Machine Learning and Pattern Recognition Self-Study sheet 8

Instructor: Iain Murray

1. **Pre-processing for Bayesian linear regression and Gaussian processes:**

   We have a dataset of inputs and outputs $\{\mathbf{x}_n, y_n\}_{n=1}^N$, describing N preparations of cells from some lab experiments. The output of interest, y_n, is the standard deviation of the radii of the cells in preparation n. The first input feature of each preparation indicates whether the cells were created in lab `A`, `B`, or `C`. That is, $x_{1,n} \in \{\texttt{A}, \texttt{B}, \texttt{C}\}$. The other features are real numbers describing experimental conditions such as temperature and concentrations of chemicals and nutrients.

   (a) Describe how you might represent the first input feature and the outputs when learning a regression model to predict the standard deviation of cell radii in future preparations from these labs. Explain your reasoning.

   (b) Compare using the lab identity as an input to your regression (as you've discussed above), with two baseline approaches: i) Ignore the lab feature, treat the data from all labs as if they came from one lab; ii) Split the dataset into three parts one for lab `A`, one for `B`, and one for `C`. Then train three separate regression models.

   Discuss both simple linear regression and Gaussian process regression. Is it possible for these models to learn to emulate either or both of the two baselines?

   (c) There's a debate in the lab about how to represent the other input features: log-temperature or temperature, and temperature in Fahrenheit, Celsius or Kelvin? Also whether to use log concentration or concentration as inputs to the regression. Discuss ways in which these issues could be resolved.

2. **Gaussian processes with non-zero mean:**

   In the lectures we assumed that the prior over any vector of function values was zero mean: $\mathbf{f} \sim \mathcal{N}(0, K)$. We focussed on the covariance or kernel function $k(\mathbf{x}_i, \mathbf{x}_j)$ that evaluates the K_{ij} elements of the covariance matrix (or 'Gram matrix').

   If we know in advance that the distribution of outputs should be centered around some other mean $\mathbf{m}$, we *could* put that into the model. Instead, we usually subtract the known mean $\mathbf{m}$ from the $\mathbf{y}$ data, and just use the zero mean model.

   Sometimes we don't really know the mean $\mathbf{m}$, but look at the data to estimate it. A fully Bayesian treatment puts a prior on $\mathbf{m}$ and, because it's an unknown, considers all possible values when making predictions. A flexible prior on the mean vector could be another Gaussian process(!). Our model for our noisy observations is now:

$$
\begin{aligned}
\mathbf{m} &\sim \mathcal{N}(0, K_m), && K_m \text{ from kernel function } k_m,\\
\mathbf{f} &\sim \mathcal{N}(\mathbf{m}, K_f), && K_f \text{ from kernel function } k_f,\\
\mathbf{y} &\sim \mathcal{N}(\mathbf{f}, \sigma_n^2\mathbb{I}), && \text{noisy observations.}
\end{aligned}
$$

   Show that — despite our efforts — the function values $\mathbf{f}$ still come from a function drawn from a zero-mean Gaussian process (if we marginalize out $\mathbf{m}$). Identify the covariance function of the zero-mean process for f.

   Identify the mean's kernel function k_m for two restricted types of mean: 1) An unknown constant $m_i = b$, with $b \sim \mathcal{N}(0, \sigma_b^2)$. 2) An unknown linear trend: $m_i = m(\mathbf{x}_i) = \mathbf{w}^\top \mathbf{x}_i + b$, with Gaussian priors $\mathbf{w} \sim \mathcal{N}(0, \sigma_w^2\mathbb{I})$, and $b \sim \mathcal{N}(0, \sigma_b^2)$.

3. **PCA and supervised learning:**

Principal Components Analysis (PCA) is an unsupervised learning technique. Given only an $N \times D$ matrix of input features X we learn a $D \times K$ matrix V that can project the data down into K dimensions. We could call the new input features $Z = XV$, where each row $\mathbf{z}_n$ is a K-dimensional feature vector.

PCA is often applied as a pre-processing step before supervised learning. For example we could apply logistic regression using the transformed input features Z instead of X. We would fit the weights $\mathbf{w}$ in the model $P(y = 1 \,|\, \mathbf{z}, \mathbf{w}) = \sigma(\mathbf{w}^\top \mathbf{z})$, where $\sigma(z) = 1/(1 + \exp(-z))$ is the logistic sigmoid.

(I'm not including bias parameters to keep any discussion of equations simpler.)

Alternatively, we could create a neural network with a linear hidden layer $\mathbf{z}_n = V^\top \mathbf{x}_n$, where V is a matrix of free parameters to learn along with the output weights during neural network training. A logistic output unit, $P(y=1 \,|\, \mathbf{z}, \mathbf{w}) = \sigma(\mathbf{w}^\top \mathbf{z})$, makes the neural network again make the same predictions as logistic regression applied to the transformed input features $Z = XV$. We can choose to have K hidden units, such that V is a $D \times K$ matrix just like in the PCA approach above.

Compare the two approaches above, saying what (if anything) they could be useful for. Suggest a modification to the neural network approach that could make it more useful.

4. **PCA and autoencoders:**

It's possible to perform unsupervised dimensionality reduction with neural networks instead of implementing PCA by computing eigenvectors or a singular value decomposition (SVD). (The previous question talked about attempting *supervised* dimensionality reduction with neural networks, finding a hidden representation from which we can predict labels well.)

An autoencoder is a neural network with D inputs $\mathbf{x}$ and D outputs $\mathbf{y}$. The goal is to find parameters where the outputs are close to the inputs. The parameters are optimized to minimize the square difference between the two vectors $\sum_n (\mathbf{x}_n - \mathbf{y}_n)^\top (\mathbf{x}_n - \mathbf{y}_n)$.

(a) Compare the autoencoder approach to PCA for an auto-encoder network with K linear hidden units, $\mathbf{z} = V^\top \mathbf{x}$, and a linear output layer $\mathbf{y} = W\mathbf{z}$.

(b) Explain why a non-linear autoencoder, where non-linearities are applied to the layer(s) of hidden units, might work better than a linear autoencoder or PCA.