Chapter 1

Additive Gaussian Processes

In section 1.8, we showed how to learn the structure of a kernel by building it up piece-by-piece. This chapter presents an alternative approach, where we start with many different types of structure in the kernel, and adjust kernel parameters to discard whatever structues are *not* present in the current dataset. The advantage of this approach is that we do not need to run an expensive discrete-and-continuous optimization problem in order to build a structured model. Implementation is also much simpler.

The type of structure our kernel will represent are sums of functions of all possible combinations of input variables. We call this model class *additive Gaussian processes*. This model can be specified by a kernel which is a sum of all possible products of one-dimensional kernels.

There are 2^D combinations of D inputs, so a naïve computation of such a kernel would be intractable. Furthermore, if each term has different kernel parameters, fitting or integrating over so many parameters would pose severe difficulty. To get around this problem, we introduce a parameterization of the kernel which allows efficient evaluation of all interaction terms. Empirically, this kernel has good predictive power in regression tasks, and its parameters are relatively interpretable.

The work in this chapter was done in collaboration with Hannes Nickisch and Carl Rasmussen, who derived and coded up the initial model. My role in the project was to examine the properties of the resulting model, clarify the connections to existing methods, to create all figures and run all experiments. That work was published in **?**. The connection to dropout regularization is an independent original contribution.

1.1 Types of Multivariate Additive Structure

In section 1.8, we saw how additive structure in a GP prior enabled long-range extrapolation in multivariate regression problems. In general, models of the form

$$f(\mathbf{x}) = g\left(f(x_1) + f(x_2) + \cdots + f(x_D)\right) \tag{1.1}$$

are widely used in machine learning and statistics, partly because they are relatively easy to fit and interpret. Examples include logistic regression, linear regression, generalized linear models (**?**) and generalized additive models (**?**).

At the other end of the spectrum are models which allow the response to depend on all input variables simultaneously, having the most general form:

$$f(\mathbf{x}) = f(x_1, x_2, \ldots, x_D) \tag{1.2}$$

An example would be a GP with an SE-ARD kernel. Such models are much more flexible than those having the form (1.1), but this flexibility can make it difficult to generalize to new combinations of input variables.

In between these extremes, we can consider function classes depending on pairs or triplets of inputs, such as

$$f(x_1, x_2, x_3) = f_{12}(x_1, x_2) + f_{23}(x_2, x_3) + f_{13}(x_1, x_3) \tag{1.3}$$

We call the number of input variables appearing in each term the *order* of a model class. Models of intermediate order such as (1.3) allow more flexibility than models of form (1.1) (D-th order), but have more structure than those of form (1.2) (first-order).

If the function being learned depends in some way on an interaction between all input variables, a Dth-order term is required in order for the model to be consistent. However, if the function also contains lower-order interactions, capturing that structure will still improve the predioctive performance on finite datasets.

1.2 Additive Kernels

We now give a precise definition of the additive kernels introduced in this chapter. We first assign each dimension $i \in \{1 \ldots D\}$ a one-dimensional *base kernel* $k_i(x_i, x_i')$. We

then define the first order, second order and nth order additive kernel as:

$$k_{add_1}(\mathbf{x}, \mathbf{x}') = \sigma_1^2 \sum_{i=1}^{D} k_i(x_i, x_i') \tag{1.4}$$

$$k_{add_2}(\mathbf{x}, \mathbf{x}') = \sigma_2^2 \sum_{i=1}^{D} \sum_{j=i+1}^{D} k_i(x_i, x_i') k_j(x_j, x_j') \tag{1.5}$$

$$k_{add_n}(\mathbf{x}, \mathbf{x}') = \sigma_n^2 \sum_{1 \le i_1 < i_2 < \ldots < i_n \le D} \left[\prod_{d=1}^{n} k_{i_d}(x_{i_d}, x_{i_d}') \right] \tag{1.6}$$

$$k_{add_D}(\mathbf{x}, \mathbf{x}') = \sigma_n^2 \sum_{1 \le i_1 < i_2 < \ldots < i_D \le D} \left[\prod_{d=1}^{D} k_{i_d}(x_{i_d}, x_{i_d}') \right] = \sigma_D^2 \prod_{d=1}^{D} k_d(x_d, x_d') \tag{1.7}$$

where D is the dimension of our input space, and σ_n^2 is the variance assigned to all nth order interactions. The nth covariance function is a sum of $\binom{D}{n}$ terms. In particular, the Dth order additive covariance function has $\binom{D}{D} = 1$ term, a product of each dimension's covariance function. In the case where each base kernel is a one-dimensional squared-exponential kernel, the Dth-order term corresponds to the multivariate squared-exponential kernel, also known as SE-ARD:

$$k_{add_D}(\mathbf{x}, \mathbf{x}') = \sigma_D^2 \prod_{d=1}^{D} k_d(x_d, x_d') = \sigma_D^2 \prod_{d=1}^{D} \exp\left(-\frac{(x_d - x_d')^2}{2l_d^2} \right) = \sigma_D^2 \exp\left(-\sum_{d=1}^{D} \frac{(x_d - x_d')^2}{2l_d^2} \right) \tag{1.8}$$

also commonly known as the Gaussian kernel.

The full additive kernel is a sum of the additive kernels of all orders.

The only design choice necessary to specify an additive kernel is the selection of a one-dimensional base kernel for each input dimension. Parameters of the base kernels (such as length-scales) can be learned as usual by maximizing the marginal likelihood of the training data.

1.3 Weighting Different Orders of Interaction

In addition to the parameters of each dimension's kernel, additive kernels are equipped with a set of D parameters $\sigma_1^2 \ldots \sigma_D^2$. These "order variance" parameters have a useful interpretation: the dth order variance hyperparameter controls how much of the target function's variance comes from interactions of the dth order. Table 1.1 shows examples of the variance contributed by different orders of interaction, learned on real datasets.

Table 1.1 Percentage of variance contributed by each order of the additive model, on different datasets. The maximum order of interaction is set to the input dimension or 10, whichever is smaller.

	Order of interaction									
Dataset	1st	2nd	3rd	4th	5th	6th	7th	8th	9th	10th
pima	0.1	0.1	0.1	0.3	1.5	**96.4**	1.4	0.0		
liver	0.0	0.2	**99.7**	0.1	0.0	0.0				
heart	**77.6**	0.0	0.0	0.0	0.1	0.1	0.1	0.1	0.1	22.0
concrete	**70.6**	13.3	13.8	2.3	0.0	0.0	0.0	0.0		
pumadyn-8nh	0.0	0.1	0.1	0.1	0.1	0.1	0.1	**99.5**		
servo	**58.7**	27.4	0.0	13.9						
housing	0.1	0.6	**80.6**	1.4	1.8	0.8	0.7	0.8	0.6	12.7

On different datasets, the dominant order of interaction estimated by the additive model varies widely. An additive GP with all of its variance coming from the 1st order is equivalent to a sum of one-dimensional functions. An additive GP with all its variance coming from the Dth order is equivalent to a GP with an SE-ARD kernel.

Because the variance parameters can specify which degrees of interaction are important, the additive GP can capture many different types of structure. The marginal likelihood will usually favor using lower orders if possible, since the $|K|^{-\frac{1}{2}}$ term in the GP marginal likelihood (**??**) will usually be larger for less flexible model classes. Low-order structure allows long-range extrapolation, as shown in **??**. If low-dimensional additive structure is not present, the kernel parameters can specify a suitably flexible model, with interactions between as many variables as necessary.

1.3.1 Efficient Evaluation of Additive Kernels

An additive kernel over D inputs with interactions up to order n has $O(2^n)$ terms. Naïvely summing over these terms quickly becomes intractable. In this section, we show how one can evaluate the sum over all terms in $O(D^2)$, while still weighting each order seperately.

To efficiently compute the additive kernel, we exploit the fact that the nth order additive kernel corresponds to the nth *elementary symmetric polynomial* (**?**) of the base kernels, which we denote e_n. For example: if $\mathbf{x}$ has 4 input dimensions ($D = 4$), and if

we use the shorthand notation $k_d = k_d(x_d, x'_d)$, then

$$k_{\text{add}_0}(\mathbf{x}, \mathbf{x}') = e_0(k_1, k_2, k_3, k_4) = 1 \tag{1.9}$$

$$k_{\text{add}_1}(\mathbf{x}, \mathbf{x}') = e_1(k_1, k_2, k_3, k_4) = k_1 + k_2 + k_3 + k_4 \tag{1.10}$$

$$k_{\text{add}_2}(\mathbf{x}, \mathbf{x}') = e_2(k_1, k_2, k_3, k_4) = k_1k_2 + k_1k_3 + k_1k_4 + k_2k_3 + k_2k_4 + k_3k_4 \tag{1.11}$$

$$k_{\text{add}_3}(\mathbf{x}, \mathbf{x}') = e_3(k_1, k_2, k_3, k_4) = k_1k_2k_3 + k_1k_2k_4 + k_1k_3k_4 + k_2k_3k_4 \tag{1.12}$$

$$k_{\text{add}_4}(\mathbf{x}, \mathbf{x}') = e_4(k_1, k_2, k_3, k_4) = k_1k_2k_3k_4 \tag{1.13}$$

The Newton-Girard formulae give an efficient recursive form for computing these polynomials:

$$k_{\text{add}_n}(\mathbf{x}, \mathbf{x}') = e_n(k_1, \ldots, k_D) = \frac{1}{n}\sum_{a=1}^{n}(-1)^{(a-1)}e_{n-a}(k_1, \ldots, k_D)\sum_{i=1}^{D}k_i^a \tag{1.14}$$

The Newton-Girard formulae have time complexity $\mathcal{O}(D^2)$, while computing a sum over an exponential number of terms.

Conveniently, we can use the same trick to efficiently compute all of the necessary derivatives of the additive kernel with respect to the base kernels. We merely need to remove the kernel of interest from each term of the polynomials:

$$\frac{\partial k_{add_n}}{\partial k_j} = e_{n-1}(k_1, \ldots, k_{j-1}, k_{j+1}, \ldots k_D) \tag{1.15}$$

This allows us to optimize the base kernel parameters with respect to the marginal likelihood using gradient-based methods.

1.3.2 Computational Cost

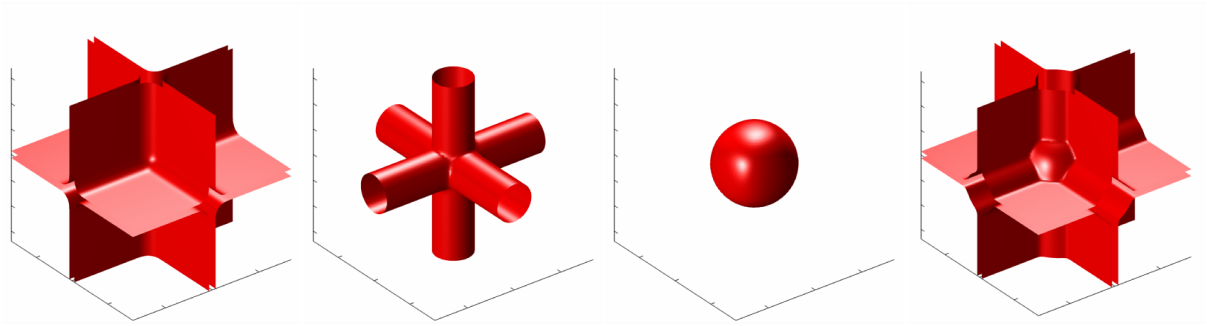
The computational cost of evaluating the Gram matrix $k(\mathbf{X}, \mathbf{X})$ of a product kernel such as the SE-ARD scales as $\mathcal{O}(N^2D)$, while the cost of evaluating the Gram matrix of the additive kernel scales as $\mathcal{O}(N^2DR)$, where R is the maximum degree of interaction allowed (up to D). In higher dimensions, this can be a significant cost, even relative to the fixed $\mathcal{O}(N^3)$ cost of inverting the Gram matrix. However, table 1.1 shows that sometimes only the first few orders of interaction are important. Hence if one is computationally limited, one may be able to limit the maximum degree of interaction without losing much accuracy.

1.4 Non-local Interactions

By far the most popular kernels for regression and classification tasks are *local* kernels, such as the SE, RQ or Matérn kernels. These kernels all depend only on the scaled Euclidean distance between two points, having the form:

$$k(\mathbf{x}, \mathbf{x}') = g\left(\sum_{d=1}^{D}\left(\frac{x_d - x'_d}{l_d}\right)^2\right) \tag{1.16}$$

For some function $g(\cdot)$. **?** argue that models based on local kernels are particularly susceptible to the curse of dimensionality, and are generally unable to extrapolate away from the training data. Thus, methods based solely on local kernels will sometimes require training examples at exponentially-many combinations of inputs.



1st-order terms $k_1 + k_2 + k_3$ | 2nd-order terms $k_1k_2 + k_2k_3 + k_1k_3$ | 3rd-order terms $k_1k_2k_3$ Squared-exp kernel | All interactions Additive kernel

Figure 1.1 Isocontours of additive kernels in 3 dimensions. The third-order kernel only considers nearby points relevant, while the lower-order kernels allow the output to depend on distant points, as long as they share one or more input value.

Additive kernels have a more complex structure, and allow extrapolation far from the training data. For example, additive kernels of the second order give high covariance between function values at points which are similar in any two input dimensions. Figure 1.1 provides a geometric comparison between squared-exponential kernels and additive kernels in 3 dimensions. **??** contains an example of how additive kernels extrapolate differently than local kernels.

1.5 Dropout in Gaussian Processes

Dropout is a method for regularizing neural networks (**??**). Training with dropout entails randomly and independently setting to zero ("dropping") some proportion p of features or inputs, in order to improve the robustness of the resulting network by reducing co-dependence between neurons. To maintain similar overall activation levels, weights are multiplied by $1/p$ at test time. Alternatively, feature activations are multiplied by $1/p$ during training. Test-time predictions are made by approximately averaging over all possible ways of dropping out neurons.

? and **?** analyzed dropout in terms of the effective prior induced by this procedure in several models, such as linear and logistic regression. In this section, perform a similar analysis for GPs, examining the priors on functions that result from performing dropout in the one-hidden-layer neural network implicitly defined by a GP.

Recall from **??** that GPs can be derived as an infinitely-wide one-layer neural network, with fixed activation functions $\mathbf{h}(\mathbf{x})$ (where $k(\mathbf{x}, \mathbf{x}') = \mathbf{h}(\mathbf{x})^\mathsf{T}\mathbf{h}(\mathbf{x}')$), and independent random weights $\boldsymbol{\alpha}$ with finite variance $\sigma_{\boldsymbol{\alpha}}^2$:

$$f(\mathbf{x}) = \frac{1}{K}\boldsymbol{\alpha}^\mathsf{T}\mathbf{h}(\mathbf{x}) = \frac{1}{K}\sum_{i=1}^{K}\alpha_i h_i(\mathbf{x}) \tag{1.17}$$

$$\implies f \overset{K\to\infty}{\sim} \mathcal{GP}\Big(\mathbb{E}\left[\boldsymbol{\alpha}\right]^\mathsf{T}\mathbf{h}(\mathbf{x}), \sigma_{\boldsymbol{\alpha}}^2\mathbf{h}(\mathbf{x})^\mathsf{T}\mathbf{h}(\mathbf{x}')\Big) \tag{1.18}$$

Mercer's theorem implies that we can write any GP prior equivalently in this way. Having expressed a GP as a neural network, we can examine the prior we get from performing dropout in this network.

1.5.1 Dropout on Hidden Units

First, we will examine the prior we get from independently dropping features from $\mathbf{h}(\mathbf{x})$ by setting some of the weights $\boldsymbol{\alpha}$ to zero with probability p. For simplicity, we assume that $\mathbb{E}\left[\boldsymbol{\alpha}\right] = \mathbf{0}$. If the weights initially have finite variance σ_α^2 before dropout, then after dropout they'll have variance

$$r_i \overset{\text{iid}}{\sim} \text{Ber}(p) \qquad \mathbb{V}\left[r_i\alpha_i\right] = p\sigma_\alpha^2 \,. \tag{1.19}$$

Because equation (1.18) is a result of the central limit theorem, it does not depend on the form of the distribution on $\boldsymbol{\alpha}$, only its mean and variance. Thus, dropping out features of an infinitely-wide MLP does not change the model at all, except to rescale the output

variance. Indeed, multiplying all weights by $p^{-1/2}$ restores the initial variance:

$$\mathbb{V}\left[\frac{1}{p^{\frac{1}{2}}} r_i \alpha_i\right] = \frac{p}{p}\sigma_\alpha^2 = \sigma_\alpha^2 \,. \tag{1.20}$$

In which case dropout on the hidden units has no effect at all. Intuitively, this is because no individual feature can have more than an infinitesimal contribution to the network output.

1.5.2 Dropout on Inputs

In a GP with a product kernel $k(\mathbf{x}, \mathbf{x}') = \prod_{d=1}^{D} k_d(x_d, x_d')$, exactly averaging over all possible ways of dropping out inputs with probability $1/2$ results in a mixture of GPs, each depending on only a subset of the inputs:

$$p\left(f(\mathbf{x})\right) = \frac{1}{2^D} \sum_{\mathbf{r}\in\{0,1\}^D} \mathrm{GP}\left(f(\mathbf{x}) \,\middle|\, 0, \prod_{d=1}^{D} k_d(x_d, x_d')^{r_d}\right) \tag{1.21}$$

We present two results ways to gain intuition about this model.

First, if the kernel on each dimension has the form $k_d(x_d, x_d') = g\left(\frac{x_d - x_d'}{w_d}\right)$, as does the SE kernel, then any input dimension can be dropped out by setting its lengthscale w_d to ∞. Thus, performing dropout on the inputs of a GP corresponds to putting independent spike-and-slab priors on the lengthscales, with each dimension independently having $w_d = \infty$ with probability $1/2$.

Another way to understand the resulting prior is to note that the dropout mixture of GPs (1.21) has the same covariance as an additive GP, scaled by a factor of 2^{-D}:

$$f(\mathbf{x}) \sim \mathrm{GP}\left(0, \frac{1}{2^D} \sum_{\mathbf{r}\in\{0,1\}^D} \prod_{d=1}^{D} k_d(x_d, x_d')^{r_d}\right) \tag{1.22}$$

Therefore, ignoring higher moments, dropout on the inputs of a GP can be approximated by an additive GP with all orders equally weighted. This suggests an interpretation of additive GPs as an approximation to a mixture of models where each model only depends on a subset of the input variables.

1.6 Related Work

Since additive models are a relatively natural and easy-to-analyze model class, the literature on similar model classes is extensive. We try to give a broad overview in this section.

Previous work on Additive GPs

Since the non-local structure capturable by additive kernels is necessarily axis-aligned, we can naturally consider that an initial transformation of the input space might allow us to recover non-axis aligned additivity in functions. This avenue was explored by **?**, who developed a linearly-transformed first-order additive GP model, called projection-pursuit GP regression. They further showed that inference in this model was possible in $\mathcal{O}(N)$ time.

? also examined the properties of additive GPs, and proposed a layer-wise optimization strategy for kernel hyperparameters in these models.

? constructed an additive GP having only first-order and Dth-order terms. This model is motivated by the desire to trade off the interpretability of first-order models with the flexibility of full-order models. Our experiments show that sometimes, the intermediate degrees of interaction contribute most of the variance.

? used a closely related procedure called Gaussian process ANOVA to perform a Bayesian analysis of meteorological data using 2nd and 3rd-order interactions. They also introduce a weighting scheme to ensure that each order's total contribution sums to zero. It is not clear if this weighting scheme permits the use of the Newton-Girard formula to speed computation of the Gram matrix.

Hierarchical Kernel Learning

A similar model class was recently explored by **?** called hierarchical kernel learning (HKL). HKL uses a regularized optimization framework to learn a weighted sum over an exponential number of kernels which can be computed in polynomial time. This method chooses among *hull* of kernels, defined as a set of terms such that if $\prod_{j \in J} k_j(\mathbf{x}, \mathbf{x}')$ is included in the set, then so are all lower-order terms containing the same elements: $\prod_{j \in J/i} k_j(\mathbf{x}, \mathbf{x}')$, for all $i \in J$. HKL computes the sum over all orders in $\mathcal{O}(D)$ time by the formula:

$$k_a(\mathbf{x}, \mathbf{x}') = v^2 \prod_{d=1}^{D} (1 + \alpha k_d(x_d, x'_d)) \tag{1.23}$$

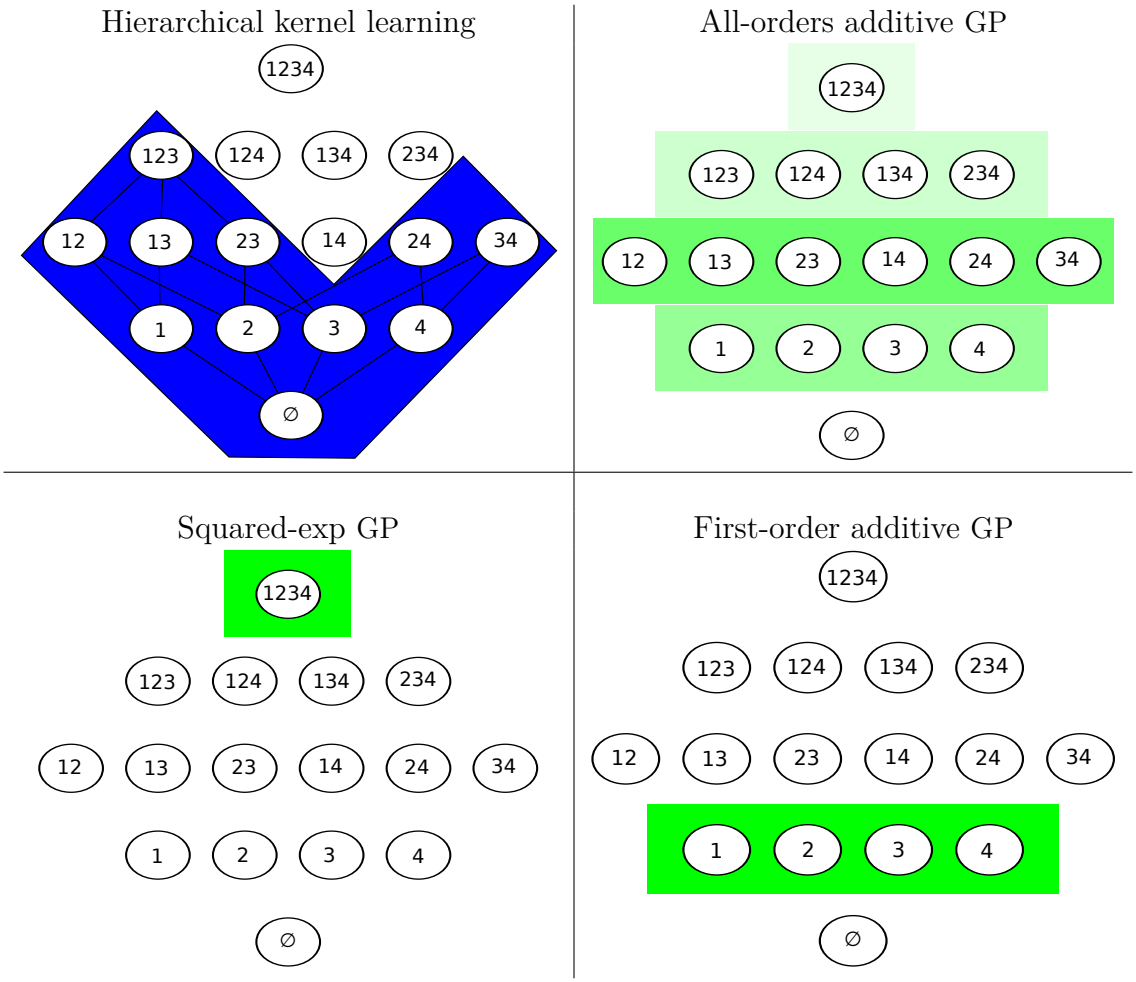


Figure 1.2 A comparison of different additive model classes. Nodes represent different interaction terms, ranging from first-order to fourth-order interactions. Coloured boxes represent the weightings of different terms. *Top left:* HKL can select a hull of interaction terms, but must use a pre-determined weighting over those terms. *Top right:* the additive GP model can weight each order of interaction separately, but weights all terms equally within each order. *Bottom row:* The SE-GP and first-order additive GP models are special cases of the all-orders additive GP.

which forces the weight of all nth order terms to be weighted by α^n.

Figure 1.2 contrasts the HKL model class with the additive GP model. Neither method is strictly more flexible than the other. The main difficulty with the approach of **?** is that the kernel parameters are hard to set other than by cross-validation.

Support Vector Machines

? introduced the support vector ANOVA decomposition, which has the same form as our additive kernel. They recommend approximating the sum over all interactions with only one of the D sets of interactions "of appropriate order", presumably because of the difficulty of setting the parameters of an SVM. This is an example of a model choice which can be automated in the GP framework.

? performed experiments which favourably compared the predictive accuracy of the support vector ANOVA decomposition against polynomial and spline kernels. They too allowed only one order to be active, and set parameters by cross-validation.

Other Related Models

A closely related procedure from **?** is smoothing-splines ANOVA (SS-ANOVA). An SS-ANOVA model is a weighted sum of splines along each dimension, plus a sum of splines over all pairs of dimensions, all triplets, etc, with each individual interaction term having a separate weighting parameter. Because the number of terms to consider grows exponentially in the order, in practice, only terms of first and second order are usually considered.

This more general model class, in which each interaction term is estimated separately, is known in the physical sciences as High Dimensional Model Representation (HDMR). **?** review some properties and applications of this model class.

The main benefits of the model setup and parameterization proposed in this chapter are the ability to include all D orders of interaction with differing weights, and the ability to learn kernel parameters individually per input dimension, allowing automatic relevance determination to operate.

1.7 Experiments

Choosing the Base Kernel

A D-dimensional SE-ARD kernel has D lengthscale parameters and one output variance parameter. An first-order additive SE model has D lengthscale parameters and one D output variance parameters. A fully-parametrized model including all orders of interaction with a separate output variance for each scale will have $3 \times D - 1$ effective parameters. Because each additional parameter increases the tendency to overfit, in our experiments we fixed each one-dimensional kernel's output variance to be 1, and only learned the length-scale of each kernel.

Methods

We compare six different methods. In the results tables below, GP Additive refers to a GP using the additive kernel with squared-exp base kernels. For speed, we limited the maximum order of interaction to 10. GP-1st denotes an additive GP model with only first-order interactions - a sum of one-dimensional kernels. GP Squared-exp is a GP model with a SE-ARD kernel. HKL was run using the all-subsets kernel, which corresponds to the same set of interaction terms as considered by the additive GP with a squared-exp base kernel.

For all GP models, we fit kernel parameters by the standard method of maximizing training-set marginal likelihood, using L-BFGS (**?**) for 500 iterations, allowing five random restarts. In addition to learning kernel parameters, we fit a constant mean function to the data. In the classification experiments, approximate GP inference was done using expectation propagation (**?**).

For the regression experiments, we also compared against the structure search method from section 1.8, run up to depth 10, using the SE and RQ base kernel families.

1.7.1 Datasets

We compared these methods on a diverse set of regression and classification datasets from the UCI repository (**?**). Their size and dimension are given in tables 1.2 and 1.3:

Bach Synthetic Dataset

In addition to standard UCI repository datasets, we generated a synthetic dataset following the same recipe as **?**. This dataset was designed to demonstrate the advantages

Table 1.2 Regression Dataset Statistics

Method	bach	concrete	pumadyn	servo	housing
Dimension	8	8	8	4	13
Number of datapoints	200	500	512	167	506

Table 1.3 Classification Dataset Statistics

Method	breast	pima	sonar	ionosphere	liver	heart
Dimension	9	8	60	32	6	13
Number of datapoints	449	768	208	351	345	297

of HKL over GP-SE. It is generated by passing correlated Gaussian-distributed inputs $x_1, x_2, \ldots, x_8$ through the quadratic function

$$f(\mathbf{x}) = \sum_{i=1}^{4} \sum_{j=1+1}^{4} x_i x_j + \epsilon \qquad \epsilon \sim \mathcal{N}(0, \sigma_\epsilon) \tag{1.24}$$

This dataset will presumably be well-modeled by an additive kernel which includes all two-way interactions over the first 4 variables, but does not depend on the extra 4 correlated nuisance inputs or the higher-order interactions.

1.7.2 Results

Tables 1.4 to 1.7 show mean performance across 10 train-test splits. Because HKL does not specify a noise model, it could not be included in the likelihood comparisons.

The model with best performance on each dataset is in bold, along with all other models that were not significantly different under a paired t-test. The additive and structure search methods usually outperformed the other methods, especially on regression problems.

The structure search outperforms the additive GP, but at the cost of a slow search over kernels. The additive GP performed best on datasets well-explained by low orders of interaction, and approximately as well as the SE-GP model on datasets which were well explained by high orders of interaction (see table 1.1). Because the additive GP is a superset of both the GP-1st model and the SE-GP model, instances where the

Table 1.4 Regression mean squared error

Method	bach	concrete	pumadyn-8nh	servo	housing
Linear Regression	1.031	0.404	0.641	0.523	0.289
GP-1st	1.259	0.149	0.598	0.281	0.161
HKL	**0.199**	0.147	0.346	0.199	0.151
GP Squared-exp	**0.045**	0.157	0.317	0.126	**0.092**
GP Additive	**0.045**	**0.089**	**0.316**	**0.110**	0.102
Structure Search	**0.044**	**0.087**	**0.315**	**0.102**	**0.082**

Table 1.5 Regression negative log-likelihood

Method	bach	concrete	pumadyn-8nh	servo	housing
Linear Regression	2.430	1.403	1.881	1.678	1.052
GP-1st	1.708	0.467	1.195	0.800	0.457
GP Squared-exp	**−0.131**	0.398	0.843	0.429	0.207
GP Additive	**−0.131**	**0.114**	**0.841**	**0.309**	0.194
Structure Search	**−0.141**	**0.065**	**0.840**	**0.265**	**0.059**

Table 1.6 Classification percent error

Method	breast	pima	sonar	ionosphere	liver	heart
Logistic Regression	7.611	24.392	26.786	16.810	45.060	**16.082**
GP-1st	**5.189**	**22.419**	**15.786**	**8.524**	**29.842**	**16.839**
HKL	**5.377**	24.261	**21.000**	9.119	**27.270**	**18.975**
GP Squared-exp	**4.734**	**23.722**	**16.357**	**6.833**	**31.237**	**20.642**
GP Additive	**5.566**	**23.076**	**15.714**	**7.976**	**30.060**	**18.496**

Table 1.7 Classification negative log-likelihood

Method	breast	pima	sonar	ionosphere	liver	heart
Logistic Regression	0.247	0.560	4.609	0.878	0.864	0.575
GP-1st	**0.163**	**0.461**	**0.377**	**0.312**	**0.569**	**0.393**
GP Squared-exp	**0.146**	0.478	**0.425**	**0.236**	**0.601**	0.480
GP Additive	**0.150**	**0.466**	**0.409**	**0.295**	**0.588**	**0.415**

additive GP performs slightly worse are presumably due to over-fitting, or due to the hyperparameter optimization becoming stuck in a local maximum. Performance could be expected to benefit from approximately integrating over the kernel parameters.

The performance of HKL is consistent with the results in **?**, performing competitively but slightly worse than SE-GP.

1.7.3 Source Code

Additive Gaussian processes are particularly appealing in practice because their use requires only the specification of the base kernel; all other aspects of GP inference remain the same. Note that we are also free to choose a different covariance function along each dimension.

All of the experiments in this chapter were performed using the standard GPML toolbox, available at `gaussianprocess.org/gpml/code`. The additive kernel described in this chapter is included in the latest release. Code to perform all experiments in this chapter is available at `github.com/duvenaud/additive-gps`

1.8 Conclusion

In this chapter, we presented a tractable GP model consisting of a sum of exponentially-many functions, each depending on a different subset of the inputs. Our experiments indicate that, to varying degrees, such additive structure is useful for modeling real datasets. When it is present, modeling this structure allows our model to perform better than standard GP models. In the case where no such structure exists, the higher-order interaction terms present in the kernel can recover arbitrarily flexible models, as well. The additive GP also affords some degree of interpretability: the variance parameters on each order of interaction indicate which sorts of structure are present the data.

The model class considered in this chapter is a subset of that explored by the structure search presented in section 1.8. Thus additive GPs can be considered a quick-and-dirty structure search, being strictly more limited in the types of structure that it can discover, but much faster and simpler to implement.

Related model classes have been previously explored, most notably smoothing-splines ANOVA, and the support vector ANOVA decomposition. However, these models are difficult to apply in practice because kernel parameters, regularization penalties, and the relevant orders of interaction must all be set by hand or by cross-validation. This

chapter illustrates that the GP framework allows these model choices to be performed automatically.