Lecture 18, Friday w9, 2015-11-20

This lecture finished talking about Gaussian processes. And had a quick look at Principal Components Analysis (PCA).

Gaussian processes

I did some review, including looking at the squared exponential kernel that are in the last lecture's notes, but that I'd only shown you very quickly.

We can learn the parameters of a kernel such as ℓ and σ_f^2 (and perhaps the weights in a combination of kernels), and the observation noise level σ_n^2, from the data. The likelihood is simply the Gaussian probability of the observed outputs given all these (hyper-)parameters. It's common to find maximum likelihood fits, or we could be fully Bayesian (using approximate methods such as MCMC) to integrate over the possible hyper-parameters.

It's often said that we can fit hyper-parameters without overfitting because there aren't many of them. However, for small datasets we've seen that different hyperparameters can give different and reasonable explanations of the data. So we might want to consider multiple settings of the hyperparameters. As a practical warning, letting the noise level σ_n^2 go close to zero can easily lead to overfitting. If you do fit the parameters, I recommend some regularization of at least the noise parameter.

Gaussian processes are computationally expensive for large datasets. Just computing and storing the kernel can be a burden, $O(n^2)$. Then factorizing the kernel to solve linear systems and compute determinants costs $O(n^3)$. There are fast methods that work in special circumstances. But for some regression problems the most competitive approximate Gaussian process method is to simply throw away some of the data(!).

Final remark: Although Gaussian processes are flexible models, as with all machine learning methods, it's still a good idea to consider standard pre-processing. For example, try taking the logs of positive inputs and positive outputs. (Non-examinable: a better transformation might be learned from the data; perhaps even using Gaussian processes!)

Principal Components Analysis (PCA)

In this lecture I only showed the first 8 slides of the PCA slide set.

Principal Components Analysis reduces the dimensionality of an $N \times D$ data matrix, by multiplying by a $D \times K$ matrix V.

The columns of V are the K orthonormal eigenvectors of the covariance matrix associated with the largest K eigenvalues. If X is zero-mean, V contains the

eigenvectors of $X^\top X$. See the slides for example code.

An intuition is that many datasets don't fill their native D-dimensional space. There are many strong constraints between the variables. For example, the location of a point on a mesh representing a human body is strongly constrained by the surrounding mesh-points. PCA describes the *principal* ways in which variables can jointly change when moving away from the mean object.

PCA is *widely* used, across many different types of data. It can give a quick first visualization of a dataset. Or reduce the number of dimensions of a data matrix if overfitting or the computational cost of fitting is a concern.

Given an $N \times D$ matrix, we can run PCA to visualize the N rows. Or we can transpose the matrix and instead visualize the D columns. Example: given a matrix relating students and courses, we might be interested in visualizing either the students or the courses or both.

Test your understanding

Here are questions about the lengthscale parameter(s) ℓ_d in the squared exponential kernel. The answers aren't explicitly in the slides, you'll have to think about what the prior model says, and so what will be imposed on the posterior.

- If a GP uses a kernel with a lengthscale that is too short, what will happen and why? You don't need to do maths: sketch what draws from the prior would be like, and use your intuition to see what posterior samples would look like given a few datapoints.
- What will happen as the lengthscale ℓ is driven to infinity?
- If a kernel has a different lengthscale ℓ_d for each feature-vector dimension, what happens if we drive one of these lengthscales to infinity?

Further reading

Optional further reading on PCA: Murphy 12.2.1 p387–389, and 12.2.3 pp392–395.

Bonus notes

Non-examinable!

The truncated SVD view of PCA shows how to simultaneously reduce the dimensionality of the rows and columns of a matrix.

Singular Value Decomposition (SVD) is a standard technique, available in most linear algebra packages. It factors a $N \times D$ matrix into a product of three

matrices, U of size $N \times K$, S a diagonal $K \times K$ matrix, and $V^\top$ of size $K \times D$. The V matrix is the same as before, its columns (or the rows of $V^\top$ contain eigenvectors of $X^\top X$. The columns of U contain eigenvectors of $XX^\top$. The rows of U give a K-dimensional embedding of the rows of X. The columns of $V^\top$ (or the rows of V) give a K-dimensional embedding of the columns of X.

A truncated SVD is known to be the best low-rank approximation of a matrix (as measured by square error). PCA is the linear dimensionality reduction method that minimizes the least squares error in the distortion if we project back to the original space: $X \approx XVV^\top$.

In the past I've been asked: if X is a square symmetrical matrix, doesn't the SVD of X give me the eigenvectors of X? Yes it does. That's potentially confusing because above I said that it gives the eigenvectors of $XX^\top$ and $X^\top X$. For a square symmetrical matrix, the SVD therefore gives the eigenvectors of X^2. These are in fact the same as the eigenvectors of X, so there's no contradiction.

X^2 is the square function applied to the matrix X. A way to apply a function to a covariance matrix is to decompose the matrix using the full SVD: $X = USV^\top$, apply the function to the diagonal elements of S (in this case square the values), and then put the matrix back together again. The eigenvectors don't change! It may interest you to know that other functions are applied to matrices in this way. For example the matrix exponential of a covariance matrix (expm in Matlab), equal to $X + \frac{1}{2}X^2 + \frac{1}{3!}X^3 + \frac{1}{4!}X^4...$, can be computed by taking the SVD, exponentiating the singular values, and putting the matrix back together again. For a general square matrix, a function is applied to eigenvalues in an eigendecomposition $X = U\Lambda U^{-1}$.