COMS30035, Machine learning: Kernels 1

James Cussens
james.cussens@bristol.ac.uk

Department of Computer Science, SCEEM
University of Bristol

October 2, 2020

Agenda

- Dual representations
- Kernel functions
- The ‘kernel trick’
- The reading associated with this lecture is [Bis06, p.291–294].

Linear regression revisited

Consider a linear regression model:

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

where

1. $\mathbf{w}$ is the M-dimensional parameter vector to be learned from the data (which includes a component for the intercept)
2. $\mathbf{x}$ is some datapoint, and
3. $\phi(\mathbf{x})$ is the M-dimensional feature vector which $\mathbf{x}$ gets mapped to by the *basis functions* [Bis06, §3.1].

Dual representations

- Let N be the size of the data. Let Φ be the *design matrix* whose nth row is just the feature vector for the nth datapoint (so it is basically 'the data'). It turns out that we can reformulate in terms of an N-dimensional parameter vector $\mathbf{a}$ as follows:

$$\mathbf{w} = \Phi^T \mathbf{a} \tag{2}$$

so that

$$y(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x}) = \mathbf{a}^T \Phi \phi(\mathbf{x}) \tag{3}$$

- This is known as a *dual representation*.
- So we have replaced an M-dimensional parameter vector with an N-dimensional one and moreover, to make a prediction for a new datapoint we have to use the entire training set (i.e. Φ)
- On the face of it this does not seem such a great idea, (unless perhaps M is much bigger than N).

Kernel functions are scalar products in feature space

Let's have a look at $\mathbf{\Phi}\phi(\mathbf{x})$. Suppose, for example, that we had 3 datapoints $\mathbf{x}_1$, $\mathbf{x}_2$ and $\mathbf{x}_3$ and 2 features so

$$\mathbf{\Phi}\phi(\mathbf{x}) = \begin{pmatrix} \phi_1(\mathbf{x}_1) & \phi_2(\mathbf{x}_1) \\ \phi_1(\mathbf{x}_2) & \phi_2(\mathbf{x}_2) \\ \phi_1(\mathbf{x}_3) & \phi_2(\mathbf{x}_3) \end{pmatrix} \begin{pmatrix} \phi_1(\mathbf{x}) \\ \phi_2(\mathbf{x}) \end{pmatrix} = \begin{pmatrix} \phi(\mathbf{x}_1)^T\phi(\mathbf{x}) \\ \phi(\mathbf{x}_2)^T\phi(\mathbf{x}) \\ \phi(\mathbf{x}_3)^T\phi(\mathbf{x}) \end{pmatrix} \tag{4}$$

If we define a *kernel function*

$$k(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^T\phi(\mathbf{x}') \tag{5}$$

then we can write:

$$\mathbf{\Phi}\phi(\mathbf{x}) = \begin{pmatrix} k(\mathbf{x}_1, \mathbf{x}) \\ k(\mathbf{x}_2, \mathbf{x}) \\ k(\mathbf{x}_3, \mathbf{x}) \end{pmatrix} \tag{6}$$

The kernel trick

So the prediction for datapoint $\mathbf{x}$ in our example is:

$$\mathbf{a}^T \Phi \phi(\mathbf{x}) = \mathbf{a}^T \begin{pmatrix} k(\mathbf{x}_1, \mathbf{x}) \\ k(\mathbf{x}_2, \mathbf{x}) \\ k(\mathbf{x}_3, \mathbf{x}) \end{pmatrix} \tag{7}$$

- The key point is that **we just need the kernel function** to make the prediction.
- The 'kernel trick' is to evaluate the kernel function values, e.g. $k(\mathbf{x}_1, \mathbf{x})$ without first computing $\phi(\mathbf{x}_1)$ and $\phi(\mathbf{x})$ and then computing their scalar product.
- This allows us to use very high-dimensional (even infinite dimensional!) feature spaces since features are never directly computed.

Kernel functions and similarity

- A kernel function represents the degree of 'similarity' between its two arguments (so kernel functions are always symmetric).
- A high kernel value represents a high degree of similarity.
- Returning to our example, the prediction for $\mathbf{x}$ is:

$$y(\mathbf{x}) = \mathbf{a}^T \begin{pmatrix} k(\mathbf{x}_1, \mathbf{x}) \\ k(\mathbf{x}_2, \mathbf{x}) \\ k(\mathbf{x}_3, \mathbf{x}) \end{pmatrix} = a_1 k(\mathbf{x}_1, \mathbf{x}) + a_2 k(\mathbf{x}_2, \mathbf{x}) + a_3 k(\mathbf{x}_3, \mathbf{x}) \qquad (8)$$

- So the prediction for $\mathbf{x}$ is a linear function of the 'similarities' between $\mathbf{x}$ and each element of the training data.
- So unlike, say, linear regression it looks like we have to keep the entire training set around to make predictions.

Kernel functions and similarity

- A kernel function represents the degree of 'similarity' between its two arguments (so kernel functions are always symmetric).
- A high kernel value represents a high degree of similarity.
- Returning to our example, the prediction for $\mathbf{x}$ is:

$$y(\mathbf{x}) = \mathbf{a}^T \begin{pmatrix} k(\mathbf{x}_1, \mathbf{x}) \\ k(\mathbf{x}_2, \mathbf{x}) \\ k(\mathbf{x}_3, \mathbf{x}) \end{pmatrix} = a_1 k(\mathbf{x}_1, \mathbf{x}) + a_2 k(\mathbf{x}_2, \mathbf{x}) + a_3 k(\mathbf{x}_3, \mathbf{x}) \qquad (8)$$

- So the prediction for $\mathbf{x}$ is a linear function of the 'similarities' between $\mathbf{x}$ and each element of the training data.
- So unlike, say, linear regression it looks like we have to keep the entire training set around to make predictions.
- But in fact we only need those $\mathbf{x}_i$ where $a_i \neq 0$. (See later on *support vector machines*.)

Learning with kernels

- So far we have focused on making predictions using a learned value of the dual parameter vector $\mathbf{a}$.
- If we needed to compute feature values $\phi(\mathbf{x})$ to learn $\mathbf{a}$, then the advantage of using kernels would disappear.
- But the good news is that (for many models) we can learn $\mathbf{a}$ just using kernels.
- So both learning and predicting just require evaluating kernel functions.

Learning with kernels example (1)

- Suppose we want to add a quadratic regulariser (aka *weight decay*) term when minimising the squared error on the training set $\mathbf{w}$ [Bis06, §3.1.4].
- Then, if we were not using kernels, our goal [Bis06, (6.2)] is to minimise $J(\mathbf{w})$ where:

$$J(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\left\{\mathbf{w}^T\phi(\mathbf{x}_n) - t_n\right\}^2 + \frac{\lambda}{2}\mathbf{w}^T\mathbf{w} \tag{9}$$

- N is the number of training datapoints, λ is the regularisation parameter and t_n is the observed target value of the nth training datapoint.
- The *Gram matrix* $\mathbf{K}$ is defined to be $\mathbf{\Phi}\mathbf{\Phi}^T$.
- So $K_{nm} = \phi(\mathbf{x}_n)^T\phi(\mathbf{x}_m) = k(\mathbf{x}_n, \mathbf{x}_m)$ is the 'similarity' between the nth and mth datapoint.

Learning with kernels example (2)

- Let $\mathbf{t} = (t_1, \ldots, t_N)^T$ and let $\mathbf{I}_N$ be the $N \times N$ identity matrix then it turns out that setting:

$$\mathbf{a} = (\mathbf{K} + \lambda \mathbf{I}_N)^{-1} \mathbf{t} \tag{10}$$

- is equivalent to minimising $J(\mathbf{w})$ (see [Bis06, §6.1] for the necessary algebra).
- The key point is that the dual parameters can be learned just using kernels and without computing any feature values.
- Although if N is large then this involves inverting a large matrix.


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