Course 02457 Non-Linear Signal Processing, Exercise 10

This exercise in application of kernel machines is based on C. M. Bishop: *Pattern Recognition and Machine Learning*, section 6.1,6.2,6.4.1-6.4.3.

Print and comment on the figures produced by the software as outlined below at the **Checkpoints**.

The kernel matrix

A training set of N data points $D = \{(t_1, \mathbf{x}_1), (t_2, \mathbf{x}_2), ..., (t_N, \mathbf{x}_N)\}$ is given and let the dimension of input space be d. Assume that we want to adapt a linear model with weights $\mathbf{w}$ based on least squares, i.e., minimizing

$$E(\mathbf{w}) = \sum_{n=1}^{N}(t_n - \sum_{j=0}^{d} w_j x_{j,n})^2, \tag{1}$$

with $x_{0,n} = 1$ for all n. Assume further that the dimension of input space exceeds the sample size, $d > N$. In this case it is useful to write the $d+1$-dimensional weight vector as a sum of two orthogonal components with respect to the linear subspace spanned by the data vectors,

$$\mathbf{w} = \mathbf{w}_\perp + \mathbf{w}_\parallel, \tag{2}$$

where $\mathbf{w}_\parallel$ is a vector in the subspace spanned by the input data points, while $\mathbf{w}_\perp$ is in the orthogonal subspace. In other words $\mathbf{w}_\parallel$ can be written as the linear combination

$$\mathbf{w}_\parallel = \sum_{n=1}^{N} a_n \mathbf{x}_n, \tag{3}$$

while $\mathbf{w}_\perp$ is orthogonal to all data points: $\mathbf{w}_\perp^\top \mathbf{x}_n = 0$. The sum of squared errors can now be rewritten as

$$\begin{aligned} E(\mathbf{w}) &= \sum_{n=1}^{N}(t_n - \mathbf{w}^\top \mathbf{x}_n)^2 = \sum_{n=1}^{N}(t_n - \mathbf{w}_\parallel^\top \mathbf{x}_n)^2 \\ &= \sum_{n=1}^{N}(t_n - \sum_{m=1}^{N} a_m \mathbf{x}_m^\top \mathbf{x}_n)^2 = \sum_{n=1}^{N}(t_n - \sum_{m=1}^{N} a_m K_{m,n})^2 = \sum_{n=1}^{N}(t_n - (\mathbf{a}^\top \mathbf{K})_n)^2. \end{aligned} \tag{4}$$

where we introduced the notation $(\mathbf{K})_{m,n} = K_{m,n} = \mathbf{x}_m^\top \mathbf{x}_n$ for the *kernel* matrix of inner products among all input vectors.

More generally we see that for a general feature mapping $\mathbf{x} \mapsto \boldsymbol{\phi}(\mathbf{x})$, we can represent a linear model on feature vectors in terms of their inner products

$$E(\mathbf{w}) = \sum_{n=1}^{N}(t_n - \mathbf{w}^\top \boldsymbol{\phi}(\mathbf{x}_n))^2 = \sum_{n=1}^{N}(t_n - (\mathbf{a}^\top \mathbf{K})_n)^2. \tag{5}$$

Here the kernel matrix is $(\mathbf{K})_{m,n} = K_{m,n} = \boldsymbol{\phi}(\mathbf{x}_m)^\top \boldsymbol{\phi}(\mathbf{x}_n)$. Conversely, we may define feature vectors *implicitly* through an $N \times N$ inner product matrix $\mathbf{K}$. An example of this is the Gaussian kernel which is widely used in applications,

$$K_{m,n} = e^{-\|\mathbf{x}_m - \mathbf{x}_n\|^2 / 2\sigma^2} \ , \tag{6}$$

with width parameter σ^2.

The general idea of kernel methods is to use the kernel matrix to encode input similarity, hence, assuming that targets are similar $\mathbf{t}_m \approx \mathbf{t}_n$ when inputs are similar ($\mathbf{x}_m \approx \mathbf{x}_n$), hence, when $K_{m,n}$ has a high value.

Checkpoint 10.1

Here we analyze the structure of the kernel matrix. First let us inspect the gaussian kernel matrix for a simple two cluster simulated data set in $d = 2$. Run the script `main10a.m` and discuss the structure of kernel matrix for these simple simulated data sets. How does the width of the Gaussian kernel affect the kernel matrix? Imagine the kernel matrix columns as new "pseudo-features", how well do they separate classes? Next we investigate the kernel matrix for the case of the sunspot data. Run script `main10b.m` to illustrate the kernel matrix for simple case of 2-dimensional history $d = 2$. Explain the kernel matrix used for predicting targets for test data. Explain the patterns you see in the training and test kernel matrices and relate them to the sunspot time series.

Gaussian process priors for function approximation

The Gaussian process implements the relation $(\mathbf{x}_m \approx \mathbf{x}_n) \Rightarrow (\mathbf{t}_m \approx \mathbf{t}_n)$ in a probabilistic setting. In particular, for function approximation, we assume that the target function $y(\mathbf{x})$ values follow a multivariate normal distribution with the kernel $\mathbf{K}$ matrix as covariance matrix. For a sample of N points this amounts to

$$\mathrm{cov}(y_1, ..., y_N) = \mathbf{K}, \quad p(\mathbf{y}|\mathbf{K}) = \frac{1}{|2\pi\mathbf{K}|^{\frac{1}{2}}} \exp(-\tfrac{1}{2}\mathbf{y}^\top \mathbf{K}^{-1}\mathbf{y}). \tag{7}$$

To model an additive noise process we let the observed targets be given as $t(\mathbf{x}) = y(\mathbf{x}) + \epsilon$, with ϵ being zero mean, Gaussian noise, with variance β^{-1}. Assuming that noise and inputs are independent, the covariance matrix between a set of targets becomes the sum of covariances: $\mathrm{cov}(t_1, ..., t_N) \equiv \mathbf{C} = \mathbf{K} + \beta^{-1}\mathbf{I}$.

In machine learning we are given training and test sets and we are interested in the predictive distribution of the test targets given training data and the $(N_{\text{test}} + N_{\text{train}}) \times (N_{\text{test}} + N_{\text{train}})$ kernel matrix $\mathbf{C}_{\text{train+test}}$ measuring the similarity between all training and test data

$$p(\mathbf{t}_{\text{test}}|\mathbf{t}_{\text{train}}, \mathbf{C}_{\text{train+test}}) = \frac{p(\mathbf{t}_{\text{test}}, \mathbf{t}_{\text{train}}|\mathbf{C}_{\text{train+test}})}{p(\mathbf{t}_{\text{train}}|\mathbf{C}_{\text{train}})}. \tag{8}$$

The predictive distribution is also multivariate Gaussian with mean $\boldsymbol{\mu}_{\text{test|train}}$ and covariance matrix $\mathbf{C}_{\text{test|train}}$ given by

$$\boldsymbol{\mu}_{\text{test|train}} = \mathbf{C}_{\text{test,train}}\mathbf{C}_{\text{train}}^{-1}\mathbf{t}_{\text{train}} \tag{9}$$

$$\mathbf{C}_{\text{test|train}} = \mathbf{C}_{\text{test}} - \mathbf{C}_{\text{test,train}}\mathbf{C}_{\text{train}}^{-1}\mathbf{C}_{\text{train,test}} \tag{10}$$

following the rules for conditioning in the Gaussian distribution (Bishop Appendix B eq. (B.58-B.60), or the Matrix Cookbook eq. (331-332)). Here $\mathbf{C}_{\text{test,train}}$ is the $N_{\text{test}} \times N_{\text{train}}$ is the sub-matrix of $\mathbf{C}_{\text{train+test}}$ connecting test and training inputs.

The maximum posterior prediction of a test target is $\hat{t}_m = (\boldsymbol{\mu}_{\text{test|train}})_m$. The probabilistic representation of the Gaussian process allows us to infer uncertainties for the predictions as well. Using the posterior covariance in eq. (11) we obtain the estimate

$$\text{std}(t_m) = \sqrt{(\mathbf{C}_{\text{test|train}})_{m,m}} = \sqrt{(\mathbf{C}_{\text{test}})_{m,m} - (\mathbf{C}_{\text{test,train}}\mathbf{C}_{\text{train}}^{-1}\mathbf{C}_{\text{train,test}})_{m,m}} \quad (11)$$

Checkpoint 10.2

We apply a simple Gaussian process (GP) model to the prediction of sunspots. Run the script `main10c.m` to optimize the two parameters (β, σ^2) of the kernel given by $(\mathbf{C})_{m,n} = e^{-\|\mathbf{x}_m - \mathbf{x}_n\|^2/2\sigma^2} + \beta^{-1}\delta_{n,m}$. Explain the relation between the conditional mean in eq. (9) above and the test set predictions. How do we optimize parameters?. What is the difference between the two estimates of the test data in figure 2 (green and blue)?. How well does a GP with a simple Gaussian kernel work? Comment on the quality of the posterior uncertainty estimate, cf. eq. (11).

Support vector machine (SVM) classification

The support vector machine implements a classifier which is very similar to the GP, however, without the probabilistic framework. Rather, it is based on geometry, aimed at maximizing the so-called margin, which is a measure of how well separated the classes are by the given decision boundary. The classifier for two classes, with class one targets encoded as $t_n = -1$ and class two targets as $t_n = +1$, is given as,

$$\hat{t}_m = \text{sign}\left(\sum_{n=1}^{N} a_n t_n K(\mathbf{x}_m, \mathbf{x}_n) + b\right). \quad (12)$$

Here b is a offset parameter. The main difference to the GP is the way we estimate the parameters of the SVM, i.e., the set of non-negative weight parameters $\{a_n\}$. We apply a widely used representation which leads to a box constrained, but still convex, optimization problem

$$\begin{aligned}
\tilde{L}(\mathbf{a}) &= \sum_{n=1}^{N} a_n - \sum_{n,m=1}^{N} a_n a_m t_n t_m K(\mathbf{x}_n, \mathbf{x}_m), \\
&\quad 0 \le a_n \le C, \\
&\quad \sum_{n=1}^{N} a_n t_n = 0, \\
b &= \frac{1}{|M|}\sum_{n \in M}(t_n - \sum_{m \in S} a_m t_m K(\mathbf{x}_n, \mathbf{x}_m)).
\end{aligned}$$

Parameter C controls the amount to which points appear inside the margin. S denotes the set of support vectors ($a_n > 0$), while M is the subset of these for which $a_n < C$. The quadratic problem is solved using a build-in Matlab function `quadprog`.

Pima indian data set

Again, the task is to classify a population of women according to the risk of diabetes (two class classification). There are 7 input variables, 200 training examples and 332 test examples. 68 (34%) in the training set and 109 (32.82%) in the test set have been diagnosed with diabetes. In Brian Ripley's textbook *Pattern Recognition and Neural Networks* he states that his best method obtains about 20% misclassification on this data set so this is what we can use for reference. The input variables are:

1. Number of pregnancies
2. Plasma glucose concentration
3. Diastolic blood pressure
4. Triceps skin fold thickness
5. Body mass index (weight/height^2)
6. Diabetes pedigree function
7. Age

The target output in the data set is 1 for examples diagnosed as diabetes, and 2 for healthy subjects.

Checkpoint 10.3

Compare the SVM decision function in eq. (12) with the nearest neighbor voting scheme discussed in Exercise 9. In the scripts `main10d.m` and `main10e.m` we use the simple Gaussian kernel function as earlier

$$K_{m,n} = e^{-\|\mathbf{x}_m - \mathbf{x}_n\|^2/2\sigma^2}. \tag{13}$$

First analyze a 2-dimensional synthetic data set with two classes. Run `main10d.m`. Inspect the solution given in the Matlab structure `SVM`, what is the number of support vectors?

Next, we analyze the Pima indian data with the script `main10e`. We optimize the parameters of the support vector machine by maximizing the accuracy on the test data. How many parameters are optimized? How many support vectors do you get?

Consider classification from a subset of the seven input variable measures. Estimate the performance for a few subsets, can you find a subset with performance equal or better than that of the full set?