Support Vector Machine
(and Statistical Learning Theory)
Tutorial

Jason Weston
NEC Labs America
4 Independence Way, Princeton, USA.
jasonw@nec-labs.com
1 Support Vector Machines: history

- SVMs introduced in COLT-92 by Boser, Guyon & Vapnik. Became rather popular since.

- Theoretically well motivated algorithm: developed from Statistical Learning Theory (Vapnik & Chervonenkis) since the 60s.

- Empirically good performance: successful applications in many fields (bioinformatics, text, image recognition, …)
2 Support Vector Machines: history II

- Several textbooks, e.g. ”An introduction to Support Vector Machines” by Cristianini and Shawe-Taylor is one.
- A large and diverse community work on them: from machine learning, optimization, statistics, neural networks, functional analysis, etc.
3 Support Vector Machines: basics

[Boser, Guyon, Vapnik ’92], [Cortes & Vapnik ’95]

Nice properties: convex, theoretically motivated, nonlinear with kernels.
4 Preliminaries:

- Machine learning is about learning structure from data.
- Although the class of algorithms called ”SVM”s can do more, in this talk we focus on pattern recognition.
- So we want to learn the mapping: $\mathcal{X} \mapsto \mathcal{Y}$, where $x \in \mathcal{X}$ is some object and $y \in \mathcal{Y}$ is a class label.
- Let’s take the simplest case: 2-class classification. So: $x \in \mathbb{R}^n$, $y \in \{\pm 1\}$. 
5 Example:

Suppose we have 50 photographs of elephants and 50 photos of tigers.

We digitize them into 100 x 100 pixel images, so we have $x \in \mathbb{R}^n$ where $n = 10,000$.

Now, given a new (different) photograph we want to answer the question: is it an elephant or a tiger? [we assume it is one or the other.]
6 Training sets and prediction models

- input/output sets $\mathcal{X}, \mathcal{Y}$
- training set $(x_1, y_1), \ldots, (x_m, y_m)$
- ”generalization”: given a previously seen $x \in \mathcal{X}$, find a suitable $y \in \mathcal{Y}$.
- i.e., want to learn a classifier: $y = f(x, \alpha)$, where $\alpha$ are the parameters of the function.
- For example, if we are choosing our model from the set of hyperplanes in $\mathbb{R}^n$, then we have:

$$f(x, \{w, b\}) = \text{sign}(w \cdot x + b).$$
We can try to learn \( f(x, \alpha) \) by choosing a function that performs well on training data:

\[
R_{\text{emp}}(\alpha) = \frac{1}{m} \sum_{i=1}^{m} \ell(f(x_i, \alpha), y_i) = \text{Training Error}
\]

where \( \ell \) is the zero-one loss function, \( \ell(y, \hat{y}) = 1 \), if \( y \neq \hat{y} \), and 0 otherwise. \( R_{\text{emp}} \) is called the empirical risk.

By doing this we are trying to minimize the overall risk:

\[
R(\alpha) = \int \ell(f(x, \alpha), y) dP(x, y) = \text{Test Error}
\]

where \( P(x,y) \) is the (unknown) joint distribution function of \( x \) and \( y \).
8 Choosing the set of functions

What about \( f(x, \alpha) \) allowing \textit{all} functions from \( \mathcal{X} \) to \( \{\pm 1\} \)?

\textbf{Training set} \((x_1, y_1), \ldots, (x_m, y_m) \in \mathcal{X} \times \{\pm 1\}\)

\textbf{Test set} \(\bar{x}_1, \ldots, \bar{x}_m \in \mathcal{X}\),

such that the two sets do not intersect.

For any \( f \) there exists \( f^* \):

1. \( f^*(x_i) = f(x_i) \) for all \( i \)
2. \( f^*(x_j) \neq f(x_j) \) for all \( j \)

Based on the training data alone, there is no means of choosing which function is better. On the test set however they give different results. So generalization is not guaranteed.

\( \implies \) a restriction must be placed on the functions that we allow.
9 Empirical Risk and the true Risk

Vapnik & Chervonenkis showed that an upper bound on the true risk can be given by the empirical risk + an additional term:

\[ R(\alpha) \leq R_{emp}(\alpha) + \sqrt{h(\log\left(\frac{2m}{h} + 1\right) - \log\left(\frac{n}{4}\right)} \frac{m}{m} \]

where \(h\) is the VC dimension of the set of functions parameterized by \(\alpha\).

- The VC dimension of a set of functions is a measure of their capacity or complexity.
- If you can describe a lot of different phenomena with a set of functions then the value of \(h\) is large.

\[ VC \ dim = \ the \ maximum \ number \ of \ points \ that \ can \ be \ separated \ in \ all \ possible \ ways \ by \ that \ set \ of \ functions. \]
10 VC dimension:

The VC dimension of a set of functions is the maximum number of points that can be separated in all possible ways by that set of functions. For hyperplanes in $\mathbb{R}^n$, the VC dimension can be shown to be $n + 1$. 
11 VC dimension and capacity of functions

Simplification of bound:

\[ \text{Test Error} \leq \text{Training Error} + \text{Complexity of set of Models} \]

- Actually, a lot of bounds of this form have been proved (different measures of capacity). The complexity function is often called a regularizer.

- If you take a high capacity set of functions (explain a lot) you get low training error. But you might ”overfit”.

- If you take a very simple set of models, you have low complexity, but won’t get low training error.
12 Capacity of a set of functions (classification)

[Images taken from a talk by B. Schoelkopf.]
13 Capacity of a set of functions (*regression*)
14 Controlling the risk: model complexity

- Bound on the risk
- Confidence interval
- Empirical risk (training error)

Diagram showing the relationship between model complexity and risk, with various regions denoted by $S_1$, $S^*_1$, $S_n$, and $h_1$, $h^*$, $h_n$.
15 Capacity of hyperplanes

Vapnik & Chervonenkis also showed the following:

Consider hyperplanes \((w \cdot x) = 0\) where \(w\) is normalized w.r.t a set of points \(X^*\) such that: \(\min_i |w \cdot x_i| = 1\).

The set of decision functions \(f_w(x) = \text{sign}(w \cdot x)\) defined on \(X^*\) such that \(|w| \leq A\) has a VC dimension satisfying

\[ h \leq R^2 A^2. \]

where \(R\) is the radius of the smallest sphere around the origin containing \(X^*\).

\[ \implies \text{minimize } |w|^2 \text{ and have low capacity} \]

\[ \implies \text{minimizing } |w|^2 \text{ equivalent to obtaining a large margin classifier} \]
\[\langle w, x \rangle + b < 0\]

\[\langle w, x \rangle + b > 0\]

\[\{x \mid \langle w, x \rangle + b = 0\}\]
\[ \{ x \mid \langle w, x \rangle + b = -1 \} \]

\[ \{ x \mid \langle w, x \rangle + b = +1 \} \]

Note:

\[ \langle w, x_1 \rangle + b = +1 \]
\[ \langle w, x_2 \rangle + b = -1 \]

\[ \Rightarrow \langle w, (x_1 - x_2) \rangle = 2 \]
\[ \Rightarrow \frac{w}{\|w\|}, (x_1 - x_2) = \frac{2}{\|w\|} \]
So, we would like to find the function which minimizes an objective like:

\[ \text{Training Error} + \text{Complexity term} \]

We write that as:

\[
\frac{1}{m} \sum_{i=1}^{m} \ell(f(x_i, \alpha), y_i) + \text{Complexity term}
\]

For now we will choose the set of hyperplanes (we will extend this later), so \( f(x) = (w \cdot x) + b \):

\[
\frac{1}{m} \sum_{i=1}^{m} \ell(w \cdot x_i + b, y_i) + ||w||^2
\]

subject to \( \min_i |w \cdot x_i| = 1 \).
That function before was a little difficult to minimize because of the step function in $\ell(y, \hat{y})$ (either 1 or 0).

Let’s assume we can separate the data perfectly. Then we can optimize the following:

Minimize $||w||^2$, subject to:

$$(w \cdot x_i + b) \geq 1, \text{ if } y_i = 1$$

$$(w \cdot x_i + b) \leq -1, \text{ if } y_i = -1$$

The last two constraints can be compacted to:

$$y_i(w \cdot x_i + b) \geq 1$$

This is a quadratic program.
18 SVMs : non-separable case

To deal with the non-separable case, one can rewrite the problem as:

Minimize:

\[ ||w||^2 + C \sum_{i=1}^{m} \xi_i \]

subject to:

\[ y_i(w \cdot x_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0 \]

This is just the same as the original objective:

\[ \frac{1}{m} \sum_{i=1}^{m} \ell(w \cdot x_i + b, y_i) + ||w||^2 \]

except \( \ell \) is no longer the zero-one loss, but is called the ”hinge-loss”:

\[ \ell(y, \hat{y}) = \max(0, 1 - y\hat{y}) \]. This is still a quadratic program!
19 Support Vector Machines - Primal

- **Decision function:**
  \[ f(x) = w \cdot x + b \]

- **Primal formulation:**
  \[
  \min P(w, b) = \frac{1}{2} ||w||^2 + C \sum_i H_1[y_i f(x_i)] \\
  \text{maximize margin} \quad \text{minimize training error}
  \]

Ideally \( H_1 \) would count the number of errors, approximate with:

**Hinge Loss** \( H_1(z) = \max(0, 1 - z) \)
Linear classifiers aren’t complex enough sometimes. SVM solution: 
*Map data into a richer feature space including nonlinear features, then construct a hyperplane in that space so all other equations are the same!*

Formally, preprocess the data with:

\[ x \mapsto \Phi(x) \]

and then learn the map from \( \Phi(x) \) to \( y \):

\[ f(x) = w \cdot \Phi(x) + b. \]
21 SVMs: polynomial mapping

\[ \Phi : \mathbb{R}^2 \rightarrow \mathbb{R}^3 \]

\[ (x_1, x_2) \mapsto (z_1, z_2, z_3) := (x_1^2, \sqrt{2}x_1x_2, x_2^2) \]
For example MNIST hand-writing recognition.
60,000 training examples, 10000 test examples, 28x28.
Linear SVM has around 8.5% test error.
Polynomial SVM has around 1% test error.
## SVMs: full MNIST results

<table>
<thead>
<tr>
<th>Classifier</th>
<th>Test Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>linear</td>
<td>8.4%</td>
</tr>
<tr>
<td>3-nearest-neighbor</td>
<td>2.4%</td>
</tr>
<tr>
<td>RBF-SVM</td>
<td>1.4%</td>
</tr>
<tr>
<td>Tangent distance</td>
<td>1.1%</td>
</tr>
<tr>
<td>LeNet</td>
<td>1.1%</td>
</tr>
<tr>
<td>Boosted LeNet</td>
<td>0.7%</td>
</tr>
<tr>
<td>Translation invariant SVM</td>
<td>0.56%</td>
</tr>
</tbody>
</table>

Choosing a good mapping $\Phi(\cdot)$ (encoding prior knowledge + getting right complexity of function class) for your problem improves results.
24 SVMs: the kernel trick

Problem: the dimensionality of $\Phi(x)$ can be very large, making $w$ hard to represent explicitly in memory, and hard for the QP to solve.

The Representer theorem (Kimeldorf & Wahba, 1971) shows that (for SVMs as a special case):

$$w = \sum_{i=1}^{m} \alpha_i \Phi(x_i)$$

for some variables $\alpha$. Instead of optimizing $w$ directly we can thus optimize $\alpha$.

The decision rule is now:

$$f(x) = \sum_{i=1}^{m} \alpha_i \Phi(x_i) \cdot \Phi(x) + b$$

We call $K(x_i, x) = \Phi(x_i) \cdot \Phi(x)$ the **kernel function**.
We can rewrite all the SVM equations we saw before, but with the \( w = \sum_{i=1}^{m} \alpha_i \Phi(x_i) \) equation:

- **Decision function:**

\[
    f(x) = \sum_i \alpha_i \Phi(x_i) \cdot \Phi(x) + b
\]

\[
    = \sum_i \alpha_i K(x_i, x) + b
\]

- **Dual formulation:**

\[
    \min P(w, b) = \frac{1}{2} \left\| \sum_{i=1}^{m} \alpha_i \Phi(x_i) \right\|^2 + C \sum_i H_1[y_i f(x_i)]
\]

\[\text{maximize margin} \quad \text{minimize training error}\]
But people normally write it like this:

- **Dual formulation:**

\[
\min_{\alpha} D(\alpha) = \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j \Phi(x_i) \cdot \Phi(x_j) - \sum_i y_i \alpha_i \quad \text{s.t.} \quad \begin{cases} 
\sum_i \alpha_i = 0 \\
0 \leq y_i \alpha_i \leq C
\end{cases}
\]

- **Dual Decision function:**

\[
f(x) = \sum_i \alpha_i K(x_i, x) + b
\]

- **Kernel function** \( K(\cdot, \cdot) \) is used to make (implicit) nonlinear feature map, e.g.
  - Polynomial kernel: \( K(x, x') = (x \cdot x' + 1)^d \).
  - RBF kernel: \( K(x, x') = \exp(-\gamma \|x - x'\|^2) \).
The kernel $K(x, x') = (x \cdot x')^d$ gives the same result as the explicit mapping $+ \cdot$ dot product that we described before:

$$\Phi : R^2 \rightarrow R^3 \quad (x_1, x_2) \mapsto (z_1, z_2, z_3) := (x_1^2, \sqrt(2)x_1x_2, x_2^2)$$

$$\Phi((x_1, x_2) \cdot \Phi((x'_1, x'_2) = (x_1^2, \sqrt(2)x_1x_2, x_2^2) \cdot (x'_1, \sqrt(2)x'_1x'_2, x'_2^2)$$

$$= x_1^2x'_1^2 + 2x_1x'_1x_2x'_2 + x_2^2x'_2^2$$

is the same as:

$$K(x, x') = (x \cdot x')^2 = ((x_1, x_2) \cdot (x'_1, x'_2))^2$$

$$= (x_1x'_1 + x_2x'_2)^2 = x_1^2x'_1^2 + x_2^2x'_2^2 + 2x_1x'_1x_2x'_2$$

Interestingly, if $d$ is large the kernel is still only requires $n$ multiplications to compute, whereas the explicit representation may not fit in memory!
The RBF kernel $K(x, x') = \exp(-\gamma \|x - x'\|^2)$ is one of the most popular kernel functions. It adds a ”bump” around each data point:

$$f(x) = \sum_{i=1}^{m} \alpha_i \exp(-\gamma \|x_i - x\|^2) + b$$

Using this one can get state-of-the-art results.
29 SVMs : more results

There is much more in the field of SVMs/ kernel machines than we could cover here, including:

- Regression, clustering, semi-supervised learning and other domains.
- Lots of other kernels, e.g. string kernels to handle text.
- Lots of research in modifications, e.g. to improve generalization ability, or tailoring to a particular task.
- Lots of research in speeding up training.

Please see text books such as the ones by Cristianini & Shawe-Taylor or by Schoelkopf and Smola.
30 SVMs : software

Lots of SVM software:

- LibSVM (C++)
- SVMLight (C)

As well as complete machine learning toolboxes that include SVMs:

- Torch (C++)
- Spider (Matlab)
- Weka (Java)

All available through www.kernel-machines.org.