Support Vector Machines
(Vapnik, 1979)

• Assume a binary classification problem.
  – Instances are represented by vector $x \in \mathbb{R}^n$.
  
  – Training examples: $x = (f_1, f_2, \ldots, f_n)$

$$S = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n) \mid (x_i, y_i) \in \mathbb{R}^n \times \{+1, -1\} \}$$

  – Hypothesis: A function $h: \mathbb{R}^n \rightarrow \{+1, -1\}$.

$$h(x) = h(f_1, f_2, \ldots, f_n) \in \{+1, -1\}$$
• Here, assume positive and negative instances are to be separated by the hyperplane

\[ \mathbf{w} \cdot \mathbf{x} + b = 0 \]

Equation of line:

\[ \mathbf{w} \cdot \mathbf{x} + b = w_1 f_1 + w_2 f_2 + b = 0 \]
• Intuition: the best hyperplane (for future generalization) will “maximally” separate the examples
Definition of Margin

- The margin of the positive examples, $d_+$ with respect to that hyperplane, is the shortest distance from a positive example to the hyperplane:
The margin of the negative examples, $d_-$ with respect to that hyperplane, is the shortest distance from a negative example to the hyperplane:
• The margin of the training set $S$ with respect to the hyperplane is $d_+ + d_-$. 
• The margin of the training set $S$ with respect to the hyperplane is $d_+ + d_-$. 

Vapnik showed that the hyperplane maximizing the margin of $S$ will have minimal VC dimension in the set of all consistent hyperplanes, and will thus be optimal.
The margin of the training set $S$ with respect to the hyperplane is $d_+ + d_-$. 

Vapnik showed that the hyperplane maximizing the margin of $S$ will have minimal VC dimension in the set of all consistent hyperplanes, and will thus be optimal.

This is an optimization problem!
• Note that the hyperplane is defined as

\[ \mathbf{w} \cdot \mathbf{x} + b = 0 \]

• To make the math easier, we will rescale \( \mathbf{w} \) and \( b \) such that the hyperplane is halfway in-between the closest positive and negative examples, and

\[
x_i \cdot \mathbf{w} + b \geq +1 \quad \text{for positive instances} \ (y_i = +1)
\]
\[
x_i \cdot \mathbf{w} + b \leq -1 \quad \text{for negative instances} \ (y_i = -1)
\]
From M. A. Hearst et al. paper (on class web page)

\[ \{ x \mid (w \cdot x) + b = -1 \} \]

\[ \{ x \mid (w \cdot x) + b = +1 \} \]

\[ y_i = -1 \]

\[ y_i = +1 \]

Note:

\[ (w \cdot x_1) + b = +1 \]

\[ (w \cdot x_2) + b = 1 \]

\[ \Rightarrow (w \cdot (x_1 - x_2)) = 2 \]

\[ \Rightarrow \left( \frac{w}{||w||} \cdot (x_1 - x_2) \right) = \frac{2}{||w||} \]
• In this case, the margin is

\[ \frac{2}{\|w\|} = \frac{2}{\sqrt{w \cdot w}} \]

• So to maximize the margin, we need to minimize \( \|w\| \).
Minimizing the margin

Find $w$ and $b$ by doing the following minimization:

$$\min_{w,b} \left( \frac{1}{2} \|w\|^2 \right)$$

subject to:

$$y_i \left( \langle w, x_i \rangle + b \right) \geq 1, \quad i = 1, \ldots, n$$

$$(y_i \in \{-1, +1\})$$

This is a quadratic optimization problem. Use “standard optimization tools” to solve it.
• **Dual formulation:** It turns out that $w$ can be expressed as a linear combination of a small subset of the training examples: those that lie exactly on margin (minimum distance to hyperplane):

$$w = \sum y_i \alpha_i x_i$$

such that $x_i$ lie exactly on the margin.

• These training examples are called “support vectors”. They carry all relevant information about the classification problem.
• The result of the SVM training algorithm (involving solving a quadratic programming problem), is the $\alpha_i$’s and the $x_i$’s.
• For a new example \( x \), We can now classify \( x \) using the support vectors:

\[
\text{class}(x) = \text{sgn}\left( \sum_i \alpha_i \langle x, x_i \rangle + b \right)
\]

• This is the resulting SVM classifier.
Non-linearly separable training examples

• What if the training examples are not linearly separable?

• Use old trick: find a function that maps points to a higher dimensional space ("feature space") in which they are linearly separable, and do the classification in that higher-dimensional space.
Need to find a function $\Phi$ that will perform such a mapping:

$$\Phi: \mathbb{R}^n \rightarrow F$$

Then can find hyperplane in higher dimensional feature space $F$, and do classification using that hyperplane in higher dimensional space.
• Problem:

- Recall that classification of instance $x$ is expressed in terms of dot products of $x$ and support vectors.

$$\text{Class}(x) = \text{sgn} \left( \sum_{i} \alpha_i (x \cdot x_i) + w_0 \right)$$

- The quadratic programming problem of finding the support vectors and coefficients also depends only on dot products between training examples, rather than on the training examples outside of dot products.
- So if each $x_i$ is replaced by $\Phi(x_i)$ in these procedures, we will have to calculate a lot of dot products, $\Phi(x_i) \cdot \Phi(x_j)$

- But in general, if the feature space $F$ is high dimensional, $\Phi(x_i) \cdot \Phi(x_j)$ will be expensive to compute.

- Also $\Phi(x)$ can be expensive to compute
• **Second trick:**

  – Suppose that there were some magic function,

    \[ k(\mathbf{x}_i, \mathbf{x}_j) = \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j) \]

    such that \( k \) is cheap to compute even though \( \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j) \) is expensive to compute.

  – Then we wouldn’t need to compute the dot product directly; we’d just need to compute \( k \) during both the training and testing phases.

  – The good news is: such \( k \) functions exist! They are called “kernel functions”, and come from the theory of integral operators.
Example: Polynomial kernel:

Suppose \( \mathbf{x} = (x_1, x_2) \) and \( \mathbf{y} = (y_1, y_2) \).

\[
k(\mathbf{x}, \mathbf{y}) = (\mathbf{x} \cdot \mathbf{y})^2
\]

Let \( \Phi(\mathbf{x}) = \left( x_1^2, \sqrt{2} \cdot x_1 x_2, x_2^2 \right) \).

Then:

\[
\Phi(\mathbf{x}) \cdot \Phi(\mathbf{y}) = \left( \begin{pmatrix} x_1^2 \\ \sqrt{2} \cdot x_1 x_2 \\ x_2^2 \end{pmatrix} \right) \cdot \left( \begin{pmatrix} y_1^2 \\ \sqrt{2} \cdot y_1 y_2 \\ y_2^2 \end{pmatrix} \right)
\]

\[
= \left( \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \cdot \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} \right)^2 = (\mathbf{x} \cdot \mathbf{y})^2 = k(\mathbf{x}, \mathbf{y})
\]
Recap of SVM algorithm

Given training set
\[ S = \{(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m) \mid (x_i, y_i) \in \mathbb{R}^n \times \{+1, -1\}\]  

1. Choose a map \( \Phi: \mathbb{R}^n \rightarrow F \), which maps \( x_i \) to a higher dimensional feature space. (Solves problem that \( X \) might not be linearly separable in original space.)

2. Choose a cheap-to-compute kernel function
\[ k(x,z) = \Phi(x) \cdot \Phi(z) \]
(Solves problem that in high dimensional spaces, dot products are very expensive to compute.)

3. Map all the \( x_i \)'s to feature space \( F \) by computing \( \Phi(x_i) \).
4. Apply quadratic programming procedure (using the kernel function $k$) to find a hyperplane $(w, w_0)$, such that

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

where the $\Phi(x_i)$’s are support vectors, the $\alpha_i$’s are coefficients, and $w_0$ is a threshold, such that $(w, w_0)$ is the hyperplane maximizing the margin of $S$ in $F$. 
Now, given a new instance, $x$, find the classification of $x$ by computing

$$\text{class}(x) = \text{sgn} \left( \sum_i \alpha_i (\Phi(x) \cdot \Phi(x_i)) + w_0 \right)$$

$$= \text{sgn} \left( \sum_i \alpha_i k(x, x_i) + w_0 \right)$$
Demo: Spam classification using SVMs
Example:
Applying SVMs to text classification
(Dumais et al., 1998)

• Used Reuters collection of news stories.

• Question: Is a particular news story in the category “grain” (i.e., about grain, grain prices, etc.)?

• Training examples: Vectors of features such as appearance or frequency of key words. (Similar to our spam-classification task.)

• Resulting SVM: weight vector defined in terms of support vectors and coefficients, plus threshold.
### Precision / Recall

- Confusion matrix for a classifier:

<table>
<thead>
<tr>
<th></th>
<th>Classified Positive</th>
<th>Classified Negative</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Positive Examples</strong></td>
<td>True positives (TP)</td>
<td>False negatives (FN)</td>
</tr>
<tr>
<td><strong>Negative Examples</strong></td>
<td>False positives (FP)</td>
<td>True negatives (TN)</td>
</tr>
</tbody>
</table>
Some performance measures

• **Accuracy**: proportion of classifications, over all the $N$ examples, that were correct:
  \[
  \text{accuracy} = \frac{TP + TN}{N}
  \]

• **Recall (or true positive rate, or “detection rate”)**: Proportion of positive examples that were classified correctly:
  \[
  \text{recall} = \frac{TP}{TP + FN}
  \]

• **Precision**: Proportion of correct positive classifications over all positive classifications:
  \[
  \text{precision} = \frac{TP}{TP + FP}
  \]
## Example

<table>
<thead>
<tr>
<th>Test data</th>
<th>Correct Classification</th>
<th>Model’s Classification</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1$</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>$x_2$</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>$x_3$</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>$x_4$</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>$x_5$</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>$x_6$</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>$x_7$</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>$x_8$</td>
<td>F</td>
<td>T</td>
</tr>
</tbody>
</table>

Accuracy =
Recall =
Precision =
Interpretation of precision and recall

• Precision and recall are often plotted against one another, especially in “detection” applications (such as spam detection), when positive examples are sparse in the observed data.

• **Recall**: How often did the system correctly identify positive examples when it encountered them?

• **Precision**: How often did the system get positive classifications correct?

• How do these two measures trade off against one another?
Precision / Recall Curves
SVMs also used in Watson for question classification

e.g., see Moschitti et al., Using syntactic and semantic structural kernels for classifying definition questions in Jeopardy!
SVM Demo