SVMs / Kernel Machines

Helen Yannakoudakis and Oana Cocarascu

Department of Informatics
King's College London

Support Vector Machines

- Easier to convert into a quadratic optimisation problem with linear constraints

$$\min_{\mathbf{w},b} \frac{1}{2}||\mathbf{w}||^2$$

such that

$$y_i(\mathbf{w}^T\mathbf{x}_i + b) \geqslant 1, i = 1, \ldots, m$$

- Objective function + inequality constraints → constrained optimisation problem

Support Vector Machines

- Such problems are "dealt with" using Lagrange multipliers $\alpha_i \geqslant 0$ and a Lagrangian:

$$L(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}||\mathbf{w}||^2 - \sum_{i=1}^{m} \alpha_i (y_i(\mathbf{w}^T \mathbf{x}_i + b) - 1)$$

$$L(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}||\mathbf{w}||^2 - \sum_{i=1}^{m} \alpha_i y_i(\mathbf{w}^T \mathbf{x}_i + b) + \sum_{i=1}^{m} \alpha_i$$

- The minimum is found by taking its partial derivatives and setting them equal to zero:

$$\frac{\partial L}{\partial \mathbf{w}} = \mathbf{w} - \sum_i \alpha_i y_i \mathbf{x}_i = 0$$

$$\frac{\partial L}{\partial b} = -\sum_i \alpha_i y_i = 0$$

Support Vector Machines

- Such problems are "dealt with" using Lagrange multipliers $\alpha_i \geqslant 0$ and a Lagrangian:

$$L(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}||\mathbf{w}||^2 - \sum_{i=1}^{m} \alpha_i y_i(\mathbf{w}^T \mathbf{x}_i + b) + \sum_{i=1}^{m} \alpha_i$$

- The minimum is found by taking its partial derivatives and setting them equal to zero:

$$\frac{\partial L}{\partial \mathbf{w}} = \mathbf{w} - \sum_i \alpha_i y_i \mathbf{x}_i = 0$$

$$\frac{\partial L}{\partial b} = -\sum_i \alpha_i y_i = 0$$

- Substituting these (and simplifying) we get a dual representation:

$$\arg\max_{\alpha} \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j)$$

Decision function

- Linear classifier has the form $y = \mathbf{w}^T\mathbf{x} + b$
- Remember that for a specific $\mathbf{x}$ we want to know whether it is $+1$ or a -1.
- We can stay in the dual representation.
- If we substitute, the decision function can be written as:

$$h(\mathbf{x}) = sign\left(\sum_i \alpha_i y_i (\mathbf{x} \cdot \mathbf{x}_i) + b\right)$$

- We only need to do this for the $\mathbf{x}_i$ that are support vectors.

Decision function

- If we substitute, the decision function can be written as:

$$h(\mathbf{x}) = sign\left(\sum_i \alpha_i y_i (\mathbf{x} \cdot \mathbf{x}_i) + b\right)$$

- We only need to do this for the $\mathbf{x}_i$ that are support vectors.
- How about b?
- Any support vector $\mathbf{x}$ satisfies:

$$y_i(\mathbf{w}^T \mathbf{x}_i + b) = 1$$

$$y_i\left(\sum_i \alpha_i y_i (\mathbf{x} \cdot \mathbf{x}_i) + b\right) = 1$$

Decision function

- How about b?
- Any support vector $\mathbf{x}$ satisfies:

$$y_i \left(\sum_i \alpha_i y_i (\mathbf{x} \cdot \mathbf{x}_i) + b \right) = 1$$

- Multiply through by y_i and use $y_i^2 = 1$
- Average these equations over all support vectors and solve for b:

$$b = \frac{1}{N} \sum_j \left(y_i - \sum_i \alpha_i y_i (\mathbf{x} \cdot \mathbf{x}_i) \right)$$

where N is the total number of support vectors.

Soft margin classifiers

- We do this by allowing slack variables:

$$\xi_i \geqslant 0$$

  for all $i = 1 \ldots m$

- Then, instead of looking for

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geqslant 1$$

  for all i in the original weight/feature space, we look for:

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geqslant 1 - \xi_i$$

  for all i.

- Points for which $\xi = 0$ are correctly classified; $0 < \xi \leqslant 1$ lie inside the margin; $\xi > 1$ lie on the wrong side of the boundary and are misclassified.

Soft margin classifiers

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

- The constant $C > 0$ determines the amount of slack that is allowed.
- This controls the trade-off between margin maximization
  - which needs more slack

  and minimizing training errors
  - which needs less slack
- Typically have to search to find the optimal C for a given problem.

Soft margin classifiers

- We end up solving:

$$\arg\max_{\alpha} \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$$

subject to the additional constraints:

$$0 \leqslant \alpha_i \leqslant C$$

for all $i = 1, \ldots, m$ and:

$$\sum_{i=1}^{m} \alpha_i y_i = 0$$

Soft margin classifiers

- The only difference with the separable case is the upper bound C on the α_i.
- This limits the influence of individual data points.
- The intuition is that this prevents outliers from having a big effect on the separator.

Kernels

- SVM can be re-cast into a dual representation based on the dot product between our feature vectors
- Replace the scalar product with a kernel
- We can use a kernel function $K(\mathbf{x}, \mathbf{z})$
- $K(\mathbf{x}, \mathbf{z}) = F(\mathbf{x}) \cdot F(\mathbf{z})$
- e.g. for quadratic, $K(\mathbf{x}, \mathbf{z}) = (\mathbf{x} \cdot \mathbf{z})^2$
- Since $\mathbf{x} \cdot \mathbf{z}$ is a scalar, once we have computed the dot product, just have to do the kernel computation on this one number.
- We've computed the inner product in the feature space $\mathbb{R}^3$ without ever having to transform data into $\mathbb{R}^3$!

Kernels

- How can we construct kernels?
- Choose a feature mapping $F(\mathbf{x})$ and use this to find your kernel
- Or construct kernel functions directly
- Ensure the function is a valid kernel

$$K(\mathbf{x}, \mathbf{z}) = (\mathbf{x}^\intercal \mathbf{z})^2$$

$$K(\mathbf{x}, \mathbf{z}) = (x_1 z_1 + x_2 z_2)^2 = x_1^2 z_1^2 + 2 x_1 z_1 x_2 z_2 + x_2^2 z_2^2 =$$

$$= (x_1^2, \sqrt{2} x_1 x_2, x_2^2)(z_1^2, \sqrt{2} z_1 z_2, z_2^2)^\intercal$$

$$= F(\mathbf{x})^\intercal F(\mathbf{z})$$

Kernels

$$
\begin{aligned}
k(\mathbf{x}, \mathbf{x}') &= c k_1(\mathbf{x}, \mathbf{x}') \\
k(\mathbf{x}, \mathbf{x}') &= f(\mathbf{x}) k_1(\mathbf{x}, \mathbf{x}') f(\mathbf{x}') \\
k(\mathbf{x}, \mathbf{x}') &= q\left(k_1(\mathbf{x}, \mathbf{x}')\right) \\
k(\mathbf{x}, \mathbf{x}') &= \exp\left(k_1(\mathbf{x}, \mathbf{x}')\right) \\
k(\mathbf{x}, \mathbf{x}') &= k_1(\mathbf{x}, \mathbf{x}') + k_2(\mathbf{x}, \mathbf{x}') \\
k(\mathbf{x}, \mathbf{x}') &= k_1(\mathbf{x}, \mathbf{x}') k_2(\mathbf{x}, \mathbf{x}') \\
k(\mathbf{x}, \mathbf{x}') &= k_3\left(\boldsymbol{\phi}(\mathbf{x}), \boldsymbol{\phi}(\mathbf{x}')\right) \\
k(\mathbf{x}, \mathbf{x}') &= \mathbf{x}^{\mathrm{T}} \mathbf{A} \mathbf{x}' \\
k(\mathbf{x}, \mathbf{x}') &= k_a(\mathbf{x}_a, \mathbf{x}'_a) + k_b(\mathbf{x}_b, \mathbf{x}'_b) \\
k(\mathbf{x}, \mathbf{x}') &= k_a(\mathbf{x}_a, \mathbf{x}'_a) k_b(\mathbf{x}_b, \mathbf{x}'_b)
\end{aligned}
$$

KING'S College LONDON

References

Christopher M. Bishop. Pattern Recognition and Machine Learning. Springer. 2006