• SVMs for regression
• Non-parametric/instance based classification method

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

Soft-margin SVM

• Allows some flexibility on crossing the separating hyperplane
**Soft-margin SVM**

\[
\begin{align*}
\text{minimize} & \quad \|w\|^2 / 2 + C \sum_{i=1}^{n} \xi_i \\
\text{subject to} & \quad w^T x_i + w_0 \geq 1 - \xi_i \quad \text{for} \quad y_i = +1 \\
& \quad w^T x_i + w_0 \leq -1 + \xi_i \quad \text{for} \quad y_i = -1 \\
& \quad \xi_i \geq 0
\end{align*}
\]

- Rewrite \( \xi_i = \max \left[ 0, \ 1 - y_i (w^T x_i + w_0) \right] \) in \( \|w\|^2 / 2 + C \sum_{i=1}^{n} \xi_i \).

\[
\|w\|^2 / 2 + C \sum_{i=1}^{n} \max \left[ 0, \ 1 - y_i (w^T x_i + w_0) \right]
\]

**Classification learning**

- **General form:**
  \[
  \min_{w} \quad L(w, D) + \lambda Q(w)
  \]
  
  \[
  \begin{array}{ll}
  \text{Loss} & \quad \text{Regularization} \\
  \text{function} & \quad \text{penalty}
  \end{array}
  \]

- **Loss functions:**
  - Negative loglikelihood (used in the LR)
  - Hinge loss (used in SVM)

- **Regularization terms:**
  - L1 (lasso)
  - L2 (ridge)
Support vector machines

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

- **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 \( (x_i^T x) \)

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

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 \( x \to \phi(x) \)
- 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 \( x = [x_1, x_2]^T \) and a feature mapping that maps the input into a quadratic feature set

\[
x \rightarrow \phi(x) = [x_1^2, x_2^2, \sqrt{2}x_1x_2, \sqrt{2}x_1, \sqrt{2}x_2, 1]^T
\]

• Kernel function for the feature space:

\[
K(x', x) = \phi(x')^T \phi(x)
\]

\[
= x_1'^2x_1^2 + x_2'^2x_2^2 + 2x_1x_2x_1'x_2' + 2x_1x_1' + 2x_2x_2' + 1
\]

\[
= (x_1x_1' + x_2x_2' + 1)^2
\]

\[
= (1 + (x^T x'))^2
\]

• The computation of the linear separation in the higher dimensional space is performed implicitly in the original input space

Nonlinear extension

Kernel trick

• Replace the inner product with a kernel

• A well chosen kernel leads to an efficient computation
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

- **Kernels** define a similarity measure:
  - define a distance in between two objects
- **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
- NP-hard problems abound with graphs
Kernels

- Research 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 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:
  \[ \{(x_1, y_1), \ldots, (x_n, y_n)\}, \quad x \in \mathbb{R}^n, \ y \in \mathbb{R} \]
- 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 function:

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

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:

\[
\text{minimize} \quad \frac{1}{2} \|w\|^2
\]

subject to

\[
\begin{align*}
  y_i - \langle w, x_i \rangle - b & \leq \varepsilon \\
  \langle w, x_i \rangle + b - y_i & \leq \varepsilon
\end{align*}
\]

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
  \[
  f(x) = w^T x + b = \langle w, x \rangle + b
  \]
- Idea: penalize points that fall outside the \( \varepsilon \) neighborhood

Linear function:

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

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

\(\varepsilon\)-intensive loss function

---

**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**

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

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

\[
\frac{\partial L}{\partial \xi_i^{(*)}} = C - a_i^{(*)} - \eta_i^{(*)} = 0
\]

\[
\frac{\partial L}{\partial \eta_i} = C - a_i - \eta_i = 0
\]

---

### 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

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

subject to:

\[ \sum_{i=1}^{l} (a_i - a_i^*) = 0 \]
\[ a_i, a_i^* \in [0, C] \]
### SVM solution

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

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

We can get:

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

at the optimal solution the Lagrange multipliers are non-zero only for points outside the \( \varepsilon \) band.

---

### Nonparametric vs Parametric Methods

**Nonparametric models:**
- More flexibility – no parametric model is needed
- But require storing the entire dataset
- and the computation is performed with all data examples.

**Parametric models:**
- Once fitted, only parameters need to be stored
- They are much more efficient in terms of computation
- But the model needs to be picked in advance

---

CS 2750 Machine Learning
Non-parametric Classification methods

• Given a data set with \( N_k \) data points from class \( C_k \) and \( \sum_k N_k = N \), we have
  \[
p(x) = \frac{K}{NV}
\]
  \( K \)
• and correspondingly
  \[
p(x|C_k) = \frac{K_k}{N_k V}.
\]

• Since \( p(C_k) = N_k/N \), Bayes’ theorem gives
  \[
p(C_k|x) = \frac{p(x|C_k)p(C_k)}{p(x)} = \frac{K_k}{K}.
\]

K-Nearest-Neighbours for Classification
Nonparametric kernel-based classification

- **Kernel function:** $k(x, x')$
  - Models similarity between $x$, $x'$
  - **Example:** Gaussian kernel we used in the kernel density estimation
    
    $$
    k(x, x') = \frac{1}{(2\pi h^2)^{D/2}} \exp\left(-\frac{(x - x')^2}{2h^2}\right)
    $$

    $$
    p(x) = \frac{1}{N} \sum_{i=1}^{N} k(x, x_i)
    $$

- **Kernel for classification**
  
  $$
  p(y = C_k \mid x) = \frac{\sum_{x' : y' = C_k} k(x, x')}{\sum_{x'} k(x, x')}
  $$