Support vector machines

Milos Hauskrecht
milos@cs.pitt.edu
5329 Sennott Square

Linearly separable classes

There is a **hyperplane** that separates training instances with no error

**Hyperplane:**
\[ w^T x + w_0 = 0 \]

<table>
<thead>
<tr>
<th>Class</th>
<th>[ w^T x + w_0 ]</th>
</tr>
</thead>
<tbody>
<tr>
<td>(+1)</td>
<td>&gt; 0</td>
</tr>
<tr>
<td>(-1)</td>
<td>&lt; 0</td>
</tr>
</tbody>
</table>
Algorithms for linearly separable sets

• **Separating hyperplane**  \( \mathbf{w}^T \mathbf{x} + w_0 = 0 \)

• We can use **gradient methods** or Newton Rhapsod for sigmoidal switching functions and learn the weights

• Recall that we learn the linear decision boundary

---

Algorithms for linearly separable sets

• **Linear program solution:**
  – Find weights that satisfy the following constraints:
    \[
    \begin{align*}
    \mathbf{w}^T \mathbf{x}_i + w_0 &\geq 0 & \text{For all } i, \text{ such that } y_i = +1 \\
    \mathbf{w}^T \mathbf{x}_i + w_0 &\leq 0 & \text{For all } i, \text{ such that } y_i = -1
    \end{align*}
    \]

  **Property:** if there is a hyperplane separating the examples, the linear program finds the solution

• **Other methods:**
  - Fisher linear discriminant
  - Perceptron algorithm
Optimal separating hyperplane

- There are multiple hyperplanes that separate the data points
  - Which one to choose?
- **Maximum margin** choice: the maximum distance of \( d_+ + d_- \)
  - where \( d_+ \) is the shortest distance of a positive example from the hyperplane (similarly \( d_- \) for negative examples)

Maximum margin hyperplane

- For the maximum margin hyperplane only examples on the margin matter (only these affect the distances)
- These are called **support vectors**
Finding maximum margin hyperplanes

- **Assume** that examples in the training set are \((x_i, y_i)\) such that \(y_i \in \{+1, -1\}\)
- **Assume** that all data satisfy:
  \[
  w^T x_i + w_0 \geq 1 \quad \text{for} \quad y_i = +1 \\
  w^T x_i + w_0 \leq -1 \quad \text{for} \quad y_i = -1
  \]
- The inequalities can be combined as:
  \[
  y_i (w^T x_i + w_0) - 1 \geq 0 \quad \text{for all} \quad i
  \]
- Equalities define two hyperplanes:
  \[
  w^T x_i + w_0 = 1 \quad \text{and} \quad w^T x_i + w_0 = -1
  \]

Finding the maximum margin hyperplane

- **Geometrical margin:** \(\rho_{w, w_0}(x, y) = y(w^T x + w_0) / \|w\|\)
  - measures the distance of a point \(x\) from the hyperplane
  - \(w\) - normal to the hyperplane
  - \(\|\cdot\|\) - Euclidean norm
  
  For points satisfying:
  \[
  y_i (w^T x_i + w_0) - 1 = 0
  \]
  
  The distance is
  \[
  \frac{1}{\|w\|}
  \]

  **Width of the margin:**
  \[
  d_+ + d_- = \frac{2}{\|w\|}
  \]
Maximum margin hyperplane

• **We want to maximize** \( d_+ + d_- = \frac{2}{\|w\|} \)

• We do it by minimizing

\[
\|w\|^2 / 2 = w^T w / 2
\]

\( w, w_0 \) - variables

– But we also need to enforce the constraints on points:

\[
[y_i (w^T x + w_0) - 1] \geq 0
\]

---

Maximum margin hyperplane

• **Solution**: Incorporate constraints into the optimization

• **Optimization problem** (Lagrangian)

\[
J(w, w_0, \alpha) = \|w\|^2 / 2 - \sum_{i=1}^{n} \alpha_i [y_i (w^T x + w_0) - 1]
\]

\( \alpha_i \geq 0 \) - Lagrange multipliers

• **Minimize** with regard to \( w, w_0 \) (primal variables)

• **Derivatives of L** with regard to \( \alpha \) should vanish at the solution

Lagrange multipliers enforce the satisfaction of constraints

If \( [y_i (w^T x + w_0) - 1] > 0 \) \( \iff \alpha_i \to 0 \)

Else \( \iff \alpha_i > 0 \) Active constraint
Maximum margin hyperplane

- **Solution:** Incorporate constraints into the optimization
- **Optimization problem** (Lagrangian)

\[
J(w, w_0, \alpha) = \|w\|^2 / 2 - \sum_{i=1}^{n} \alpha_i [y_i (w^T x_i + w_0) - 1]
\]

\[\alpha_i \geq 0 \quad \text{Lagrange multipliers}\]

- **Dual formulation (Wolfe dual):**
  - **Maximize** with regard to \(\alpha\) (dual variables)
  - **Derivatives of** \(L\) **with regard to** \(w, w_0\) **should vanish**

---

Max margin hyperplane solution

- Set derivatives to 0 (Kuhn-Tucker conditions)

\[
\nabla_w J(w, w_0, \alpha) = w - \sum_{i=1}^{n} \alpha_i y_i x_i = 0
\]

\[
\frac{\partial J(w, w_0, \alpha)}{\partial w_0} = -\sum_{i=1}^{n} \alpha_i y_i = 0
\]

- Now we need to solve for Lagrange parameters only

\[
J(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j (x_i^T x_j) \quad \text{maximize}
\]

Subject to constraints

\[\alpha_i \geq 0 \quad \text{for all } i, \quad \text{and} \quad \sum_{i=1}^{n} \alpha_i y_i = 0\]

- **Quadratic optimization problem:** solution \(\hat{\alpha}_i\) for all \(i\)
Maximum hyperplane solution

- The resulting parameter vector $\hat{w}$ can be expressed as:
  $$\hat{w} = \sum_{i=1}^{n} \hat{\alpha}_i y_i x_i$$
  $\hat{\alpha}_i$ is the solution of the dual problem

- The parameter $w_0$ is obtained through Karush-Kuhn-Tucker conditions
  $$\hat{\alpha}_i [y_i (\hat{w} x_i + w_0) - 1] = 0$$

Solution properties

- $\hat{\alpha}_i = 0$ for all points that are not on the margin
- $\hat{w}$ is a linear combination of support vectors only

- The decision boundary:
  $$\hat{w}^T x + w_0 = \sum_{i \in SV} \hat{\alpha}_i y_i (x_i^T x) + w_0 = 0$$

Support vector machines

- The decision boundary:
  $$\hat{w}^T x + w_0 = \sum_{i \in SV} \hat{\alpha}_i y_i (x_i^T x) + w_0$$

- The decision:
  $$\hat{y} = \text{sign} \left( \sum_{i \in SV} \hat{\alpha}_i y_i (x_i^T x) + w_0 \right)$$
Support vector machines

- The decision boundary:
  \[ \hat{w}^T x + w_0 = \sum_{i \in SV} \hat{\alpha}_i y_i (x_i^T x) + w_0 \]

- The decision:
  \[ \hat{y} = \text{sign} \left[ \sum_{i \in SV} \hat{\alpha}_i y_i (x_i^T x) + w_0 \right] \]

- (!!):
  - Decision on a new \( x \) requires to compute the inner product between the examples \( (x_i^T x) \)
  - Similarly, the optimization depends on \( (x_i^T x_j) \)
  \[ J(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j (x_i^T x_j) \]

Extension to a linearly non-separable case

- Idea: Allow some flexibility on crossing the separating hyperplane
Extension to the linearly non-separable case

• Relax constraints with variables $\xi_i \geq 0$
  \[ w^T x_i + w_0 \geq 1 - \xi_i \quad \text{for} \quad y_i = +1 \]
  \[ w^T x_i + w_0 \leq -1 + \xi_i \quad \text{for} \quad y_i = -1 \]

• Error occurs if $\xi_i \geq 1$, $\sum_{i=1}^{n} \xi_i$ is the upper bound on the number of errors

• Introduce a penalty for the errors
  \[ \text{minimize} \quad \|w\|^2 / 2 + C \sum_{i=1}^{n} \xi_i \]

Subject to constraints

$C$ – set by a user, larger $C$ leads to a larger penalty for an error

Extension to linearly non-separable case

• Lagrange multiplier form (primal problem)
  \[ J(w, w_0, \alpha) = \|w\|^2 / 2 + C \sum_{i=1}^{n} \xi_i - \sum_{i=1}^{n} \alpha_i \left[ y_i (w^T x + w_0) - 1 + \xi_i \right] - \sum_{i=1}^{n} \mu_i \xi_i \]

• Dual form after $w, w_0$ are expressed ($\xi_i$ s cancel out)
  \[ J(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j (x_i^T x_j) \]

Subject to: $0 \leq \alpha_i \leq C$ for all $i$, and $\sum_{i=1}^{n} \alpha_i y_i = 0$

Solution: $\hat{w} = \sum_{i=1}^{n} \hat{\alpha}_i y_i x_i$

The difference from the separable case: $0 \leq \alpha_i \leq C$

The parameter $w_0$ is obtained through KKT conditions
Support vector machines

• The decision boundary:
\[ \hat{w}^T x + w_0 = \sum_{i \in SV} \hat{\alpha}_i y_i \phi(x_i^T x) + w_0 \]

• The decision:
\[ \hat{y} = \text{sign} \left[ \sum_{i \in SV} \hat{\alpha}_i y_i \phi(x_i^T x) + w_0 \right] \]

• (!!):
  • Decision on a new \( x \) requires to compute the inner product between the examples \( x_i^T x \)
  • Similarly, the optimization depends on \( x_i^T x_j \)
\[ J(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j (x_i^T x_j) \]

Nonlinear case

• The linear case requires to compute \( x_i^T x \)
• The non-linear case can be handled by using a set of features. Essentially we map input vectors to (larger) feature vectors
\[ x \rightarrow \phi(x) \]
• It is possible to use SVM formalism on feature vectors
\[ \phi(x)^T \phi(x') \]
• Kernel function
\[ K(x, x') = \phi(x)^T \phi(x') \]

• Crucial idea: If we choose the kernel function wisely we can compute linear separation in the feature space implicitly such that we keep working in the original input space !!!!
Kernel function example

- Assume \( \mathbf{x} = [x_1, x_2]^T \) and a feature mapping that maps the input into a quadratic feature set
  \[
  \mathbf{x} \rightarrow \varphi(\mathbf{x}) = [x_1^2, x_2^2, \sqrt{2}x_1x_2, \sqrt{2}x_1, \sqrt{2}x_2, 1]^T
  \]
- Kernel function for the feature space:
  \[
  K(\mathbf{x}', \mathbf{x}) = \varphi(\mathbf{x}')^T \varphi(\mathbf{x})
  \]
  \[
  = x_1^2x_1'^2+x_2^2x_2'^2+2x_1x_2x_1'x_2'+2x_1x_1'+2x_2x_2'+1
  \]
  \[
  = (x_1x_1'+x_2x_2'+1)^2
  \]
  \[
  = (1+(\mathbf{x}^T\mathbf{x}')^2
  \]
- The computation of the linear separation in the higher dimensional space is performed implicitly in the original input space
**Kernel functions**

- **Linear kernel**
  
  \[ K(x, x') = x^T x' \]

- **Polynomial kernel**
  
  \[ K(x, x') = \left[ 1 + x^T x' \right]^k \]

- **Radial basis kernel**
  
  \[ K(x, x') = \exp\left[-\frac{1}{2} \|x - x'\|^2\right] \]

---

**Kernels**

- The dot product \( x^T x \) is a **distance measure**
- **Kernels** can be seen as distance measures
  - Or conversely express degree of similarity
- Design criteria - we want kernels to be
  - valid – Satisfy Mercer condition of positive semi-definiteness
  - good – embody the “true similarity” between objects
  - appropriate – generalize well
  - efficient – the computation of \( k(x,x') \) is feasible
Kernels

• SVM researchers have proposed kernels for comparison of variety of objects:
  – Strings
  – Trees
  – Graphs
• Cool thing:
  – SVM algorithm can be now applied to classify a variety of objects

Support vector machine SVM

• SVM maximize the margin around the separating hyperplane.
• The decision function is fully specified by a subset of the training data, the support vectors.
Support vector machine for regression

- **Regression** = find a function that fits the data.
- A data point may be wrong due to the noise
- **Idea:** Error from points which are close should count as a valid noise
- Line should be influenced by the real data not the noise.

Linear model

- **Training data:**
  \[ \{(x_1, y_1), \ldots, (x_i, y_i)\}, \quad x \in \mathbb{R}^n, \quad y \in \mathbb{R} \]
- Our goal is to find a function \( f(x) \) that has at most \( \varepsilon \) deviation from the actually obtained target for all the training data.
  \[ f(x) = w^T x + b = \langle w, x \rangle + b \]
**Linear model**

**Linear function:**

\[ f(x) = w^Tx + b = \langle w, x \rangle + b \]

We want a function that is:

- **flat:** means that one seeks small \( w \)
- all data points are within its \( \varepsilon \) neighborhood

The problem can be formulated as a **convex optimization problem**:

\[
\begin{align*}
\text{minimize} & \quad \frac{1}{2} \|w\|^2 \\
\text{subject to} & \quad \begin{cases} 
  y_i - \langle w_i, x_i \rangle - b \leq \varepsilon \\
  \langle w_i, x_i \rangle + b - y_i \leq \varepsilon
\end{cases}
\end{align*}
\]

All data points are assumed to be in the \( \varepsilon \) neighborhood

---

**Linear model**

- **Real data:** not all data points always fall into the \( \varepsilon \) neighborhood

\[ f(x) = w^Tx + b = \langle w, x \rangle + b \]

- **Idea:** penalize points that fall outside the \( \varepsilon \) neighborhood
Linear model

Linear function:

\[ f(x) = w^T x + b = \langle w, x \rangle + b \]

**Idea:** penalize points that fall outside the \( \varepsilon \) neighborhood

\[
\begin{align*}
\text{minimize} & \quad \frac{1}{2} \|w\|^2 + C \sum_{i=1}^{l} (\xi_i + \xi_i^*) \\
\text{subject to} & \quad \begin{cases} 
    y_i - \langle w_i, x_i \rangle - b \leq \varepsilon + \xi_i \\
    \langle w_i, x_i \rangle + b - y_i \leq \varepsilon + \xi_i^* \\
    \xi_i, \xi_i^* \geq 0
\end{cases}
\end{align*}
\]

**\( \varepsilon \)-intensive loss function**

\[
|\xi|_\varepsilon = \begin{cases} 
0 & \text{for } |\xi| \leq \varepsilon \\
|\xi| - \varepsilon & \text{otherwise}
\end{cases}
\]
Optimization

Lagrangian that solves the optimization problem

\[ L = \frac{1}{2} \langle w, w \rangle + C \sum_{i=1}^{l} (\xi_i + \xi_i^*) \]
\[ - \sum_{i=1}^{l} a_i (\varepsilon - \xi_i - y_i + \langle w, x_i \rangle + b) - \sum_{i=1}^{l} a_i^* (\varepsilon + \xi_i^* + y_i - \langle w, x_i \rangle - b) \]
\[ - \sum_{i=1}^{l} (\eta_i \xi_i + \eta_i^* \xi_i^*) \]

Subject to \( a_i, a_i^*, \eta_i, \eta_i^* \geq 0 \)

Primal variables \( w, b, \xi_i, \xi_i^* \)

Optimization

Derivatives with respect to primal variables

\[ \frac{\partial L}{\partial b} = \sum_{i=1}^{l} (a_i^* - a_i) = 0 \]
\[ \frac{\partial L}{\partial w} = w - \sum_{i=1}^{l} (a_i^* - a_i) x_i = 0 \]
\[ \frac{\partial L}{\partial \xi_i^*} = C - a_i^{(*)} - \eta_i^{(*)} = 0 \]
\[ \frac{\partial L}{\partial \xi_i} = C - a_i - \eta_i = 0 \]
Optimization

\[ L = \frac{1}{2} \langle w, w \rangle + \sum_{i=1}^l C \xi_i + \sum_{i=1}^l C \xi_i^* \]

\[ - \sum_{i=1}^l a_i \varepsilon - \sum_{i=1}^l a_i \xi_i - \sum_{i=1}^l a_i y_i - \sum_{i=1}^l a_i \langle \omega, x_i \rangle + \sum_{i=1}^l a_i b \]

\[ - \sum_{i=1}^l a_i^* \varepsilon - \sum_{i=1}^l a_i^* \xi_i - \sum_{i=1}^l a_i^* y_i + \sum_{i=1}^l a_i^* \langle \omega, x_i \rangle + \sum_{i=1}^l a_i^* b \]

\[ - \sum_{i=1}^l \eta_i \xi_i - \sum_{i=1}^l \eta_i^* \xi_i^* \]
Optimization

\[ L = -\frac{1}{2} \langle w, w \rangle - \sum_{i=1}^{l} (a_i + a_i^*) e - \sum_{i=1}^{l} (a_i + a_i^*) y_i \]

Maximize the dual

\[ L(a, a^*) = -\frac{1}{2} \sum_{i=1}^{l} (a_i - a_i^*)(a_j - a_j^*) \langle x_i, x_j \rangle - \sum_{i=1}^{l} (a_i + a_i^*) e - \sum_{i=1}^{l} (a_i + a_i^*) y_i \]

subject to

\[ \begin{cases} \sum_{i=1}^{l} (a_i - a_i^*) = 0 \\ a_i, a_i^* \in [0, C] \end{cases} \]

Solution

\[ \frac{\partial L}{\partial w} = w - \sum_{i=1}^{l} (a_i^* - a_i) x_i = 0 \]

\[ w = \sum_{i=1}^{l} (a_i - a_i^*) x_i \]

We can get:

\[ f(x) = \sum_{i=1}^{l} (a_i - a_i^*) \langle x_i, x \rangle + b \]

at the optimal solution the Lagrange multipliers are non-zero only for points outside the ε band.
Kernel trick

- Replace the inner product with a kernel
- A well chosen kernel leads to efficient computation