     \[ L(\lambda) = \sum_{i=1}^{N} \lambda_i - \frac{1}{2} \sum_{i,j=1}^{N} \lambda_i \lambda_j y_i y_j k(x_i, x_j) \]
     is maximized subject to \( 0 \leq \lambda_i \leq C \, \forall i \) and \( \sum_i \lambda_i y_i = 0 \) using a QP solver
  2. Determine the set of support vectors \( S \) by finding the indices such that \( 0 < \lambda_i \leq C \, \forall i \)
  3. Calculate the **bias** \( b = \frac{1}{N_S} \sum_{s_i \in S} (y_{s_i} - \sum_{s_j \in S} \lambda_{s_j} y_{s_j} k(x_{s_j}, x_{s_i})) \)

- **Inference and decision**
  4. **Predict class** for new points \( x' \) by evaluating
     \[ y' = \text{sign}(\sum_{i=1}^{N} \lambda_i y_i k(x_i, x') + b) \]
Summary SVM

- A Support Vector Machine is a non-probabilistic discriminative classifier.
- Its approach to minimize the generalization error is to maximize the margin (it’s an instance of a maximum margin classifier).
- Learning is framed as a constraint quadratic optimization problem.
- The learned classifier only depends sparsely on the training set.
- Non-linear SVM transform input data which are not linearly separable into a higher dimensional feature space and apply linear separation there.
- The kernel trick is an efficient transformation of input data to some space and taking an inner product in that space without ever going there. Works even for infinite dimensional feature spaces.
- For non-linearly separable data there are two cases: for outliers use soft-margin SVM, for data with inherently non-linear class distributions, use non-linear, kernelized SVM.
Summary SVM

- **Advantages**
  - Kernel-based framework is very **powerful**
  - Quadratic optimization problem is **convex** and has a **unique** solution (as opposed to other classifiers such as NN, RVM)
  - Efficient inference due to **sparsity**
  - SVM classifiers work usually **very well** in practice

- **Drawbacks**
  - **Not probabilistic**
  - **Binary classifier**, extension to multi-class not straightforward
  - Learning may be very **slow** for large training sets
  - Constraint QP may run into **numerical instabilities**
Sources and Further Reading

These slides contain material by Russell and Norvig [2] (chapter 18), Bishop [1] (chapter 7 and 9), Ng’s lecture notes on SVM [3] and Fletcher [4]. Several images were produced using Karpathy’s nice and very instructive SVM applet [5].


To be continued in Supervised Learning, part 3/3