# Rapid Uncertainty Computation with Gaussian Processes and Histogram Intersection Kernels

Alexander Freytag[1], Erik Rodner[1,2], Paul Bodesheim[1], and Joachim Denzler[1]

[1]Computer Vision Group, Friedrich Schiller University Jena, Germany
[2]Vision Group, ICSI, UC Berkeley, United States

**Abstract.** An important advantage of Gaussian processes is the ability to directly estimate classification uncertainties in a Bayesian manner. In this paper, we develop techniques that allow for estimating these uncertainties with a runtime linear or even constant with respect to the number of training examples. Our approach makes use of all training data without any sparse approximation technique while needing only a linear amount of memory. To incorporate new information over time, we further derive online learning methods leading to significant speed-ups and allowing for hyperparameter optimization on-the-fly. We conduct several experiments on public image datasets for the tasks of one-class classification and active learning, where computing the uncertainty is an essential task. The experimental results highlight that we are able to compute classification uncertainties within microseconds even for large-scale datasets with tens of thousands of training examples.

## 1 Introduction

Learning with kernel-based methods is one of the main techniques used in the area of visual object recognition to cope with the high complexity and difficulty of this task. Their main limitation is the high computation time for learning as well as for classifying a new test example. Apart from several (sparse) approximation methods [1] presented to reduce the number of necessary evaluations, recent work [2, 3] has shown that for histogram intersection kernels (HIK), several kernel terms can be efficiently evaluated. This allows for SVM learning and classification in sub-quadratic and constant computation time, respectively.

A recent work [4] demonstrated how to utilize the inherent properties of the HIK to speed up Gaussian process (GP) regression [5] allowing for large-scale Bayesian inference and hyperparameter optimization. In this paper, we extend the work of [4] in several aspects. First, we show how to perform incremental or online GP learning in an efficient manner also allowing for estimating hyperparameter values on-the-fly. Furthermore, we present how to obtain estimates of the predictive GP variance for large-scale scenarios when confronted with tens of thousands of training examples. The predictive variance, which is directly available in the original GP framework, is one of the main advantages of a Bayesian method and has been used for active learning [6] as well as for one-class classification [7]. In contrast to sparse GP approaches [1], our proposed methods and

approximations take every single learning example into account. Evaluations are done for one-class classification (OCC) and active learning (AL) showing the benefits of GP variance estimates as uncertainty values as well as the suitability of our approximations. Summarizing, the contributions of this paper are:

1. We extend [4] towards efficient online learning with GP and HIK to incorporate new data over time and to adapt involved hyperparameters.
2. We present and analyze several approximations and methods that are able to compute the predictive GP variance even for large-scale scenarios with linear memory demand and linear or even constant runtimes.

In addition, we develop a new active learning query strategy and compare it to known approaches. The remainder of the paper is structured as follows: first, we quickly review the work of [4] in Sect. 2, which is adapted towards online learning in Section 3. Our approximation techniques for efficiently estimating the GP predictive variance are explained in Sect. 4. A short overview of active learning using Gaussian processes is given in Sect. 5. Experiments in Sect. 6 show the validity of the approximations as well as the benefits of our approach for active and online learning. A summary of our findings and a discussion of future research directions conclude the paper.

## 2 Fast Gaussian Process Inference with HIK

In this section, we review the Gaussian process framework and explain how to exploit the histogram intersection kernel for efficient inference.

**Gaussian Process Regression and Classification** For the fundamental classification framework in this paper, we use a Gaussian process model [5]. Given learning data $\mathcal{D} = \{(\boldsymbol{x}_i, y_i\})_{i=1}^n$, we would like to estimate the underlying latent function $f$, which maps inputs $\boldsymbol{x}$ to outputs $y$. The GP framework casts this problem into a non-parameterized Bayesian formulation by marginalizing the latent function $f$ and assuming that $f$ is sampled from a Gaussian process with zero mean and covariance (kernel) function $K$. If we assume that outputs $y$ are disturbed by Gaussian noise, *i.e.*, $y = f(\boldsymbol{x}) + \varepsilon$ with $\varepsilon \sim \mathcal{N}(0, \sigma_n^2)$, we can directly obtain the predictive distribution $\mathcal{N}(\mu_*, \sigma_*^2)$ of the output $y_*$ for a new test input $\boldsymbol{x}^*$:

$$\mu_* = \boldsymbol{k}_*^T (\mathbf{K} + \sigma_n^2 \cdot \mathbf{I})^{-1} \boldsymbol{y} = \boldsymbol{k}_*^T \boldsymbol{\alpha} \ , \tag{1}$$

$$\sigma_*^2 = k_{**} - \boldsymbol{k}_*^T (\mathbf{K} + \sigma_n^2 \cdot \mathbf{I})^{-1} \boldsymbol{k}_* + \sigma_n^2 \ , \tag{2}$$

where $k_{**}, \boldsymbol{k}_*$, and $\mathbf{K}$ are the kernel values of the test input, between training set and test input, and of the training set itself, respectively. Furthermore, $\boldsymbol{y}$ denotes the vector containing the output values of the learning set.

In this paper, we consider classification with the GP framework, *i.e.*, we have discrete outputs (labels) $y \in \{-1, 1\}$. Although the Gaussian noise assumption is in this case critical from a theoretical point of view, it has been shown [6] that applying GP regression directly to discrete labels leads to a powerful classification method. This technique is called label regression and allows for exact inference without approximation methods [5]. In the case of multiple classes, we use the one-vs-all approach as proposed in [6] and also used in [4].

**Exploiting the Histogram Intersection Kernel** Inference in the GP framework is costly. Computing the predictive mean in Eq. (1) requires $\mathcal{O}(n^3)$ for learning and $\mathcal{O}(n)$ for prediction. Furthermore, computing the predictive variance even involves an asymptotic runtime of $\mathcal{O}(n^2)$. This prevents its direct use for large-scale datasets.

To speed up Gaussian process inference, there are multiple methods (see [1] and references therein). However, nearly all of them rely on approximations and use only a subset of the learning examples directly. In contrast, [4] showed how to exploit the intrinsic properties of the histogram intersection kernel (HIK) [2, 3], to speed up GP inference significantly. We briefly review the key points of their work in the following.

The HIK is defined for non-negative histogram features $\boldsymbol{x} \in \mathbb{R}^D_{\geq 0}$ as follows:

$$K^{\text{HIK}}(\boldsymbol{x}, \boldsymbol{x}') = \sum_{d=1}^{D} \min(x_d, x'_d) \ . \tag{3}$$

Let us first assume that the weight vector $\boldsymbol{\alpha}$ in Eq. (1) has already been calculated. Computing the predictive mean then reduces to determining the value of the scalar product of the weight vector and the kernel vector $\boldsymbol{k}_*$ [2]:

$$\boldsymbol{k}_*^T \boldsymbol{\alpha} = \sum_{i=1}^{n} \alpha_i \left( \sum_{d=1}^{D} \min(x_d^{(i)}, x_d^*) \right) = \sum_{d=1}^{D} \Big( \sum_{\{i: x_d^{(i)} < x_d^*\}} \alpha_i \, x_d^{(i)} + x_d^* \sum_{\{j: x_d^{(j)} \geq x_d^*\}} \alpha_j \Big) \ . \tag{4}$$

As can be seen, we divided the summation in two parts, which depend on the location of the feature values $x_d^*$ in the sorted lists of the training examples $x_d^{(i)}$. Sorted lists are obtained by computing permutations $\pi_d$ during learning. With these permutations, we can further rewrite the above sum as follows:

$$\boldsymbol{k}_*^T \boldsymbol{\alpha} = \sum_{d=1}^{D} \Big( \underbrace{\sum_{i=1}^{r_d} \alpha_{\pi_d^{-1}(i)} x_k^{(\pi_d^{-1}(i))}}_{\doteq A(d, r_d)} + \underbrace{x_d^* \sum_{i=r_d+1}^{n} \alpha_{\pi_d^{-1}(i)}}_{\doteq B(d, r_d)} \Big) \ , \tag{5}$$

where we defined $r_d$ as the location of the feature values $x_d^*$ and matrices $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{D \times n}$ that do not depend on the test example and can be directly computed during learning. With these matrices, the time needed for calculating the predictive mean reduces to $\mathcal{O}(D \log n)$. Furthermore, applying the quantization idea of [2] resulting in a look-up table $\mathbf{T}$ allows for constant runtimes.

As shown in [4], computing products $\mathbf{K} \cdot \boldsymbol{v}$ with the kernel matrix can be done using the same ideas and allows for applying conjugate gradient methods to estimate the weight vector $\boldsymbol{\alpha}$. The paper also showed how to perform efficient estimation of kernel hyperparameters. Due to the lack of space, we refer the reader to [4] for details. Throughout this paper we refer to the resulting classifier as GP-HIK. In the following, we show how to extend their methods towards online learning as well as efficiently computing the predictive variance, which allows for interesting applications such as novelty detection and active learning.

## 3 Fast Incremental Learning of the GP-HIK

The usual pipeline for object recognition systems is to train a classifier on a given set of labeled examples and apply the resulting model on unseen examples. Although current research lead to impressive results even on highly challenging datasets with this strategy [8, 9, 6], it suffers from two main drawbacks: (1) there is no possibility to exploit labeled examples that are available after the training process, which consequently neglects potentially useful information, and (2) it will fail in situations where existing categories vary over time or new categories become available. Although trivial training from scratch as soon as new data is accessible would resolve these drawbacks, it suffers from huge computational costs and does not exploit information about the model currently used. Incremental or online learning methods combine both aspects, namely adapting the classifier over time while using previously computed models. We show how to efficiently retrain a GP-HIK in the following.

As presented in [4], training a GP-HIK mainly consists of four stages: (1) sort training examples in every dimension, (2) compute the weight vector $\boldsymbol{\alpha}$ using an iterative linear solver, (3) optimize involved hyperparameters, and (4) compute the matrices $\mathbf{A}$ and $\mathbf{B}$ and the look-up table $\mathbf{T}$ if required. For new training examples, we can exploit the previous calculations of every step to significantly speed-up the process of retraining:

(1) We can build on the given sorting of each dimension and find the correct position of new values in $\mathcal{O}(\log n)$.
(2) Using the previously calculated $\boldsymbol{\alpha}$ as an initialization for the iterative linear solver, we can significantly speed-up the process until convergence since the variations of $\boldsymbol{\alpha}$ are smooth, especially for large training sizes.
(3) Again we can use the optimal parameter settings known so far as an initial guess to speed up the process of hyperparameter optimization.
(4) For updating the arrays $\mathbf{A}$ and $\mathbf{B}$ as well as the look-up table $\mathbf{T}$, we only need to correct entries that are affected by the new examples.

The authors of [4] proposed to use a linear conjugate gradient method as iterative linear solver and for hyperparameter optimization the downhill-simplex method. Regarding these algorithms, a proper initialization leads to both fast convergence rates and stable optima.

Summarizing, we can significantly benefit from previous calculations in every training step, which is further validated in our experiments in Sect. 6.4. Note that one could even further speed-up the process of retraining if the optimization of hyperparameters was done less regularly, since they are typically found to be much less sensitive to new data than model parameters.

## 4 Efficient Computations of the Predictive Variance

Up to now, we only considered fast computations and efficient updates of the predictive mean derived from GP regression. However, in many scenarios such

as active learning or novelty detection it is important to get an estimate for the uncertainty of the prediction as well. The uncertainty is mostly measured in terms of entropy and for Gaussian distributions it is directly related to the variance. Due to this reason, we develop efficient methods to compute the GP predictive variance (Eq. (2)) also in large-scale scenarios. As presented in Sect. 2, the predictive variance $\sigma_*^2$ :

$$\sigma_*^2 = k_{**} - \boldsymbol{k}_*^T(\mathbf{K} + \sigma_n^2 \cdot \mathbf{I})^{-1}\boldsymbol{k}_* + \sigma_n^2 \tag{6}$$

depends on three terms. While the first and third terms reflect the a-priori uncertainties $k_{**} = K(\boldsymbol{x}^*, \boldsymbol{x}^*)$ and $\sigma_n^2$ without considering previously known training examples, the second term reduces the a-priori uncertainty based on the similarities between test example $\boldsymbol{x}^*$ and training examples $\mathbf{X}$. Since this term is a quadratic instead of a linear form in $\boldsymbol{k}_*$, the previously presented techniques for fast computations of scalar products $\boldsymbol{k}_*^T\boldsymbol{\alpha}$ [4] can not be applied here. In the following, we show how to approximate the second term in an efficient manner using fast kernel evaluations. An overview of the presented approaches as well as their resulting runtimes and decision functions is given in Table 1.

### 4.1 Bounds on Quadratic Forms

From linear algebra we know that any real-valued, symmetric matrix $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ can be transformed into $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{D}\boldsymbol{U}^T$ where $\boldsymbol{D}$ is a positive diagonal matrix containing the eigenvalues of $\boldsymbol{A}$ and $\boldsymbol{U}$ is an orthogonal matrix of the same size as $\boldsymbol{A}$. Therefore, we notice that for any vector $\boldsymbol{x} \in \mathbb{R}^n$ the following holds:

$$\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x} = \boldsymbol{x}^T\boldsymbol{U}\boldsymbol{D}\boldsymbol{U}^T\boldsymbol{x} = \tilde{\boldsymbol{x}}^T\boldsymbol{D}\tilde{\boldsymbol{x}} = \sum_{i=1}^{n} \lambda_i \tilde{x}_i^2 \ . \tag{7}$$

We denoted with $\lambda_i$ the decreasingly ordered eigenvalues of $\boldsymbol{A}$, *i.e.*, $\lambda_1 \geq \ldots \geq \lambda_N$, and $\tilde{x}_i$ contains the projection of $\boldsymbol{x}$ onto the $i$th row of $\boldsymbol{U}$, which is the $i$th eigenvector of $\boldsymbol{A}$. As a result, we can bound the quadratic form in Eq. (7) as follows:

$$\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x} = \sum_{i=1}^{k} \lambda_i \tilde{x}_i^2 + \sum_{j=k+1}^{n} \lambda_j \tilde{x}_j^2 \leq \sum_{i=1}^{k} \lambda_i \tilde{x}_i^2 + \lambda_{k+1} \sum_{j=k+1}^{n} \tilde{x}_j^2 \ . \tag{8}$$

Since $\boldsymbol{U}$ is an orthonormal basis, it does not influence the length of vectors, *i.e.*, $||\boldsymbol{U}\boldsymbol{x}|| = ||\boldsymbol{x}||$. Therefore, we can obtain the following upper bound:

$$\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x} \leq \sum_{i=1}^{k} \lambda_i \tilde{x}_i^2 + \lambda_{k+1}\Big(||\boldsymbol{x}||^2 - \sum_{i=1}^{k} \tilde{x}_i^2\Big) \ . \tag{9}$$

Equivalently, we get the following lower bound considering the $k$ smallest eigenvalues of $\boldsymbol{A}$ and bounding the remaining eigenvalues with the $(k+1)$th smallest:

$$\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x} \geq \sum_{j=N-k+1}^{N} \lambda_j \tilde{x}_j^2 + \lambda_{N-k}\Big(||\boldsymbol{x}||^2 - \sum_{j=N-k+1}^{N} \tilde{x}_j^2\Big) \ . \tag{10}$$

For the special cases of $k = 0$ in Eq. (9) and Eq. (10), we obtain the well known bounds for any positive definite matrix $\boldsymbol{A}$ and any vector $\boldsymbol{x}$ of corresponding size:

$$\lambda_{\min}(\boldsymbol{A}) \, ||\boldsymbol{x}||^2 \leq \boldsymbol{x}^T \boldsymbol{A} \boldsymbol{x} \leq \lambda_{\max}(\boldsymbol{A}) \, ||\boldsymbol{x}||^2 \quad . \tag{11}$$

### 4.2 RAPU – Rough Approximation of the Predictive Uncertainty

Since the kernel matrix $\mathbf{K}$ is symmetric and positive definite, the same holds for its inverse. Therefore, we can use the previously presented bounds to obtain suitable approximations for $\sigma_*^2$. We start with the special case analysis using $k = 0$ in Eq. (10) and arrive at the following approximation:

$$\begin{aligned} \sigma_*^2 &= k_{**} - \boldsymbol{k}_*^T (\mathbf{K} + \sigma_n^2 \cdot \mathbf{I})^{-1} \boldsymbol{k}_* + \sigma_n^2 \\ &\leq k_{**} - ||\boldsymbol{k}_*||^2 \left(\lambda_{\max}(\mathbf{K} + \sigma_n^2 \cdot \mathbf{I})\right)^{-1} + \sigma_n^2 \quad . \end{aligned} \tag{12}$$

In the following, we derive fast computations of the involved terms using properties of the histogram intersection kernel.

**Efficient HIK computation** It was already shown [4] that the computation of $\lambda_{\max}$ can be done efficiently using histogram intersection kernels and the Arnoldi iteration method. Therefore, it remains to efficiently compute $||\boldsymbol{k}_*||^2$:

$$||\boldsymbol{k}_*||^2 = \boldsymbol{k}_*^T \cdot \boldsymbol{k}_* = \sum_{i=1}^{n} \Big( \sum_{d=1}^{D} \min \left( x_d^*, x_d^{(i)} \right) \Big)^2 \quad . \tag{13}$$

If we approximate $||\boldsymbol{k}_*||^2$ by a lower bound, we still obtain a valid upper bound approximation for the predictive variance as given in Eq. (12). For this purpose, we observe that especially when dealing with sparse features having only few non-zero entries, the majority of mixed terms between different dimensions $\min(x_{d_1}^*, x_{d_1}^{(i)}) \cdot \min(x_{d_2}^*, x_{d_2}^{(i)})$ will vanish given the histogram intersection kernel. For a sparsity ratio of 0.1, these are 99% of all terms. Therefore, neglecting the mixed terms is well justifiable and we obtain a squared Parzen-like expression:

$$||\boldsymbol{k}_*||^2 \geq \sum_{i=1}^{n} \sum_{d=1}^{D} \left( \min(x_d^*, x_d^{(i)}) \right)^2 = \sum_{i=1}^{n} \sum_{d=1}^{D} \min \left( (x_d^*)^2, (x_d^{(i)})^2 \right) \doteq \left|\left| \widehat{\boldsymbol{k}_*} \right|\right|^2 \tag{14}$$

This expression is equivalent to the one in Eq. (4) for squared features and $\alpha_i = 1$. Therefore, we can directly apply the same fast computation methods as described in Sect. 2 with squared feature values. Furthermore, we can even use the same permutations of the learning data and only have to compute a new matrix $\tilde{\mathbf{A}} \in \mathbb{R}^{D \times n}$ storing the cumulative sums of squared feature values similar to $\mathbf{A}$. Finally, for an unseen example $\boldsymbol{x}^*$, we can compute the squared kernel vector within $\mathcal{O}(D \log n)$ operations or in $\mathcal{O}(D)$ time when using the quantization approach (q-RAPU) presented in [4]. Note that the predictive variance is the same for all known classes [6], thus, our computation times are efficient even for extremely large numbers of classes.

### 4.3 FAPU – Fine Approximation of the Predictive Uncertainty

In the previous section, we derived squared Parzen-like approximations of the GP predictive variance using a special case of quadratic form approximations (see Eq. (12)). Although these scores are efficiently computable using histogram intersection kernels, the assumption of sparse features is not always valid. In these cases, using the RAPU-method is not justifiable. Therefore, we derive a finer approximation scheme based on the general inequalities given in Eq. (10):

$$\sigma_*^2 \leq k_{**} - \Big( \sum_{j=n-k+1}^{n} \lambda_j v_j^2 + \lambda_{n-k} \Big( ||\boldsymbol{k}_*||^2 - \sum_{j=n-k+1}^{n} v_j^2 \Big) \Big) + \sigma_n^2 \qquad (15)$$

where $\lambda_j$ denotes the $j$th decreasingly ordered eigenvalue of $(\mathbf{K} + \sigma_n^2 \cdot \mathbf{I})^{-1}$ and $v_j$ the projection of $\boldsymbol{k}_*$ onto the $j$th eigenvector of the inverse kernel matrix.

Since we can not access the inverse kernel matrix, but $(\mathbf{K} + \sigma_n^2 \cdot \mathbf{I})$ indirectly, we transfer the previous bound. Therefore, we define $\mu_i$ as the $i$th decreasingly ordered eigenvalue of the kernel matrix $(\mathbf{K} + \sigma_n^2 \cdot \mathbf{I})$, *i.e.*, $\lambda_j = \frac{1}{\mu_{n-j+1}}$. In addition, we define $\nu_i$ to be the projection of $\boldsymbol{k}_*$ onto the $i$th eigenvector of the kernel matrix. We further recall that for any symmetric and positive definite matrix $\boldsymbol{A}$ the eigenvector corresponding to the $i$th largest eigenvalue $\mu_i$ is the same as the eigenvector of $\boldsymbol{A}^{-1}$ belonging to the $(n-i+1)$th largest eigenvalue $\lambda_{n-i+1}$ of the inverse matrix, and consequently $\nu_i = v_{n-i+1}$. Therefore, we can express the approximated variance given in Eq. (15) with the following term:

$$\sigma_*^2 \leq k_{**} - \Big( \sum_{i=1}^{k} \frac{1}{\mu_i} \nu_i^2 + \frac{1}{\mu_{k+1}} \Big( ||\boldsymbol{k}_*||^2 - \sum_{i=1}^{k} \nu_i^2 \Big) \Big) + \sigma_n^2 \ . \qquad (16)$$

To compute $k$ eigenvalues and eigenvectors we need $\mathcal{O}(kT_1 n)$ operations using Arnoldi iteration with $T_1$ as the number of iterations until convergence. In our experiments, $T_1$ was almost constant about 10 steps for various numbers of training examples. For the computation of the kernel vector $\boldsymbol{k}_*$, we have to spend $\mathcal{O}(Dn)$ operations. Projections $\nu_i^2$ of $\boldsymbol{k}_*$ onto eigenvectors can be computed in $\mathcal{O}(n)$ as well as the norm $||\boldsymbol{k}_*||^2$. Summarizing, we need $\mathcal{O}(Dn + kT_1 n)$ operations in total to compute the approximation of $\sigma_*^2$ as given in Eq. (16).

### 4.4 PUP – Precise Uncertainty Prediction

While the previously presented approaches were based on approximations, we can also exploit the properties of histogram intersection kernels to compute the exact predictive variance. To obtain the score for a single test example $\boldsymbol{x}^*$, we first compute the kernel vector $\boldsymbol{k}_*$ requiring $\mathcal{O}(Dn)$ operations. After that, we apply an iterative linear solver to compute the vector $\boldsymbol{\alpha}_* = (\mathbf{K} + \sigma_n^2 \cdot \mathbf{I})^{-1} \boldsymbol{k}_*$. This takes $\mathcal{O}(T_2 Dn)$ operations where $T_2$ denotes the number of iterations needed for convergence. In our experiments, we noticed a dependence between $T_2$ and the condition of the implicit kernel matrix, which is related to $n$ and can be corrected

**Table 1.** Overview of the presented approaches to compute the predictive variance $\sigma_*^2$. For details see the derivations in the corresponding sections.

| Approach | Asymptotic runtime | Resulting score |
|---|---|---|
| q-RAPU (Sect. 4.2) | $\mathcal{O}(D)$ | $k_{**} - \left\|\widehat{\boldsymbol{k}_*}\right\|^2 \frac{1}{\lambda_{\max}(\mathbf{K}+\sigma_n^2 \cdot \mathbf{I})} + \sigma_n^2$ |
| RAPU (Sect. 4.2) | $\mathcal{O}(D \log n)$ | $k_{**} - \left\|\widehat{\boldsymbol{k}_*}\right\|^2 \frac{1}{\lambda_{\max}(\mathbf{K}+\sigma_n^2 \cdot \mathbf{I})} + \sigma_n^2$ |
| FAPU (Sect. 4.3) | $\mathcal{O}(D \log n + kT_1 n)$ | $k_{**} - \left(\sum_{i=1}^{k} \frac{1}{\mu_i} \nu_i^2 + \frac{1}{\mu_{k+1}} \left( \|\boldsymbol{k}_*\|^2 - \sum_{i=1}^{k} \nu_i^2 \right)\right) + \sigma_n^2$ |
| PUP (Sect. 4.4) | $\mathcal{O}(T_2 Dn)$ | $k_{**} - \boldsymbol{k}_*^T (\mathbf{K} + \sigma_n^2 \cdot \mathbf{I})^{-1} \boldsymbol{k}_* + \sigma_n^2$ |
| GP-standard | $\mathcal{O}(n^2 + nD)$ | $k_{**} - \boldsymbol{k}_*^T (\mathbf{K} + \sigma_n^2 \cdot \mathbf{I})^{-1} \boldsymbol{k}_* + \sigma_n^2$ |

by adapting $\sigma_n^2$. After the linear solver, we can compute the product of $\boldsymbol{k}_*$ and $\boldsymbol{\alpha}_*$ in $\mathcal{O}(n)$ operations to obtain the second term needed to compute $\sigma_*^2$. In total, we need $\mathcal{O}(T_2 Dn)$ operations to compute the exact predictive variance for an unseen example during testing.

Summarizing, we are able to efficiently compute the predictive variance with selectable precision as well as selectable time to spent. In the next section, we analyze the usability of our methods for the prominent task of active learning.

## 5 Active Learning with Gaussian Processes

For active learning, one typically has a small set $\mathcal{L} = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)\}$ consisting of labeled data and an arbitrary large set $\mathcal{U} = \{\widehat{\boldsymbol{x}}_1, \ldots, \widehat{\boldsymbol{x}}_m\}$ of unlabeled examples. To obtain a classifier $\mathcal{A}$ trained with most informative examples, one exploits a query function $\mathcal{Q}$ that scores each unlabeled example and asks for the ground-truth label of the example with best score. Consequently, an active learning scenario is a quadruple $(\mathcal{A}, \mathcal{Q}, \mathcal{L}, \mathcal{U})$. One further distinguishes query strategies in two groups [10]: exploitative methods utilize examples of $\mathcal{L}$ including the labels and rely on scores derived from outputs of the involved classifier whereas explorative methods neglect the label information and query new examples only based on the distribution of the current examples.

For the choice of Gaussian processes, there exist mainly three possible query strategies so far [6] : (1) the predictive mean $\mathcal{Q}_{\mu_*}(\mathcal{U}) = \text{argmin } |\mu_*(\widehat{\boldsymbol{x}}_i)| : \widehat{\boldsymbol{x}}_i \in \mathcal{U}$ is an exploitative method and selects examples possibly close to the current decision boundary, (2) the predictive variance $\mathcal{Q}_{\sigma_*^2}(\mathcal{U}) = \text{argmax } \sigma_*^2(\widehat{\boldsymbol{x}}_i) : \widehat{\boldsymbol{x}}_i \in \mathcal{U}$, is explorative and selects examples with highest classification uncertainty regarding the known training examples, and (3) the *uncertainty*[1] $\mathcal{Q}_{unc}(\mathcal{U}) = \text{argmin } \frac{|\mu_*(\widehat{\boldsymbol{x}}_i)|}{\sqrt{\sigma_n^2 + \sigma_*^2(\widehat{\boldsymbol{x}}_i)}} : \widehat{\boldsymbol{x}}_i \in \mathcal{U}$ as a combination of both. The authors of [6] showed that especially $\mathcal{Q}_{\mu_*}$ and $\mathcal{Q}_{unc}$ lead to impressive performance gains compared to randomly querying new samples. In contrast, [11] argued that the $\mathcal{Q}_{unc}$ method can also lead to a loss of performance compared to random picking.

We argue that this is most likely the case due to a drawback in the design of $\mathcal{Q}_{unc}$: originally, the authors of [6] invented this method to obtain a query

[1] Note that in the rest of the paper, the term *uncertainty* refers to classification uncertainty, and not to the query strategy introduced by [6].

function similar to the minimum margin approach suitable for SVMs [12] but with the additional consideration of the classification uncertainty. However, this method will tend to pick examples maximal far from the training samples and not those lying close to the decision boundary between two clusters in the input space. This behavior is due to the zero mean assumption and the upper bound of $k_{**}$ for the predictive variance. To overcome these problems, we propose to use a criterion that picks examples that both result in small absolute scores and are additionally slightly similar to currently known training examples:

$$\mathcal{Q}_{\text{Unc+}}(\mathcal{U}) = \text{argmin} \left( |\mu_*(\widehat{\boldsymbol{x}}_i)| + \sqrt{\sigma_n^2 + \sigma_*^2(\widehat{\boldsymbol{x}}_i)} \right) : \widehat{\boldsymbol{x}}_i \in \mathcal{U} \ . \tag{17}$$

We experimentally prove the suitability of this query function for the task of active learning in Sect. 6.5.

## 6 Experiments

We evaluate our approach on synthetic and real world datasets. Our main findings can be summarized as follows:

1. Our approximation techniques allow for computing classification uncertainties in microseconds while requiring only linear amount of memory.
2. Approximating uncertainties with the presented methods leads to valid scores and similar or even improved one-class classification results with respect to the GP baseline.
3. The techniques for incrementally training a GP-HIK lead to significant reductions of computation times compared to learning from scratch and allow for parameter optimization on-the-fly.
4. Combining our approaches for efficient uncertainty prediction and incremental learning allows for active learning in an efficient manner.

### 6.1 Experimental Setup

To obtain a setup as similar to [4] as possible, we represent images with histogram features extracted using the toolkit provided for the ILSVRC'10 challenge[2]. Vector quantization is achieved using the provided codebook with $1,000$ clusters following the standard bag of visual words approach. Histograms are $L_1$-normalized. As in [4], we do not include any spatial information. Nonetheless it should be noted, that every image representation given as normalized histograms is suitable for our approach, *e.g.*, the spatial pyramid match kernel introduced by [8]. If not stated otherwise, we use the FAPU method with $k = 2$ eigenvectors to compute $\sigma_*^2$. For a fair comparison, all experiments are conducted on a 3.4 GHz CPU using a C++ implementation[3] without any parallelization. In classification experiments, we use the area under the receiver-operator curve (AUC) for binary setups and averaged class-wise recognition rates (ARR) for multi-class scenarios.

[2] http://www.image-net.org/challenges/LSVRC/2010

[3] Source code will be available at http://www.inf-cv.uni-jena.de/en/gp_hik.html

**Table 2.** Runtimes needed for the computation of the predictive variance using the presented techniques from Sect. 4 in comparison to the baseline GP on two image categorization datasets (see Sect. 6.2). * not possible due to excessive memory demand.

| **Approach** | **1,500** | **10,090** | **50,050** |
|---|---|---|---|
| q-RAPU (Sect. 4.2) | 13.32$\mu$s | 33.89$\mu$s | 33.89$\mu$s |
| RAPU (Sect. 4.2) | 5.92ms | 62.07ms | 266.97ms |
| FAPU (Sect. 4.3), $k = 2$ | 13.35ms | 105.37ms | 1.15 s |
| FAPU (Sect. 4.3), $k = 8$ | 13.73ms | 105.63ms | 1.47 s |
| PUP (Sect. 4.4) | 2.11 s | 22.92 s | $>$ 1min |
| GP-standard | 60.36ms | 1.54 s | —* |

### 6.2 Fast Computation of the Predictive Variance

To evaluate the efficiency of our proposed techniques, we perform experiments on two datasets. The first one is the 15 scenes dataset [8] consisting of 15 classes with in total $4,530$ images. As a second dataset we choose the part of the large-scale ImageNet dataset that was previously used for the ILSVRC'10 challenge with $1,000$ classes each consisting of $1,000$ images. For the small dataset, we randomly pick 100 examples of each class for training whereas for the large-scale evaluation we pick 10 or 50 examples resulting in 1 500, 10 090, and 50 050 training examples. Times are averaged over remaining examples.

Experimental results are shown in Table 2. Especially for large-scale datasets, we obtain a significant speed-up compared to the standard GP computation. For rapid uncertainty prediction the quantized RAPU methods turns out to be highly suitable with computation times in the order of microseconds. It should be noted that the PUP method is relatively slow due to the involved computations of the iterative linear solver. Although the authors of [4] pointed out the efficiency of the linear conjugate gradient method for this problem, in our experiments it still needed some hundreds of iterations until convergence, especially for large training sets. Therefore, we argue to use the precise method only in cases where time is not the limiting factor, but the GP baseline can not be computed explicitly due to the huge memory demand.

### 6.3 Uncertainty Approximations for One-Class Classification

To demonstrate the adequacy of our approximations, we conduct experiments in scenarios where the predictive variance is highly beneficial. As demonstrated by [7], this is the case for one-class classification, where only data from a single class is available during training. The decision whether or not a new example belongs to the target class can be made based on the classification uncertainty for that example. We evaluate our approximations in comparison to the exact variance on $1,000$ OCC tasks derived from the ImageNet dataset. In each task, we train the GP classifier with 100 randomly chosen examples. AUC scores are computed using 50 examples of the target class as well as 50 examples of each of the remaining 999 classes.

The results are given in Fig. 1. From the results, we clearly see that the exact computation and the fine approximations with 2 or 8 eigenvectors lead to

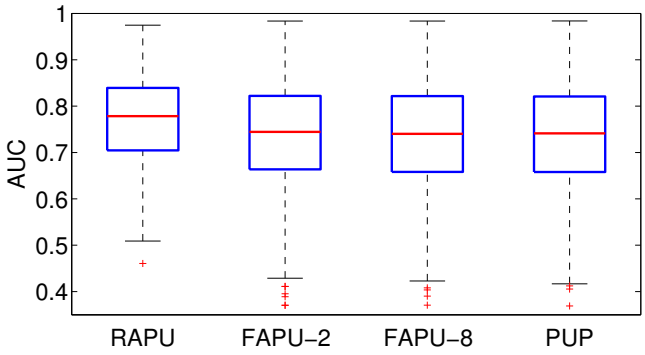



**Fig. 1.** Results for OCC scenarios derived from ImageNet using the presented strategies for computing the predictive variance. Results are given as AUC scores.

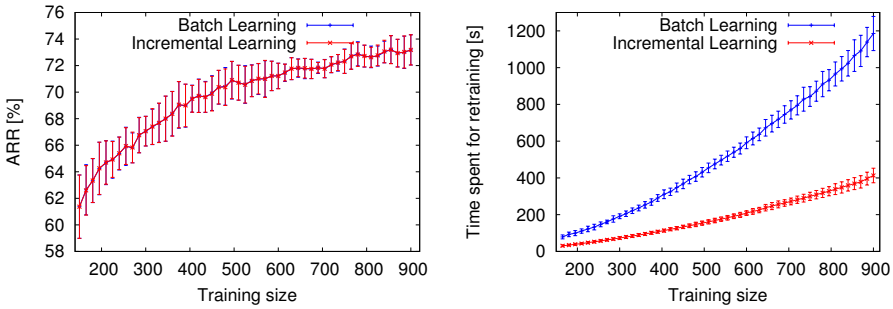



**Fig. 2.** Comparison of incremental learning and learning from the scratch. *Left*: classification results (ARR) of both methods. *Right*: Corresponding times for retraining.

almost identical results. Interestingly, the rough approximation even improves the novelty detection results. For an comparison of the predictive variance with other established methods, we refer the reader to [7]. From our results, we can conclude that by using our approximation schemes we are able to reproduce or even improve the results that we would obtain using the exact variance.

### 6.4 Comparing Incremental and Batch Learning

In Sect. 3, we analyzed how we can efficiently handle new data without the necessity of retraining the classifier from the scratch. To evaluate the resulting benefit, we present experiments conducted on the 15 scenes dataset [8]. We perform multi-class experiments using all 15 classes. In 100 runs, we randomly pick 10 examples per class as an initialization. During each run, we incrementally add 1 example per class over 50 iterations resulting in maximal 900 examples used for training the model. Every iteration consists of training the classifier as well as optimizing kernel hyperparameters to perform parameter optimization on-the-fly. Performances are evaluated on a disjoint test set consisting of 50 examples per class. We visualized experimental results in Fig.2.

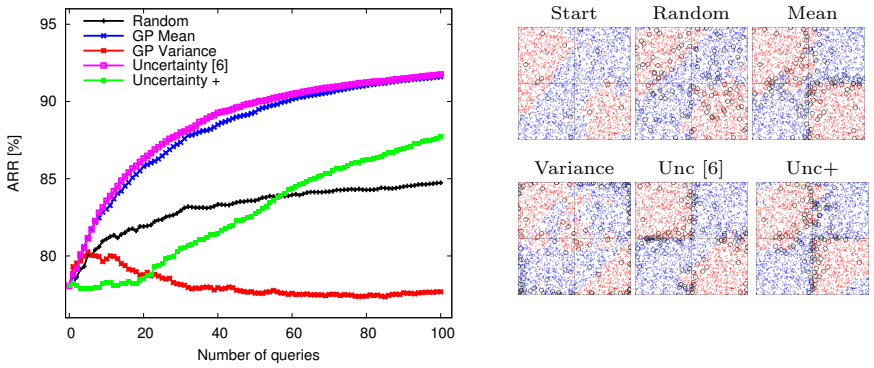



**Fig. 3.** Classification results (ARR) on a 2x2 checker board using different query strategies. *Left:* Results averaged over 100 runs. *Right:* Results of a single run after querying 100 examples. Queried examples are marked with black circles. Best viewed in color.

From the left plot in Fig. 2, we make the well-known observation that using more examples is beneficial for building more robust models. In addition, we notice that the models learned in an incremental manner lead to almost identical results as those from models trained from the scratch. However, when taking the computation times given in the right plot of Fig. 2 into account, we obtain a clear advantage of our incremental learning approach compared to simple retraining.

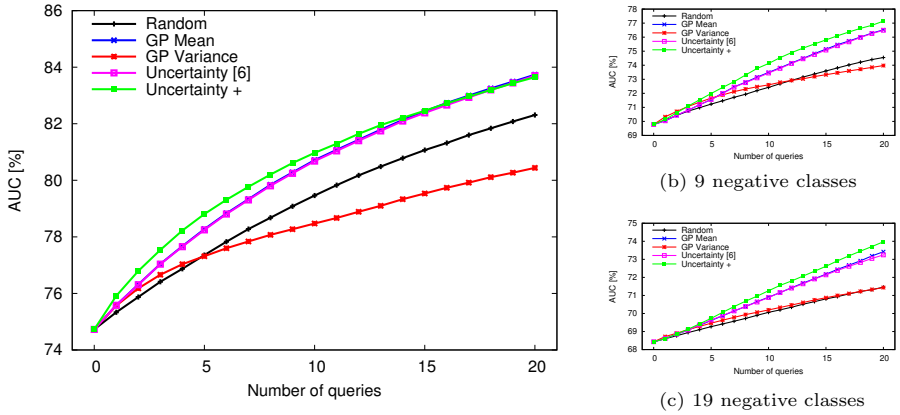
Summarizing, we are able to efficiently update our model when new data is available even with an involved parameter optimization, which allows for using Gaussian processes for large-scale scenarios in lifelong or active learning.

### 6.5 Active Learning with the GP-HIK

In the previous sections, we observed that using a histogram intersection kernel allows for rapidly computing classification uncertainties and incorporating new examples without costly retraining. With these methods on hand, we are able to efficiently perform active learning with Gaussian processes as explained in Sect. 5. We give experimental proofs of this fact in the following.

**Active Learning on a Synthetic Example** We start by considering the synthetic example of a 2x2 checker board structure. We use positions as features and add a linear dependent third dimension in order to obtain $L_1$-normalized histograms. In each run, we sample 200 examples per sector for $\mathcal{U}$. As an initial training set, we randomly pick 8 examples per class from $\mathcal{U}$. After each query, we evaluate the resulting performance with 750 examples per sector. We average results over 100 runs.

Active learning results on the synthetic example are shown in Fig. 3. In terms of recognition rates, we clearly observe the general trend of $\mathcal{Q}_{\mu_*}$ and $\mathcal{Q}_{Unc}$ to be superior to random sampling for this scenario. As already noticed in [6] the predictive variance $\mathcal{Q}_{\sigma_*^2}$ tends to query outliers and building unreliable models.




(a) 4 negative classes

(b) 9 negative classes

(c) 19 negative classes

**Fig. 4.** Active learning results (AUC) on 100 binary classification tasks derived from the ImageNet dataset. See text below for details. Best viewed in color.

Surprisingly, the proposed strategy $\mathcal{Q}_{Unc+}$ (see Eq. (17)) is inferior to random sampling at the beginning but outperforms it for larger numbers of queries.

In the right part of Fig. 3, the behavior of the different query strategies are visualized for a single run. We first note that the random strategy spends a lot of effort in non-informative regions and leads to severe generalization errors. We again notice that the predictive variance $\mathcal{Q}_{\sigma_*^2}$ leads to queries far away from the currently seen examples and results in bad generalizations as well. In contrast, $\mathcal{Q}_{\mu_*}$ and $\mathcal{Q}_{Unc}$ query examples close to the currently known decision boundaries. However, $\mathcal{Q}_{Unc+}$ is appealing from a visual point of view, since it is able to resolve wrong decision boundaries as in the bottom right sector by querying informative examples close to the actual borders. In summary, it is beneficial to have both, mean and variance, available for the task of active learning to significantly improve learning rates.

**Active Learning on ImageNet** We further evaluate our methods on the challenging real-world ImageNet dataset. For each experiment, we randomly pick a single positive class as well as four, nine, or nineteen classes serving as negative examples. Starting with two randomly chosen examples per class, we query new examples using the proposed methods. Each task is repeated with 100 random initializations. Final results are achieved by averaging over 100 different tasks.

Experimental results are given in Fig. 4. We first notice that $\mathcal{Q}_{\mu_*}$ and $\mathcal{Q}_{Unc}$ tend to query similar examples even on this challenging dataset resulting in almost identical performances. In contrast, our query strategy $\mathcal{Q}_{Unc+}$, which combines predictive mean and variance in a suitable way, leads to a remarkable gain in performance. Concerning $\mathcal{Q}_{\sigma_*^2}$ we again obtain interesting results, since it is inferior to random sampling for a small number of negative classes but

slightly superior for larger number of classes. As a concluding remark we adhere that with a suitable combination of mean and variance we can significantly outperform random sampling even on highly challenging real-world datasets.

## 7 Conclusions and Future Work

This paper considered incremental and active learning with Gaussian processes in the presence of tens of thousands of learning examples. A key ingredient is the estimation of the predictive variance for a new test example, for which we presented several approximations and efficient methods. Our approach is based on exploiting the properties of the histogram intersection kernel, which allows for computing the kernel terms in an efficient manner. Our methods break the limits of non-parametric Bayesian inference and allow for obtaining complete estimates of the predictive distribution for a new test example in constant time and without any sparse approximation of the kernel matrix. Furthermore, we studied several active learning criteria and showed their suitability and benefits on synthetic as well as image categorization tasks.

For future work, we plan to improve incremental learning by only updating parts of the weight vector. In addition it would be interesting to replace queries of single examples by sets of examples, which allows for non-myopic active learning, and complement it by adaptive active learning methods to combine single criteria depending on the dataset used.

## References

1. Quiñonero Candela, J., Rasmussen, C.E.: A unifying view of sparse approximate gaussian process regression. J. Mach. Learn. Res. **6** (2005) 1939–1959
2. Maji, S., Berg, A., Malik, J.: Classification using intersection kernel support vector machines is efficient. In: CVPR. (2008) 1 –8
3. Wu, J.: A fast dual method for hik svm learning. In: ECCV. (2010) 552–565
4. Rodner, E., Freytag, A., Bodesheim, P., Denzler, J.: Large-scale gaussian process classification with flexible adaptive histogram kernels. In: ECCV. (2012)
5. Rasmussen, C.E., Williams, C.K.I.: Gaussian Processes for Machine Learning. Adaptive Computation and Machine Learning. The MIT Press (2006)
6. Kapoor, A., Grauman, K., Urtasun, R., Darrell, T.: Gaussian processes for object categorization. International Journal of Computer Vision **88** (2010) 169–188
7. Kemmler, M., Rodner, E., Denzler, J.: One-class classification with gaussian processes. In: ACCV. (2010) 489–500
8. Lazebnik, S., Schmid, C., Ponce, J.: Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In: CVPR. (2006) 2169–2178
9. Vedaldi, A., Gulshan, V., Varma, M., Zisserman, A.: Multiple kernels for object detection. In: ICCV. (2009)
10. Ebert, S., Fritz, M., Schiele, B.: Ralf: A reinforced active learning formulation for object class recognition. In: CVPR. (2012)
11. Jain, P., Kapoor, A.: Active learning for large multi-class problems. In: CVPR. (2009) 762 –769
12. Tong, S., Koller, D.: Support vector machine active learning with applications to text classification. J. Mach. Learn. Res. **2** (2002) 45–66