The Support Vector Machine

Nuno Vasconcelos
(Ken Kreutz-Delgado)

UC San Diego
**Geometric Interpretation**

- Summarizing, the linear discriminant decision rule

\[
\hat{h}^*(x) = \begin{cases} 
0 & \text{if } g(x) > 0 \\
1 & \text{if } g(x) < 0 
\end{cases}
\]

with

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

has the following properties

- It divides \(X\) into two “half-spaces”
- The boundary is the hyperplane with:
  - normal \(w\)
  - distance to the origin \(b/||w||\)
- \(g(x)/||w||\) gives the signed distance from point \(x\) to the boundary
  - \(g(x) = 0\) for points on the plane
  - \(g(x) > 0\) for points on the side \(w\) points to (“positive side”)
  - \(g(x) < 0\) for points on the “negative side”
For now, our goal is to explore the simplicity of the linear discriminant.

Let’s assume linear separability of the training data.

One handy trick is to use class labels $y \in \{-1, 1\}$ instead of $y \in \{0, 1\}$, where

- $y = 1$ for points on the positive side
- $y = -1$ for points on the negative side

The decision function then becomes

$$ h^*(x) = \begin{cases} 
1 & \text{if } g(x) > 0 \\
-1 & \text{if } g(x) < 0
\end{cases} \iff h^*(x) = \text{sgn}[g(x)] $$
Linear Discriminants & Separable Data

- We have a classification error if
  - \( y = 1 \) and \( g(x) < 0 \) or \( y = -1 \) and \( g(x) > 0 \)
  - i.e., if \( y g(x) < 0 \)

- We have a correct classification if
  - \( y = 1 \) and \( g(x) > 0 \) or \( y = -1 \) and \( g(x) < 0 \)
  - i.e., if \( y g(x) > 0 \)

- Note that, if the data is linearly separable, given a training set
  \[ D = \{(x_1, y_1), \ldots, (x_n, y_n)\} \]

  we can have zero training error.

- The necessary & sufficient condition for this is that
  \[ y_i \left( w^T x_i + b \right) > 0, \quad \forall i = 1, \ldots, n \]
The Margin

- The margin is the distance from the boundary to the closest point

\[ \gamma = \min_i \frac{w^T x_i + b}{\|w\|} \]

- There will be no error on the training set if it is strictly greater than zero:

\[ y_i (w^T x_i + b) > 0, \quad \forall i \iff \gamma > 0 \]

- Note that this is ill-defined in the sense that \( \gamma \) does not change if both \( w \) and \( b \) are scaled by a common scalar \( \lambda \)

- We need a normalization
A convenient normalization is to make \(|g(x)| = 1\) for the closest point, i.e.

\[
\min_i |w^T x_i + b| = 1
\]

under which

\[
y = \frac{1}{\|w\|}
\]

The Support Vector Machine (SVM) is the linear discriminant classifier that maximizes the margin subject to these constraints:

\[
\min_{w,b} \|w\|^2 \quad \text{subject to} \quad y_i \left( w^T x_i + b \right) \geq 1 \quad \forall i
\]
Duality

We must solve an optimization problem with constraints

There is a rich theory on how to solve such problems

- We will not get into it here (take 271B if interested)
- The main result is that we can often formulate a dual problem which is easier to solve
- In the dual formulation we introduce a vector of Lagrange multipliers $\alpha_i > 0$, one for each constraint, and solve

$$\max_{\alpha \geq 0} \ q(\alpha) = \max_{\alpha \geq 0} \ \left\{ \min_w \ L(w, b, \alpha) \right\}$$

- where

$$L(w, b, \alpha) = \frac{1}{2} \|w\|^2 - \sum_i \alpha_i \left[ y_i \left( w^T x_i + b \right) - 1 \right]$$

is the Lagrangian
The Dual Optimization Problem

For the SVM, the dual problem can be simplified into:

\[
\max_{\alpha \geq 0} \left\{ -\frac{1}{2} \sum_{ij} \alpha_i \alpha_j y_i y_j x_i^T x_j + \sum_i \alpha_i \right\}
\]

subject to \( \sum_i y_i \alpha_i = 0 \)

Once this is solved, the vector

\[
w^* = \sum_i \alpha_i y_i x_i
\]

is the normal to the maximum margin hyperplane

Note: the dual solution does not determine the optimal \( b^* \), since \( b \) drops out when we solve

\[
\min_w L(w, b, \alpha)
\]
The Dual Problem

There are various possibilities for determining \( b^* \). For example:

- Pick one point \( x^+ \) on the margin on the \( y = 1 \) side and one point \( x^- \) on margin on the \( y = -1 \) side
- Then use the margin constraint

\[
\begin{align*}
    w^T x^+ + b &= 1 \\
    w^T x^- + b &= -1
\end{align*}
\]

\( \iff \)

\[
b^* = -\frac{w^T (x^+ + x^-)}{2}
\]

Note:

- The maximum margin solution guarantees that there is always at least one point “on the margin” on each side
- If not, we could move the hyperplane and get an even larger margin (see figure on the right)
Support Vectors

It turns out that:

- An inactive constraint always has zero Lagrange multiplier $\alpha_i$
- That is,
  - i) $\alpha_i > 0$ and $y_i(w^*T x_i + b^*) = 1$
  - or
  - ii) $\alpha_i = 0$ and $y_i(w^*T x_i + b^*) > 1$
- Hence $\alpha_i > 0$ only for points $|w^*T x_i + b^*| = 1$
  - which are those that lie at a distance equal to the margin (i.e., those that are “on the margin”). These points are the “Support Vectors”
Support Vectors

- The points with \( \alpha_i > 0 \) “support” the optimal hyperplane \((w^*, b^*)\).
- This is why they are called “Support Vectors”
- Note that the decision rule is

\[
f(x) = \text{sgn} \left[ w^{*T} x + b^* \right] = \text{sgn} \left[ \sum_i y_i \alpha_i^* x_i^T \left( x - \frac{x^+ + x^-}{2} \right) \right] = \text{sgn} \left[ \sum_{i \in SV} y_i \alpha_i^* x_i^T \left( x - \frac{x^+ + x^-}{2} \right) \right]
\]

where \( SV = \{i \mid \alpha_i^* > 0\} \) indexes the set of support vectors
Support Vectors and the SVM

Since the decision rule is

\[ f(x) = \text{sgn} \left[ \sum_{i \in SV} y_i \alpha_i^* x_i^T \left( x - \frac{x^+ + x^-}{2} \right) \right] \]

where \( x^+ \) and \( x^- \) are support vectors, we see that we only need the support vectors to completely define the classifier!

We can literally throw away all other points!!

The Lagrange multipliers can also be seen as a measure of importance of each point.

Points with \( \alpha_i = 0 \) have no influence—a small perturbation does not change the solution.
The Robustness of SVMs

- We talked a lot about the “curse of dimensionality”
  - In general, the number of examples required to achieve certain precision of pdf estimation, and pdf-based classification, is exponential in the number of dimensions.

- It turns out that SVMs are remarkably robust to the dimensionality of the feature space.
  - Not uncommon to see successful applications on 1,000D+ spaces.

- Two main reasons for this:
  - 1) All that the SVM has to do is to learn a hyperplane.

    Although the number of dimensions may be large, the number of parameters is relatively small and there is not much room for overfitting.

    In fact, \( d+1 \) points are enough to specify the decision rule in \( R^d \) !!
Robustness: SVMs as Feature Selectors

The second reason for robustness is that the data/feature space effectively is not really that large

2) This is because the SVM is a feature selector

To see this let’s look at the decision function

\[ f(x) = \text{sgn} \left[ \sum_{i \in SV} y_i \alpha_i^* x_i^T x + b^* \right] \]

This is a thresholding of the quantity

\[ \sum_{i \in SV} y_i \alpha_i^* x_i^T x \]

Note that each of the terms \( x_i^T x \) is the projection (actually, inner product) of the vector which we wish to classify, \( x \), onto the training (support) vector \( x_i \)
SVMs as Feature Selectors

Define $z$ to be the vector of the projection of $x$ onto all of the support vectors:

$$z(x) = \left( x^T x_{i_1}, \cdots, x^T x_{i_k} \right)^T$$

The decision function is a hyperplane in the $z$-space:

$$f(x) = \text{sgn} \left[ \sum_{i \in \text{SV}} y_i \alpha_i^* x_i^T x + b^* \right] = \text{sgn} \left[ \sum_k w_k^* z_k(x) + b^* \right]$$

with

$$w^* = \left( \alpha_{i_1}^* y_{i_1}, \cdots, \alpha_{i_k}^* y_{i_k} \right)^T$$

This means that

- The classifier operates only on the span of the support vectors!
- The SVM performs feature selection automatically.
SVMs as Feature Selectors

Geometrically, we have:

- 1) Projection of new data point $x$ on the span of the support vectors
- 2) Classification on this (sub)space

$$z(x) = (w^*, b^*)$$

- The effective dimension is $|SV|$ and, typically, $|SV| \ll n$
Summary of the SVM

- SVM training:
  1) Solve the optimization problem:

\[
\begin{align*}
\max_{\alpha \geq 0} & \left\{ -\frac{1}{2} \sum_{ij} \alpha_i \alpha_j y_i y_j x_i^T x_j + \sum_i \alpha_i \right\} \\
\text{subject to} & \sum_i y_i \alpha_i = 0
\end{align*}
\]

2) Then compute the parameters of the “large margin” linear discriminant function:

\[
\begin{align*}
w^* &= \sum_{i \in SV} \alpha_i^* y_i x_i \\
b^* &= -\frac{1}{2} \sum_{i \in SV} y_i \alpha_i^* \left( x_i^T x^* + x_i^T x^- \right)
\end{align*}
\]

- SVM Linear Discriminant Decision Function:

\[
f(x) = \text{sgn} \left[ \sum_{i \in SV} y_i \alpha_i^* x_i^T x + b^* \right]
\]
Non-Separable Problems

- So far we have assumed linearly separable classes.
- This is rarely the case in practice.
- A separable problem is “easy” most classifiers will do well.
- We need to be able to extend the SVM to the non-separable case.
- Basic idea:
  - With class overlap we cannot enforce a (“hard”) margin.
  - But we can enforce a “soft margin”
  - For most points there is a margin. But there are a few outliers that cross-over, or are closer to the boundary than the margin. So how do we handle the latter set of points?
Soft Margin Optimization

Mathematically this is done by introducing slack variables.

Rather than solving the “hard margin” problem:

\[
\min_{w, b} \|w\|^2 \quad \text{subject to } y_i \left( w^T x_i + b \right) \geq 1 \quad \forall i
\]

instead we solve the “soft margin” problem:

\[
\min_{w, \xi, b} \|w\|^2 \quad \text{subject to } y_i \left( w^T x_i + b \right) \geq 1 - \xi_i \quad \forall i
\]

\[
\xi_i \geq 0, \forall i
\]

The \( \xi_i \) are called slack variables.

Basically, the same optimization as before but points with \( \xi_i > 0 \) are allowed to violate the margin.
Soft Margin Optimization

- Note that, as it stands, the problem is not well defined.
- By making $\xi_i$ arbitrarily large, $w \approx 0$ is a solution!
- Therefore, we need to penalize large values of $\xi_i$.
- Thus, instead we solve the penalized, or regularized, optimization problem:

$$\min_{w, \xi, b} \|w\|^2 + C \sum_i \xi_i$$

subject to

$$y_i \left( w^T x_i + b \right) \geq 1 - \xi_i \quad \forall i$$

$$\xi_i \geq 0, \forall i$$

- The quantity $C \sum \xi_i$ is the penalty, or regularization, term. The positive parameter $C$ controls how harsh it is.
The Soft Margin Dual Problem

The dual optimization problem:

\[
\max_{\alpha \geq 0} \left\{ -\frac{1}{2} \sum_{ij} \alpha_i \alpha_j y_i y_j x_i^T x_j + \sum_i \alpha_i \right\}
\]

subject to \( \sum_i y_i \alpha_i = 0 \),

\( 0 \leq \alpha_i \leq C \)

The only difference with respect to the hard margin case is the “box constraint” on the Lagrange multipliers \( \alpha_i \)

Geometrically we have this...
Support Vectors

- They are the points with $\alpha_i > 0$
- As before, the decision rule is
  \[ f(x) = \text{sgn} \left[ \sum_{i \in SV} y_i \alpha_i^* x_i^T x + b^* \right] \]
  where $SV = \{ i \mid \alpha_i^* > 0 \}$
  and $b^*$ is chosen s.t.
  - $y_i g(x_i) = 1$, for all $x_i$ s.t. $0 < \alpha_i < C$
- The box constraint on the Lagrange multipliers:
  - makes intuitive sense as it prevents any single support vector outlier from having an unduly large impact in the decision rule.
Summary of the soft-margin SVM

**SVM training:**

1. Solve the optimization problem:
   \[
   \max_{\alpha \geq 0} \left\{ -\frac{1}{2} \sum_{ij} \alpha_i \alpha_j y_i y_j x_i^T x_j + \sum_i \alpha_i \right\}
   \]
   subject to \( \sum_i y_i \alpha_i = 0, \)
   \( 0 \leq \alpha_i \leq C \)

2. Then compute the parameters of the “large margin” linear discriminant function:
   \[
   w^* = \sum_{i \in SV} \alpha_i^* y_i x_i
   \]
   \[
   b^* = -\frac{1}{2} \sum_{i \in SV} y_i \alpha_i^* \left( x_i^T x^+ + x_i^T x^- \right)
   \]

**SVM Linear Discriminant Decision Function:**

\[
 f(x) = \text{sgn} \left[ \sum_{i \in SV} y_i \alpha_i^* x_i^T x + b^* \right]
\]
“The Kernel Trick”

What if we want a non-linear boundary?

Consider the following transformation of the feature space:

- Introduce a mapping to a “better” (i.e., linearly separable) feature space
  \[ \phi: \mathcal{X} \to \mathcal{Z} \]
  where, generally, \( \dim(\mathcal{Z}) > \dim(\mathcal{X}) \).

- If a classification algorithm only depends on the data through inner products then, in the transformed space, it depends on
  \[ \langle \phi(x_i), \phi(x_j) \rangle = \phi^T(x_i) \phi(x_j) \]
The Inner Product Implementation

- In the transformed space, the learning algorithms only require inner products

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

- Note that we do not need to store \( \phi(x_j) \), but only the \( n^2 \) (scalar) component values of the inner product matrix

- Interestingly, this holds even if \( \phi(x) \) takes its value in an infinite dimensional space.
  - We get a reduction from infinity to \( n^2! \)
  - There is, however, still one problem:
    - When \( \phi(x_j) \) is infinite dimensional the computation of the inner product \( \langle \phi(x_i), \phi(x_j) \rangle \) looks impossible.
“The Kernel Trick”

“Instead of defining $\phi(x)$, then computing $\phi(x_i)$ for each $i$, and then computing $\langle\phi(x_i),\phi(x_j)\rangle$ for each pair $(i,j)$, simply define a kernel function

$$K(x,z) = \langle\phi(x),\phi(z)\rangle$$

and work with it directly.”

$K(x,z)$ is called an inner product or dot-product kernel

Since we only use the kernel, why bother to define $\phi(x)$?

Just define the kernel $K(x,z)$ directly!

Then we never have to deal with the complexity of $\phi(x)$.

This is usually called “the kernel trick”
Kernel Summary

1. \( D \) not easy to deal with in \( \mathcal{X} \), apply feature transformation \( \phi: \mathcal{X} \rightarrow \mathcal{Z} \), such that \( \dim(\mathcal{Z}) \gg \dim(\mathcal{X}) \)

2. Constructing and computing \( \phi(x) \) directly is too expensive:
   - Write your learning algorithm in inner product form
   - Then, instead of \( \phi(x) \), we only need \( \langle \phi(x_i), \phi(x_j) \rangle \) for all \( i \) and \( j \), which we can compute by defining an “inner product kernel”

\[
K(x, z) = \langle \phi(x), \phi(z) \rangle
\]

and computing \( K(x_i, x_j) \forall i, j \) directly
   - Note: the matrix

\[
K = \begin{bmatrix}
    \vdots \\
    \cdots K(x_i, z_j) \cdots \\
    \vdots
\end{bmatrix}
\]

is called the “Kernel matrix” or Gram matrix

3. Moral: Forget about \( \phi(x) \) and instead use \( K(x, z) \) from the start!
Question?

What is a good inner product kernel?

- This is a difficult question (see Prof. Lenckriet’s work)

In practice, the usual recipe is:

- Pick a kernel from a library of known kernels
- some examples
  - the linear kernel \( K(x, z) = x^T z \)
  - the Gaussian family
    \[
    K(x, z) = e^{\frac{-\|x-z\|^2}{\sigma}}
    \]
  - the polynomial family
    \[
    K(x, z) = \left(1 + x^T z\right)^k, \quad k \in \{1, 2, \cdots\}
    \]
Kernelization of the SVM

- Note that all SVM equations depend only on $x_i^T x_j$
- The kernel trick is trivial: replace by $K(x_i, x_j)$

1) Training:

$$\max_{\alpha \geq 0} \left\{ -\frac{1}{2} \sum_{ij} \alpha_i \alpha_j y_i y_j k(x_i, x_j) + \sum_i \alpha_i \right\}$$

subject to  $\sum_i y_i \alpha_i = 0, \quad 0 \leq \alpha_i \leq C$

$$b^* = -\frac{1}{2} \sum_{i \in SV} y_i \alpha_i^* (K(x_i, x^+) + K(x_i, x^-))$$

2) Decision function:

$$f(x) = \text{sgn} \left[ \sum_{i \in SV} y_i \alpha_i^* K(x_i, x) + b^* \right]$$
Kernelization of the SVM

Notes:

• As usual, nothing we did really requires us to be in $\mathbb{R}^d$.
  
  ➢ We could have simply used $\langle x_i, x_j \rangle$ to denote for the inner product on a infinite dimensional space and all the equations would still hold

• The only difference is that we can no longer recover $w^*$ explicitly without determining the feature transformation $\phi$, since

$$w^* = \sum_{i \in SV} \alpha_i^* y_i \phi(x_i)$$

• This can be an infinite dimensional object. E.g., it is a sum of Gaussians (“lives” in an infinite dimensional function space) when we use the Gaussian kernel

• Luckily, we don’t need $w^*$, only the SVM decision function

$$f(x) = \text{sgn} \left[ \sum_{i \in SV} y_i \alpha_i^* K(x_i, x) + b^* \right]$$
Limitations of the SVM

The SVM is appealing, but there are some limitations:

- A major problem is the selection of an appropriate kernel. There is no generic “optimal” procedure to find the kernel or its parameters.
  - Usually we pick an arbitrary kernel, e.g. Gaussian
  - Then, determine kernel parameters, e.g. variance, by trial and error
  - $C$ controls the importance of outliers (larger $C$ = less influence)
    - Not really intuitive how to choose $C$

SVM is usually tuned and performance-tested using cross-validation. There is a need to cross-validate with respect to both $C$ and kernel parameters.
Practical Implementation of the SVM

In practice, we need an algorithm for solving the optimization problem of the training stage

• This is a complex problem
• There has been a large amount of research in this area
  ➢ Therefore, writing “your own” algorithm is not going to be competitive
• Luckily there are various packages available, e.g.:
  • libSVM: http://www.csie.ntu.edu.tw/~cjlin/libsvm/
  • SVM light: http://www.cs.cornell.edu/People/tj/svm_light/
  • SVM fu: http://five-percent-nation.mit.edu/SvmFu/
  • various others (see http://www.support-vector.net/software.html)
• There are also many papers and books on algorithms (see e.g. B. Schölkopf and A. Smola. Learning with Kernels. MIT Press, 2002)
END