Support Vector Machines

- Hypothesis Space
  - variable size
  - deterministic
  - continuous parameters

- Learning Algorithm
  - linear and quadratic programming
  - eager
  - batch

- SVMs combine three important ideas
  - Apply optimization algorithms from Operations Research (Linear Programming and Quadratic Programming)
  - Implicit feature transformation using kernels
  - Control of overfitting by maximizing the margin
White Lie Warning

- This first introduction to SVMs describes a special case with simplifying assumptions.
- We will revisit SVMs later in the quarter and remove the assumptions.
- The material you are about to see does not describe "real" SVMs.
The linear programming problem is the following:

- find \( w \)
- minimize \( c \cdot w \)
- subject to
  - \( w \cdot a_i = b_i \) for \( i = 1, \ldots, m \)
  - \( w_j \geq 0 \) for \( j = 1, \ldots, n \)

There are fast algorithms for solving linear programs including the simplex algorithm and Karmarkar’s algorithm.
Formulating LTU Learning as Linear Programming

- Encode classes as \{+1, −1\}
- LTU:
  \[ h(x) = +1 \text{ if } w \cdot x \geq 0 \]
  \[ = −1 \text{ otherwise} \]
- An example \((x_i, y_i)\) is classified correctly by \(h\) if
  \[ y_i \cdot w \cdot x_i > 0 \]
- Basic idea: The constraints on the linear programming problem will be of the form
  \[ y_i \cdot w \cdot x_i > 0 \]
- We need to introduce two more steps to convert this into the standard format for a linear program
Converting to Standard LP Form

- **Step 1: Convert to equality constraints by using slack variables**
  - Introduce one slack variable $s_i$ for each training example $x_i$ and require that $s_i \geq 0$:
    $$y_i \cdot w \cdot x_i - s_i = 0$$

- **Step 2: Make all variables positive by subtracting pairs**
  - Replace each $w_j$ be a difference of two variables: $w_j = u_j - v_j$, where $u_j, v_j \geq 0$
    $$y_i \cdot (u - v) \cdot x_i - s_i = 0$$

- **Linear program:**
  - Find $s_i, u_j, v_j$
  - Minimize (no objective function)
  - Subject to:
    $$y_i (\sum_j (u_j - v_j)x_{ij}) - s_i = 0$$
    $$s_i \geq 0, u_j \geq 0, v_j \geq 0$$

- The linear program will have a solution iff the points are linearly separable
Example

- 30 random data points labeled according to the line $x_2 = 1 + x_1$
- Pink line is true classifier
- Blue line is the linear programming fit
What Happens with Non-Separable Data?

Bad News: Linear Program is Infeasible
Higher Dimensional Spaces

Theorem: For any data set, there exists a mapping $\Phi$ to a higher-dimensional space such that the data is linearly separable

$$\Phi(X) = (\phi_1(x), \phi_2(x), \ldots, \phi_D(x))$$

Example: Map to quadratic space

- $x = (x_1, x_2)$ (just two features)
- $\Phi(x) = (x_1^2, \sqrt{2} x_1 x_2, x_2^2, \sqrt{2} x_1, \sqrt{2} x_2, 1)$
- compute linear separator in this space
Drawback of this approach

- The number of features increases rapidly
- This makes the linear program much slower to solve
Kernel Trick

A dot product between two higher-dimensional mappings can sometimes be implemented by a kernel function

\[ K(x_i, x_j) = \Phi(x_i) \cdot \Phi(x_j) \]

Example: Quadratic Kernel

\[
K(x_i, x_j) = (x_i \cdot x_j + 1)^2 \\
= (x_{i1}x_{j1} + x_{i2}x_{j2} + 1)^2 \\
= x_{i1}^2x_{j1}^2 + 2x_{i1}x_{i2}x_{j1}x_{j2} + x_{i2}^2x_{j2}^2 + 2x_{i1}x_{j1} + 2x_{i2}x_{j2} + 1 \\
= (x_{i1}^2, \sqrt{2} x_{i1}x_{i2}, x_{i2}^2, \sqrt{2} x_{i1}, \sqrt{2} x_{i2}, 1) \cdot \\
(x_{j1}^2, \sqrt{2} x_{j1}x_{j2}, x_{j2}^2, \sqrt{2} x_{j1}, \sqrt{2} x_{j2}, 1) \\
= \Phi(x_i) \cdot \Phi(x_j)
\]
Idea

- Reformulate the LTU linear program so that it only involves dot products between pairs of training examples.
- Then we can use kernels to compute these dot products.
- Running time of algorithm will not depend on number of dimensions $D$ of high-dimensional space.
Reformulating the LTU Linear Program

Claim: In online Perceptron, \( w \) can be written as
\[
    w = \sum_j \alpha_j y_j x_j
\]

Proof:
- Each weight update has the form
  \[
  w_t := w_{t-1} + \eta \cdot g_{i,t}
  \]
- \( g_{i,t} \) is computed as
  \[
  g_{i,t} = \text{error}_{it} y_i x_i \quad \text{(error}_{it} = 1 \text{ if } x_i \text{ misclassified in iteration } t; 0 \text{ otherwise)}
  \]
- Hence
  \[
  w_t = w_0 + \sum_t \sum_i \eta \cdot \text{error}_{it} y_i x_i
  \]
- But \( w_0 = (0, 0, ..., 0) \), so
  \[
  w_t = \sum_t \sum_i \eta \cdot \text{error}_{it} y_i x_i
  \]
  \[
  w_t = \sum_i (\sum_t \eta \cdot \text{error}_{it}) y_i x_i
  \]
  \[
  w_t = \sum_i \alpha_i y_i x_i
  \]
Rewriting the Linear Separator Using Dot Products

\[ w \cdot x_i = \left( \sum_j \alpha_j y_j x_j \right) \cdot x_i \]

\[ = \sum_j \alpha_j y_j (x_j \cdot x_i) \]

**Change of variables**

– instead of optimizing \( w \), optimize \( \{\alpha_j\} \).

– Rewrite the constraint

\[ y_i w \cdot x_i > 0 \quad \text{as} \]
\[ y_i \sum_j \alpha_j y_j (x_j \cdot x_i) > 0 \quad \text{or} \]
\[ \sum_j \alpha_j y_j y_i (x_j \cdot x_i) > 0 \]
The Linear Program becomes

– Find \( \{ \alpha_j \} \)
– minimize (no objective function)
– subject to
  \[
  \sum_j \alpha_j y_j (x_j \cdot x_i) > 0
  \]
  \[
  \alpha_j \geq 0
  \]
– Notes:
  - The weight \( \alpha_j \) tells us how “important” example \( x_j \) is. If \( \alpha_j \) is non-zero, then \( x_j \) is called a “support vector”
  - To classify a new data point \( x \), we take its dot product with the support vectors
    \[
    \sum_j \alpha_j y_j (x_j \cdot x) > 0?
    \]
Kernel Version of Linear Program

– Find \( \{\alpha_j\} \)
– minimize (no objective function)
– subject to
\[
\sum_j \alpha_j y_j y_i K(x_j, x_i) > 0
\]
\[
\alpha_j \geq 0
\]

Classify new \( x \) according to
\[
\sum_j \alpha_j y_j K(x_j, x) > 0?
\]
Two support vectors (blue) with $\alpha_1 = 0.205$ and $\alpha_2 = 0.338$

Equivalent to the line $x_2 = -0.0974 + x_1 \times 1.341$
Solving the Non-Separable Case with Cubic Polynomial Kernel
Kernels

- Dot product
  \[ K(x_i, x_j) = x_i \cdot x_j \]

- Polynomial of degree \( d \)
  \[ K(x_i, x_j) = (x_i \cdot x_j + 1)^d \]

- Gaussian with scale \( \sigma \)
  \[ K(x_i, x_j) = \exp(-||x_i - x_j||^2/\sigma^2) \]

- Polynomials often give strange boundaries. Gaussians generally work well.
Gaussian kernel with $\sigma^2 = 4$

The gaussian kernel is equivalent to an infinite-dimensional feature space!
## Evaluation of SVMs

<table>
<thead>
<tr>
<th>Criterion</th>
<th>Perc</th>
<th>Logistic</th>
<th>LDA</th>
<th>Trees</th>
<th>Nets</th>
<th>NNbr</th>
<th>SVM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mixed data</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>Missing values</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>somewhat</td>
<td>no</td>
</tr>
<tr>
<td>Outliers</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Monotone transformations</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>somewhat</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>Scalability</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>Irrelevant inputs</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>somewhat</td>
<td>no</td>
<td>no</td>
<td>yes*</td>
</tr>
<tr>
<td>Linear combinations</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>somewhat</td>
<td>yes</td>
</tr>
<tr>
<td>Interpretable</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>yes**</td>
</tr>
<tr>
<td>Accurate</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
</tr>
</tbody>
</table>

* = dot product kernel with absolute value penalty

** = dot product kernel
Support Vector Machines Summary

Advantages of SVMs
- variable-sized hypothesis space
- polynomial-time exact optimization rather than approximate methods
  - unlike decision trees and neural networks
- Kernels allow very flexible hypotheses

Disadvantages of SVMs
- Must choose kernel and kernel parameters: Gaussian, \( \sigma \)
- Very large problems are computationally intractable
  - quadratic in number of examples
  - problems with more than 20,000 examples are very difficult to solve exactly
- Batch algorithm
SVMs Unify LTUs and Nearest Neighbor

With Gaussian kernel

- compute distance to a set of “support vector” nearest neighbors
- transform through a gaussian (nearer neighbors get bigger votes)
- take weighted sum of those distances for each class
- classify to the class with most votes