The Australian National University Semester 1, 2022
School of Computing Assignment 2
Convenor: Lexing Xie Weight: 20%
Other Contributors: Alexander Soen, Chamin Hewa Koneputugodage, Evan Markou

COMP4670/8600: Statistical Machine Learning

Release Date. 22nd April 2022.
Due Date. 16th May 2022 at 1200 AEDT.
Maximum credit. 100 Marks for COMP4670 and COMP8600.

For submission, we are using Gradescope. Make sure that the files included in the submission follow the following format:

1. Ensure that if submitting a zip file (or similar), the file is uncompressed on Gradescope. Individual files should be visible in Gradescope's submission website.
2. `contribution.py` is submitted with your contribution / reference / collaboration statement filled in. State your group members (if any) and state any collaboration has in completing the assignment.
3. A **PDF for theory solutions**. We will not be accepting any other file format. This may result in a null grade or a heavy mark penalty. When referring to equations in the spec, please use the equation number, i.e., (2.1). For Section 2, you should include the answers of Q2.3 and Q2.6b inside your theory PDF file.
4. Submit the coding folder `boframework` that should **only** include the following files with your code implementation where appropriate (`FIXME` sections):
   - `kernels.py`
   - `gp.py`
   - `acquisitions.py`
   - `bayesopt.py`

   Do not delete any setup code provided or change the functions you are required to complete.
5. Submit the file `bayesopt_implementation_viewer.ipynb`. Note that you should submit your executed version of the file, and not the default one. Similarly, do not delete or change any of the code inside the jupyter notebook.

Section 1: Generalisation and Data Noise (50 Total Points)

In the lectures, we explored several aspects of the generalisation bound. One simple variant we discussed is that of a finite hypothesis class, where it was shown that the generalisation gap can upper-bounded by a quantity scaling with $\mathcal{O}(1/\sqrt{N})$ many samples. However, this requires the independent samples to follow the true distribution. Inspired by the discussions and questions given by students in that lecture, we will consider a very simple method[1] of adapting the uniform bound for a finite hypothesis class in the presence of noise.

Setup. Let us set up the scene. Suppose we are interested in the classification of inputs $\mathcal{X}$ to classes $\mathcal{Y}$, with corresponding distribution $\mathcal{D}$ and corresponding joint probability density function p. We also consider the random variables of input and output classes given by the distribution $(X, Y) \sim \mathcal{D}$.

Define a hypothesis class $\mathcal{H}$ as a set of classifiers of the form $f : \mathcal{X} \to \mathcal{Y}$. The objective of learning is to find the best possible $f \in \mathcal{H}$, where "best" means the classifier that gives us the lowest risk based on some loss function $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$. Recall from the lectures that there are two ways we can aggregate the loss: either using the theoretical distribution $\mathcal{D}$ to get the *true risk*, or using the empirical distribution (through an i.i.d. dataset) to get the *empirical risk*. The following definition will make it clearer.

Definition 1 (Risk[2])**.** The *(true) risk* of a classifier $f : \mathcal{X} \to \mathcal{Y}$ with respect to (w.r.t.) a distribution $\mathcal{D}$ on $\mathcal{X} \times \mathcal{Y}$ and loss function $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ is given by

$$\mathcal{R}_{\mathcal{D}}[f] \stackrel{\text{def}}{=} \mathop{\mathbb{E}}_{(X,Y)\sim\mathcal{D}}[\ell(f(X), Y)]; \tag{1.1}$$

and the *empirical risk* w.r.t. samples $\mathcal{S} \sim \mathcal{D}^N$ is given by

$$\widehat{\mathcal{R}}_{\mathcal{S}}[f] \stackrel{\text{def}}{=} \mathop{\widehat{\mathbb{E}}}_{(X,Y)\sim\mathcal{S}}[\ell(f(X), Y)] = \frac{1}{N}\sum_{i=1}^{N} \ell(f(x_i), y_i). \tag{1.2}$$

Remark. Note the slight difference in notation from lectures. The empirical risk is denoted by a hat $\widehat{\mathcal{R}}_{\square}[f]$ and each type of risk has an explicit subscript to denote what distribution we are considering. This is so we can compare the true risk and empirical risk for different distributions.

From the lecture, we know that the the difference between these two types of risk — also called the *generalisation gap* — is important. It determines how close we can get to the true risk given only a finite amount of data. We also know in the lecture that this gap can be bounded no matter what classifier f we choose, using the two-sided Hoeffding's inequality (see Proposition 1 in Appendix A). You may use this result without proof later.

Assumption: For the rest of this section, we will make two assumptions. First, the hypothesis class $\mathcal{H}$ is finite and contains $|\mathcal{H}|$ classifiers. Second, the loss function ℓ is bounded between 0 and 1 on its domain—that is, for any $y, y' \in \mathcal{Y}$,

$$0 \leq \ell(y, y') \leq 1. \tag{1.3}$$

General noise. To consider the effects *data noise* has on our generalisation bounds, we will consider two different distributions: (1) a clean distribution which we want to learn from; and (2) a noisy distribution which can cause problems in learning. For example, we consider a noisy distribution $\widetilde{\mathcal{D}}$ as a noisy version of $\mathcal{D}$. The "tilde" $\widetilde{\square}$ symbol is used to indicate noisy variants of object we have already introduced. Table 1 in Appendix A summarises the differences.

In this general setup, we have a natural question about the effect of noise: how close can we get to the true risk if we only have a noisy sample of data? Or mathematically, can the generalisation gap when we have noisy data, defined as

$$\Delta_{\text{noise\_gap}}[f] \stackrel{\text{def}}{=} |\mathcal{R}_{\mathcal{D}}[f] - \widehat{\mathcal{R}}_{\widetilde{\mathcal{S}}}[f]|, \tag{1.4}$$

be bounded? To answer this, let us first examine the difference between the true risk of clean distribution and that of the noisy version.

[1] There are more slick bounds available if we use more sophisticated frameworks, *i.e.*, PAC-Bayes.

[2] Technically, the co-domain of the classifier does not need to match that of the output space. However, we simplify to this case for convenience of notation.

Question 1.1: Noise and True Risk (10 Points)

Let $f \in \mathcal{H}$ with a bounded loss function (as per Eq. (1.3)). Prove that

$$|\mathcal{R}_{\mathcal{D}}[f] - \mathcal{R}_{\widetilde{\mathcal{D}}}[f]| \leq D_{TV}(p, \widetilde{p}), \tag{1.5}$$

where $p, \widetilde{p}$ are the density functions of $\mathcal{D}, \widetilde{\mathcal{D}}$, respectively; and

$$D_{\mathrm{TV}}(p, \widetilde{p}) \stackrel{\text{def}}{=} \iint |p(x, y) - \widetilde{p}(x, y)| \, \mathrm{d}x \, \mathrm{d}y. \tag{1.6}$$

The integral term in Eq. (1.6) is actually a well known distance function called the *total variation distance*. One nice characteristic of this distance is that it is (polynomially) upper-bounded by the KL-divergence[3]

A hint for later: The integral in Eq. (1.6) can be simplified to a "summation" if we are considering a countable domain, *i.e.*, a classification task with finite $\mathcal{Y}$.

Thus putting everything together, let us establish a *uniform bound* for the noisy sample gap (given by Eq. (1.4)) for a finite hypothesis class $\mathcal{H}$.

Question 1.2: Uniform Bound for Noisy Sample Gap (15 Points)

Let $\mathcal{H}$ be a finite hypothesis class. Prove that given a finite sample $\mathcal{S} \sim \mathcal{D}^N$, for $\delta > 0$

$$\Pr\left\{\forall f \in \mathcal{H} : |\mathcal{R}_{\mathcal{D}}[f] - \widehat{\mathcal{R}}_{\widetilde{\mathcal{S}}}[f]| \leq D_{\mathrm{TV}}(p, \widetilde{p}) + \sqrt{\frac{\ln|\mathcal{H}| + \ln(1/\delta)}{N}}\right\} \geq 1 - \delta. \tag{1.7}$$

Hint: Use the triangle inequality. Then reuse the result in Question 1.1 to find an upper bound for the left-hand distance above. Finally use Proposition 1 to derive the bound and change ϵ to δ to get Eq. (1.7).

Question 1.3: Interpreting the Uniform Bound for Noisy Sample Gap (10 Points)

Compare Eq. (1.7) with the bound we found in the lecture in the absence of noisy data, namely, for any $\delta > 0$,

$$\Pr\left\{\forall f \in \mathcal{H} : |\mathcal{R}_{\mathcal{D}}[f] - \widehat{\mathcal{R}}_{\mathcal{S}}[f]| \leq \sqrt{\frac{\ln|\mathcal{H}| + \ln(1/\delta)}{N}}\right\} \geq 1 - \delta.$$

What is the effect of noisy data on the generalisation bound and the rate at which it decreases with the number of data points?

Label noise. Now that we have a "general" bound for different noises, let us specialise the bound for label noise. In particular, for the next couple questions we will consider label noise for binary classification, where $\mathcal{Y} = \{-1, +1\}$. Simply, label noise can be summarised the case where class labels used to train have a chance of being incorrect.

Definition 2 ((Asymmetric) Label Noise)**.** Let $(X, Y) \sim \mathcal{D}$ be a data distribution. Given label flipping probabilities $\sigma_{-1}, \sigma_{+1} \in [0, 1]$, the corresponding noisy distribution under *(asymmetric) label noise* is the distribution $(\widetilde{X}, \widetilde{Y}) \sim \widetilde{\mathcal{D}}$ in which noisy random variables $\widetilde{X}, \widetilde{Y}$ satisfy the following properties:

(**LN1**)**:** Label flipping probability $\Pr(\widetilde{Y} = -y \mid Y = y) = \sigma_y$ for all $y \in \mathcal{Y}$;

(**LN2**)**:** Identical input marginals $\Pr(\widetilde{X} = x) = \Pr(X = x)$ for all $x \in \mathcal{X}$;

(**LN3**)**:** Conditional independence $\widetilde{Y} \perp\!\!\!\perp X \mid Y$.

[3]Pinsker's Inequality gives for every probability distribution p, q on $\mathcal{X} \times \mathcal{Y}$: $\mathcal{D}_{\mathrm{TV}}(p, q) \leq \sqrt{\frac{1}{2} D_{\mathrm{KL}}(p, q)}$.

Symmetric label noise is the noise model when $\sigma = \sigma_{-1} = \sigma_{+1}$.

Given the definition of asymmetric label noise, we can calculate certain posterior probability quantities which are more intuitive. For instance, one can summarise Definition 2 by considering the noise posterior $\Pr(\widetilde{Y} \mid X)$.

Question 1.4: Noisy Posterior for Label Noise (5 Points)

Given a clean distribution $\mathcal{D}$ and a noisy version of the distribution $\widetilde{\mathcal{D}}$ under asymmetric label noise with flipping probabilities $\sigma_{-1}, \sigma_{+1} \in [0, 1]$, show that for all $y \in \mathcal{Y}$

$$\Pr(\widetilde{Y} = y \mid X) = \sigma_{-y} \cdot (1 - \Pr(Y = y \mid X)) + (1 - \sigma_y) \cdot \Pr(Y = y \mid X). \tag{1.8}$$

Using Eq. (1.8) we can simply adapt the uniform bound given in the general case for label noise. In particular, we will look at the symmetric label noise version of the bound.

Question 1.5: Uniform Bound for Symmetric Label Noise (10 Points)

Suppose that a noisy distribution $\widetilde{\mathcal{D}}$ is given by symmetric label noise, with flipping $\sigma \in [0, 1]$, applied to clean distribution $\mathcal{D}$. Prove that given a finite sample $\mathcal{S} \sim \mathcal{D}^N$, for $\delta > 0$

$$\Pr\left\{\forall f \in \mathcal{H} : |\mathcal{R}_{\mathcal{D}}[f] - \widehat{\mathcal{R}}_{\widetilde{\mathcal{S}}}[f]| \leq 2\sigma + \sqrt{\frac{\ln|\mathcal{H}| + \ln(1/\delta)}{N}}\right\} \geq 1 - \delta. \tag{1.9}$$

Hint: Start by bounding $|\mathcal{R}_{\mathcal{D}}[f] - \mathcal{R}_{\widetilde{\mathcal{D}}}[f]|$ like in Question 1.1. Then reuse the technique in Question 1.2.

Remark. In Eq. (1.9), we can see that the noise term σ is linear in σ: Thus the noise is maximally harmful with $\sigma = 1$ and vanishes when $\sigma = 0$.

Section 2: Gaussian Processes and Bayesian Optimisation (50 Total Points)

Gaussian Processes. For reference, we have defined the key concepts and notations of a Gaussian Process in Appendix B.

Gaussian Process Regression. Having defined the prior and the joint distributions (check Appendix B), we are now in position to define the Gaussian Process predictive distribution as,

$$\mathbf{f}_* \mid X, \mathbf{y}, X_* \sim \mathcal{N}(\bar{\mathbf{f}}_*, \operatorname{cov}(\mathbf{f}_*)) \tag{2.1}$$

where we have,

$$\bar{\mathbf{f}}_* \stackrel{\text{def}}{=} \mathbb{E}[\mathbf{f}_* \mid X, \mathbf{y}, X_*] = K_*^\top K_{\mathbf{y}}^{-1} \mathbf{y} \tag{2.2a}$$

$$\operatorname{cov}(\mathbf{f}_*) = K_{**} - K_*^\top K_{\mathbf{y}}^{-1} K_*. \tag{2.2b}$$

If we have only a single point to predict, the above equations simplify to,

$$\bar{f}_* = \mathbf{k}_*^\top K_{\mathbf{y}}^{-1} \mathbf{y} \tag{2.3a}$$

$$\mathbb{V}(f_*) = k_{**} - \mathbf{k}_*^\top K_{\mathbf{y}}^{-1} \mathbf{k}_*. \tag{2.3b}$$

Depending on our choice of the covariance function, we would have to choose its parameters as well. This choice must be diligent, otherwise we will have trouble predicting new points with high confidence. It would be ideal to have a field expert for every exact problem we are trying to solve to inform us of the underlying parameters of the model, but this isn't always possible. In those cases, we have to treat those covariance function parameters as *hyperparameters*, which we denote as $\boldsymbol{\theta}$, and optimise them in our algorithm. There are many ways to achieve this hyperparameter optimisation, but here we will maximise the log marginal likelihood with respect to those parameters $\boldsymbol{\theta}$, with the log marginal likelihood defined as,

$$\log p(\mathbf{y} \mid \mathbf{X}, \boldsymbol{\theta}) = -\frac{1}{2} \mathbf{y}^\top K_{\mathbf{y}}^{-1} \mathbf{y} - \frac{1}{2} \log |K_{\mathbf{y}}| - \frac{n}{2} \log 2\pi. \tag{2.4}$$

Now, as you have noticed we are required many times to compute the inverse of the training covariance matrix, $K_{\mathbf{y}}^{-1}$. Taking the inverse straight away would be a highly inefficient and numerically unstable way, especially as the training dataset grows. So, instead we should use a more appropriate method by utilising techniques from matrix factorisation theory. More specifically, we will perform Cholesky factorisation on the training covariance matrix. An analytical implementation of the steps to implement this decomposition is given in Algorithm 1. Note the backslash notation $A \backslash \mathbf{b}$ is the vector $\mathbf{x}$ which solves $A\mathbf{x} = \mathbf{b}$.

Algorithm 1 Efficient Gaussian Process Regression

1: $L \leftarrow \text{cholesky}(K_{\mathbf{y}})$ ▷ Cholesky factorisation for predictions and log marginal likelihood
2: $\boldsymbol{\alpha} \leftarrow L^\top \backslash (L \backslash \mathbf{y})$
3: $\bar{f}_* \leftarrow \mathbf{k}_*^\top \boldsymbol{\alpha}$ ▷ predictive mean Eq. 2.3
4: $\mathbf{v} \leftarrow L \backslash \mathbf{k}_*$
5: $\mathbb{V}(f_*) \leftarrow k_{**} - \mathbf{v}^\top \mathbf{v}$ ▷ predictive variance Eq. 2.3
6: $\log p(\mathbf{y}|\mathbf{X}) \leftarrow -\frac{1}{2} \mathbf{y}^\top \boldsymbol{\alpha} - \sum_i \log L_{ii} - \frac{n}{2} \log 2\pi$ ▷ log marginal likelihood Eq. 2.4

Choice of Covariance. As we discussed before, the choice of covariance plays an important factor in the success of a GP regression. For this assignment, we will consider a general class of Covariance functions, called *Matérn*. This class is defined mathematically as,

$$k_{\text{Matern}}(x, y) = \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\frac{\sqrt{2\nu} d}{\ell} \right)^\nu K_\nu \left(\frac{\sqrt{2\nu} d}{\ell} \right) \tag{2.5}$$

with positive parameters ν and ℓ. K_ν is a modified Bessel function and $d = \text{dist}(x, y)$ denotes the pairwise Euclidean distance.

We can easily observe that when $\nu \to \infty$, we obtain the well-known Radial Basis function (RBF). Therefore, this class of covariance functions is a generalisation of the RBF class. However, RBF can be

too smooth sometimes and this is not always appropriate in a machine learning setting. For machine learning solutions, we usually tend to use two other covariance functions. Matérn32 and Matérn52, when $\nu = 3/2$ and $\nu = 5/2$ respectfully. Matérn32 covariance function works for single-differentiable functions, whereas Matérn52 could work for up to twice-differentiable functions. In general, a Gaussian process with Matérn covariance is $\lceil \nu \rceil - 1$ times differentiable in the mean-square sense.

The two covariance functions are defined below as,

$$k_{\nu=3/2}(x, y) = \sigma_f^2 \cdot \left(1 + \frac{\sqrt{3}d}{\ell}\right) \exp\left(-\frac{\sqrt{3}d}{\ell}\right) \quad \rightarrow \text{In Matérn32 case} \tag{2.6a}$$

$$k_{\nu=5/2}(x, y) = \sigma_f^2 \cdot \left(1 + \frac{\sqrt{5}d}{\ell} + \frac{5d^2}{3\ell^2}\right) \exp\left(-\frac{\sqrt{5}d}{\ell}\right) \quad \rightarrow \text{In Matérn52 case} \tag{2.6b}$$

with σ_f^2 being the signal variance.

Question 2.1: Matérn Class Covariance Functions (5 Points)

Implement the Matérn32 and Matérn52 covariance functions (`__call__()`) under the `FIXME` sections in `kernels.py`.

You should use the Equations 2.6a and 2.6b for your implementation.

Question 2.2: Gaussian Process Regression (15 Points)

Implement the `optimisation()`, `fit()`, `predict()`, and `log_marginal_likelihood()` functions under the respective `FIXME` sections in `gp.py`.

You should use Algorithm 1, where appropriate for your implementation.

Bayesian Optimisation - Intuition. Bayesian Optimisation (BayesOpt) is a zeroth-order optimisation method of finding the maximum of expensive cost functions. BayesOpt employs Bayesian techniques of setting a prior over the objective function and combining it with evidence to get a posterior function. This permits a utility-based selection of the next observation to make on the objective function, which must take into account both exploration (sampling from areas of high uncertainty) and exploitation (sampling areas likely to offer improvement over the current best observation). More specifically, it builds a surrogate model for the objective and quantifies the uncertainty in that surrogate using Gaussian process regression, and then uses an acquisition function defined from this surrogate to decide where to sample. We will define all those terms later.

Let's assume we want to solve the following optimisation problem

$$\max_{x \in \mathcal{X}} f(x). \tag{2.7}$$

The function defined in 2.7 also has the following important properties:

- The objective function f is *L-Lipschitz-continuous* in order to use Gaussian process regression.
- f is expensive to evaluate, either in the sense of time or money or other factors that will result in hurdles along the way.
- f lacks known special structure, such as convexity or quasi-convexity (similarly with concavity) to allow us to use known gradient-based optimisation algorithms. We say that f is a black-box in those cases.

Finally, our focus in on finding a global rather than local optimum.

Surrogate Model. As we said before, BayesOpt consists of two main components: a Bayesian statistical model for modelling the objective function, and an acquisition function for deciding where to sample next.

For the surrogate model we will use the Gaussian process regression model that we have implemented before. Since our evaluation functions will be continuous, a GPR is an ideal surrogate model. In general, you can read about surrogate models here.

Improvement-based Acquisition Functions. Now, let's turn over focus on the acquisition functions. The role of the acquisition function is to guide the search for the optimum. Typically, acquisition functions are defined such that high acquisition corresponds to potentially high values of the objective function, whether because the prediction is high, the uncertainty is great, or both. Maximising the acquisition function is used to select the next point at which to evaluate the function. That is, we wish to sample f at $\arg\max_x u(x \mid D)$, where $u(\cdot)$ is the generic symbol for an acquisition function.

There are many types of acquisition functions, but in this assignment we will focus only on improvement-based acquisition functions. This class of functions is based on the random variable *Improvement*:

$$\mathrm{I}(\mathbf{x}) = \max\{0, f(\mathbf{x}) - f(\mathbf{x}^+) - \xi\}, \tag{2.8}$$

where $f(\mathbf{x}^+)$ is the incumbent solution and $\mathbf{x}^+ = \arg\max_{\mathbf{x}_i \in \mathbf{x}_{1:t}} f(\mathbf{x}_i)$. Note that $\xi \geq 0$ is the exploration-exploitation trade-off *hyperparameter*.

Intuitively, $\mathrm{I}(\mathbf{x})$ assigns a reward of $\{f(\mathbf{x}) - f(\mathbf{x}^+) - \xi\}$ if $f(\mathbf{x}) > f(\mathbf{x}^+) + \xi$, and zero otherwise. Therefore, we should select the point that has the highest probability of improving upon the incumbent function value $f(\mathbf{x}^+)$. This effectively means that we should maximise the probability of the event $\{\Pr(\mathrm{I}(\mathbf{x}) > 0)\}$ or equivalently maximise the probability of the event $\{f(\mathbf{x}) > f(\mathbf{x}^+) + \xi\}$. Another way of seeing this, is by taking the expectation of the utility function $\mathrm{u}(\mathbf{x}) = 1_{\{f(\mathbf{x}) > f(\mathbf{x}^+) + \xi\}}$, which is $\mathrm{u}(\mathbf{x}) = 1$ when $f(\mathbf{x}) > f(\mathbf{x}^+) + \xi$ and zero otherwise.

A well-known acquisition function is the maximisation of this *probability of improvement* (PI) over the $f(\mathbf{x}^+)$, and is defined as such,

$$\begin{aligned} \mathrm{PI}(\mathbf{x}) &= \Pr(\mathrm{I}(\mathbf{x}) > 0) \\ &= 1 - \Pr\left(\frac{f(\mathbf{x}) - \mu(\mathbf{x})}{\sigma(\mathbf{x})} \leq \frac{f(\mathbf{x}^+) + \xi - \mu(\mathbf{x})}{\sigma(\mathbf{x})}\right) \Rightarrow \text{reparameterisation trick } u \\ &= \Phi(Z), \end{aligned} \tag{2.9}$$

where

$$Z = \frac{\mu(\mathbf{x}) - f(\mathbf{x}^+) - \xi}{\sigma(\mathbf{x})}. \tag{2.10}$$

The full derivation can be found in Appendix B.

Also, $\Phi(\cdot)$ and $\phi(\cdot)$ is the normal cumulative and probability distribution functions respectively, $\mu(\mathbf{x})$ is the mean of $f(\mathbf{x})$, $\sigma(\mathbf{x})$ is the standard deviation of $f(\mathbf{x})$, and ξ is the exploration-exploitation trade-off *hyperparameter*, as before.

However, the solution of the PI acquisition function is *greedy* as it selects the point most likely to offer an improvement of at least ξ.

A somewhat more satisfying alternative acquisition function would be one that takes into account not only the probability of improvement, but also the magnitude of the improvement a point can potentially yield. In particular, we want to minimise the expected deviation from the true maximum $f(\mathbf{x}^\star)$, when choosing a new trial point. This can be formulated mathematically as,

$$\begin{aligned} \mathbf{x}_{t+1} &= \arg\min_{\mathbf{x}} \mathbb{E}[\|f_{t+1}(\mathbf{x}) - f(\mathbf{x}^\star)\| \mid \mathcal{D}_{1:t}] \\ &= \arg\min_{\mathbf{x}} \int \|f_{t+1}(\mathbf{x}) - f(\mathbf{x}^\star)\| \Pr(f_{t+1} \mid \mathcal{D}_{1:t}) \, \mathrm{d}f_{t+1} \end{aligned} \tag{2.11}$$

In the above definition, we have consider only one step $t+1$, and if we want to account for many steps, we just have to use a recursive procedure with dynamic programming to find the solution. However, this is expensive, and so we shall propose an alternative procedure. That is to maximise this expected improvement with respect to $f(\mathbf{x}^+)$. More specifically, we can re-introduce the *improvement* (or window function in stochastic modelling in engineering) from the PI acquisition function.

The new query point is found by maximizing the expected improvement:

$$\mathbf{x} = \arg\max_{\mathbf{x}} \mathbb{E}[\mathrm{I}(\mathbf{x}) \mid \mathcal{D}_t]. \tag{2.12}$$

Thus, taking the expectation of the random variable $\mathrm{I}(\mathbf{x})$ yields the *expected improvement* (EI) acquisition function, defined as follows,

$$\mathrm{EI}(\mathbf{x}) = \begin{cases} (\mu(\mathbf{x}) - f(\mathbf{x}^+) - \xi)\Phi(Z) + \sigma(\mathbf{x})\phi(Z) & \text{if } \sigma(\mathbf{x}) > 0, \\ 0 & \text{if } \sigma(\mathbf{x}) = 0. \end{cases} \tag{2.13}$$

where

$$Z = \begin{cases} \frac{\mu(\mathbf{x}) - f(\mathbf{x}^+) - \xi}{\sigma(\mathbf{x})} & \text{if } \sigma(\mathbf{x}) > 0, \\ 0 & \text{if } \sigma(\mathbf{x}) = 0. \end{cases} \tag{2.14}$$

Question 2.3: Expected Improvement (6 Points)

Derive mathematically the result of the Expected Improvement (EI) acquisition function in Equation 2.13. That is show that,

$$\mathbb{E}\left[\mathrm{I}(\mathbf{x})\right] = \begin{cases} (\mu(\mathbf{x}) - f(\mathbf{x}^+) - \xi)\Phi(Z) + \sigma(\mathbf{x})\phi(Z) & \text{if } \sigma(\mathbf{x}) > 0, \\ 0 & \text{if } \sigma(\mathbf{x}) = 0. \end{cases}$$

where Z is defined in Equation 2.14.

Bayesian Optimisation. To summarise the BayesOpt algorithm can be described in Algorithm 2.

Algorithm 2 Bayesian Optimisation

1: **for** t=1,2,... **do**
2: Choose $\mathbf{x}_t$ by optimising the acquisition function u over the Gaussian Process such that:
3: $\mathbf{x}_t = \arg\max_{\mathbf{x}} u(\mathbf{x} \mid \mathcal{D}_{1:t-1})$. ▷ *
4: Sample the objective function: $y_t = f(\mathbf{x}_t) + \epsilon$.
5: Augment the data $\mathcal{D}_{1:t} = \mathcal{D}_{1:t-1} \cup \{(\mathbf{x}_t, y_t)\}$ and update the GP.
6: **end for**

***:** Step 3 of Algorithm 2 has been implemented for you in the `sample_next_point()` function which can be found in `bayesopt.py`.

Question 2.4: Acquisition Function (5 Points)

Implement the Probability of Improvement (PI) and Expected Improvement (EI) acquisitions functions under the respective `FIXME` sections in `acquisitions.py`. You should use the Equations 2.9 (the final result) and 2.13 for your implementation.

Note: For the coding questions Q2.1, Q2.2, and Q2.4 we have provided a testing suite for you (`tests.py`) to validate your implementation before running the whole algorithm in Q2.5. However, these tests are not complete and passing them won't necessary mean that you will receive full marks for these questions. They are here for your guidance and debugging purposes. To test the code, run the following command:

```
$ python -m unittest tests
```

Question 2.5: Bayesian Optimisation (15 Points)

Implement the Bayesian Optimisation algorithm function (`__call__()`) under the respective `FIXME` section in `bayesopt.py`. You should use Algorithm 2 for your implementation.

Note: The default choices for the exploitation-exploration trade-off hyperparameter ξ, for both acquisition functions, is set to 0.01 by default. However, given the choice of the initial points, you will see that the algorithm doesn't converge. If fact, since the function is multi-modal it will require a scheduling of this parameter ξ to achieve the global maximum in 10 iterations. A simple scheduler would be at first start with a higher ξ to account for more exploration and as the iterations proceed, decay the parameter for more exploitation. However this is a suggestion and feel free to come up with a different scheduling strategy.

As long as you can justify your strategy and the algorithm converges, you will receive full marks. To check for convergence, run the `bayesopt_implementation_viewer.ipynb` for the single dimensional case.

Question 2.6: Intuition on Higher Dimensions (4 Points)

a) Implement the `fit_and_predict()` function under the `FIXME` section in `gp.py`. (2 Points)

b) After you run the code on `bayesopt_implementation_viewer.ipynb` for the higher dimensional case, comment on the benefit of including a hyperparameter (meaning the parameters of the covariance function $\boldsymbol{\theta}$ of the surrogate model) optimisation approach on your Bayesian Optimisation algorithm. (2 Points)

Note: For the coding questions, feel free to use the packages that are already imported in your solutions but do not import any other package. There will be a penalty if you import extra packages.

A Q1: Generalisation Appendix

The following appendix section states the two-sided Hoeffding's inequality and summaries the notation used in Question 1.

We recommend using the following version of Hoeffding's inequality for this assignment.

Proposition 1 (Two-sided Hoeffding's inequality)**.** Let $Z_1, \ldots, Z_N$ be i.i.d. random variables and take values in $[0, 1]$. Let the true mean be μ_Z and define the sample mean as $\hat{\mu} \stackrel{\text{def}}{=} \frac{1}{N}\sum_{i=1}^{N} Z_i$. Then, for any $\epsilon > 0$,

$$\Pr\left\{|\hat{\mu} - \mu_Z| \geq \epsilon\right\} \leq 2\exp(-2N\epsilon^2). \tag{A.1}$$

The notation can be summarised by the following:

	Clean	Noisy
Distribution	$\mathcal{D}$	$\widetilde{\mathcal{D}}$
Density	p	$\widetilde{p}$
Samples	$\mathcal{S}$	$\widetilde{\mathcal{S}}$
Data point	(x, y)	$(\tilde{x}, \tilde{y})$
Random Variable	(X, Y)	$(\widetilde{X}, \widetilde{Y})$

Table 1: Summary of notation for noisy objects.

B Q2: Gaussian Process Appendix

Definition 3. *A Gaussian process (GP) is a collection of random variables, any finite number of which have a joint Gaussian distribution.*

Setup. We specify a GP by its mean function and covariance function as such:

$$m(\mathbf{x}) = \mathbb{E}[f(\mathbf{x})], \tag{B.1a}$$

$$k(\mathbf{x}, \mathbf{x}') = \mathbb{E}[(f(\mathbf{x}) - m(\mathbf{x}))(f(\mathbf{x}') - m(\mathbf{x}'))] \tag{B.1b}$$

and mathematically define a Gaussian process as,

$$f(\mathbf{x}) \sim \mathcal{GP}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}')) \tag{B.2}$$

In this assignment, we will perform Gaussian Process Regression (GPR) using noisy observations to account for more realistic modelling situations. Thus, our evaluation functions will have the form of

$$\mathbf{y} = f(\mathbf{x}) + \epsilon \tag{B.3}$$

where ϵ is some random noise.

Without loss of generality, we can assume the prior mean is zero (we can always add the mean vector later), and incorporating the noise into the covariance function, we can define the prior with respect to the covariance (using matrix notation) as,

$$K_{\mathbf{y}} = K(X, X) + \sigma_n^2 I \tag{B.4}$$

where σ_n^2 is the variance of the Gaussian noise, assuming it's derived from *i.i.d.* data.

In addition, for the noise training outputs $\mathbf{y}$, and the testing outputs $\mathbf{f}_*$, we can define their joint distribution, by carefully partitioning the covariance matrix. This will be produce the following result,

$$\begin{bmatrix} \mathbf{y} \\ \mathbf{f}_* \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} K_{\mathbf{y}} & K_* \\ K_*^\top & K_{**} \end{bmatrix}\right) \tag{B.5}$$

where for simplicity, we have defined each block component of the covariance matrix as,

$$
\begin{aligned}
K_{\mathbf{y}} &: \text{defined as the noisy training covariance matrix in Eq. B.4} \\
K_{*} &: \text{defined as the training-testing covariance matrix, i.e., } K(X, X_{*}) \\
K_{*}^{\top} &: \text{defined as the testing-training covariance matrix, i.e., } K(X_{*}, X) \\
K_{**} &: \text{defined as the testing covariance matrix, i.e., } K(X_{*}, X_{*})
\end{aligned}
$$

Derivation of Eq. (2.9)

$$
\begin{aligned}
\mathrm{PI}(\mathbf{x}) &= \Pr\left(\mathrm{I}(\mathbf{x}) > 0\right) \\
&= \mathbb{E}[1_{\{f(\mathbf{x}) > f(\mathbf{x}^{+}) + \xi\}}] \\
&= \Pr\left(f(\mathbf{x}) > f(\mathbf{x}^{+}) + \xi\right) \\
&= \Pr\left(\frac{f(\mathbf{x}) - \mu(\mathbf{x})}{\sigma(\mathbf{x})} > \frac{f(\mathbf{x}^{+}) + \xi - \mu(\mathbf{x})}{\sigma(\mathbf{x})}\right) \\
&= 1 - \Pr\left(\frac{f(\mathbf{x}) - \mu(\mathbf{x})}{\sigma(\mathbf{x})} \leq \frac{f(\mathbf{x}^{+}) + \xi - \mu(\mathbf{x})}{\sigma(\mathbf{x})}\right) \Rightarrow \text{reparameterisation trick } u \\
&= 1 - \int_{-\infty}^{-Z} \phi(u) du \\
&= 1 - \Phi(-Z) \\
&= \Phi(Z),
\end{aligned}
\tag{B.6}
$$

where

$$
Z = \frac{\mu(\mathbf{x}) - f(\mathbf{x}^{+}) - \xi}{\sigma(\mathbf{x})}. \tag{B.7}
$$