Efficient Active Learning for Gaussian Process Classification by Error Reduction

Anonymous Author(s)
Affiliation
Address
email

Abstract

Active learning sequentially selects the best instance for labeling by optimizing an acquisition function to enhance data/label efficiency. The selection can be either from a discrete instance set (pool-based scenario) or a continuous instance space (query synthesis scenario). In this work, we study both active learning scenarios for Gaussian Process Classification (GPC). The existing active learning strategies that maximize the Estimated Error Reduction (EER) aim at reducing the classification error after training with the new acquired instance in a one-step-look-ahead manner. The computation of EER-based acquisition functions is typically prohibitive as it requires retraining the GPC with every new query. Moreover, as the EER is not smooth, it can not be combined with gradient-based optimization techniques to efficiently explore the continuous instance space for query synthesis. To overcome these critical limitations, we develop computationally efficient algorithms for EER-based active learning with GPC. We derive the joint predictive distribution of label pairs as a one-dimensional integral, as a result of which the computation of the acquisition function avoids retraining the GPC for each query, remarkably reducing the computational overhead. We also derive the gradient chain rule to efficiently calculate the gradient of the acquisition function, which leads to the first query synthesis active learning algorithm implementing EER-based strategies. Our experiments clearly demonstrate the computational efficiency of the proposed algorithms. We also benchmark our algorithms on both synthetic and real-world datasets, which show superior performance in terms of sampling efficiency compared to the existing state-of-the-art algorithms.

1 Introduction

Compared to traditional passive learning with randomly sampled training instances, active learning aims at "optimally" querying instances for labeling to achieve label efficiency when training machine learning models, especially when labeling is difficult or costly. There are two fundamental scenarios of active learning discussed in the literature: query synthesis and pool-based sampling [14]. In query synthesis, the leaner can request labels for any instance generated from a continuous feature space while pool-based sampling selects the instance from a finite set. Query synthesis is more challenging due to the infinite search space and inherent higher label uncertainty. Recent research on query synthesis with deep generative models has shown promising potential [18, 11]. However, in many science and engineering applications, acquiring the label for even one instance is prohibitively resource-demanding and therefore active learning with deep models may not be practical.

For example, one of critical materials science research questions is to identify phase transition diagrams, where the phase transition response surface can be complex [8]. Identifying phase transitions can be formulated as finding the optimal classification boundaries between different

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

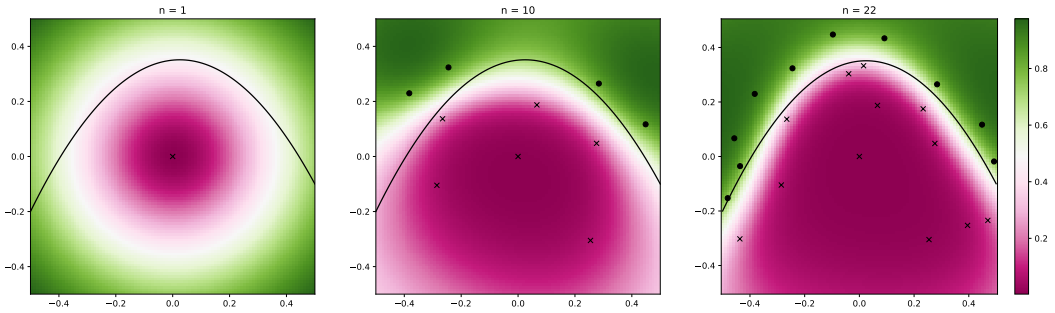


Figure 1: Phase diagram identification by active learning with Gaussian Process Classification (GPC).

phases in the enormous materials design space. However, precisely knowing the phase of each design with the corresponding input features requires costly and time-consuming materials synthesis and profiling experiments or running complex simulation models. Furthermore, there may exist significant uncertainty in acquired phase labels due to technical limitations. Hence, active learning for optimal Bayesian classifiers is a natural solution to help identify the phase diagram with as few as possible synthesized or simulated materials under such uncertainty and complexity. Gaussian Process Classification (GPC) as a popular and powerful Bayesian classifier with the flexibility of adopting different kernels [15], is suitable for solving this problem with appropriate active learning strategies. Fig.1 shows an example of phase identification in a two-dimensional design space by active learning with GPC (Details in the Appendix). In the figure, the black solid line indicates the phase transition response surface, the crosses and dots indicate the queries of different phases, and the colorbar indicates the predictive distribution of GPC. From the figure we can see, the predictive distribution identifies the phase transition boundary with a few samples guided by active learning using GPC surrogate models, which has significant cost and time savings compared to the traditional trial-and-error phase diagram identification paradigm in the current materials science practice.

Many active learning strategies have been proposed for GPC. For example, Bayesian Active Learning by Disagreement (BALD) selects the instance with the maximum mutual information between the observation and the derived uncertain model [5]. There also have been strategies targeting at reducing classification error directly or indirectly. Estimated Error Reduction (EER) strategies optimize for the error reduction after training with the new queries [13, 10, 6]. We note here that EER-based active learning has been studied for both Gaussian Process Regression (GPR) and GPC [13, 6]. For GPR, the regression error can be represented by the posterior predictive variance with the analytical expression and the acquisition function is easy to calculate for efficient active learning. In contrast, for GPC, there is no analytical expression for the posterior predictive. The model updates need approximate inference, such as Expectation Propagation (EP), an algorithm iteratively approximating the GPC posterior [7]. Moreover, the classification error computation requires the new query labels; so the calculation of the corresponding EER-based acquisition function requires incrementally retraining the model for each possible query label to calculate the expected future error as the utility to guide active learning. The complexity of training GPC with EP approximation is $O(n^3)$, where n is the number of observed data. This has been the main hurdle to develop active learning for GPC models.

To reduce the calculation cost of EER, the paper [6] proposed to use a non-iterative but less accurate method approximating the new posterior when calculating the acquisition functions. The paper [3] proposed a novel approximated error reduction (AER) criterion, in which the error reduction of a candidate is estimated based on the impact over its nearby instances. The approximated estimation avoids re-inferring the labels of massive instances. These methods are relatively efficient in computation. But besides EP approximation, they need additional approximations in calculating acquisition functions so the acquisition functions are not precisely calculated, which may degrade the desired data efficiency.

In this paper, within the EER-based active learning framework for GPC, we develop computationally efficient algorithms to compute EER to guide both query synthesis and pool-based active learning. In particular, we consider EER as the reduction of the Mean Objective Cost of Uncertainty (MOCU) [16] since the learning objective of GPC, in particular for identifying phase diagram, is to reduce the classification error. By deriving the joint distribution of queries as a one-dimensional integral, we avoid retraining the GPC for each query when calculating the EER/MOCU-based acquisition function. We further leverage a smooth approximation of MOCU, Soft MOCU (SMOCU) [17], to enable

efficient gradient computation of the SMOCU reduction by deriving the corresponding chain rule for efficient query synthesis with GPC. To the best of our knowledge, this is the first algorithm for query synthesis active learning based on EER strategies. We show in our experiments that our algorithms accelerate the computation of the acquisition functions. Compared with other existing algorithms, we demonstrate consistent data efficiency of our algorithms with both synthetic and real-world datasets.

2 Problem Setting and Background

2.1 Gaussian Process Classification (GPC)

Consider a binary classification problem with the instance space $\mathcal{X}$ and binary label set $\mathcal{Y} = \{0, 1\}$, we aim to train a classifier $\psi : \mathcal{X} \to \mathcal{Y}$ to predict labels for unobserved instances $\psi(\boldsymbol{x}_*), \boldsymbol{x}_* \in \mathcal{X}$. The Gaussian Process classification (GPC) framework connects the instance and the label by a latent function f, which is a random process depending on $\boldsymbol{x}$. Assume that f follows a Gaussian Process (GP) prior $f \sim \text{GP}(\mu(\cdot), k(\cdot, \cdot))$, where $\mu(\cdot)$ is a mean function and $k(\cdot, \cdot)$ is a covariance kernel function [9]. GP is a popular and powerful model for both regression and classification. A good property of GP is that given any finite number of instances $\boldsymbol{x}_i$, the joint distribution of $f(\boldsymbol{x}_i)$ is still Gaussian. For classification problems, given f, the label y takes a Bernoulli distribution with probability $p(y = 1|\boldsymbol{x}, f) = \Phi(f(\boldsymbol{x}))$, where Φ is the Gaussian cumulative distribution function.

Given a sequence of observations $D = \{X, Y\}$ with $X = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n\}$ and $Y = \{y_1, y_2, \ldots, y_n\}$, the class labels are conditionally independent given the latent function. Therefore, the joint likelihood can be factorized as: $p(Y|X, f) = \prod_i p(y_i|\boldsymbol{x}_i, f)$. Since the likelihood probability is non-Gaussian, the posterior $\pi(f|X, Y) \propto \pi(f)p(Y|f, X)$ can not be computed analytically and approximation are often adopted. The Expectation Propagation (EP) algorithm approximates $\pi(f|X, Y)$ with a Gaussian approximation $q(f|X, Y) = \mathcal{N}(f|\tilde{\mu}(\cdot), \tilde{\Sigma}(\cdot, \cdot))$ by iteratively moment matching marginal posteriors [9].

With the approximated posterior, the marginal distribution of $f_* = f(\boldsymbol{x}_*)$ is still Gaussian. Denote the mean and variance as μ_* and σ_{**} respectively, then there is a closed-form expression for $p(y_*|\boldsymbol{x}_*, X, Y)$ as:

$$p(y_* = 1|X, Y, \boldsymbol{x}_*) = \int \Phi(f_*)\phi(f_*|\mu_*, \sigma_*^2)df_* = \Phi(\frac{\mu_*}{\sqrt{1 + \sigma_{**}}}). \tag{1}$$

Given the predictive probability, we assume the prediction of the instance is the most probable label $\arg\max_{y_*} p(y_*|X, Y, \boldsymbol{x}_*)$, which is known as the Optimal Bayesian Classifier (OBC) [1].

2.2 Bayesian Active Learning Based on Estimated Error Reduction (EER)

In the Bayesian active learning procedure, we consider a general case that the label given by the annotator is not deterministic, but the label follows an unknown distribution instead. Given the GPC model and observations, active learning sequentially chooses a query from a discrete instance set or a continuous instance space, to efficiently improve the prediction performance of GPC. In each iteration, the query is chosen by optimizing an acquisition function. The acquisition function reflects the anticipated benefit of the queries, and it usually depends on the model posterior $\pi(f|X, Y)$. After observing the new data $(\boldsymbol{x}_*, y_*)$, the model is updated by Bayes' rule to get the posterior $\pi(f|X, Y, \boldsymbol{x}_*, y_*)$. In the following content, we'll discuss the iteration of active learning after observing training data (X, Y), for the sake of clarity, we omit X, Y from the notations of posterior and posterior predictive distributions as $\pi(f)$ and $p(y|\boldsymbol{x})$.

Assume the distribution of instances over $\mathcal{X}$ is $\boldsymbol{x}_s \sim p(\boldsymbol{x}_s)$, denote the optimal Bayesian classifier (OBC) as $\psi_{\pi(f)}(\cdot)$. Based on GPC, the expected classification error of the OBC is:

$$\mathbb{E}_{\boldsymbol{x}_s}\{1 - p(y_s = \psi_{\pi(f)}(\boldsymbol{x}_s)|\boldsymbol{x}_s)\} = \mathbb{E}_{\boldsymbol{x}_s}\{1 - \max_{y_s} p(y_s|\boldsymbol{x}_s)\}. \tag{2}$$

In the GPC setting, if the latent function f is known, we can accordingly label each instance with the most probable label as $\psi_f(\boldsymbol{x}) = \arg\max_y p(y|\boldsymbol{x}, f)$, where $\psi_f(\cdot)$ denotes the optimal classifier. The classification error of ψ_f is $\mathbb{E}_{\boldsymbol{x}_s}\{1 - \max_y p(y|\boldsymbol{x}, f)\}$. Due to the model uncertainty, the classification error of OBC should be larger than that of the optimal classifier. The Mean Objective Cost of Uncertainty (MOCU) measures the classification error increase due to the model

uncertainty [16]. It is defined as the OBC classification error minus the expected classification error of the optimal classifier under model uncertainty:

$$\mathcal{M}(\pi(f)) = \mathbb{E}_{\boldsymbol{x}_s}\{1 - \max_{y_s} p(y_s|\boldsymbol{x}_s)\} - \mathbb{E}_{\pi(f)}\{\mathbb{E}_{\boldsymbol{x}_s}[1 - \max_{y_s} p(y_s|\boldsymbol{x}_s, f)]\}. \tag{3}$$

As MOCU captures the direct influence of the model uncertainty on the classification error, we can take the MOCU reduction in a one-step-look-ahead setting as the acquisition function:

$$U^{\mathrm{M}}(\boldsymbol{x}_*; \pi(f)) = \mathcal{M}(\pi(f)) - \mathbb{E}_{y_*|\boldsymbol{x}_*}[\mathcal{M}(\pi(f|\boldsymbol{x}_*, y_*))] \tag{4}$$

The second term is the MOCU after observing $(\boldsymbol{x}_*, y_*)$, and averages over $p(y_*|\boldsymbol{x}_*)$ since we do not know the label of y_* before observation. It is easy to verify that in (4), the $\max_{y_s} p(y_s|\boldsymbol{x}_s, f)$ terms in $\mathcal{M}(\pi(f))$ and $\mathcal{M}(\pi(f|\boldsymbol{x}_*, y_*))$ are cancelled and the acquisition function can be proved to be equivalent to the Expected Error Reduction (EER) of OBC [10, 17]:

$$\begin{aligned} U^{\mathrm{M}}(\boldsymbol{x}_*) &= \mathbb{E}_{\boldsymbol{x}_s}\{1 - \max_{y_s} p(y|\boldsymbol{x}_s)\} - \mathbb{E}_{y_*|\boldsymbol{x}_*}\{\mathbb{E}_{\boldsymbol{x}_s}[1 - \max_{y_s} p(y_s|\boldsymbol{x}_s, \boldsymbol{x}_*, y_*)]\} \\ &= \mathbb{E}_{\boldsymbol{x}_s}\{\mathbb{E}_{y_*|\boldsymbol{x}_*}[\max_{y_s} p(y_s|\boldsymbol{x}_s, \boldsymbol{x}, y)] - \max_{y_s} p(y_s|\boldsymbol{x}_s)\}, \end{aligned} \tag{5}$$

Although MOCU reduction or EER is optimal for single queries, with noisy observation labels, it may get stuck before converging to the optimal classifier as discussed in [16]. To address this issue, the paper [17] proposed a smooth concave approximation of MOCU, called Soft-MOCU (SMOCU), which replaces the $\max_{y_s} p(y_s|\boldsymbol{x}_s)$ term in (3) with a `LogSumExp` term:

$$\mathcal{M}^{\mathrm{S}}(\pi(f)) = \mathbb{E}_{\boldsymbol{x}_s}\{1 - \frac{1}{k}\texttt{LogSumExp}(k \cdot p(y|\boldsymbol{x}_s)\} - \mathbb{E}_{\pi(f)}\{\mathbb{E}_{\boldsymbol{x}_s}[1 - \max_{y_s} p(y_s|\boldsymbol{x}_s, f)]\}, \tag{6}$$

where k is a parameter controlling the approximation to MOCU. The resulting acquisition function based on the SMOCU reduction can guarantee the convergence to the optimal classifier:

$$U^{\mathrm{S}}(\boldsymbol{x}_*) = \mathbb{E}_{\boldsymbol{x}_s}\{\mathbb{E}_{y_*|\boldsymbol{x}_*}[\frac{1}{k}\texttt{LogSumExp}(k \cdot p(y_s|\boldsymbol{x}_s, \boldsymbol{x}_*, y_*))] - \frac{1}{k}\texttt{LogSumExp}(k \cdot p(y_s|\boldsymbol{x}_s))\}. \tag{7}$$

The resulting acquisition function is a smooth function of $p(y_s|\boldsymbol{x}_s, \boldsymbol{x}, y)$ and has no maximization operators. We will leverage this smooth acquisition function to derive the first efficient gradient-based query synthesis active learning algorithm for GPC.

3 Efficient Active Learning for GPC

In this section, we present our EER-based active learning algorithms for GPC based on the acquisition functions defined by the MOCU and SMOCU reduction. At each iteration, with the updated GPC given previous observations, the acquisition function (5) or (7) can be optimized to guide the selection of the next query for active learning. We first present a straightforward algorithm for both the discrete instance set (pool-based sampling) and continuous instance space (query synthesis) scenarios, where the acquisition function is optimized by random optimization. Random optimization first collects a random sample set $\mathcal{X}_* \subset \mathcal{X}$ of size M_1, calculates the acquisition function for each sample in the set and then takes the instance with the maximum acquisition function value as the query.

To calculate (5) or (7), the integral over $\mathcal{X}$ space is not analytical. Hence we need to calculate the integral by Monte Carlo sampling with M_2 samples of $\boldsymbol{x}_s \in \mathcal{X}$. Define $g^{\mathrm{M}}(\boldsymbol{x}_s; \boldsymbol{x}_*)$ and $g^{\mathrm{S}}(\boldsymbol{x}_s; \boldsymbol{x}_*)$ such that $U^{\mathrm{M}}(\boldsymbol{x}_*) = \mathbb{E}_{\boldsymbol{x}_s}\{g^{\mathrm{M}}(\boldsymbol{x}_s; \boldsymbol{x}_*)\}$ and $U^{\mathrm{S}}(\boldsymbol{x}_*) = \mathbb{E}_{\boldsymbol{x}_s}\{g^{\mathrm{S}}(\boldsymbol{x}_s; \boldsymbol{x}_*)\}$. Let $g(\boldsymbol{x}_s; \boldsymbol{x}_*)$ denote either $g^{\mathrm{M}}(\boldsymbol{x}_s; \boldsymbol{x}_*)$ or $g^{\mathrm{S}}(\boldsymbol{x}_s; \boldsymbol{x}_*)$. For each $\boldsymbol{x}_s$, the calculation of $g(\boldsymbol{x}_s; \boldsymbol{x}_*)$ requires deriving the probability distribution of $p(y_s|\boldsymbol{x}_s)$ and $p(y_s|\boldsymbol{x}_s, \boldsymbol{x}_*, y_*), \forall y_* \in \mathcal{Y}$. Here, $p(y_s|\boldsymbol{x}_s)$ can be calculated directly from (1) while updating $p(y_s|\boldsymbol{x}_s, \boldsymbol{x}_*, y_*)$ needs incremental training of GPC with observations $\{X, Y, \boldsymbol{x}_*, y_*\}$ based on the EP approximation, and then calculating (1). The whole procedure of optimizing the acquisition function at the n-th iteration need to retrain GPC for each possible pair $(\boldsymbol{x}_*, y_*)$, therefore, the EP approximation needs to be performed $2 \times M_1$ times. The whole procedure is illustrated as the pseudocode *Algorithm 1* in the Appendix.

There are three issues of the acquisition function calculation in the straight forward algorithm. First, we need a large number of samples to have a reliable estimation of the integral in (5) or (7). Second, the calculation of $p(y_s|\boldsymbol{x}_s, \boldsymbol{x}_*, y_*)$ requires incremental retraining of the GPC model for each pair of $(\boldsymbol{x}_*, y_*)$, with computational complexity $O(M_1 n^3)$. Third, even though $U^{\mathrm{S}}(\boldsymbol{x}_*)$ is a differentiable

function of $p(y_s|\boldsymbol{x}_s, \boldsymbol{x}_*, y_*)$, in the algorithm we actually use the EP approximation $q(f|\boldsymbol{x}_*, y_*)$ to calculate $U^{\mathrm{S}}(\boldsymbol{x}_*)$, and it is impossible to calculate the gradient $\nabla q(f|\boldsymbol{x}_*, y_*)$ during the EP procedure.

We develop our EER-based active learning algorithms to address these three presented challenges: 1) By importance sampling leveraging inhere GPC assumptions, we reduce the required number of samples for estimating acquisition functions; 2) We compute $p(y_s|\boldsymbol{x}_s, \boldsymbol{x}_*, y_*)$ by deriving analytic solution to marginalize the joint distribution $p(y_s, y_*|\boldsymbol{x}_s, \boldsymbol{x}_*)$ from the approximated posterior $q(f|X, Y)$, which avoids the retraining with EP approximation; 3) More critically, we derive the first gradient-based query synthesis algorithm when using the SMOCU-based acquisition function by deriving the gradient $\nabla q(f|\boldsymbol{x}_*, y_*)$ for efficient active learning.

3.1 Importance Sampling

Regarding the issue in requiring the high sampling number of $\boldsymbol{x}_s$ for reliable estimation of the acquisition function, we notice that most of the kernels applied in GPC assume that the observed data only have influence on neighboring regions. Hence, we only need to account for the samples near $\boldsymbol{x}_*$ to estimate its influence on the classification error, thereafter the acquisition function for each new query. More specifically, we can use importance sampling to reduce the required sampling number. Reformulate the expectation $U(\boldsymbol{x}_*) = \mathbb{E}_{\boldsymbol{x}_s \sim p(\boldsymbol{x}_s)}[g(\boldsymbol{x}_s; \boldsymbol{x}_*)]$ over an assistant distribution $\tilde{p}(\boldsymbol{x}_s; \boldsymbol{x}_*)$ as: $U(\boldsymbol{x}_*) = \mathbb{E}_{\boldsymbol{x}_s \sim \tilde{p}(\boldsymbol{x}_s; \boldsymbol{x}_*)}[p(\boldsymbol{x}_s) g(\boldsymbol{x}_s; \boldsymbol{x}_*)/\tilde{p}(\boldsymbol{x}_s; \boldsymbol{x}_*)]$.

The variance of the expectation is minimized when $\tilde{p}(\boldsymbol{x}_s; \boldsymbol{x}_*)$ is proportional to $p(\boldsymbol{x}_s) g(\boldsymbol{x}_s; \boldsymbol{x}_*)$. But this requires knowledge of the value of $U(\boldsymbol{x}_*)$, which is the acquisition function that we try to estimate. In practice, we can choose $\tilde{p}(\boldsymbol{x}_s; \boldsymbol{x}_*) \propto k(\boldsymbol{x}_s, \boldsymbol{x}_*) p(\boldsymbol{x}_s)$ instead, as $k(\boldsymbol{x}_s, \boldsymbol{x}_*)$ reflects the non-zero region of $g(\boldsymbol{x}_s; \boldsymbol{x}_*)$: $k(\boldsymbol{x}_s, \boldsymbol{x}_*) \approx 0$ means $(\boldsymbol{x}_s, y_s)$ and $(\boldsymbol{x}_*, y_*)$ are independent, thus $p(y_s|\boldsymbol{x}_s, \boldsymbol{x}_*, y_*) \approx p(y_s|\boldsymbol{x}_s)$ and $g(\boldsymbol{x}_s; \boldsymbol{x}_*) \approx 0$. For example, if $p(\boldsymbol{x}_s)$ is uniformly distributed within a finite region, $k(\boldsymbol{x}_s, \boldsymbol{x}_*)$ is a square exponential kernel, then $\tilde{p}(\boldsymbol{x}_s; \boldsymbol{x}_*)$ can be chosen as a truncated Gaussian distribution. Note that importance sampling and random optimization is only suitable for continuous instance space, or discrete set with large cardinality, For small instance set, we can traverse all the elements for calculating the expectation and optimizing the acquisition function.

3.2 Joint Distribution Calculation

To avoid the retraining of GPC for each $(\boldsymbol{x}_*, y_*)$, we can calculate the posterior predictive by $p(y_s|\boldsymbol{x}_s, \boldsymbol{x}_*, y_*) = p(y_s, y_*|\boldsymbol{x}_s, \boldsymbol{x}_*)/p(y_*|\boldsymbol{x}_*)$, which requires computing the joint distribution of $p(y_s, y_*|\boldsymbol{x}_s, \boldsymbol{x}_*)$. We remind the reader all the probabilities are conditioned on $\{X, Y\}$ and we omit them for the seek of brevity. Denote $f_s = f(\boldsymbol{x}_s)$ and $f_* = f(\boldsymbol{x}_*)$. Now we show how to simplify the calculation of $p(y_s, y_*|\boldsymbol{x}_s, \boldsymbol{x}_*)$. In the Gaussian approximation of the posterior $q(f|X, Y)$, the joint distribution of f_s, f_* is still Gaussian. Since y_s and y_* are conditionally independent given f_s and f_*, the joint distribution can be expressed as:

$$p(y_s = 1, y_* = 1|\boldsymbol{x}_s, \boldsymbol{x}_*) = \iint \Phi(f_s)\Phi(f_*)\phi(f_s, f_*|\mu_{s*}, \Sigma_{s*}) df_s df_*, \tag{8}$$

where μ_{s*} and Σ_{s*} are the marginal mean and covariance matrix of f_s and f_*. This integral can be simplified as a one-dimensional integral. Denote $\mu_{s*} = \begin{pmatrix} \mu_s \\ \mu_* \end{pmatrix}$ and $\Sigma_{s*} = \begin{pmatrix} \sigma_{ss} & \sigma_{s*} \\ \sigma_{s*} & \sigma_{**} \end{pmatrix}$. We can decompose the joint Gaussian distribution as the marginal distribution of f_s times the conditional distribution of f_* given f_s, i.e. $\phi(f_s, f_*|\mu_{s*}, \Sigma_{s*}) = \phi(f_s|\mu_s, \sigma_{ss})\phi(f_*|\tilde{\mu}_*(f_s), \tilde{\sigma}_{**})$, where $\tilde{\mu}_*(f_s) = \mu_* + (f_s - \mu_s)\sigma_{s*}/\sigma_{ss}$ and $\tilde{\sigma}_{**} = \sigma_{**} - \sigma_{s*}^2/\sigma_{ss}$. Then, (8) can be transformed as:

$$\begin{aligned} p(y_s = 1, y_* = 1|\boldsymbol{x}_s, \boldsymbol{x}_*) &= \iint \Phi(f_*)\Phi(f_s)\phi(f_s, f_*|\mu_{s*}, \Sigma_{s*}) df_s df_*, \\ &= \iint \Phi(f_*)\phi(f_*|\tilde{\mu}_*(f_s), \tilde{\sigma}_{**}) df_* \ \Phi(f_s)\phi(f_s|\mu_s, \sigma_{ss}) df_s \\ &= \int \Phi(\frac{\tilde{\mu}_*(f_s)}{\sqrt{\tilde{\sigma}_{**} + 1}})\Phi(f_s)\phi(f_s|\mu_s, \sigma_{ss}) df_s. \end{aligned} \tag{9}$$

The last line is based on the integral equation introduced in [9]. The above equation (9) calculates the joint distribution with the 1-d integral in constant time. With the joint distribution $p(y_s = 1, y_* = 1|\boldsymbol{x}_s, \boldsymbol{x}_*)$, we can easily obtain the joint distribution of y_s, y_* with other label pairs, and

finally we can obtain the posterior predictive $p(y_s|\boldsymbol{x}_s, \boldsymbol{x}_*, y_*)$ without retraining the GPC with EP approximation. Based on this solution with importance sampling, we can develop efficient algorithms estimating MOCU/SMOCU reduction. Combining with Random Optimization, we develop the active learning algorithm with MOCU reduction named Non-Retraining MOCU reduction with Random Optimization (NR-MOCU-RO), and similarly for SMOCU reduction NR-SMOCU-RO. The pseudocode of NR-(S)MOCU-RO can be found in the Appendix (*Algorithm 2*).

3.3 Gradient Calculation

With the introduced marginalization strategy and importance sampling, we can significantly improve the computational efficiency for pool-based active learning with GPC. However, in the query synthesis problems, we would like to optimize the acquisition functions with gradient-based algorithms. Usually the acquisition functions are multi-modal in the feature space, so the common practice is to perform random optimization first, and then take the optimal point as the initial point to perform the gradient-based algorithms [4].

Here we consider the gradient calculation of the acquisition function based on the SMOCU reduction $\nabla U^{\mathrm{S}}(\boldsymbol{x}_*)$ for query synthesis active learning as $U^{\mathrm{S}}(\boldsymbol{x}_*)$ is a smooth function whose gradients exist everywhere. For a Gaussian Process, the gradients of its mean and covariance functions have closed-form expressions of the gradients of its adopted kernel function. Here we assume that given the EP approximation $q(f|X, Y)$ and any pair of points $(\boldsymbol{x}_s, \boldsymbol{x}_*)$, we already know the gradient of $\nabla\mu_*$, $\nabla\sigma_{**}$ and $\nabla\sigma_{s*}$ with respect to $\boldsymbol{x}_*$. With this assumption, we can use the chain rule to compute the gradients and finally express the gradients $U^{\mathrm{S}}(\boldsymbol{x}_*)$ in the form of $\nabla\mu_*$, $\nabla\sigma_{**}$ and $\nabla\sigma_{s*}$.

In (7), the second term is unrelated to $\boldsymbol{x}_*$, so the gradient of the SMOCU reduction is:

$$\begin{aligned}\nabla U^{\mathrm{S}}(\boldsymbol{x}_*) &= \mathbb{E}_{\boldsymbol{x}_s}[\nabla g^{\mathrm{S}}(\boldsymbol{x}_s, \boldsymbol{x}_*)] = \nabla\mathbb{E}_{\boldsymbol{x}_s}\{\sum_{y_*} p(y_*|\boldsymbol{x}_*)\frac{1}{k}\texttt{LogSumExp}[k \cdot p(y_s|\boldsymbol{x}_s, \boldsymbol{x}_*, y_*)]\} \\ &= \sum_{y_*} \nabla p(y_*|\boldsymbol{x}_*) \cdot \mathbb{E}_{\boldsymbol{x}_s}\{\frac{1}{k}\texttt{LogSumExp}[k \cdot p(y_s|\boldsymbol{x}_s, \boldsymbol{x}_*, y_*)]\} \\ &+ \sum_{y_*} p(y_*|\boldsymbol{x}_*) \cdot \mathbb{E}_{\boldsymbol{x}_s}\{\nabla\frac{1}{k}\texttt{LogSumExp}[k \cdot p(y_s|\boldsymbol{x}_s, \boldsymbol{x}_*, y_*)]\}. \end{aligned} \tag{10}$$

In the first term, $\nabla p(y_*|\boldsymbol{x}_*)$ can be calculated with $\nabla\mu_*$ and $\nabla\sigma_{**}$ based on (1). For the second term, we can use the chain rule to compute $\nabla\texttt{LogSumExp}[k \cdot p(y_s|\boldsymbol{x}_s, \boldsymbol{x}_*, y_*)] = g_1 \cdot g_2$, where:

$$g_1 = \frac{\partial\texttt{LogSumExp}[k \cdot p(y_s|\boldsymbol{x}_s, \boldsymbol{x}_*, y_*)]}{\partial p(y_s|\boldsymbol{x}_s, \boldsymbol{x}_*, y_*)}, \qquad g_2 = \nabla p(y_s|\boldsymbol{x}_s, \boldsymbol{x}_*, y_*). \tag{11}$$

Since $p(y_s|\boldsymbol{x}_s, \boldsymbol{x}_*, y_*) = p(y_s, y_*|\boldsymbol{x}_s, \boldsymbol{x}_*)/p(y_*|\boldsymbol{x}_*)$ and we have calculated $\nabla p(y_*|\boldsymbol{x}_*)$, now we only need to calculated $\nabla p(y_s, y_*|\boldsymbol{x}_s, \boldsymbol{x}_*)$ based on (9):

$$\begin{aligned}\nabla p(y_s = 1, y_* = 1|\boldsymbol{x}_s, \boldsymbol{x}_*) =& \nabla \int \Phi(f_s)\phi(f_s|\mu_s, \sigma_{ss})\Phi(\frac{\tilde{\mu}_*(f_s)}{\sqrt{\tilde{\sigma}_{**}+1}})df_s \\ =& \int \Phi(f_s)\phi(f_s|\mu_s, \sigma_{ss})\phi(\frac{\tilde{\mu}_*(f_s)}{\sqrt{\tilde{\sigma}_{**}+1}}) \cdot \nabla(\frac{\tilde{\mu}_*(f_s)}{\sqrt{\tilde{\sigma}_{**}+1}})df_s, \end{aligned} \tag{12}$$

which is again a 1-d integral. The gradient of $\nabla(\frac{\tilde{\mu}_*(f_s)}{\sqrt{\tilde{\sigma}_{**}+1}})$ can be again calculated by chain rule, and connected to the calculation of $\nabla\tilde{\mu}_*(f_s)$ and $\nabla\tilde{\sigma}_{**}$:

$$\nabla\tilde{\mu}_*(f_s) = \nabla\mu_* + \frac{f_s - \mu_s}{\sigma_{ss}}\nabla\sigma_{s*}, \qquad \nabla\tilde{\sigma}_{**} = \nabla\sigma_{**} - \frac{2\nabla\sigma_{s*}}{\sigma_{ss}}. \tag{13}$$

Therefore, $\nabla(\frac{\tilde{\mu}_*(f_s)}{\sqrt{\tilde{\sigma}_{**}+1}})$ is a linear function of f_s, and we can use numerical integral methods to calculate (12). The query synthesis algorithm with the integral computation is summarized in the pseudocode: NR-SMOCU with Stochastic Gradient Descent (NR-SMOCU-SGD).

In summary, to reduce the number of samples $\boldsymbol{x}_s$ for relable estimation of acquisition functions, our algorithm utilizes an importance sampling with an assistant distribution chosen according to the kernel function. By calculating the posterior predictive directly from the joint distribution, the algorithm avoids retraining GPC with EP approximation. The introduction of the joint distribution also enables the efficient calculation of the gradient of the smooth acquisition function, with which we develop an efficient active learning algorithm for query synthesis.

NR-SMOCU-SGD: n-th iteration

function GRADIENTOPT($p(\boldsymbol{x}), q(f|X,Y)$)
  Obtain initial point $\boldsymbol{x}_*$ from RANDOMOPT($p(\boldsymbol{x}), q(f|X,Y)$)
  while not converge **do**
    Sample M_2 samples of $\boldsymbol{x}_s \sim \tilde{p}(\boldsymbol{x}_s; \boldsymbol{x}_*)$
    Calculate $p(y_*|\boldsymbol{x}_*)$ and $\nabla p(y_*|\boldsymbol{x}_*)$ by (1)
    for each $\boldsymbol{x}_s$ **do**
      for y in $\{0,1\}$ **do**
        Calculate $p(y_s, y_*|\boldsymbol{x}_s, \boldsymbol{x}_*)$, $\nabla p(y_s, y_*|\boldsymbol{x}_s, \boldsymbol{x}_*)$ and $p(y_s|\boldsymbol{x}_s)$ by (1, 9, 12)
      end for
      Calculate $\nabla g^{\mathrm{S}}(\boldsymbol{x}_s, \boldsymbol{x}_*)$ by (10 - 12)
    end for
    $\nabla U^{\mathrm{S}}(\boldsymbol{x}_*) = \frac{1}{M_2} \sum_{\boldsymbol{x}_s} p(\boldsymbol{x}_s) \nabla g^{\mathrm{S}}(\boldsymbol{x}_s; \boldsymbol{x}_*) / \tilde{p}(\boldsymbol{x}_s; \boldsymbol{x}_*)$
    Update $\boldsymbol{x}_*$ with $\nabla U^{\mathrm{S}}(\boldsymbol{x}_*)$
  end while
  return $\boldsymbol{x}_*$
end function

4 Experiments

In this section we demonstrate the efficiency of our active learning algorithms combined with either random optimization (NR-MOCU-RO, NR-SMOCU-RO) or Adagrad (NR-SMOCU-SGD) in the following sets of experiments. In the first set of experiments, we analyze and benchmark the running time of our algorithm by comparing to the naive computation of the MOCU/SMOCU reduction. Then we benchmark our algorithms with other active learning algorithms, including random sampling, Maximum Entropy Search (MES) [12] and Bayesian Active Learning by Disagreement (BALD) [5] for both query synthesis on synthetic benchmark datasets, and pool-based active learning on real-world datasets. In our experiments, we use GP prior for f with the squared-exponential kernels $k(\boldsymbol{x}, \boldsymbol{x}') = \gamma^2 \exp(-\|\boldsymbol{x} - \boldsymbol{x}'\|^2/l^2)$, where $\{\gamma, l\}$ are model hyperparameters. The label probability is modeled with the probit function as $p(y|\boldsymbol{x}, f) = \Phi(f(\boldsymbol{x}))$.

4.1 Estimation Accuracy And Running Time Comparison

We first evaluate the effect of using the joint distribution integral in calculating the acquisition functions. We compare the estimation of $p(y_s|\boldsymbol{x}_s, \boldsymbol{x}_*, y_*)$ through the joint distribution integral (9), with $p'(y_s|\boldsymbol{x}_s, \boldsymbol{x}_*, y_*)$ estimated by retraining GPC with EP approximation. For this set of experiments, we generate the initial data points using a latent function f sampled form the GP prior. The instance space $\mathcal{X} = [-4, 4]$, and the hyperparameters $\gamma^2 = 0.5$, $l^2 = 1$. We initially sample 100 data points to train GPC, then we compare the values $g = \texttt{LogSumExp}(k \cdot p(y_s|\boldsymbol{x}_s, \boldsymbol{x}, y))/k$ and $g' = \texttt{LogSumExp}(k \cdot p'(y_s|\boldsymbol{x}_s, \boldsymbol{x}, y))/k$, since these are related to the calculation of the SMOCU reduction. With 1000 pairs of $(\boldsymbol{x}_s, \boldsymbol{x}_*)$ randomly sampled from $\mathcal{X}$, we calculate the absolute error ratio as $|g - g'|/g'$. The average absolute error ratio is *1.8e-5* and the maximum error ratio is *2.4e-3*. We also compare the values $g = \max p(y_s|\boldsymbol{x}_s, \boldsymbol{x}, y)$ and $g' = \max p'(y_s|\boldsymbol{x}_s, \boldsymbol{x}, y)$, which is used to calculate the MOCU reduction. The average absolute error ratio is *1.3e-5* and the maximum ratio is *2.2e-3*. These results validate that using (9) can provide accurate estimates of the acquisition functions.

Next, we compare the running time of estimating acquisition functions by three algorithms: 1) a naive algorithm calculating $p(y_s|\boldsymbol{x}_s, \boldsymbol{x}_*, y_*)$ with GPC retraining (naive), 2) sampling $\boldsymbol{x}_s$ by Importance Sampling and calculating $p(y_s|\boldsymbol{x}_s, \boldsymbol{x}_*, y_*)$ with GPC retraining (IS), 3) sampling $\boldsymbol{x}_s$ by Importance Sampling and calculating $p(y_s|\boldsymbol{x}_s, \boldsymbol{x}_*, y_*)$ with the Joint Distribution Integral (IS+JDI). The algorithms are implemented in Python 3.7 on a personal computer with Intel i5-10400 2.9 GHz CPU and 16G RAM. We set the initial datasets of size $n = 10$, 50 and 100, respectively. For all three competing algorithms, we sample 1000 $\boldsymbol{x}_s$'s, and calculate $U^{\mathrm{S}}(\boldsymbol{x}_*)$. We benchmark them with a "ground truth" algorithm: calculating $U^{\mathrm{S}}(\boldsymbol{x}_*)$ by the naive algorithm with *1e6* $\boldsymbol{x}_s$ samples. We perform this comparison for 100 $\boldsymbol{x}_*$'s. The average running time and absolute error rate are shown in Table 1. From the table, the absolute error rate of the naive algorithm with 1000 samples is much larger than the other two as expected since the other two algorithms use importance sampling with

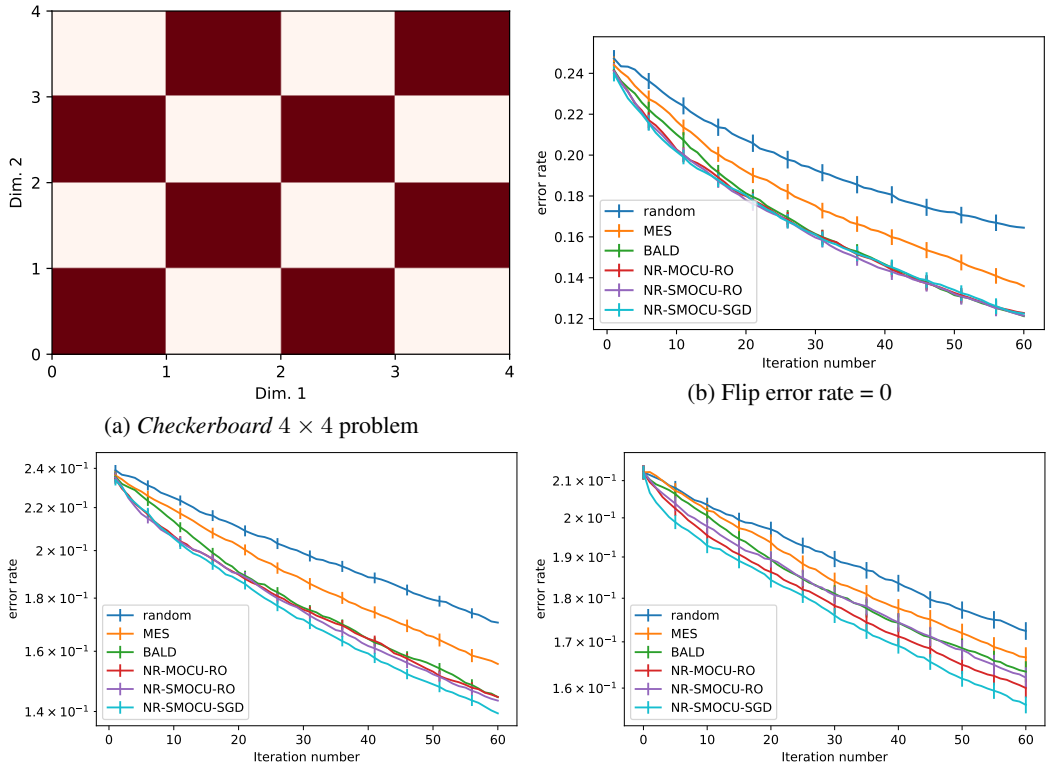
Table 1: Running time (in seconds) and estimation accuracy.

	n = 10		n = 50		n = 100	
Algorithm	**Time (s)**	**Error rate**	**Time (s)**	**Error rate**	**Time (s)**	**Error rate**
naive	1.390	0.166	2.496	0.184	2.704	0.162
IS	1.364	0.012	2.465	0.032	2.685	0.024
IS+JDI	1.950	0.012	2.188	0.028	2.209	0.025

smaller estimation variance. For the running time, retraining GPC in the naive and IS algorithms is the main time-consuming component while IS+JDI spends most of the time in calculating the joint distribution integral for marginalization. As n increase, the running time of both naive and IS algorithms increases fast since GPC training has a complexity of $O(n^3)$, while the running time of IS+JDI does not change much because the joint distribution integral is calculated in constant time. Note that in all three algorithms, inferring the predictive distribution for each $\boldsymbol{x}_s$ also takes a considerable part of computations, especially when n is small. The results show that importance sampling does not impose much extra computation load in active learning. When n is large, the joint distribution integral is faster than retraining GPC with EP approximation.

4.2 Query Synthesis

We now benchmark our proposed algorithms with a challenging synthetic dataset *checkerboard* 4×4, similar as the one tested in [5], which emulates the setup of phase diagram identification in materials science. Fig. 2a illustrates the classification boundaries. We perform all the competing active learning algorithms on this dataset. MES, BALD, NR-MOCU-RO and NR-SMOCU-RO are all optimized by random optimization with $M_2 = 1000$. In NR-SMOCU-SGD, we first perform random optimization with $M_2 = 800$ and set the best point as the initial point for Adagrad so that NR-SMOCU-SGD has similar running time compared with NR-SMOCU-RO at their corresponding setups for fair comparison. Algorithm performance is measured in terms of the error regret defined as the OBC error at n-th iteration minus the optimal classifier error of the simulated ground truth. To



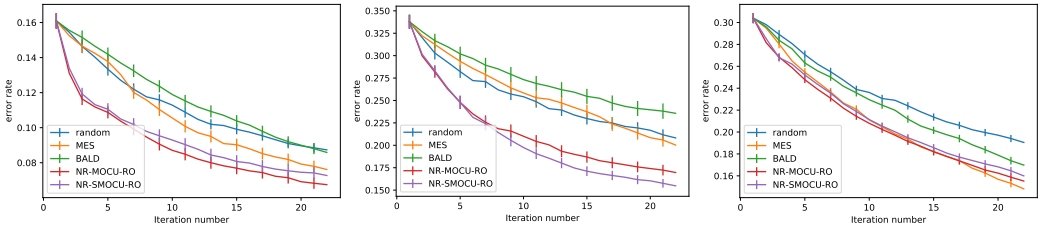
(a) *Checkerboard* 4×4 problem

(b) Flip error rate = 0

(c) Flip error rate = 0.1

(d) Flip error rate = 0.2

Figure 2: Comparison of expected OBC error regret on the 4×4 *checkerboard* problem.



(a) WDBC (b) Ionosphere (c) Vehicle

Figure 3: Classification error rate comparison on UCI datasets.

show the influence of the observation error on the performance of different algorithms, we further assume that there is a constant flip error rate on the observing labels, and we take different error rate equal to 0, 0.1 and 0.2. In the experiments, we initially draw 30 samples for labeling to estimate the hyperparameters $\{\gamma, l\}$, then we perform different algorithms to collect new data. We repeat the procedure for 150 runs and plot the average error regret with standard deviations in Figs. 2b-d.

We can observe from these figures that MES performs bad, which is because MES tends to query the point close to the decision boundary, while this problem has multiple intertwined boundaries and MES cannot differentiate different boundaries. We also observe that as the error rate increases, the difference between NR-MOCU and NR-SMOCU also increases, that is because as the error rate increases, MOCU reduction tends to ignore the long-term effect of a query, leading to the degraded long-term performance, also discussed in [16]. Among these algorithms, NR-SMOCU-SGD performs better than other proposed algorithms by leveraging the gradient information in the optimization procedure.

4.3 Pool-Based Active Learning with Real-World Datasets

We also compare algorithms on the UCI datasets [2] for pool-based active learning. NR-SMOCU-SGD is not included as it is designed to search the continuous space. For each dataset, we split it into training and testing datasets. We take the training dataset as the sampling pool for active learning, initially we randomly choose two samples from each class for labelling, and use them to estimate the GPC hyperparameters. Then we apply the competing active learning algorithms to sequentially select the query from the training dataset and estimate the OBC error with the testing dataset after each iteration. Details of the tested UCI datasets and additional results are provided in the Appendix.

Similarly, we repeat the active learning procedures for 100 runs and plot the average OBC error with standard deviation values for each algorithm for performance comparison shown in Fig. 3. Overall, NR-MOCU-RO and NR-SMOCU-RO are better than MES and BALD, which validates that our algorithms achieve better sample or in particular label efficiency. Note that in the *Vehicle* dataset, MES performs closely to our algorithms. When checking the converging GPC model, the classification boundary is relatively simple on the the *Vehicle* dataset with relatively low-dimensional feature space ($d = 18$). The *WDBC* and *Ionosphere* data are in higher dimensions ($d \geq 30$), for which our proposed algorithms perform significantly better.

5 Conclusions

We have proposed efficient EER-based active learning algorithms with GPC, which estimate the error reduction by querying instances based on the joint distribution of label pairs. We have derived the joint distribution as a one-dimensional integral with constant computation cost and calculate the predictive posterior based on it. Together with importance sampling, the acquisition function can be estimated efficiently by 1-d integral of the joint distribution without incrementally retraining GPC with EP approximation, which has a computation complexity of $O(n^3)$. Without the need for EP approximation, we can further derive the chain rule to calculate the gradient of the SMOCU reduction, which provides us the first efficient gradient-based query synthesis active learning algorithm. Our experiments have demonstrated not only the accuracy and the running speed of our algorithms but also consistently better data/label efficiency compared with competing algorithms for both query synthesis and pool-based active learning.

References

[1] Lori A Dalton and Edward R Dougherty. Optimal classifiers with minimum expected error within a Bayesian framework—part i: Discrete and Gaussian models. *Pattern Recognition*, 46(5):1301–1314, 2013.

[2] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.

[3] Weijie Fu, Meng Wang, Shijie Hao, and Xindong Wu. Scalable active learning by approximated error reduction. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1396–1405, 2018.

[4] José Miguel Hernández-Lobato, Matthew W Hoffman, and Zoubin Ghahramani. Predictive entropy search for efficient global optimization of black-box functions. *arXiv preprint arXiv:1406.2541*, 2014.

[5] Neil Houlsby, Ferenc Huszár, Zoubin Ghahramani, and Máté Lengyel. Bayesian active learning for classification and preference learning. *arXiv preprint arXiv:1112.5745*, 2011.

[6] Ashish Kapoor, Eric Horvitz, and Sumit Basu. Selective supervision: Guiding supervised learning with decision-theoretic active learning. In *IJCAI*, volume 7, pages 877–882, 2007.

[7] Malte Kuss, Carl Edward Rasmussen, and Ralf Herbrich. Assessing approximate inference for binary Gaussian process classification. *Journal of machine learning research*, 6(10), 2005.

[8] GE Poirier, WP Fitts, and JM White. Two-dimensional phase diagram of decanethiol on au (111). *Langmuir*, 17(4):1176–1183, 2001.

[9] Carl Edward Rasmussen. Gaussian processes in machine learning. In *Summer school on machine learning*, pages 63–71. Springer, 2003.

[10] Nicholas Roy and Andrew McCallum. Toward optimal active learning through sampling estimation of error reduction. In *Proceedings of the 18th International Conference on Machine Learning*, ICML '01, pages 441—-448, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[11] Raphael Schumann and Ines Rehbein. Active learning via membership query synthesis for semi-supervised sentence classification. In *Proceedings of the 23rd Conference on Computational Natural Language Learning (CoNLL)*, pages 472–481, 2019.

[12] Paola Sebastiani and Henry P Wynn. Maximum entropy sampling and optimal Bayesian experimental design. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 62(1):145–157, 2000.

[13] Sambu Seo, Marko Wallat, Thore Graepel, and Klaus Obermayer. Gaussian process regression: Active data selection and test point rejection. In *Mustererkennung 2000*, pages 27–34. Springer, 2000.

[14] Burr Settles. Active learning literature survey. Computer Science Technical Report 1648, University of Wisconsin-Madison. 2009.

[15] Andrew Gordon Wilson, Zhiting Hu, Ruslan Salakhutdinov, and Eric P Xing. Deep kernel learning. In *Artificial intelligence and statistics*, pages 370–378. PMLR, 2016.

[16] Guang Zhao, E Dougherty, Byung-Jun Yoon, F Alexander, and Xiaoning Qian. Uncertainty-aware active learning for optimal Bayesian classifier. In *Proc. 9th Int. Conf. Learn. Represent.(ICLR)*, pages 1–20, 2021.

[17] Guang Zhao, Edward Dougherty, Byung-Jun Yoon, Francis J Alexander, and Xiaoning Qian. Bayesian active learning by soft mean objective cost of uncertainty. In *International Conference on Artificial Intelligence and Statistics*, pages 3970–3978. PMLR, 2021.

[18] Jia-Jie Zhu and José Bento. Generative adversarial active learning. *arXiv preprint arXiv:1702.07956*, 2017.

Checklist

The checklist follows the references. Please read the checklist guidelines carefully for information on how to answer these questions. For each question, change the default **[TODO]** to [Yes] , [No] , or [N/A] . You are strongly encouraged to include a **justification to your answer**, either by referencing the appropriate section of your paper or providing a brief inline description. For example:

- Did you include the license to the code and datasets? [Yes] See Section.
- Did you include the license to the code and datasets? [No] The code and the data are proprietary.
- Did you include the license to the code and datasets? [N/A]

Please do not modify the questions and only use the provided macros for your answers. Note that the Checklist section does not count towards the page limit. In your paper, please delete this instructions block and only keep the Checklist section heading above along with the questions/answers below.

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] See Sections 1 & 5. In addition to deriving label-efficient EER-based active learning for GPC with improved computational efficiency, we have derived the first gradient-based query synthesis algorithm for GPC.
   (b) Did you describe the limitations of your work? [Yes] See the discussions in Sections 2 & 3. Active learning algorithms in this work are derived for GPC using probit likelihood, which may have its inherent limitation in high-dimensional feature space.
   (c) Did you discuss any potential negative societal impacts of your work? [No] The paper focuses on active learning algorithm development, which helps reduce the cost and time of machine learning and experimental design. We do not foresee immediate negative societal impact if not implemented for malicious machine learning applications.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes] The settings for deriving the joint distribution and gradient calculation are based on basic calculus as presented in Section 3.
   (b) Did you include complete proofs of all theoretical results? [Yes] The derivation of the joint distribution and gradient calculation is detailed in Section 3.
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] Included in the supplementary
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See the corresponding descriptions in all the experiments in Section 4.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] For all the experimental results, we have shown the error bars with standard deviation values.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] All the experiments are implemented on a PC as described in Section 4.1.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] In Section 4, we have explicitly stated and cited the corresponding programming environment and competing algorithms for performance comparison.
   (b) Did you mention the license of the assets? [No] The licenses of the assets can be viewed in their respective documentation.

(c) Did you include any new assets either in the supplemental material or as a URL? [Yes] We have included our code in the supplementary.

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]

5. If you used crowdsourcing or conducted research with human subjects...

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]