Contents

Chapter 1

Evaluating Cross Sections with Gaussian Processes

This chapter is dedicated to evaluating cross sections for squark pair production with Gaussian processes. The learning of a benchmark process is considered, and compared to Gaussian processes with changes in the data set, the kernel and different features. One change to the benchmark settings is considered at a time. Then the effect of adding more data in the form of distributed Gaussian processes is investigated, and finally learning curves for the optimal parameters are calculated to decide whether the estimator benefits from adding more data.

1.1 Data Generation

In this section the generation of MSSM-24 training and test data is discussed, following closely the discussion in [5].

Sampling of Data

An MPI parallelized script generates a sample point in parameter space by drawing random values from the distributions in Tab. 1.1. The table contains log and flat priors, and a combination of these is used to cover more of the parameter space. When a parameter point has been sampled, it is run through the program `softpoint.x` which calculates its SUSY spectrum using the `Softsusy 3.6.2`-package [1]. The spectrum is then written to a `slha`-file and given as input to `Prospino 2.1`, which calculates the LO and NLO cross sections according to the method outlined in Section 3.4.1. The relevant features and NLO cross sections are harvested to `.dat`-files, which are used by the Gaussian processes.

Parameter	Log prior range	Flat prior range
M_1	[0,100,4000]	[0,4000]
M_2	[0,100,4000]	[0,4000]
M_3	[0,100,4000]	[0,4000]
A_t	[-4000, -100, 100, 4000]	[-4000, 4000]
A_b	[-4000, -100, 100, 4000]	[-4000, 4000]
A_τ	[-4000, -100, 100, 4000]	[-4000, 4000]
μ	[-4000, -100, 100, 4000]	[-4000, 4000]
m_A^{pole}	[0,100,4000]	[0,4000]
$\tan\beta$	[2, 60]	[2, 60]
m_{L_1}	[0, 100, 4000]	[0, 4000]
m_{L_2}	[0, 100, 4000]	[0, 4000]
m_{L_3}	[0, 100, 4000]	[0, 4000]
m_{e_1}	[0, 100, 4000]	[0, 4000]
m_{e_2}	[0, 100, 4000]	[0, 4000]
m_{e_3}	[0, 100, 4000]	[0, 4000]
m_{Q_1}	[0, 100, 4000]	[0, 4000]
m_{Q_2}	[0, 100, 4000]	[0, 4000]
m_{Q_3}	[0, 100, 4000]	[0, 4000]
m_{u_1}	[0, 100, 4000]	[0, 4000]
m_{u_2}	[0, 100, 4000]	[0, 4000]
m_{u_3}	[0, 100, 4000]	[0, 4000]
m_{d_1}	[0, 100, 4000]	[0, 4000]
m_{d_2}	[0, 100, 4000]	[0, 4000]
m_{d_3}	[0, 100, 4000]	[0, 4000]

Table 1.1: Table showing the sampling intervals used for the parameters when sampling the MSSM-24 model, where the soft breaking scale is set to $Q = 1$ TeV. The log priors have three and four limit values, which are of the form [start_flat, start_log, end_log] and [start_log, start_flat, start_log, end_log]. All values in GeV except $\tan\beta$ which is unitless. Table from [5].

Priors

To get a reasonable distribution in parameter space it is necessary to use an objective prior distribution of parameters. This means that priors are assigned according to a set of principles of how information should be encoded in a probability distribution. More details on objective priors can be found in [4].

The first of these principles is the *transformation group invariance*, which states that the probability distribution should be invariant under any transformation that is considered irrelevant to the problem. In other words, the *pdf* should satisfy

$$\pi(x|I)dx = \pi(x+a|I)d(x+a) \qquad \Rightarrow \qquad \pi(x|I) = \pi(x+a|I), \tag{1.1}$$

where a is some translation. This is often referred to as a uniform or *flat prior*. Invariance under scale transformations, which are transformations that introduce a scale to the problem $m \to m' = cm$, requires

$$\pi(m|I)dm = \pi(cm|I)cdm, \tag{1.2}$$

which is satisfied if $\pi(m|I) \propto 1/m$. Since this corresponds to $\pi(\log m|I)$ being flat, it is called the *log prior*.

The flat prior covers the edges of parameter space well, while a log prior covers the innermost points [1]. Therefore, a combination of the log and flat priors is used in order to properly cover parameter space. To avoid divergence of the log prior close to zero this region is covered by a flat prior. The limits on the priors are `[start, flat_start, flat_end, end]` for priors that include negative values, and `[flat_start, start, end]` for priors with only positive values. An illustration of a prior with positive values is shown in Fig. 1.1, where a flat distribution is used close to $x = 0$ to avoid divergences.

The weak scale MSSM model used in this project, MSSM-24, requires a soft breaking scale Q. This scale is set to $Q = 1$ TeV. It is also worth noting that the parameter space for the cross sections is significantly reduced from that of the MSSM-24. The cross sections depend on the values $m_{\tilde{g}}$, $m_{\tilde{q}}$, $\tilde{g}_s$, $\hat{g}_s$ and s. Since only first and second generation squarks are considered, this contributes with 8 masses [2] which combined with the 4 other parameters reduces the parameter space to 12 dimensions. The parameter space is thus better sampled than it would appear from the 24 parameters in MSSM-24.

Data Quality

To ensure the data is properly distributed in parameter space data quality plots are generated. In Fig. 1.2 distributions for $m_{\tilde{g}}$, $m_{\tilde{d}_L}$ and $m_{\tilde{u}_L}$ are shown, as the

[1] The points close to 0.

[2] 4 quarks with a pair of lefthanded and righthanded squarks each.

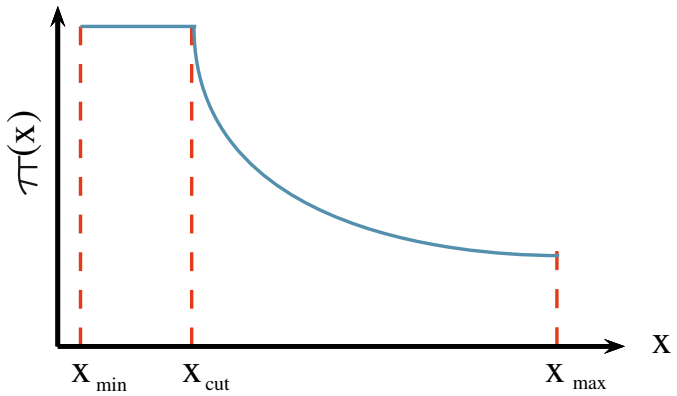


Figure 1.1: Illustration of the log prior distribution $\pi(x)$. Around $x = 0$ the prior would blow up, so at a limit x_{cut} a flat prior is used.

benchmark processes will be for the production of $\tilde{u}_L\tilde{d}_L$ and $\tilde{d}_L\tilde{d}_L$. The scatter plots use 40 000 points. All parts of the parameter space seem to be covered, although the density of points is higher for small masses. This could affect predictions with few training points.

The panel in Fig. 1.2b shows a scatter plot of $m_{\tilde{d}_L}$ versus $m_{\tilde{u}_L}$. The plot is almost linear, which comes from the mass splittings of the MSSM. Same-generation left-handed squarks come in $SU(2)$-doublets, and their masses are predominantly determined by *one* mass parameter, namely Q_i for generation i. Right-handed squarks, on the other hand, get their masses from different parameters $m^2_{\tilde{d}_i}$ and $m^2_{\tilde{u}_i}$, and so are independent of each other. The mass splitting between same-generation left-handed squarks, *e.g.* $\tilde{d}_L$ and $\tilde{u}_L$, was given in Sec. 1.4.5,

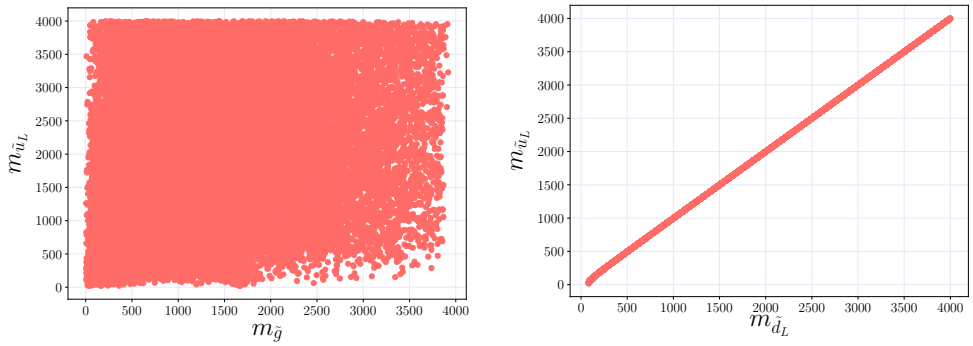
$$m^2_{\tilde{d}_L} - m^2_{\tilde{u}_L} = -\cos 2\beta \; m^2_W,$$

where $-\cos 2\beta \; m^2_W$ is relatively small ($\leq$ 15 GeV). It is therefore a possibility that training on processes with same-generation left-handed squarks, $\tilde{u}_L\tilde{d}_L$, $\tilde{s}_L\tilde{c}_L$, will effectively be training on a single squark mass, such as $m_{\tilde{d}_L}$, in stead of two, such as $m_{\tilde{d}_L}, m_{\tilde{u}_L}$.

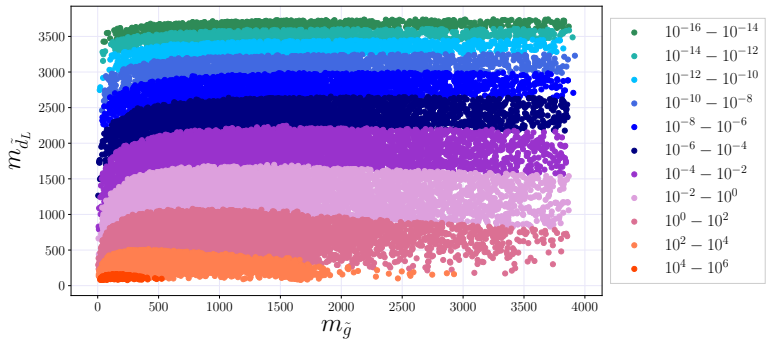
Noise

The data contains some noise that originates in the `Prospino 2.1` calculation. In a parameter point chosen at random the relative error has a standard deviation of $\varepsilon = 0.002$. This error fluctuates little between parameter points, so it is considered a good approximation for the order of magnitude of errors in all points. The goal is now to incorporate this information in the Gaussian process. For that purpose the following relation is considered,

$$Y_i = y_i^{true} + \epsilon_i = y_i^{true}(1 + \varepsilon_i), \tag{1.3}$$



(a) $m_{\tilde{g}}$-$m_{\tilde{u}_L}$ **(b)** $m_{\tilde{d}_L}$-$m_{\tilde{u}_L}$



(c) $m_{\tilde{g}}$-$m_{\tilde{d}_L}$

Figure 1.2: Data quality plots of the distribution of mass parameters $m_{\tilde{d}_L}$, $m_{\tilde{u}_L}$ and $m_{\tilde{g}}$ for 40 000 points. In **(c)** different orders of magnitude of the cross sections for $\tilde{d}_L\tilde{d}_L$ are shown in different colors, indicating that there are few points in the very high cross sections $\sigma \propto 10^4 - 10^6$, and that lower cross sections are more spread across the gluino mass spectrum.

where cross sections provided by Prospino are denoted as Y_i, real cross sections as y_i^{true} and $\varepsilon_i \sim \mathcal{N}(0, \varepsilon^2)$. The distribution of Y_i can thus be written as

$$Y_i = \mathcal{N}(y_i^{true}, (\varepsilon y_i^{true})^2), \tag{1.4}$$

where the only random variable is ε. The target values will in fact be the logarithm of cross sections, and changing variables to $\log_{10} Y_i$ gives

$$X_i = \log_{10} Y_i \to Y_i = 10^{X_i} \tag{1.5}$$

$$P_{X_i}(X_i) = P_{Y_i}(Y_i)\Big|\frac{\partial Y_i}{\partial X_i}\Big| \tag{1.6}$$

$$= P_{Y_i}(y_i)10^{X^i}\log 10. \tag{1.7}$$

So the relevant distribution is in fact

$$X_i = \log_{10} Y_i = \log_{10} y_i^{true} + \log_{10}(1 + \mathcal{N}(0, \varepsilon^2)),$$

where the following expansion can be made

$$\log_{10}(1 + \mathcal{N}(0, \varepsilon^2)) \simeq \frac{\mathcal{N}(0, \varepsilon^2)}{\log 10} - \frac{\mathcal{N}(0, \varepsilon^2)^2}{\log 100} + ... \tag{1.8}$$

Since the leading order term is dominant for small ε the logarithm of the cross section may be approximated as

$$X_i \simeq \log_{10} y_i^{true} + \frac{1}{\log 10}\mathcal{N}(0, \varepsilon^2) \tag{1.9}$$

Equation (1.9) is now on the form of the Gaussian noise model discussed in Sec. **??**. The error distribution has a standard deviation $\varepsilon = 2 \cdot 10^{-3}$, so the variance of the Gaussian distributed noise should be

$$\left(\frac{\varepsilon}{\log 10}\right)^2 = \frac{(2 \cdot 10^{-3})^2}{(\log 10)^2} = \frac{4 \cdot 10^{-6}}{5.301} \simeq 7.544 \cdot 10^{-7}.$$

The Gaussian noise term predicted by the GP, $\varepsilon_{GP} \sim \mathcal{N}(0, \sigma_n^2)$, should therefore have a variance of the order $\mathcal{O}(10^{-7} - 10^{-6})$.

1.2 Dataset Transformations

The plot in Fig. 1.2c indicates that cross sections, especially those of the order $\mathcal{O}(1$ fb) and lower, are very spread as a function of $m_{\tilde{g}}$. This is more evident in the upper left panel of Fig. 1.3, where the cross section for the production of $\tilde{d}_L\tilde{d}_L$, σ, is plotted as a function of the gluino mass. The upper left panel shows σ as a function of the squark mass $m_{\tilde{q}}$, which is more defined but still has some spread. This section will be devoted to reducing the spread caused by the gluino mass dependency.

Scaling Functions

As discussed in Sec. 2.4, the partonic cross sections can be written in terms of scaling functions f

$$\hat{\sigma}_{ij} = \frac{\alpha_s^2(Q^2)}{m^2}\left\{ f_{ij}^B(\eta, r) + 4\pi\alpha_s(Q^2)\left[f_{ij}^{V+S}(\eta, r, r_t) + f_{ij}^H(\eta, r) + \bar{f}_{ij}(\eta, r)\log\left(\frac{Q^2}{m^2}\right)\right]\right\}, \tag{1.10}$$

where

$$\eta = \frac{s}{m^2} - 1, \qquad r = \frac{m_{\tilde{g}}^2}{m_{\tilde{q}}^2}, \qquad r_t = \frac{m_t^2}{m^2},$$

where $m = (\sqrt{p_1^2} + \sqrt{p_2^2})/2$ is the average mass of the particles produced. The scaling functions are the different contributions to the cross section, as explained in Sec. 2.4. As the total cross section only differs from the partonic cross section by an integral over parton distribution functions, the mass dependencies in Eq. (1.10) are relevant for the total cross sections as well.

The energy near the threshold is the base for an important part of the contributions to the cross section [2]. In this region the scaling functions can be expanded in the low velocity of produced particles β, leading to the following expressions [2]

$$\begin{aligned}
f_{qq}^B &= \frac{8\pi\beta m_{\tilde{q}}^2 m_{\tilde{g}}^2}{27(m_{\tilde{q}}^2 + m_{\tilde{g}}^2)^2}, & f_{q'q}^B &= \frac{8\pi\beta m_{\tilde{q}}^2 m_{\tilde{g}}^2}{9(m_{\tilde{q}}^2 + m_{\tilde{g}}^2)^2} \\
f_{qq}^{V+S} &= f_{qq}^B \frac{1}{24\beta} & f_{q'q}^{V+S} &= f_{q'q}^B \frac{1}{24\beta} \\
f_{qq}^H &= f_{qq}^B\Big[\frac{2}{3\pi^2}\log^2(8\beta^2) - \frac{7}{2\pi^2}\log(8\beta^2)\Big] & f_{q'q}^H &= f_{q'q}^B\Big[\frac{2}{3\pi^2}\log^2(8\beta^2) - \frac{19}{6\pi^2}\log(8\beta^2)\Big] \\
\bar{f}_{qq} &= -f_{qq}^B \frac{2}{3\pi^2}\log(8\beta^2) & \bar{f}_{q'q} &= -f_{q'q}^B \frac{2}{3\pi^2}\log(8\beta^2).
\end{aligned} \tag{1.11}$$

As the main contributions come from this energy region, it may be possible to remove some of the complexity of the function using the expressions in Eq. 1.11. Note that all terms are proportional to f_{qq}^B ($f_{q'q}^B$), which is again proportional to $m_{\tilde{q}}^2 m_{\tilde{g}}^2$. Since the partonic cross section is proportional to $\sigma \propto 1/m^2$, and $m^2 = m_{\tilde{q}}^2$ for squark pair production, this $m_{\tilde{q}}^2$-dependency is automatically cancelled. However, the following transformation can be made

$$\sigma \to \sigma_{m_{\tilde{g}}} = \frac{\sigma}{m_{\tilde{g}}^2}, \tag{1.12}$$

reducing the gluino mass dependency. The lower panels in Fig. 1.3 show $\sigma_{m_{\tilde{g}}}$ as a function of $m_{\tilde{g}}$ and $m_{\tilde{q}}$. The spread as a function of the squark mass is much

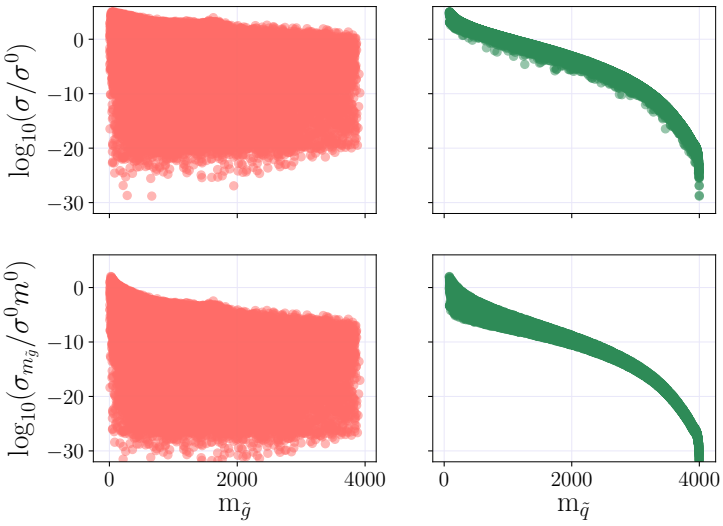


Figure 1.3: The cross section, σ, and the modified cross section, $\sigma_{m_{\tilde{g}}}$, as functions of the gluino mass $m_{\tilde{g}}$ and squark mass $m_{\tilde{q}} = m_{\tilde{d}_L}$ for the production of $\tilde{d}_L\tilde{d}_L$. The functions have less spread when some of the mass dependency is removed.

smaller, and for high cross sections the shape of $\sigma_{m_{\tilde{g}}}$ as a function of squark and gluino mass are very similar, which may make it easier to find a kernel that fits well in both dimensions. Therefore, the target value in this thesis will be the logarithm of $\sigma_{m_{\tilde{g}}}$. As mentioned, the logarithm is used because of the large span in function values.

In the threshold region, the partonic version of $\sigma_{m_{\tilde{g}}}$ is given by

$$\hat{\sigma}_{ij,m_{\tilde{g}}} = \frac{8\pi\beta\alpha_s^2(Q^2)}{27(m_{\tilde{q}}^2 + m_{\tilde{g}}^2)^2}\left\{1 + 4\pi\alpha_s(Q^2)\left[\frac{1}{24\beta} + \frac{2}{3\pi^2}\log^2(8\beta^2) - \frac{7}{2\pi^2}\log(8\beta^2) - \frac{2}{3\pi^2}\log(8\beta^2)\log\left(\frac{Q^2}{m^2}\right)\right]\right\}. \quad (1.13)$$

Another possibility is to further reduce the mass dependecy, by defining σ_{fac} as

$$\sigma_{fac} = \sigma\frac{(m_{\tilde{g}}^2 + m_{\tilde{q}}^2)^2}{m_{\tilde{g}}^2}. \quad (1.14)$$

This transformation provides an even smoother function, but has been left as further work with GPs.

1.3 Learning the Gaussian Process

In this section a single Gaussian process is learned with benchmark settings, and a selected set of modifications to the benchmark are introduced. The quality of estimators are quantified using plots of the relative deviance defined in Sec. ??.

1.3.1 The Benchmark

In this project the benchmark processes will be the production of $\tilde{d}_L\tilde{d}_L$ and $\tilde{d}_L\tilde{u}_L$. The benchmark settings (BM) are a single GP with 2000 training points and 20 000 test points that uses $m_{\tilde{g}}$ and $m_{\tilde{d}_L}$ as features (as well as $m_{\tilde{u}_L}$ for $\tilde{d}_L\tilde{u}_L$ production). The exponential squared kernel (RBF) with a white noise term is used. The kernel is implemented in `scikit-learn` in the following way

```
kernel_BM = C(constant_value=10,
    constant_value_bounds=(1e-3, 1e4)) * RBF(length_scale =
    np.array([1000, 1000]), length_scale_bounds=(1, 1e6)) +
    WhiteKernel(noise_level=1, noise_level_bounds=(2e-10,1e2))
```

where the features are `(m_gluino, m_dL)` for $\tilde{d}_L\tilde{d}_L$ and `(m_gluino, m_dL, m_uL)` for $\tilde{d}_L\tilde{u}_L$. Note that the length scale of the RBF is given a a vector of the same dimension as the feature vector $\mathbf{x}, \boldsymbol{\ell} \in \mathbb{R}^D$ [3]. This is in case the different features have different characteristic length scales, as they appear to have from the lower panels in Fig. 1.3.

$m(\varepsilon_i)$ and $\mathbb{V}(\epsilon_i)$, as defined in Eq. (??)-(??), of the decades are calculated and plotted for the BM settings in Fig. 1.4 for $\tilde{d}_L\tilde{d}_L$. For $\tilde{d}_L\tilde{u}_L$ the results are very similar, and therefore not shown here. The optimal kernel parameters found by the GP are found in Tab. 1.2 - 1.3, while computation times on a laptop are found in Tab. 1.4. The predicted noise level variances, α, are very high for both processes, at $\alpha = 0.47$ for $\tilde{d}_L\tilde{d}_L$ and $\alpha = 0.593$ for $\tilde{d}_L\tilde{u}_L$, which is far from the expected value of $\alpha \sim 7.544 \cdot 10^{-7}$. In addition, the prediction is very unstable, with the estimator over prediction for low cross sections.

1.3.2 Outliers

Calculations in `Prospino 2.1` set some NLO terms to zero as discussed in Sec. ??. This creates outliers in the dataset, as shown in Fig. 1.5, where the outliers can be seen as a cluster of points for large masses, well below the other cross sections. These points have zero cross sections, which are set to 10^{-32} fb in the calculation to avoid divergences. A GP is trained on a dataset where outliers have been removed, otherwise the settings are the BM. Removing the ouliers makes the

[3] This example is for $\tilde{d}_L\tilde{d}_L$ production, for $\tilde{d}_L\tilde{u}_L$ production $D = 3$ as $m_{\tilde{u}_L}$ is inluded as well.

	C	$\ell_{m_{\tilde{g}}}$	$\ell_{m_{\tilde{d}_L}}$	$\ell_{\bar{m}}$	α
BM	2981	5470	2190		0.47
No outliers	9702	5740	215		0.00372
$\sigma > 10^{-16}$ fb	515	1170	998		0.0036
$\bar{m}$	1095	1190	200	846	0.0000119
Matern	1000	30200	8600		0.462

Table 1.2: Optimal kernel parameters for different settings for $\tilde{d}_L\tilde{d}_L$.

	C	$\ell_{m_{\tilde{g}}}$	$\ell_{m_{\tilde{d}_L}}$	$\ell_{m_{\tilde{u}_L}}$	$\ell_{\bar{m}}$	α
BM	2490	5870	4000	2220		0.593
No outliers	6400	4420	3100	240		0.00348
$\sigma > 10^{-16}$ fb	10000	1540	2900	2150		0.0031
$\bar{m}$	3806	1340	3590	251	748	0.0000119
Matern	1000	31100	1000000	8260		0.585

Table 1.3: Optimal kernel parameters for different settings for $\tilde{d}_L\tilde{u}_L$.

	Time $\tilde{d}_L\tilde{d}_L$	Time $\tilde{d}_L\tilde{u}_L$
BM	00:07:48	00:08:40
Outliers	00:08:42	00:11:20
Cut	00:07:24	00:11:07
Features	00:11:20	00:16:37
Kernel	00:07:28	00:13:05

Table 1.4: Computation times for GP with 2000 training points and 20 000 test points on a laptop with 4 cores.

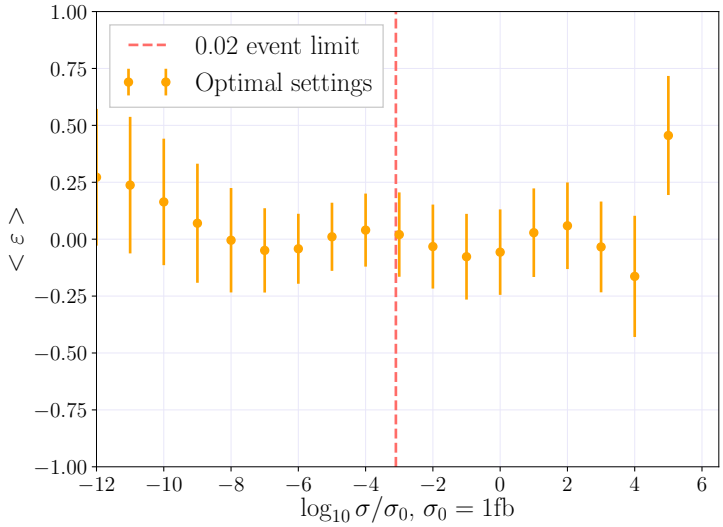


Figure 1.4: The mean and variance of the relative deviances, $m(\varepsilon_i)$ and $\mathbb{V}(\varepsilon_i)$, as a function of $i = \log_{10}\sigma/\sigma_0$, for the process $\tilde{d}_L\tilde{d}_L$ with benchmark settings. 2000 training points and 20 000 test points were used on a regular GP, with the final kernel parameters $C = 2981$, $\ell_{m_{\tilde{g}}} = 5470$, $\ell_{m_{\tilde{d}_L}} = 2190$ and $\alpha = 0.47$. Features are the physical masses $m_{\tilde{g}}$ and $m_{\tilde{d}_L}$.

prediction much better for small cross sections, but also stabilizes the prediction for larger values, as can be seen in Fig. 1.6 a) where the prediction without outliers is compared to the BM. The predicted noise levels are significantly reduced with the removal of outliers for both processes, from $\alpha = 0.47$ to $\alpha = 0.00372$ for $\tilde{d}_L\tilde{d}_L$ and from $\alpha = 0.593$ to $\alpha = 0.00348$ for $\tilde{d}_L\tilde{u}_L$. This implies that including the outliers leads the GP to underfit, which means that much of the signal is considered noise.

1.3.3 Cuts on Cross Sections

Smooth functions are easier to fit with Gaussian processes, and the target values are very steep functions of large squark masses. This can be seen from the righthand panels in Fig. 1.3. In addition, small target values comprise the regions with the most spread as a function of the gluino mass. Since the limit for 0.02 events is at $\sigma = 10^{-3}$ fb [4], a lower cut is set at $\sigma_{cut} = 10^{-16}$ fb. The cut excludes all cross sections lower than σ_{cut} from both training and testing. The resulting error distributions for $\tilde{d}_L\tilde{d}_L$ are shown in Fig. 1.6 b), and the optimal kernel parameters are found in Tab. 1.2 - 1.3. Noise levels are further reduced from

[4] Cross sections with lower values than this predict less than 0.02 events, and are therefore less interesting in this project.

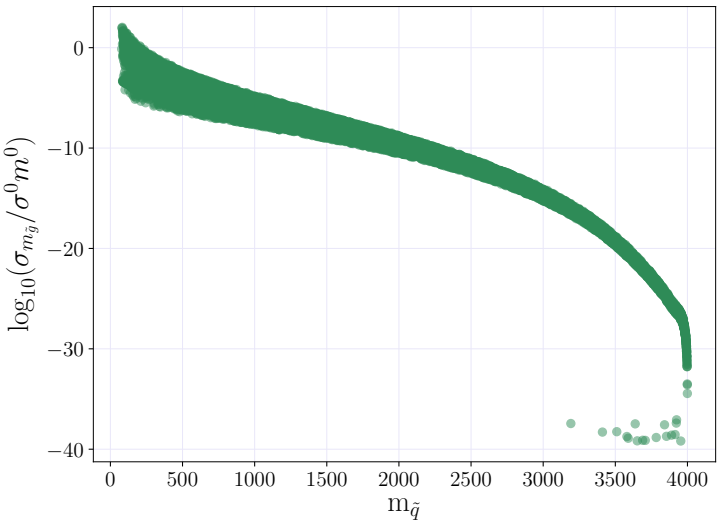


Figure 1.5: Plot of $\log(\sigma_{m_{\tilde{g}}})$ for $\tilde{d}_L\tilde{d}_L$ as a function of $m_{\tilde{d}_L}$, where outliers are included. The outliers are originally $\sigma = 0$ fb, but are set to $\sigma = 10^{-32}$ fb so as to avoid infinities.

the case where outliers are removed, with the variance going from $\alpha = 0.00372$ to $\alpha = 0.00336$ for $\tilde{d}_L\tilde{d}_L$ and from $\alpha = 0.00348$ to $\alpha = 0.0031$ for $\tilde{d}_L\tilde{u}_L$. The estimated noise level seems to be approaching the expected theoretical value, indicating that the exclusion of low cross sections improves the prediction. The error in the prediction of the highest targets is significantly improved from the cases of the BM and the removed outliers. Training and testing only on high values renders the prediction very stable for all (included) orders of magnitude.

1.3.4 Features

`Prospino 2.1` calculates the NLO cross section for the mean of the squark masses, $\bar{m}$, and uses this to find the K-factor. The K-factor is then multplied with the LO cross sections for non-degenerate squark masses to find the individual NLO terms, as discussed in Sec. **??**. Adding the mean squark mass as a feature could therefore improve the prediction. The features used in this section are

```
features_dLdL = (m_gluino, m_dL, m_mean)
features_dLuL = (m_gluino, m_dL, m_uL, m_mean)
```

The optimal kernel values are found in Tab. 1.2 - 1.3. Adding the mean mass as a feature reduces the estimated noise level considerably, and estimates the same level, $\alpha = 1.19 \cdot 10^{-5}$, for $\tilde{d}_L\tilde{d}_L$ and $\tilde{d}_L\tilde{d}_L$. This is an indication that the prediction

is improved, considering the two processes should have approximately the same level of noise, as discussed in Sec. 1.1.

Further, the estimator predicts a shorter length scale for $\bar{m}$ than for $m_{\tilde{g}}$. This implies that there is a higher dependence on the mean mass than the gluino mass, since GPs attribute large length scales to irrelevant features. The resulting error distributions for $\tilde{d}_L\tilde{d}_L$ are shown in Fig. 1.6 c) and compared to the BM. With some exceptions at $\log_{10}\sigma/\sigma_0 \in [2,4]$, where the variances are very large, adding $\bar{m}$ as a feature gives a mean of almost zero and very small variances for cross sections over the 0.02 event limit.

1.3.5 Kernel

The exponential squared kernel is very smooth, while the Matérn kernel has a hyperparameter ν to control its smoothness. It is sometimes argued that this makes Matérn a better kernel for physical processes. In this section the Matérn kernel with $\nu = 1.5$ is compared to the BM RBF kernel, and the resulting error distributions for $\tilde{d}_L\tilde{d}_L$ are found in Fig. 1.4 d). The hyperparameter ν is set to 1.5 as this is one of the values for which covariances are quick to calculate, as mentioned in Sec. **??**. The predictions are somewhat more stable for high cross sections than with the RBF kernel. For low cross sections the error distributions have larger variances, but as this is currently not the region of interest the Matérn kernel is considered a better fit than the RBF. As can be seen from the optimal kernel values in Tab. 1.2 - 1.3, the Matérn kernel predicts a slightly lower noise level than the RBF. The noise variances go from $\alpha = 0.47$ with RBF to $\alpha = 0.462$ with Matérn for $\tilde{d}_L\tilde{d}_L$, and from $\alpha = 0.593$ with RBF to $\alpha = 0.585$ with Matérn for $\tilde{d}_L\tilde{u}_L$.

1.3.6 Optimal Settings

A combination of the improved settings found in the foregoing sections is tested in this section. The cummulative settings are; a single GP with outlier points removed; a lower cut on cross sections $\sigma > 10^{-16}$ fb; the Matérn kernel with $\nu = 1.5$; and the gluino mass, the relevant squark mass(es) and the mean of all squark masses as features. The resulting error distributions are found in Fig. 1.7. The prediction is now very good, as all error distribution variances, $\mathbb{V}(\varepsilon_i)$, are less than 5%. The computation with optimal settings for 2000 training points takes 00:09:30 for $\tilde{d}_L\tilde{d}_L$ and 00:10:28 for $\tilde{d}_L\tilde{u}_L$ on a regular laptop with 4 cores.

Noise

Since the noise level is known to some degree, it is worth testing whether this knowledge be used in Gaussian processes. As previously shown, the variance of the Gaussian distribution of noise is around $\alpha_{fix} = 7.544 \cdot 10^{-7}$. In `scikit-learn`

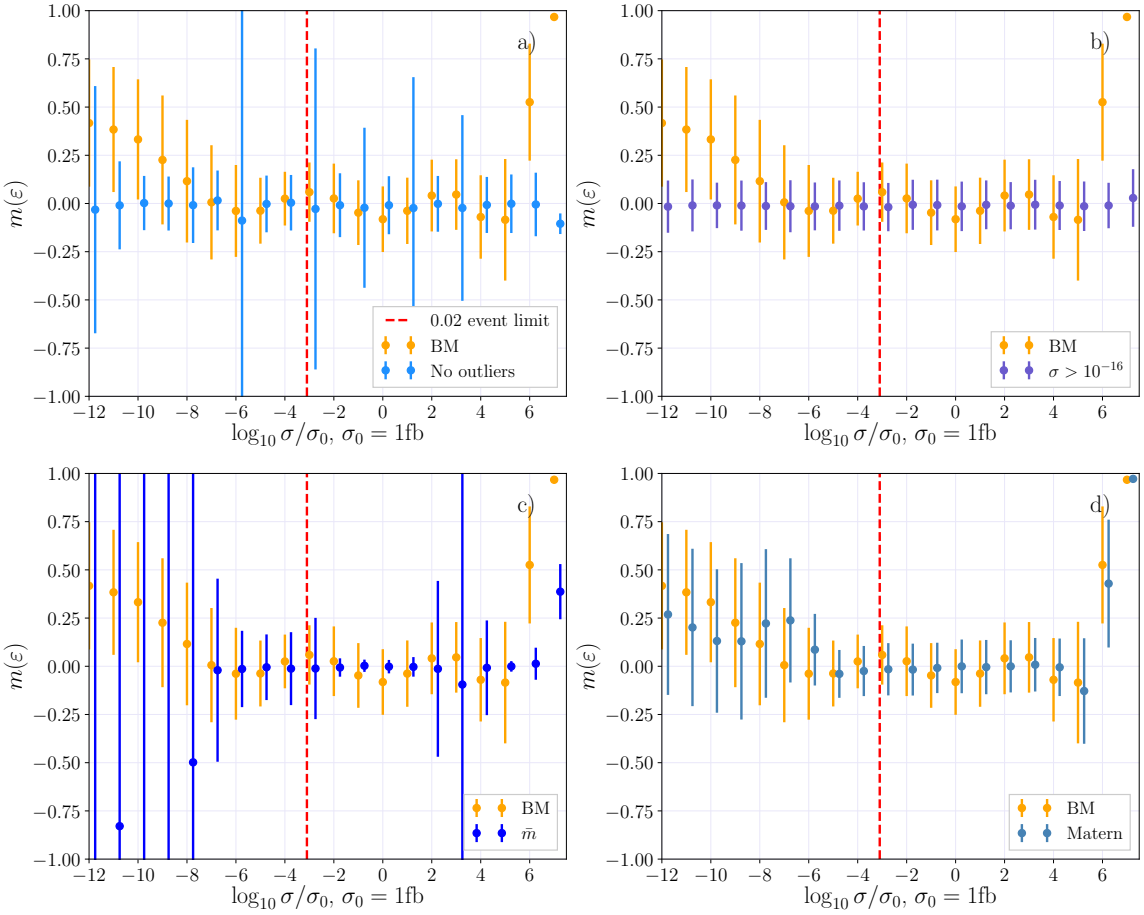


Figure 1.6: The distribution of the relative deviance ε as a function of the logarithm of the normalized cross section for $\tilde{d}_L\tilde{d}_L$ with a) benchmark settings (orange) and removed outliers (blue); b) benchmark settings (orange) and a lower cut on cross sections (blue); c) benchmark settings (orange) and the added feature $\bar{m}$ (blue); d) the benchmark settings (orange) and the Matérn kernel (blue). Benchmark settings are abbreviated BM.

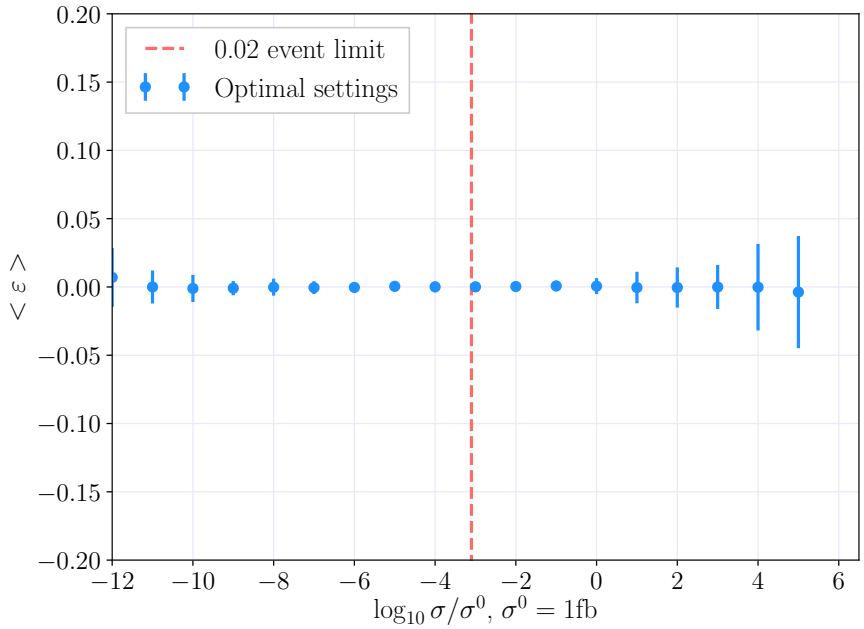


Figure 1.7: The distribution of the relative deviance ε as a function of the logarithm of the normalized cross section for $\tilde{d}_L\tilde{d}_L$ with the optimal settings; a single GP with 2000 training points that uses the Matérn kernel with $\nu = 1.5$. Outliers are removed and a lower cut at $\sigma = 10^{-16}$ fb is introduced. The features are $m_{\tilde{g}}$, $m_{\tilde{d}_L}$ and $\bar{m}$, and the target is $\sigma_{m_{\tilde{g}}}$. All error distibutions have σ_{std} smaller than 5%.

	C	$\ell_{m_{\tilde{g}}}$	$\ell_{m_{\tilde{d}_L}}$	$\ell_{m_{\tilde{u}_L}}$	$\ell_{\bar{m}}$	α
$\tilde{d}_L\tilde{d}_L$	750	30500	17400		74800	$1 \cdot 10^{-5}$
$\tilde{d}_L\tilde{d}_L$	1000	28300	18300		69900	$7.544 \cdot 10^{-7}$ (fixed)
$\tilde{d}_L\tilde{u}_L$	1000	30200	79600	18500	89000	$5.63 \cdot 10^{-6}$
$\tilde{d}_L\tilde{u}_L$	1000	29100	66200	18300	76000	$7.544 \cdot 10^{-7}$ (fixed)

Table 1.5: Kernel parameters for the optimal settings with the noise level estimated by the GP, and given as a constant $\alpha = 7.544 \cdot 10^{-7}$.

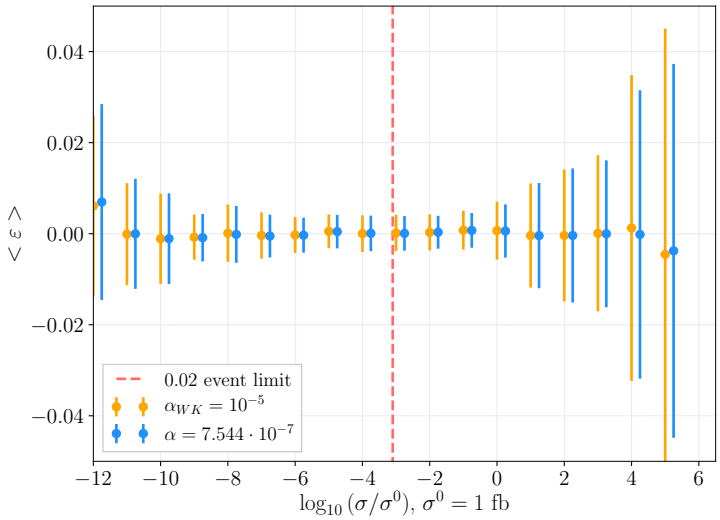


Figure 1.8: The distribution of the relative deviance ε as a function of the logarithm of the normalized cross section for $\tilde{d}_L\tilde{d}_L$ with the optimal settings. In one case the noise level is estimated by the GP (orange), and in the other it is given as a parameter α (blue).

an option to letting the `WhiteKernel` estimate the level of noise is to specify the noise by passing it as `alpha` to the Gaussian process regressor function

```
gp = GaussianProcessRegressor(kernel=kernel, alpha=7.544e-7)
```

For the optimal settings the `WhiteKernel` predicts values close to α_{fix}, with $\alpha = 10^{-5}$ for $\tilde{d}_L\tilde{d}_L$ and $5.63 \cdot 10^{-6}$ for $\tilde{d}_L\tilde{u}_L$. The remaining kernel parameters therefore hardly change when α is fixed, as seen in Tab. 1.5. As expected, the prediction changes very little. A marginal improvement can be seen in the variance for α_{fix} in Fig. 1.8. For calculations with few training points the computation time is not affected in any great way, but for larger datasets the removal of α from the kernel may affect the time.

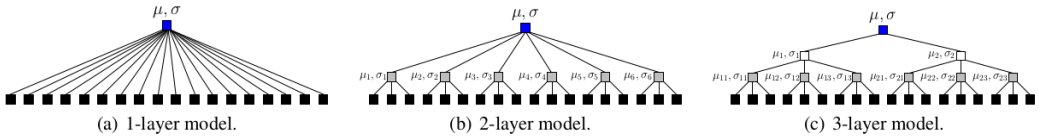


Figure 1.9: Computational graphs of hierarchical product-of-expert models. Main computations are at the leaf nodes (GP experts in black). All other nodes recombine computations from their children nodes. The blue node at the top represents the final prediction. Figure from [3].

1.4 Distributed Gaussian Processes

Limitations of Gaussian Processes

The biggest weakness of Gaussian processes is that they scale poorly with the size of the training data set, n, and testing data set, m, as discussed in Sec. **??**. The training and predicting scale as $\mathcal{O}(n^3)$ and $\mathcal{O}(m^2)$, respectively, giving GPs a practical limit of $\mathcal{O}(10^4)$.

In [3] a way of scaling GPs to large data sets is proposed, in the form of a robust Bayesian Comittee Machine (rBCM). This method is based on the product-of-experts and Bayesian Comittee Machine, and has the advantage of providing an uncertainty for the prediction.

1.4.1 Product-of-Experts

Product-of-expert (PoE) models are a way of parallelising large computations. They combine several independent computations on subsets of the total data, called 'experts'. In the case of distributed Gaussian processes each expert performs GP on a subset of the training data, and the predictions on a common test set are combined.

Consider the training data set $\mathcal{D} = \{\mathbf{X}, \mathbf{y}\}$, which is partitioned into M subsets $\mathcal{D}^{(k)} = \{\mathbf{X}^{(k)}, \mathbf{y}^{(k)}\}$, $k = 1, ..., M$. Each GP expert does learning on its training data set $\mathcal{D}^{(k)}$, then predictions are combined at the parent node. This node could also be considered an expert for a PoE with several layers, see Fig. 1.9.

1.4.2 Algorithm

The M experts are assumed to be independent [3], effectively block-diagonalizing the covariance matrix. The marginal likelihood from Eq. **??** now factorizes into the product of M individual terms because of the independence assumption. The log marginal likelihood is then

$$\log P(\mathbf{y}^{(k)}|\mathbf{X}^{(k)}, \boldsymbol{\theta}) = -\frac{1}{2}\mathbf{y}^{(k)}(\mathbf{K}_{\psi}^{(k)} + \sigma_{\varepsilon}^2\mathbf{I})^{-1}\mathbf{y}^{(k)} - \frac{1}{2}\log|\mathbf{K}_{\psi}^{(k)} + \sigma_{\varepsilon}^2\mathbf{I}|, \qquad (1.15)$$

where $a^{(k)}$ is the quantity corresponding to the kth expert. Computing the LML now entails inverting a $n_k \times n_k$ matrix, $(\mathbf{K}_\psi^{(k)} + \sigma_\varepsilon^2 \mathbf{I})$, which requires time $\mathcal{O}(n_k^3)$ and memory consumption $\mathcal{O}(n_k^2 + n_k D)$ for $\mathbf{x} \in \mathbb{R}^D$. For $n_k \ll N$, this reduces the computation time and memory use considerably, and allows for parallel computing.

Several methods for prediction are discussed in [3], but here only the robust Bayesian Comittee Machine is introduced. The PoE predicts a function value f^* at a corresponding test input $\mathbf{x}^*$ according to the predictive distribution

$$P(f^*|\mathbf{x}^*, \mathcal{D}) = \frac{\prod_{k=1}^M P_k^{\beta_k}(f^*|\mathbf{x}^*, \mathcal{D}^{(k)})}{P^{-1+\sum_k \beta_k}(f^*|\mathbf{x}^*)}, \tag{1.16}$$

where the parameters β_k control the importance of the individual experts, but also the how strong the influence of the prior is. In the article, these are chosen according to the predictive power of each expert at $\mathbf{x}^*$. More specifically, β_k is the change in differential entropy between the prior $p(f^*|\mathbf{x}^*)$ and the posterior $p(f^*|\mathbf{x}^*, \mathcal{D}^{(k)})$, which can be calculated as

$$\beta_k = \frac{1}{2}(\log \sigma_{**}^2 - \log \sigma_k^2(\mathbf{x}^*)), \tag{1.17}$$

where $\sigma_{**}^2 = k(\mathbf{x}^*, \mathbf{x}^*)$ is the prior variance of the test point, and $\sigma_k^2(\mathbf{x}^*)$ is the predictive variance of the kth expert given by Eq. **??**.

The combined predictive mean and variance are denoted μ_*^{rbcm} and σ_*^{rbcm}

$$\mu_*^{rbcm} = (\sigma_*^{rbcm})^2 \sum_k \beta_k \sigma_k^{-2}(\mathbf{x}_*)\mu_k(\mathbf{x}_*), \tag{1.18}$$

$$(\sigma_*^{rbcm})^{-2} = \sum_{k=1}^M \beta_k \sigma_k^{-2}(\mathbf{x}_*) + \big(1 - \sum_{k=1}^M \beta_k\big)\sigma_{**}^{-2}. \tag{1.19}$$

Implementation

The combined predictive mean and variance in Eq. 1.18 - 1.19 were implemented in `Python` using the `scikit-learn` library's existing framework for regular Gaussian processes. The algorithm was parallelised, so that each expert can learn and predict in parallel, before being combined to the final prediction. Pseudocode for the implementation is found in Algorithm 1, which takes the training data X, $\mathbf{y}$, the initial kernel k, the noise level variance σ_n^2, the number of experts $N_{experts}$ and the test features $\mathbf{x}^*$ as input, and computes the combined predictive mean and variance, μ_*^{rbcm}, ${\sigma_*^{rbcm}}^2$.

For parallelisation the `scikit-learn` function `Parallel` from `joblib` was used, which runs Python functions as pipeline jobs. It uses the Python function `multiprocessing` as a backend. An example of usage with 3 parallel jobs is as follows

```
>>> from joblib import Parallel, delayed
>>> from math import sqrt
>>> Parallel(n_jobs=3)(delayed(sqrt)(i**2) for i in range(10))
[0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0]
```

where `delayed` is a simple trick to be able to create a tuple with a function-call syntax.

Data: $N_{experts}$ (number of experts), X (inputs), $\mathbf{y}$ (targets), k (initial kernel), σ_n^2 (noise level), $\mathbf{x}^*$ (test input)
Split training data into N subsets: $X_k, \mathbf{y}_k$;
for *each expert* **do**
  Fit GP to training data $X_k, \mathbf{y}_k$;
  Predict μ_*, σ_*^2 for $\mathbf{x}^*$ using GP ;
  $\sigma_{**}^2 = k(x^*, x^*)$;
end
for *each expert* **do**
  $\beta = \frac{1}{2}(\log(\sigma_{**}^2) - \log(\sigma_*^2))$;
  $(\sigma_*^{rbcm})^{-2} += \beta\sigma^{-2} + \left(\frac{1}{n_{experts}} - \beta\right)\sigma_{**}^{-2}$
end
for *each expert* **do**
  $\mu_*^{rbcm} += (\sigma_*^{rbcm})^2 \beta \sigma_*^{-2} \mu_*$
end
Result: Approximative distribution of $f_* = f(\mathbf{x}_*)$ with mean μ_*^{rbcm} and variance $(\sigma_*^{rbcm})^2$.

Algorithm 1: Pseudocode for distributed Gaussian processes on a single test point $\mathbf{x}_*$. For the fit and prediction of each GP expert Algorithm (**??**) is used.

1.4.3 Benchmark

The benchmark function for parallelised distributed Gaussian processes is

$$f(x_1, x_2) = 4x_1 x_2,$$

where the vectors $\mathbf{x} = (x_1, x_2)$ were drawn from a random normal distribution using the `numpy` function `random.randn`. Gaussian processes implemented by `scikit-learn` in the function `GaussianProcessRegressor` were compared to distributed Gaussian processes with 4 experts. 2000 training points and 1000

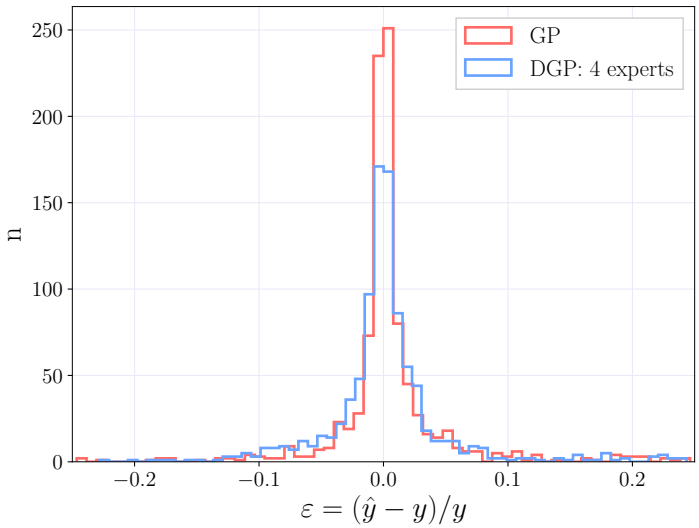


Figure 1.10: Histogram of the relative deviance between true value y and predicted value $\hat{y}$ for Gaussian process regression (GP) and Distributed gaussian process regression (DGP) for the function $f(x_1, x_2) = 4x_1x_2$.

test points were used, and the resulting times for the GP and DGP were

$$\text{Gaussian processes time: } 154.12 \text{ s} \tag{1.20}$$

$$\text{Distributed Gaussian processes time: } 5.61 \text{ s} \tag{1.21}$$

Histograms of the relative deviances for Gaussian processes (GP) and Distributed Gaussian processes (DGP) are found in Fig. (1.10).

1.5 Distributed Gaussian Processes for Cross Sections

In this section the distributed Gaussian processes described in the previous section are applied to estimators with the benchmark settings from Sec. 1.3.1 to scale the problem to larger training sets. The prediction for a single expert with 8000 training points is compared to the combined prediction of 4 experts with 2000 training points each, both in terms of computation time and quality of prediction.

Adding Experts

The addition of experts with 2000 training points improves the prediction from Sec. 1.3.1 significantly, with very little increase in computational cost. In Fig. 1.11

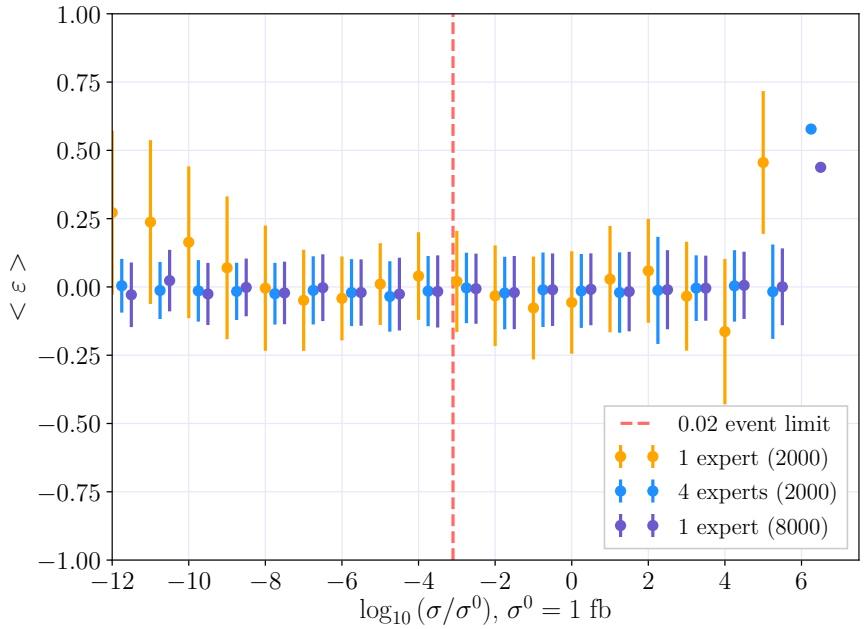


Figure 1.11: The error distributions of GP with the BM settings, using 1 (orange) and 4 (experts).

Number of experts	Points per expert	Time
1	2000	00:03:32
4	2000	00:05:46
1	8000	01:35:21

Table 1.6: Table of computation times for GP fit and prediction on Abel.

a comparison of error distributions between one expert and four experts with 2000 training points each, and one expert with 8000 training points is shown. For comparison the settings are the benchmark settings. The improvement and stability of the prediction with the addition of data is large. From the distributions of relative deviation distributions there is little difference between using four experts with 2000 training points each and using a single expert with 8000 training points. The difference in computation times, however, is very large. Four experts with 2000 points take a little under six minutes to compute, while the single expert with 8000 points takes over an hour and a half. Computation times are shown in Tab. 1.6.

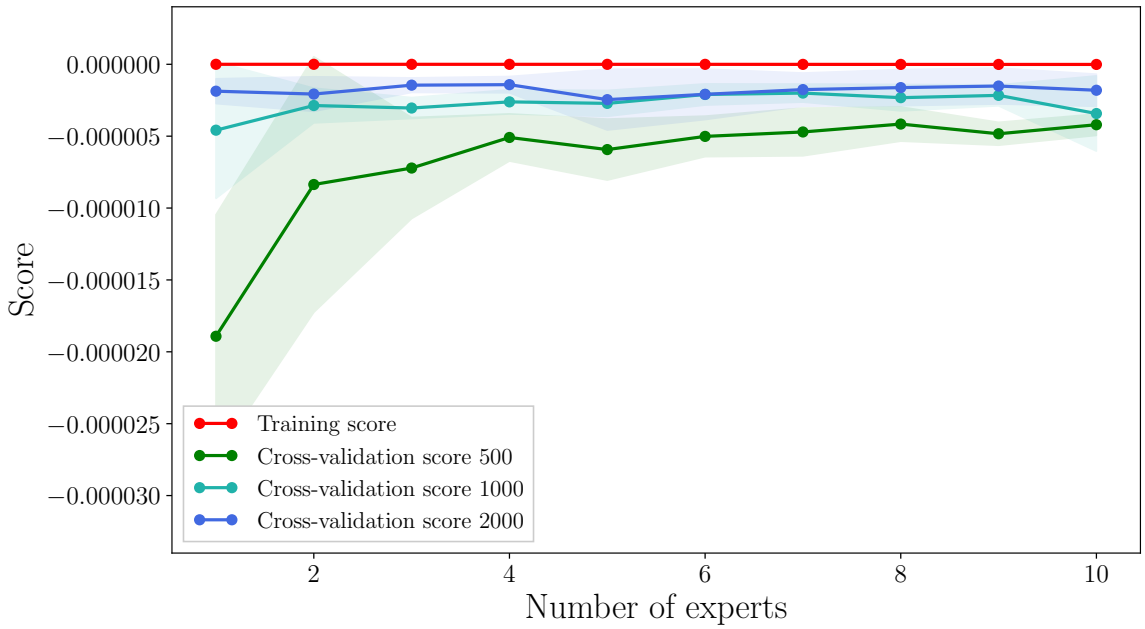


Figure 1.12: Learning curves as a function of number of experts, with 500 and 1000 training points per expert for $\tilde{d}_L\tilde{d}_L$. k-fold cross validation uses R^2-score, but here $R^2 - 1$ is plotted.

1.5.1 Cross validation for DGP

There is no `scikit-learn` function for distributed Gaussian processes, so an algorithm for k-fold cross validation as a function of experts was implemented. The algorithm uses the `scikit-learn` function `KFold` to find training and test indices for k splits of the data, and is found in the sub-library `model_selection`. For k folds it is implemented in the following way

```
from sklearn.model_selection import KFold
kf = KFold(n_splits=k, random_state=42)
```

The scoring function used in the CV is the R^2-score introduced in Sec. **??**, and pseudocode for the algorithm is found in Algorithm 2. Learning curves for the optimal settings found in Sec. 1.3.6 are shown in Fig. 1.12 for $\tilde{d}_L\tilde{d}_L$ and Fig. 1.13 for $\tilde{d}_L\tilde{u}_L$, with 500, 1000 and 2000 points per expert.

The validation curves for both processes coverge towards 1, albeit faster and for less training points per expert for $\tilde{d}_L\tilde{d}_L$ than for $\tilde{d}_L\tilde{u}_L$, indicating that the estimators benefit from adding more data. In both cases the training and validation scores are very high, even for few experts.

Data: $N_{experts}$ (max number of experts), n (training points per expert), X (inputs), $\mathbf{y}$ (targets), k (number of folds for cross validation)
number of experts $\mathbf{n} = [1, ..., N_{experts}]$;
for *each number of experts* i **do**
  Training size $= n \cdot (i + 1)$;
  Total size = training size $\cdot \frac{k}{k-1}$;
  Split training data into subsets;
  Use KFold to create k-fold cross validation instance kf;
  for *training indices, test indices in* kf **do**
    Fit GP to $k - 1$ folds of training data;
    Use 1 fold as test data;
    Predict values $\hat{y}_{train}$ for training data;
    Predict values $\hat{y}_{test}$ for test data;
    $R^2_{\text{train}}(\hat{y}, y) = 1 - \frac{\sum_{i=0}^{\text{Training size}-1}(y_i - \hat{y}_i)^2}{\sum_{i=0}^{\text{Training size}-1}(y_i - \bar{y})^2}$;
    $R^2_{\text{test}}(\hat{y}, y) = 1 - \frac{\sum_{i=0}^{n-1}(y_i - \hat{y}_i)^2}{\sum_{i=0}^{n-1}(y_i - \bar{y})^2}$;
  end
  Find mean m and std σ_{std} of R^2_{test}-values and R^2_{train}-values
end
Result: $\mathbf{n}, \mathbf{m}(R^2_{\text{train}})$, $\boldsymbol{\sigma}_{std}(R^2_{\text{train}})$, $\mathbf{m}(R^2_{\text{test}})$, $\boldsymbol{\sigma}_{std}(R^2_{\text{test}})$.

Algorithm 2: Pseudocode for k-fold cross validation of distributed Gaussian processes, which calculates the R^2-scores for training and test data as a fucntion of the number of experts, to be used for *e.g.* learning curves. In the R^2-score calculation, y_i are true values, $\hat{y}_i$ are GP predicted values, and $\bar{y}$ is the mean of all y_i.

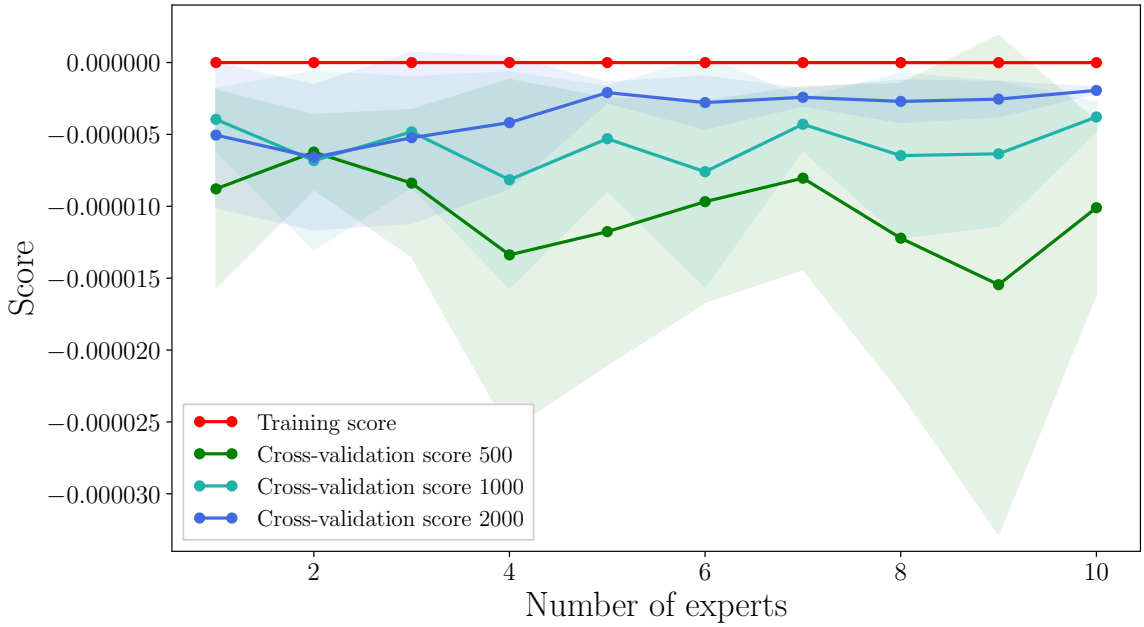


Figure 1.13: Learning curves as a function of number of experts, with 500 and 1000 training points per expert for $\tilde{d}_L\tilde{u}_L$. k-fold cross validation uses R^2-score, but here $R^2 - 1$ is plotted.

Bibliography

[1] B.C. Allanach. Softsusy: A program for calculating supersymmetric spectra. *Computer Physics Communications*, 143(3):305 – 331, 2002.

[2] Wim Beenakker, R Höpker, Michael Spira, and PM Zerwas. Squark and gluino production at hadron colliders. *Nuclear Physics B*, 492(1-2):51–103, 1997.

[3] Marc Peter Deisenroth and Jun Wei Ng. Distributed gaussian processes. *arXiv preprint arXiv:1502.02843*, 2015.

[4] Anders Kvellestad. Chasing susy through parameter space. 2015.

[5] Jon Vegard Sparre. Fast evaluation of supersymmetric cross sections. 2015.