COMS30035, Machine learning: Kernels 2

James Cussens
james.cussens@bristol.ac.uk

Department of Computer Science, SCEEM
University of Bristol

September 28, 2020

Agenda

- Max margin classification
- Support vector machines

Recap

- In the first Kernels lecture we saw an example of:
  1. learning: $\mathbf{a} = (\mathbf{K} + \lambda \mathbf{I}_N)^{-1}\mathbf{t}$
  2. and computing predicted values:
     $y(\mathbf{x}) = a_1 k(\mathbf{x}_1, \mathbf{x}) + a_2 k(\mathbf{x}_2, \mathbf{x}) + a_3 k(\mathbf{x}_3, \mathbf{x})$

  where both were done using only a kernels
- But for learning we needed to compute the kernel value for every pair of training datapoints, and for prediction we needed the entire training set.
- *Support vector machines* are a kernel-based method for classification which avoids this excessive computation.
- We still also need to address the question of which kernel function to use, more on this later . . .

Linear classification

Consider a simple linear model for two class classification:

$$y(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x}) + b \tag{1}$$

where the *bias b* has been made explicit and where the class label is either -1 or 1.

- Let's assume (rather optimistically!) that the training dataset is linearly separable, so there is some **w** and b such that $y(\mathbf{x}_n) > 0$ if $t_n = 1$ and $y(\mathbf{x}_n) < 0$ if $t_n = -1$. (So $t_n y(\mathbf{x}) > 0$ for all $\mathbf{x}_n$.)
- Typically there will be more than one hyperplane that separates the classes, so which one to choose?

Maximum margin classifiers

- A natural choice (which has a theoretical justification) is to choose the hyperplane which maximises the *margin*: the distance from the hyperplane to the closest training datapoint.
- Let's look at this using a scikit-learn Jupyter notebook
- The training data points closest to the separating hyperplane are the *support vectors*.
- In a sense, they are the training datapoints 'that matter'.

Maximising the margin

The learning (=optimisation) problem we have to solve is:

$$\arg\max_{\mathbf{w},b} \left\{ \frac{1}{\|\mathbf{w}\|} \min_n [t_n(\mathbf{w}^T \phi(\mathbf{x}_n) + b)] \right\} \tag{2}$$

But we can rescale $\mathbf{w}$ and b so that for a point $\mathbf{x}_n$ that is closest to the separating hyperplane

$$t_n(\mathbf{w}^T \phi(\mathbf{x}_n) + b) = 1 \tag{3}$$

and for all datapoints:

$$t_n(\mathbf{w}^T \phi(\mathbf{x}_n) + b) \geq 1 \quad n = 1, \ldots, N \tag{4}$$

Plugging back into (2) we now just need to maximise $\frac{1}{\|\mathbf{w}\|}$ which is the same as minimising:

$$\arg\min_{\mathbf{w},b} \frac{1}{2} \|\mathbf{w}\|^2 \tag{5}$$

subject to the linear inequality constraints (4).
This is a *quadratic programming* problem.

Dual representation

The dual representation of the maximum margin problem is:

$$\tilde{L}(\mathbf{a}) = \sum_{n=1}^{N} a_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} a_n a_m t_n t_m k(\mathbf{x}_n, \mathbf{x}_m) \tag{6}$$

subject to the constraints:

$$a_n \geq 0, \quad n = 1, \ldots, N \tag{7}$$

$$\sum_{n=1}^{N} a_n t_n = 0 \tag{8}$$

- where, of course, $k(\mathbf{x}_n, \mathbf{x}_m) = \phi(\mathbf{x}_n)^T \phi(\mathbf{x}_m)$.
- This is another quadratic program.
- This dual representation can be derived from the original one by using Lagrange multipliers (which are the a_n).

Support vector machines

- So to learn a max margin classifier we just need the $k(\mathbf{x}_n, \mathbf{x}_m)$ values (i.e. the Gram matrix).
- We do not need to compute $\phi(\mathbf{x}_n)$, so $\phi(\mathbf{x}_n)$ can be as high-dimensional as we like!
- To classify a new datapoint we compute (the sign of)

$$y(\mathbf{x}) = \sum_{n=1}^{N} a_n t_n k(\mathbf{x}, \mathbf{x}_n) + b \tag{9}$$

- So again only the kernel function is needed.
- Crucially, typically for most training datapoints $\mathbf{x}_n$ we have $a_n = 0$ and they are not needed for making predictions.
- The ones that are needed are called *support vectors*.

Choosing a kernel

- You have already seen an SVM with a particular choice of kernel: the *linear kernel* $k(\mathbf{x}_n, \mathbf{x}_m) = \mathbf{x}_n^T \mathbf{x}_m$.
- Let's look at some more interesting kernels.
- We will use this useful Jupyter notebook
- The default kernel for NuSVC is the popular RBF kernel:

$$k(\mathbf{x}, \mathbf{x}') = \exp(-\gamma \|\mathbf{x} - \mathbf{x}'\|^2) \tag{10}$$

- The (implicit) feature space for the RBF kernel is infinite dimensional.