Support Vector Machine (SVM) Classification

Greg Grudic
Last Class

- Linear separating hyperplanes for binary classification
- Rosenblatt’s Perceptron Algorithm
  - Based on Gradient Descent
  - Convergence theoretically guaranteed if data is linearly separable
    - Infinite number of solutions
- For nonlinear data:
  - Mapping data into a nonlinear space where it is linearly separable (or almost)
  - However, convergence still not guaranteed…
Questions?
Why Classification?

Signals → Classification → Symbols

(The Grounding Problem)

Uncertainty

Not typically addressed in CS
The Problem Domain for Project Test 1: Identifying (and Navigating) Paths

Data → Construct a Classifier → Classifier

Non-path → Path

Data → Data

Image 1

Path labeled Image
Today’s Lecture Goals

• Support Vector Machine (SVM) Classification
  – Another algorithm for linear separating hyperplanes

A Good text on SVMs: Bernhard Schölkopf and Alex Smola. Learning with Kernels. MIT Press, Cambridge, MA, 2002
Support Vector Machine (SVM) Classification

- Classification as a problem of finding optimal (canonical) linear hyperplanes.
- Optimal Linear Separating Hyperplanes:
  - In Input Space
  - In Kernel Space
    - Can be non-linear
Linear Separating Hyper-Planes

How many lines can separate these points?

Which line should we use?
Initial Assumption:
Linearly Separable Data
Linear Separating Hyper-Planes

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

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

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

\[ y = -1 \]

\[ y = +1 \]
Linear Separating Hyper-Planes

• Given data: \((x_1, y_1), \ldots, (x_N, y_N)\)
• Finding a separating hyperplane can be posed as a constraint satisfaction problem (CSP):

\[
\forall i \in (1, \ldots, N), \quad \text{find } w \text{ and } b \text{ such that} \]

\[
w \cdot x_i + b \geq +1 \quad \text{if } \quad y_i = +1
\]

\[
w \cdot x_i + b \leq -1 \quad \text{if } \quad y_i = -1
\]

• Or, equivalently: \(y_i (w \cdot x_i + b) - 1 \geq 0, \quad \forall i\)

• *If data is linearly separable, there are an infinite number of hyperplanes that satisfy this CSP*
The Margin of a Classifier

• Take any hyper-plane (P0) that separates the data
• Put a parallel hyper-plane (P1) on a point in class 1 closest to P0
• Put a second parallel hyper-plane (P2) on a point in class -1 closest to P0
• The margin \( M \) is the perpendicular distance between P1 and P2
Calculating the Margin of a Classifier

- **P0**: Any separating hyperplane
- **P1**: Parallel to P0, passing through closest point in one class
- **P2**: Parallel to P0, passing through point closest to the opposite class

**Margin (M)**: distance measured along a line perpendicular to P1 and P2
SVM Constraints on the Model Parameters

Model parameters \((w, b)\) must be chosen such that, for \(x_1\) on P1 and for \(x_2\) on P2:

\[
\begin{align*}
\text{P1: } \mathbf{w} \cdot \mathbf{x}_1 + b &= -1 \\
\text{P2: } \mathbf{w} \cdot \mathbf{x}_2 + b &= +1
\end{align*}
\]

For any \(P_0\), these constraints are always attainable.

Given the above, then the linear separating boundary lies half way between P1 and P2 and is given by:

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

Resulting Classifier: \(\hat{y} = \text{sgn}(\mathbf{w} \cdot \mathbf{x} + b)\)
Remember: **signed** distance from a point to a hyperplane:

\[
d(x, \text{hyperplane}) = \frac{c + \mathbf{w} \cdot \mathbf{x}}{\sqrt{\sum_{i=1}^{d} w_i^2}} = \frac{c + \mathbf{w} \cdot \mathbf{x}}{\|\mathbf{w}\|}
\]

Hyperplane defined by: \((c, \mathbf{w})\)
Calculating the Margin (1)

\[ M = d(P1, x_2) = \frac{\mathbf{w} \cdot x_2 + b + 1}{\|\mathbf{w}\|} \]

\[ M = d(P2, x_1) = \frac{\mathbf{w} \cdot x_1 + b - 1}{\|\mathbf{w}\|} \]

\[ 0 = d(P1, x_1) = \frac{\mathbf{w} \cdot x_1 + b + 1}{\|\mathbf{w}\|} \]

\[ 0 = d(P2, x_1) = \frac{\mathbf{w} \cdot x_2 + b - 1}{\|\mathbf{w}\|} \]
Calculating the Margin (2)

\[ M = d(P_1, x_2) = d(P_2, x_1) = \frac{\mathbf{w} \cdot x_2 + b + 1}{\|\mathbf{w}\|} = \frac{\mathbf{w} \cdot x_1 + b - 1}{\|\mathbf{w}\|} \]

Therefore:

\[ \frac{\mathbf{w} \cdot x_2 + b + 1}{\|\mathbf{w}\|} = \frac{\mathbf{w} \cdot x_1 + b - 1}{\|\mathbf{w}\|} \implies \mathbf{w} \cdot x_2 = \mathbf{w} \cdot x_1 - 2 \]

Therefore:

\[ M = \frac{(\mathbf{w} \cdot x_1 - 2) + b + 1}{\|\mathbf{w}\|} = -2 + \frac{(\mathbf{w} \cdot x_1 + b + 1)}{\|\mathbf{w}\|} = -2 + (0) = -2 \]

Take absolute value to get the unsigned margin:

\[ M = \frac{2}{\|\mathbf{w}\|} \]
Different P0’s have Different Margins

- P0: Any separating hyperplane
- P1: Parallel to P0, passing through closest point in one class
- P2: Parallel to P0, passing through point closest to the opposite class

Margin (M): distance measured along a line perpendicular to P1 and P2.
Different P0’s have Different Margins

- **P0**: Any separating hyperplane
- **P1**: Parallel to P0, passing through closest point in one class
- **P2**: Parallel to P0, passing through point closest to the opposite class

**Margin (M)**: distance measured along a line perpendicular to P1 and P2
Different P0’s have Different Margins

- P0: Any separating hyperplane
- P1: Parallel to P0, passing through closest point in one class
- P2: Parallel to P0, passing through point closest to the opposite class

**Margin (M):** distance measured along a line perpendicular to P1 and P2
How Do SVMs Choose the Optimal Separating Hyperplane (boundary)?

- Find the \( \mathbf{w} \) that maximizes the margin!

**Margin**\((M)\): distance measured along a line perpendicular to \( P1 \) and \( P2 \)

\[
\text{margin } (M) = \frac{2}{\|\mathbf{w}\|}
\]
SVM: Constraint Optimization Problem

- Given data:

\[(x_1, y_1), \ldots, (x_N, y_N)\]

- Minimize \[\|w\|^2\] subject to:

\[y_i (w \cdot x_i + b) - 1 \geq 0, \quad \forall i = (1, \ldots, N)\]

The Lagrange Function Formulation is used to solve this Minimization Problem
The Lagrange Function Formulation

For every constraint we introduce a Lagrange Multiplier: \( \alpha_i \geq 0 \)

The Lagrangian is then defined by:

\[
L(w, b, \alpha) = \frac{1}{2} \|w\|^2 - \sum_{i=1}^{N} \alpha_i \left( y_i [w \cdot x_i + b] - 1 \right)
\]

Where - the primal variables are \((w, b)\)
- the dual variables are \((\alpha_1, ..., \alpha_N)\)

Goal: **Minimize** Lagrangian w.r.t. primal variables,
and **Maximize** w.r.t. dual variables
Derivation of the Dual Problem

- At the saddle point (extremum w.r.t. primal)
  \[
  \frac{\partial}{\partial b} L(w, b, \alpha) = 0, \quad \frac{\partial}{\partial w} L(w, b, \alpha) = 0
  \]
- This gives the conditions
  \[
  \sum_{i=1}^{N} \alpha_i y_i = 0, \quad w = \sum_{i=1}^{N} \alpha_i y_i x_i
  \]
- Substitute into \( L(w, b, \alpha) \) to get the dual problem
Using the Lagrange Function Formulation, we get the **Dual Problem**

- **Maximize**
  
  \[ W(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j) \]

- **Subject to**
  
  \[ \alpha_i \geq 0, \quad i = 1, \ldots, N \]
  
  \[ \sum_{i=1}^{N} \alpha_i y_i = 0 \]
Properties of the Dual Problem

• Solving the Dual gives a solution to the original constraint optimization problem
• For SVMs, the Dual problem is a Quadratic Optimization Problem which has a globally optimal solution
• Gives insights into the NON-Linear formulation for SVMs
Support Vector Expansion (1)

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

\[ y_i [w \cdot x_i + b] > 1 \Rightarrow \alpha_i = 0 \rightarrow x_i \text{ irrelevant} \]

OR

\[ y_i [w \cdot x_i + b] = 1 \text{ (On Margin)} \quad x_i \quad \text{Support Vector} \]

\[ b \text{ is also computed from the optimal dual variables } \alpha_i \]
Support Vector Expansion (2)

\[ f(x) = \text{sgn}(w \cdot x + b) \]

Substitute

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

OR

\[
    f(x) = \text{sgn}\left(\sum_{i=1}^{N} \alpha_i y_i (x_i \cdot x) + b\right)
\]
What are the Support Vectors?

Maximized Margin
Why do we want a model with only a few SVs?

- Leaving out an example that does not become an SV gives the same solution!
- **Theorem (Vapnik and Chervonenkis, 1974):** Let $N_{SV}$ be the number of SVs obtained by training on N examples randomly drawn from $P(X,Y)$, and $E$ be an expectation. Then

\[
E[\text{Prob(test error)}] \leq \frac{E[N_{SV}]}{N}
\]
What Happens When Data is Not Separable: **Soft Margin SVM**

Add a Slack Variable

\[ \xi_i = \begin{cases} 
0 & \text{if } x_i \text{ correctly classified} \\
\text{distance to margin} & \text{otherwise}
\end{cases} \]
Soft Margin SVM: Constraint Optimization Problem

• Given data:
  \[(x_1, y_1), \ldots, (x_N, y_N)\]

• Minimize \[\frac{1}{2} \|w\|^2 + C \sum_{i=1}^{N} \xi_i\] subject to:
  \[y_i (w \cdot x_i + b) \geq 1 - \xi_i, \quad \forall i = (1, \ldots, N)\]
Dual Problem (Non-separable data)

• Maximize

\[ W(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j (x_i \cdot x_j) \]

• Subject to

\[ 0 \leq \alpha_i \leq C, \quad i = 1, \ldots, N \]

\[ \sum_{i=1}^{N} \alpha_i y_i = 0 \]
Same Decision Boundary

\[ f(x) = \text{sgn} \left( \sum_{i=1}^{N} \alpha_i y_i (x_i \cdot x) + b \right) \]
Mapping into Nonlinear Space

\[ \Phi = \left( x_1^2, x_2^2 \right) \]

Goal: Data is linearly separable (or almost) in the nonlinear space.
Nonlinear SVMs

- **KEY IDEA:** Note that both the decision boundary and dual optimization formulation use dot products in input space only!

\[
f(x) = \text{sgn}\left( \sum_{i=1}^{N} \alpha_i y_i (x_i \cdot x) + b \right)
\]

\[
W(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j (x_i \cdot x)
\]
Kernel Trick

Replace \( \langle \mathbf{x}_i \cdot \mathbf{x}_j \rangle \)

with

\[
K(\mathbf{x}_i, \mathbf{x}_j) = \left\langle \Phi(\mathbf{x}_i), \Phi(\mathbf{x}_j) \right\rangle
\]

Can use the same algorithms in nonlinear kernel space!
Nonlinear SVMs

Maximize:

\[ W(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j K(x_i, x_j) \]

Boundary:

\[ f(x) = \text{sgn}\left( \sum_{i=1}^{N} \alpha_i y_i K(x_i, x) + b \right) \]
Need Mercer Kernels

\[ K(x_i, x_j) = \left\langle \Phi(x_i), \Phi(x_j) \right\rangle \]
\[ = \left\langle \Phi(x_j), \Phi(x_i) \right\rangle \]
\[ = K(x_j, x_i) \]
Gram (Kernel) Matrix

Training Data: \((x_1, y_1), \ldots, (x_N, y_N)\)

\[
K = \begin{pmatrix}
K(x_1, x_1) & \cdots & K(x_1, x_N) \\
\vdots & \ddots & \vdots \\
K(x_N, x_1) & \cdots & K(x_N, x_N)
\end{pmatrix}
\]

Properties:
- Positive Definite Matrix
- Symmetric
- Positive on diagonal
- \(N\) by \(N\)
Commonly Used Mercer Kernels

- Polynomial
\[ K(x_i, x_j) = \left((x_i \cdot x_j) + c\right)^d \]

- Sigmoid
\[ K(x_i, x_j) = \tanh\left(\kappa (x_i \cdot x_j) + \theta\right) \]

- Gaussian
\[ K(x_i, x_j) = \exp\left(-\frac{1}{2\sigma^2} \|x_i - x_j\|^2 \right) \]
Why these kernels?

• There are infinitely many kernels
  – The best kernel is data set dependent
  – We can only know which kernels are good by trying them and estimating error rates on future data

• Definition: a universal approximator is a mapping that can arbitrarily well model any surface (i.e. many to one mapping)

• Motivation for the most commonly used kernels
  – Polynomials (given enough terms) are universal approximators
    • However, Polynomial Kernels are not universal approximators because they cannot represent polynomial interactions
  – Sigmoid functions (given enough training examples) are also universal approximators
  – Gaussian Kernels (given enough training examples) are universal approximators
  – Also, these kernels have shown to produce good models in practice
Picking a Model
(A Kernel for SVMs)?

• How do you pick the Kernels?
  – Kernel parameters

• These are called **learning parameters** or **hyperparameters**
  – Two approaches choosing learning parameters
    • Bayesian
      – Learning parameters must maximize probability of correct classification on future data based on prior biases
    • Frequentist
      – Use the training data to learn the model parameters \( (\hat{\beta}_0, \hat{\beta}_1, ..., \hat{\beta}_d) \)
      – Use validation data to pick the best hyperparameters.

• More on learning parameter selection later
\[ m = N \text{ (number of training examples)} \]

**Soft Margin SVMs**

**C-SVM [15]:** for \( C > 0 \), minimize

\[
\tau(w, \xi) = \frac{1}{2} ||w||^2 + C \sum_{i=1}^{m} \xi_i
\]

subject to
\[ y_i \cdot (\langle w, x_i \rangle + b) \geq 1 - \xi_i, \quad \xi_i \geq 0 \text{ (margin } 2/||w||) \]

**\( \nu \)-SVM [55]:** for \( 0 \leq \nu < 1 \), minimize

\[
\tau(w, \xi, \rho) = \frac{1}{2} ||w||^2 - \nu \rho + \frac{1}{m} \sum \xi_i
\]

subject to
\[ y_i \cdot (\langle w, x_i \rangle + b) \geq \rho - \xi_i, \quad \xi_i \geq 0 \text{ (margin } 2\rho/||w||) \]

---

B. Schölkopf, Canberra, February 2002
Duals, Using Kernels

**C-SVM** dual: maximize

\[
W(\alpha) = \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j k(x_i, x_j)
\]

subject to \(0 \leq \alpha_i \leq C\), \(\sum_i \alpha_i y_i = 0\).

**\(\nu\)-SVM** dual: maximize

\[
W(\alpha) = -\frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j k(x_i, x_j)
\]

subject to \(0 \leq \alpha_i \leq \frac{1}{m}\), \(\sum_i \alpha_i y_i = 0\), \(\sum_i \alpha_i \geq \nu\).

In both cases: *decision function*:

\[
f(x) = \text{sgn} \left( \sum_{i=1}^m \alpha_i y_i k(x, x_i) + b \right)
\]

\(m = N\) (number of training examples)

B. Schölkopf, Canberra, February 2002

Greg Grudic

Intro AI
SVM Training  \( m = N \) (number of training examples)

- naive approach: the complexity of maximizing

\[
W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j k(x_i, x_j)
\]

scales with the third power of the training set size \( m \)

- only SVs are relevant \( \longrightarrow \) only compute \((k(x_i, x_j))_{ij}\) for SVs. Extract them iteratively by cycling through the training set in chunks [63].

- in fact, one can use chunks which do not even contain all SVs [42]. Maximize over these sub-problems, using your favorite optimizer.

- the extreme case: by making the sub-problems very small (just two points), one can solve them analytically [45].
Some SVM Software

• LIBSVM
  – http://www.csie.ntu.edu.tw/~cjlin/libsvm/

• SVM Light
  – http://svmlight.joachims.org/

• TinySVM
  – http://chasen.org/~taku/software/TinySVM/

• WEKA
  – http://www.cs.waikato.ac.nz/ml/weka/
  – Has many ML algorithm implementations in JAVA
MNIST: A SVM Success Story

• Handwritten character benchmark
  – 60,000 training and 10,000 testing
  – Dimension $d = 28 \times 28$
## Results on Test Data

<table>
<thead>
<tr>
<th>Classifier</th>
<th>test error</th>
</tr>
</thead>
<tbody>
<tr>
<td>linear classifier</td>
<td>8.4%</td>
</tr>
<tr>
<td>3-nearest-neighbour</td>
<td>2.4%</td>
</tr>
<tr>
<td>SVM</td>
<td>1.4%</td>
</tr>
<tr>
<td>Tangent distance</td>
<td>1.1%</td>
</tr>
<tr>
<td>LeNet4</td>
<td>1.1%</td>
</tr>
<tr>
<td>Boosted LeNet4</td>
<td>0.7%</td>
</tr>
<tr>
<td>Translation invariant SVM</td>
<td>0.56%</td>
</tr>
</tbody>
</table>

SVM used a polynomial kernel of degree 9.
**SVM (Kernel) Model Structure**

\[ f(x) = \text{sgn}(\sum_{i} \lambda_i k(x, x_i) + b) \]

- **Classification**
- **Weights**
- **Comparison:** e.g.
  
  \[ k(x, x_i) = (x \cdot x_i)^d \]
  
  \[ k(x, x_i) = \exp(-\|x - x_i\|^2 / c) \]
  
  \[ k(x, x_i) = \tanh(\kappa(x \cdot x_i) + \theta) \]

- **Support Vectors:** \( x_1 \ldots x_4 \)

- **Input Vector:** \( x \)