Transferable MOGP for drug response prediction

A dissertation submitted to The University of Manchester for the degree of
Master of Science in Computer Science: Artificial Intelligence
in the Faculty of Science and Engineering

Year of submission
2024

Student ID
11404073

Department of Computer Science

Contents

Word count: 12,219

Abstract

Drug response prediction using machine learning approaches allows for precise control of drug dosage, helping to minimize the side effects of chemotherapy. Multi-output Gaussian Process(MOGP), as one of the models suitable for this task, features prediction(mean) with degree of uncertainty(variance). Existing research has pointed out some difficulty of overfitting when MOGP faces input data from varied cancer types or drug compounds. Transfer kernel learning aims to boost the model's performance when handling input from a diverse domain and to migrate and reuse domain knowledge from those with a rich number of samples to those without. This research applied transferable kernels to MOGP models and investigated their performance when dealing with different number of input domains. An adapted kernel(K_{MS-AFS}) with an input-weighing technique is proposed for such a task, and experiments have shown its outstanding performance over tested datasets.

Declaration of originality

I hereby confirm that this dissertation is my own original work unless referenced clearly to the contrary, and that no portion of the work referred to in the dissertation has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

Intellectual property statement

i The author of this thesis (including any appendices and/or schedules to this thesis) owns certain copyright or related rights in it (the "Copyright") and s/he has given The University of Manchester certain rights to use such Copyright, including for administrative purposes.

ii Copies of this thesis, either in full or in extracts and whether in hard or electronic copy, may be made only in accordance with the Copyright, Designs and Patents Act 1988 (as amended) and regulations issued under it or, where appropriate, in accordance with licensing agreements which the University has from time to time. This page must form part of any such copies made.

iii The ownership of certain Copyright, patents, designs, trademarks and other intellectual property (the "Intellectual Property") and any reproductions of copyright works in the thesis, for example graphs and tables ("Reproductions"), which may be described in this thesis, may not be owned by the author and may be owned by third parties. Such Intellectual Property and Reproductions cannot and must not be made available for use without the prior written permission of the owner(s) of the relevant Intellectual Property and/or Reproductions.

iv Further information on the conditions under which disclosure, publication and commercialisation of this thesis, the Copyright and any Intellectual Property and/or Reproductions described in it may take place is available in the University IP Policy (see http://documents.manchester.ac.uk/DocuInfo.aspx?DocID=24420), in any relevant Dissertation restriction declarations deposited in the University Library, and The University Library's regulations (see http://www.library.manchester.ac.uk/about/regulations/_files/Library-regulations.pdf).

Acknowledgements

I'd like to express warm gratitude to my supervisor, Dr. Mauricio A Álvarez Lopez, who has been consistently providing crucial advice that has helped me out of several technical challenges during this research. This research couldn't have gone this far without his provisional perspective and academic expertise. I would also like to thank the Computational Shared Facility and the University of Manchester IT Service team who manage it, without whom the computational task wouldn't have been possible. Last but not least, my special thanks go to Dr. Matthew Lyon, who provided essential guidance on configuring the Python environment on CSF at critical moments of this project.

1 Introduction

1.1 Background

Drug response prediction(DRP) is a critical support when making therapy decisions and clinical use of drugs. These data explore changes in important biomarkers after treatment with drugs and draw further conclusions about whether the drug is responsive or not.[1] Traditional acquisition of these data is expensive, due to the cost incurred during drug response tracking, data manifesting and management, need for professional operators and equipment involved during the process.[1] If this approach were to be revolutionized by artificial intelligence, to predict drug response before medical experiments take place in reality, this could significantly reduce the temporal and financial cost to collect these data, and therefore benefit therapeutic decisions and the development of new drugs.[2]

The detailed method used for drug response prediction varies and could be loosely classified into two categories: machine learning approaches and deep neuron network approaches. Machine learning approaches usually feature quick training, low requirements for the amount of data, and highly interpretable models. Deep neuron network approaches are usually good at excavating deeply hidden patterns in data if given an abundant amount of data.

One of the machine learning approaches is the Gaussian process(GP).In GPs, data are considered to be sampled from a latent distribution, which is assumed to be Gaussian. Prediction using GP involves finding the best likelihood of distribution parameters with given data and conditioning it with the known information for prediction. a model extended from the Gaussian Process(GP). Multi-output GP extends GP by stacking multiple GPs for multiple outputs and define the covariance across them. The linear model of coregionalization(LMC), a further extended kind of MOGP, assumes the latent distribution to be a linear combination of either identical or different GPs. With this assumption, LMC is expected to discover more complex patterns and make more precise predictions. Above that, in contrast to some traditional DRP solutions, where the whole curve is implicitly hinted with predicted statistical features, the MOGP solution is inherently capable of generating a whole drug response curve with its predicted mean and variance(serves as a measure of prediction uncertainty) acting as the curve and upper and lower boundaries.[2]

Existing research that applied the model to the DRP field indicates traditional MOGP setup doesn't propagate its knowledge well from one drug to another.[2] This cutback in performance is considered to be related to the model's insufficient ability to domain knowledge transfer. It is also pointed out that solutions proposed in paper [3] and [4] may become helpful in this regard. These papers propose the idea of introducing domain information to the kernel, such that it reacts differently and produces a covariance adapted to domain information when given data from varied domains.

1.2 Research question and Objective

Based on the background introduced, certain questions arise: If a MOGP model is designed specifically by introducing a transferable kernel for cross-domain training and testing data, will it perform better than those that are not? If so, what kind of transferable kernel can most effectively achieve such a goal? Compared to MOGP without such a special design, how much better can such an approach achieve?

This research seeks transferable kernel adaptions on MOGP to construct a transferable model. and applies them to MOGP. Instead of challenging the state-of-the-art DRP model, this research focuses on whether a quantitative improvement can be achieved and how much can the model benefit by using such an approach.

During this research, several models and kernels and a combination of them are investigated. Inspired by existing kernels and methods, a new kernel is proposed and proved to be effective in transfer kernel learning.

1.3 Report overview

This report consists of 5 chapters. Chapter 1, Introduction, covers the background and the objective of this research. Chapter 2, Literature Review, discusses the GP models and kernels investigated in the research, along with some basics in transferable learning, specifically transferable kernel learning. Chapter 3, Research Methodology explains the technical details of models, kernels, and experiments in this research. Chapter 4, Result and Discussion, gives the experiment result and corresponding analysis. Eventually, chapter 4 concludes the major findings, some known limitations, and possible future work.

Relevant code of this paper can be found at:https://github.com/Paul-A-Lavender/MOGP-Transferable-kernel-l
git

2 Literature Review

2.1 Dataset

Genomics of Drug Sensitivity in Cancer (GDSC) is a resource developed by the Wellcome Trust Sanger Institute and the Massachusetts General Hospital Cancer Center, which aims to understand how genomic features influence drug response in cancer cells.

The dataset provides extensive drug sensitivity data from approximately 75000 experiments, encompassing 138 anti-cancer compounds tested across nearly 700 cancer cell lines. These data are labeled with genomic information, including somatic mutations, gene amplifications, deletions, tissue types, and transcriptional profiles.[1]

GSDC1 and GSDC2 are two different versions provided by GSDC. GSDC1 is collected between 2009 and 2015, and GSDC2 is screened from 2015 until now. There are also discrepancies in the screening of these data, where different methods are used in the part of compound storage, cancer cell seeding, and cell viability measuring. [1]

This research plans to use GSDC1 and GSDC2 datasets, as it is used by many in drug response prediction research.[2], [5], [6]

2.2 Transfer learning

An important condition that is often assumed by common machine learning algorithms, is that there's abundant training data that follows the same distribution as the test data. Yet very often, collecting such training data may end up incurring unbearable financial or temporal costs, sometimes not even feasible. There are semi-supervised learning technique, which utilizes a massive amount of unlabelled data as a complement to a limited number of labelled data. Such idea is practised by research [7]. However, sometimes even unlabelled data becomes scarce and therefore causes unsatisfactory machine learning models.[8][9]

Such circumstances are familiar to drug profile data, as the extraction of such data would require expert personnel and specialized equipment.[1] A question naturally arises: is it possible to improve the models' performance by transfer learning?

In the paper [2], an LMC model with a customized kernel is applied to the GDSC dataset, to study drug efficacy across different dose metrics across two drug screening studies and five cancer types. It was found that the model began to overfit when the study tried to push the model across different cancer types. It is then pointed out transferable kernel learning method in paper [4] and [10] hopefully improve such a situation.

The paper [4] proposed a strategy that combines stacking and individual source-domain similarity for each source-domain pair. The paper [10] proposed a kernel function, k_{MS}, and assigns a learnable parametric coefficient to model the relatedness of each inter-domain pair.

2.3 Gaussian Process

The Gaussian Process is a non-parametric Bayesian method that can be used for classification and regression tasks. It is a probability statistics model describing a set of variables that every subset follows Gaussian distribution. It can also be considered as a distribution over function, in which a prior distribution controls what kind of function is more likely to expect in the model, and the prediction is conducted by conditioning that prior distribution.[11]

A Gaussian process can be uniquely determined by a pair of mean and covariance functions. To train a GP is to maximize the following log-likelihood function[11], w.r.t. hyper-parameters θ of the kernel function

$$logp(\mathrm{y}|\mathrm{X},\theta) = -\frac{1}{2}\mathrm{y}^T \mathrm{K}_y^{-1} y - \frac{1}{2}\log|\mathrm{K}_y| - \frac{n}{2}\log 2\pi$$

where y and X correspond to data features and labels,K_y represents the covariance matrix of y. To maximize the function is to maximize the likelihood that the given data follows the assumed prior distribution, which suggests the predictive model nicely fits the underlying pattern of training data.

After the GP is trained, the mean and covariance function[11] can be given as,

$$\mu = K(\mathrm{X}_*, \mathrm{X}_t)[K(\mathrm{X}_t, \mathrm{X}_t + \sigma_2^{\varepsilon}\mathrm{I})]^{-1}\mathrm{y}_t$$

$$Cov = \mathrm{K}(\mathrm{X}_*, \mathrm{X}_*) - K(\mathrm{X}_*, \mathrm{X}_t)[K(\mathrm{X}_t, \mathrm{X}_t) + \sigma_2^{\varepsilon} I]^{-1} K(\mathrm{X}_t, \mathrm{X}_*)$$

2.4 Muiti-output Gaussian Process

Multi-output Gaussian process is a statistic model that aims to treat multi-output problems. In this model, outputs are expected to be correlated, and MOGP models this kind of correlation by introducing GPs. Generally, MOGP is a method to extend GP to a multi-output field. According to different assumptions of latent function, MOGP can be classified into Intrinsic Coregionalization Model(ICM), Semiparametric Latent Factor Model (SLFM), and Linear Model of Coregionalization(LMC).

2.4.1 Intrinsic Coregionalization Model

The Intrinsic Coregionalization Model(ICM) assumes the latent function is linearly composed of independently sampled GP. The expression of the outputs $\{f_d(x)\}_{d=1}^{D}$(D being the number of outputs) and the covariance matrix can be expressed in the following form:

$$f_d(x) = \sum_{i=1}^{R} a_d^i u^i(x) \tag{1}$$

$$Cov[f(x), f(x')] = AA^T k(x, x') \tag{2}$$

$$= Bk(x, x') \tag{3}$$

where $A = [a^1, a^2, ..., a^R]$ and $\{u^i(x)\}_{q=1}^{Q}$ are the set of Gaussian Processes, which are all independently sampled from $u(x)$. Essentially, ICM assumes the output to be a linear combination of samples from the same Gaussian Process.[12]

2.4.2 Semiparametric Latent Factor Model

The Semiparametric Latent Factor Model, proposed by paper [13], assumes the latent function to be a linear combination with different GPs, the idea of which is alike but different to ICM. SLFM doesn't take into account the same GPs being independently sampled multiple times. For the d-th output among all D outputs $\{f_d(x)\}_{d=1}^{D}$, each output and their covariance can be expressed by:

$$f_d(x) = \sum_{q=1}^{Q} a_{d,q} u_q(x) \tag{4}$$

$$Cov[f_1(x), f(x')] = \sum_{q=1}^{Q} a_q a_Q^T k_q(x, x') = \sum_{q=1}^{Q} B_q k_q(x, x') \tag{5}$$

2.4.3 Linear Model of Coregionalization

The Linear Model of Coregionalization generalizes the ICM and the SLFM by allowing several independent samples from GPs with different covariance.[14] The output $\{f_d(x)\}_{d=1}^{D}$ and the covariance can be given by:

$$f_d(x) = \sum_{q=1}^{Q} \sum_{i=1}^{R_q} a_{d,q}^i u_q^i(x) \tag{6}$$

$$Cov[f(x), f(x')] = \sum_{q=1}^{Q} A_A a_Q^T k_q(x, x') = \sum_{q=1}^{Q} B_q k_q(x, x') \tag{7}$$

where $A_q = [a_q^1, a_q^2, ..., a_q^{R_q}]$, and R_q is the number of repetitions that the q-th GP is being sampled.[15]

Existing research [2] has applied the LMC to the field of drug response prediction. It is also shown this predictive model can be used in combination with KL-divergence sensitivity analysis, which could uncover how much influence each feature could have on the decision-making of the model. This has revealed the importance of certain biomarkers when making drug response predictions. Figure 1[2] shows how a MOGP solution generates a whole curve response, and how KL divergence is used to investigate further the importance of biomarkers in the decision-making of the model. From the picture, it's clear that beyond mean and deviation, the MOGP will provide how the possible curves spread out within the upper and lower bounds, giving more information about the uncertainty of prediction, which can be considered another advantage, compared to conventional solutions.

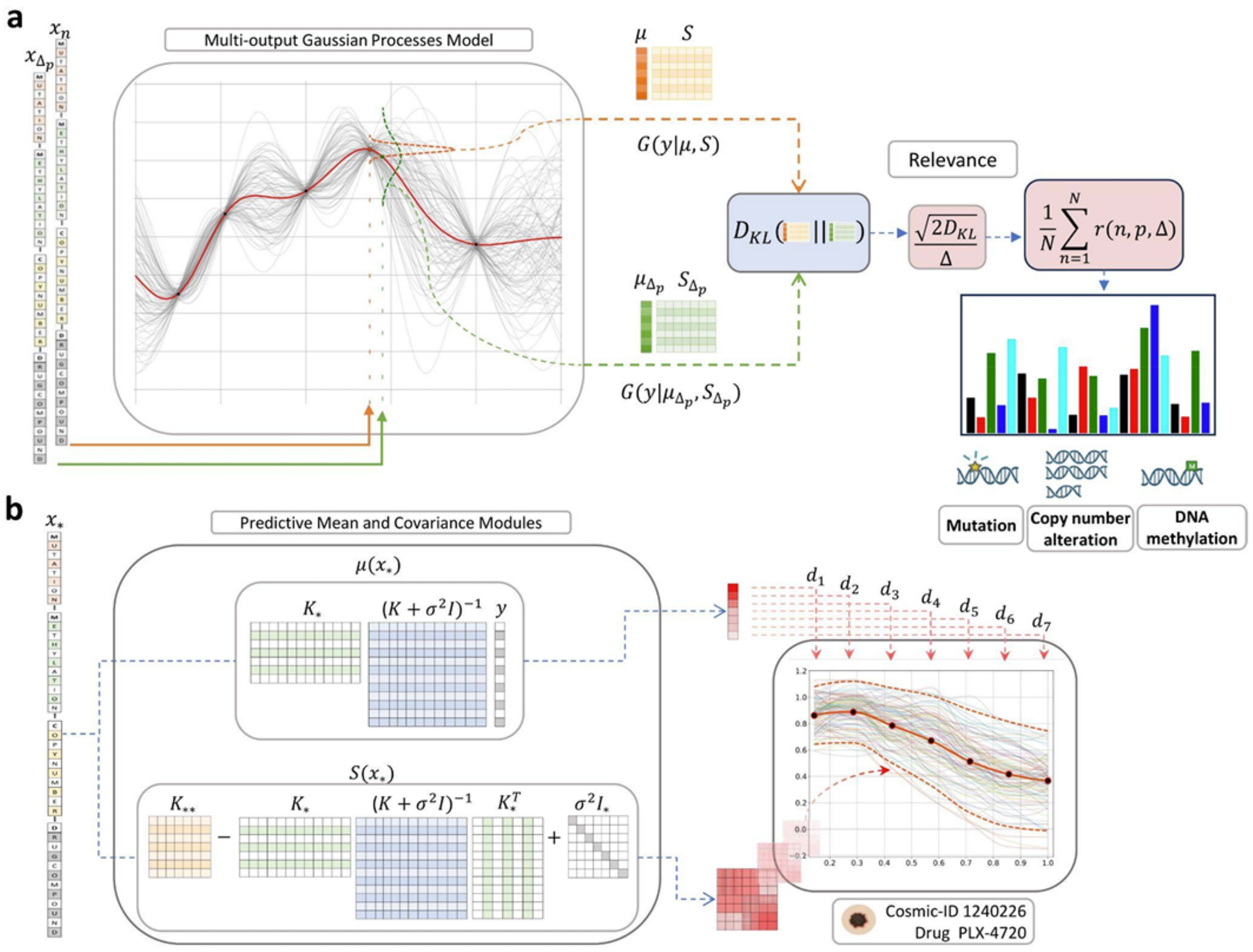
a
b
Multi-output Gaussian Processes Model
Predictive Mean and Covariance Modules
Relevance
Mutation
Copy number alteration
DNA methylation
Cosmic-ID 1240226
Drug PLX-4720
$G(y|\mu, S)$
$G(y|\mu_{\Delta_p}, S_{\Delta_p})$
$\frac{1}{N}\sum_{n=1}^{N} r(n,p,\Delta)$
$\mu(x_*)$
$S(x_*)$

Fig. 1. (a) Kullback-Leibler relevance determination to estimate feature importance; (b) Prediction of full dose-response curves using MOGP

2.5 Kernel Function

The kernel function deals with the problem when the samples in their original space are not linearly separable. The kernel function maps the samples from a lower dimension space to a higher dimension space, in the hope of these samples being linearly separable in the latter.[16]

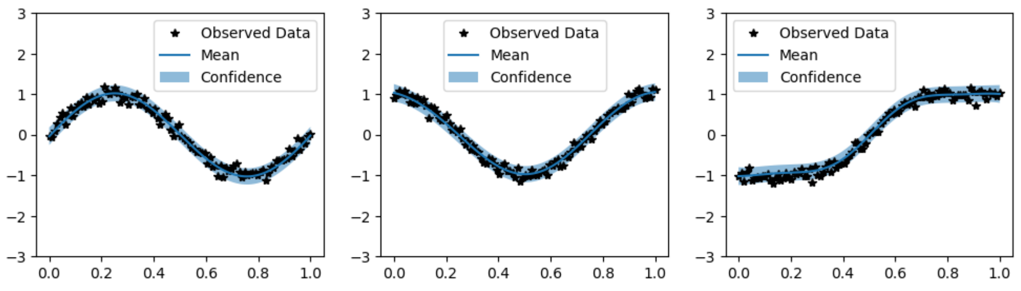

Fig. 2. Transferable model on toy dataset

It can be proven, that such kernel function is valid, if and only if the kernel matrix(or covariance matrix) is positive semi-definite.[17] This suggests that, as long as a function can generate a positive semi-definite kernel matrix, then it is potentially qualified for a kernel function.[18]

Every kernel function implicitly defines a feature space, called Reproducing Kernel Hilbert Space.[17]It's worth noting that the quality of the feature space plays an important role in the final performance of the model. If the original space is mapped into an unsuitable target space, then it is likely that the samples will not be linearly separable in the target space and therefore damage the performance of the model. The kernel function only implicitly defines such a space and hence becomes the largest variant in machine learning models.

To derive a customized kernel, it's important to know that the sum and product of the kernel preserve the P.S.D. property, and therefore qualify the obtained function as the valid kernel.[17] This technique is used in research[2], where a serial product of 4 sub-kernels is constructed as the proposed kernel.

2.6 Transferable kernel

In practice, an established model may aim for a general domain, under which can be divided into several sub-domain. It is often when the model is trained on a specific sub-domain, it may not perform as well when encountering data from another sub-domain. This is considered as a lack of ability to generalize.[10]

The transferable kernel is aimed for such a task. By training with a transferable kernel on a domain that is rich in samples, it is expected to carry its knowledge to another domain that isn't, but delivering similar performance. This allows for a flexible and efficient way of training, by extracting general knowledge and incorporating them into specific domains.[4]

To achieve this, transfer kernel learning propose a method, where each pair of input for kernel function is not treated equally, but in different way that depends on the domain that they are from. [10] A small toy dataset is crafted as demonstrated by figure 2 and 3.

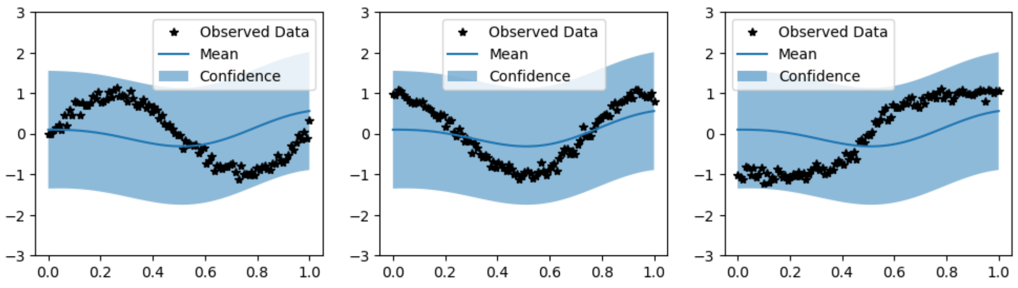

Fig. 3. Non-transferable model on toy dataset

The toy dataset consists of 3 different input patterns, each mapping a uniformly increasing sequence from 0 to 1 to 3 different function outputs, specifically sine, cosine, and sigmoid. The input of these 3 functions is completely identical, except for a single integer digit varies from 0 to 2 representing domain information. The purpose of this task is to determine whether a model has the ability to react differently according to the domain information. The result is clearly depicted in figure 3, that for non-transferable models(using MOGP-RBF, LMC-RBF), they cannot learn the different patterns from the same input(as their structures forbid). On the contrary, transferable models(using combinations between [MOGP, LMC] and [K_{MS}, $K_{\alpha\beta}$]) fit these patterns perfectly, which is shown in figure 3. Note that this task, though designed specifically to challenge models with transfer learning problem, is still extremely easy and performance-saturated for all tested models. (i.e. they all deliver the same perfect performance) In fact, if the domain information is hard-coded into the input, even a MOGP-RBF model can achieve a similar result. This reveals that transferable learning features do not necessarily rely on transferable kernels.

However, this does not undermine the advantage of transferable models, compared to methods that hard-code domain information into input. With the growth of task complexity, it turns out models without transferable kernels could gradually fall behind, presumably because lack of dedicated structure to model variance in domains. This is further shown in section 4.

Transferable kernels rely on some conditions to function. The most important and common part is how the P.S.D. property of the kernel matrix can be ensured.[10] For instance, for kernels proposed in wei2017source, the kernel would degrade to a domain-unspecific one without a transferable learning property if the P.S.D. condition were to be guaranteed. For kernels proposed in [19]–[21], P.S.D. is guaranteed by first decomposing the kernel matrix to low-triangular form and then learning it with an EM algorithm. The downside for this solution may include the counter-intuitive growth of domain relatedness from intra-domain to inter-domain and the lack of uniqueness of such factorization.[10]

3 Research Methodology

In this section, some important milestones and to-dos will be discussed.

3.1 Dataset and Preprocessing

The dataset used in this research inherits from the research [22], where a subset of GDSC1 is provided with some useful processing. This subset consists of several CSV file, which can be outer joined on certain columns and form a collection of 5584 CCLs.(refered to as Davin's subset in the following) It's a small subset from the original GDSC dataset, but is considered sufficient for this research, which is to validate the idea of transferable kernel learning with MOGP, instead of challenging the state-of-the-art models with top-tier performance.

This research further divides this subset by domain features, namely drug compound(essentially different drugs) and cancer types. In this subset, the drug "Shikonin", with a drug_id of 170, contains the most number(274) of CCLs, and is selected as the main origin where finer subsets in this research are derived from. The Shikonin subset contains CCLs from many(30) cancer types. To test the transferable model's performance in different circumstances, subsets of the following are extracted from the mother dataset:

2-shots This subset is selected from 2 types of cancer that repeat the most(i.e., the most common ones) in the Shikonin subset. This is for the purpose of a preliminary validation of transferable models, as this subset only has very limited number(62) of total CCLs, and a domain number of 2 is far from sufficient to conclude the function of any transferable model.

8-shots Similar to the 2-shots subset, this contains 153 CCLs from 8 types of cancer with the most repetition. In this research, this dataset is used for the purpose of performance comparison across different variations of models.

full-shots This is the whole Shikonin subset. In this research, this subset is only intended for specific transferable model that excelled from others in the 8-shots subset, to stress the model with most number of domain available and to observe how much improvement the transferable models can make compared to models without transferable learning features.

It is noteworthy that no dimension reduction technique is used in this research, but instead maintained all the 1085 features(1 domain feature of cancer types), which is the same as the original Davin's subset. This is because such a technique(e.g. PCA) is universal, and improvement made by such a technique can't be considered relevant to transferable learning. Therefore, while dimension reduction may be beneficial to the eventual performance of transferable models in this research, they do not help to indicate which model may transfer knowledge better than the others and is not adopted to keep irrelevant variables at a minimum.

3.2 GP Models

This research uses the GPytorch[23] as the main library to implement GP models. 2 major models tested and presented in this paper is Multi-task GP and Linear model of Corregionalization, where most generic components are implemented by GPytorch.

GPytorch is a Python library implemented with Pytorch that focuses on delivering abstracted models for GP models and their variants. Similar to Pytorch, it features an object-oriented interface and is suitable for non-basic level modification of models. For GPs, the most important mean and covariance is modulized in this library and open to the configuration of many kinds. Customized GP models and kernels, if not already provided by the library, can be easily extended from existing abstract class by declaring it as a subclass and inherits the parent's behavior.

Other GP libraries are also explored for reasons during the development of this research. GPy is another GP library developed by the University of Sheffield. It offers a high flexibility of model and kernel definition but comes at a low level of abstraction. This implies a steeper learning curve as beginners could spend a long time investigating all kinds of "how to do"s.

GPFlow is another GP library yet implemented on Tensorflow. Similiar to GPytorch, it provides encapsulation for all sorts of common kernels and GP models. An arguable downside of it is, despite extensive API document, limited use case of MOGP is provided, and the programming style is closer to procedure-oriented than object-oriented, the latter of which may require extra concern to keep the maintainability and readability of the code with the growth of the project's complexity.

In the following sections, the technical and implementation details of Multi-task GP and Linear model of Corregionalization will be introduced.

3.2.1 Multi-task GP

The Multi-task GP(MTGP in this research is equivalent to the MOGP model mentioned in previous sections). This naming is to respect the module name given by GPytorch, where a Kronecker product

$$K = K_{TT} * K_{XX}$$

is calculated given K_{TT} and K_{XX}, where T is number of tasks(outputs in MOGP terms) and X is number of X samples.

In this research, this model is implemented by declaring a subclass extended from ExactGP, and replacing the mean and covariance module by MultitaskMean and MultitaskKernel, former allows for individual mean to be assigned to each task, and the latter allows for the calculation of inter-task(inter-output in MOGP terms) covariance.

The reason to include this also as a model to test the kernel is to include the case where the transferable kernel handles most of the implicit relationships between hetero-topic data. Having more than 1 latent GP can be considered a way trying to model cross-domain information since this approach would possibly allow underlying correlations between domains to be learned in an unsupervised way. While how transferable kernel interacts with such property of LMC is interesting to investigate, to include MTGP test cases is to ensure there are test cases that cover when transferable kernel shoulders most of the capacity of domain knowledge transfer, to hopefully more accurately determine how much transferable kernel helps in DRP cases.

3.2.2 Linear model of Corregionalization

In this research, the LMC model is implemented by introducing the gpytorch.kernels.LCMKernel as the covariance module. GPytorch used a less common abbreviation for the kernel yet presenting the same as the original LMC model, which involves assuming the latent function to be a linear combination of a number of GPs. In gpytorch.kernels.LCMKernel, the number of latent GPs is controlled by the length of the kernel list passed to the base_kernels parameter. It is noteworthy that GPytorch forbids more latent GPs than the number of outputs, therefore the maximum number of latent GPs, within the scope of this research, would be the number of output concentrations.

To assign different kernels within the kernel list is possible yet not explored in the research, as it is considered unclear how such exploration relates to the purpose of the research. It is however potentially interesting to seek for a further improvement in performance by experimenting on whether different combinations of kernels may achieve any improvements.

3.3 Transferable Kernels

In this section, 2 major kernels(K_{SE} is a sub-kernel of K_{MS}) implemented in this research will be introduced. The idea of these transferable kernels originates from [4] and [10], and they are modified according to the kernel designed by [2]. Some minor adjustment has took place and they may slightly vary from the original idea, for convenience of implementation and calculation efficiency.

3.3.1 K_{SE}

$$K_{SE}^{(D_i,D_j)}(x_1,x_2) = \sqrt{s_i^2 s_j^2}\sqrt{\frac{2l_j l_j}{l_i^2+l_j^2}}exp(-\frac{||x_1-x_2||}{l_i^2+l_j^2})(i \neq j) \tag{8}$$

$$K_{SE}^{(D_i,D_j)}(x_1,x_2) = \exp(-\frac{||x_1-x_2||}{l_i^2+l_j^2})(i = j) \tag{9}$$

The idea of this kernel comes from the paper [3]. Being an independent transferable kernel itself, the K_{SE} is presented as a component of K_{MS}. Essentially, it is an RBF kernel with all quadratic terms replaced by a product of 2 separated parameters, similar to the idea to develop covariance from variance. The mathematical expression for K_{SE} is presented as follows, where s_i and s_j are the domain-specific scaling and l_i and l_j are domain-specific lengthscales.

A core idea of a transferable kernel is to introduce parameters that react differently to input from different domain. The idea of the kernel can be understood in the following way:

Having an RBF kernel in hand, what does it take to adapt it into a transferable kernel? An intuitive thought would be to have a few different RBF kernels to treat them individually. For n number of input domains, the 2 inputs can have n*n combinations. Considering the kernel should abide by the Commutative Law, i.e., $K(x_1,x_2) = K(x_2,x_1)$, the valid non-repetitive combination is reduced to $\frac{n^2+n}{2}$. In this way, the knowledge of each domain will be protected from contamination of those from other domains. Consequentially, this brings a downside, that to treat different domain input with different kernel destroy the construction of correlation between them, and this goes against the idea of MOGPs, which is to utilize the correlation and to give a more considerate prediction. Worse even, this will introduce a lot of parameters into the kernel. As is mentioned, for $\frac{n^2+n}{2}$ domain combination, an RBF kernel with hyperparameters of lengthscale and variance, 2 hyperparameters in total, will expand to a hyperparameters size of n^2+n. In practice, these 2 points mentioned before can lead to very bad overfitting, which is totally unacceptable in the field of DRP, where samples are considered limited.

An improvement can be made by a technique, hereby referred as the decomposition. The idea of this technique is, instead of having $\frac{n^2+n}{2}$ number of parameters to represent the hyperparameter for each domain pair, decompose the hyperparameter into 2 terms, each representing the domain for 1 of the 2 inputs of the kernel. In this way, $\frac{n^2+n}{2}$ of input domain combination can be approximated by the product of 2 hyperparameters from n hyperparameters for each domain.

For the K_{SE} kernel, the length scale terms will fall back to the case of a simple RBF when the input of the kernel is from the same domain. The scaling term only takes effect when the input is from different domains.

The paper[3] didn't provide an implementation of this kernel, so it has to be re-implemented in this research. For the K_{SE}, while it fall back to RBF kernel in same-domain case, it is not feasible to use RBF without making major compromise to the structure of encapsulation in GPytorch. This is because the RBF kernel module in GPytorch does not expose the lengthscale hyperparameter for modification while the K_{SE} needs the lengthscale in both same-domain case and cross-domain case. (the lengthscale is readable as a Python feature but hardly supports exporting the hyperparameter to reuse it either from cross-domain case to same-domain case, or the contrary)

The main challenge in the implementation is the input may contain data from different domain in different sequence. The final covariance matrix will be composed by different variances between each input sample. The same-domain case and cross-domain case will distribute irregularly in the covariance matrix. One can certainly argue this can be resolved by first ordering the input data by domain, treating same-domain and cross-domain cases locally, and eventually concatenating them into the final covariance matrix. It is not investigated in this research whether such a solution may lead to loss of gradient information, but it is fair to assume the existence of such risk, as irregular concatenation can potentially lead to such a problem. Worse even, ordering the data by the specific pattern is not the common behavior one would expect from a kernel function. This may lead to misunderstanding or setting obstacles for future researchers and limit the re-usability of the outcome of this research. Another noteworthy problem is that such an approach will damage the optimization for matrix multiplication and significantly add to the performance cost.

The solution given in this research is to compose 3 major matrices, for the first term $\sqrt{s_i^2 s_j^2}$, the second term $\sqrt{\frac{2l_j l_j}{l_i^2+l_j^2}}$ and the third denominator in the exponential term $l_i^2 + l_j^2$ prior to the calculation of distance, according to the domain information of the input. The denominator of the second term can use the third part directly. For the calculation of the squared distance and exponential, this research looked into the original implementation of the RBF kernel in GPytorch, and used the native method as much as possible, to approach the performance of the native RBF kernel module.

To differentiate the same-domain and cross-domain case, the scaling components s_i and s_j also need to be set to one in a single-domain case. In this research this is implemented by first augment the encoded domain information vector to the size of the final covariance matrix, by making the outer product with an all 1 vector. The next is to subtract one matrix from another and find out the locations of all 0, where all same-domain cases are. Using these locations to set the corresponding element in $\sqrt{s_i^2 s_j^2}$ matrix to 1. As the final step, do element-wise multiplication among the processed $\sqrt{s_i^2 s_j^2}$ matrix, $\sqrt{\frac{2l_j l_j}{l_i^2+l_j^2}}$ matrix, and the exponential term. The result would be the equivalent output of the K_{SE}.

3.3.2 K_{MS}

$$k_{ms}^{D_i,D_j}(x_1,x_2) = \begin{cases} \lambda_{(D_i,D_j)} K_{SE}^{(D_i,D_j)}(X_1,X_2),\ x_1 \in D_i \ \&\ x_2 \in D_j \ \&\ i \neq j \\ K_{SE}^{(D_i,D_j)}(X_1,X_2), x_1 \in D_i \ \&\ x_2 \in D_j \ \&\ i = j \end{cases} \tag{10}$$

$$\lambda_{(D_i,D_j)} = a_i * a_j,\ a_i, a_j \in [-1,1]$$

K_{MS} is a kernel based on K_{SE}, where K_{SE} is multiplied by a factorized domain coefficient, a_i and a_j. In the original paper [3], the kernel above is given as an instance of a proposed kernel framework, and not much explanation was given as to what idea is such construction is based on. It's apparent that the λ multiplier act as a measure of the relateness of the domain, and such relatedness is represented by decomposing λ into 2 terms, a_i and a_j for each domain. It remains unclear, though, that this setup has a high similarity compared to the term $\sqrt{s_i^2 s_i^2}$ in K_{SE}, aside from the part that in this case, λ can reach values below 0 and therefore modeling a negative correlation.

Among all experimented kernels, this kernel contains the most number of hyperparameters regardless of the change in a number of domains. A slight downside of it is such a kernel does not fit naturally into a multi-hyperdomain scenario. This means, for instance, if we have 2 sets of different domains(in this research, different cancer type and different drug compounds), it would be hard to model such data via this kernel as the designed parameter only takes 1. Considering a combination of specific drug compounds and cancer type will increase the number of domains dramatically, and because the growing number of hyperparameters is not accompanied by a growth of samples' number at the same scale, this will most likely lead to serious overfitting. Alternatively, we can use the decomposition trick again and decompose each hyperparameter to 2n sub-parameters(n being the number of hyperdomains), yet this would require extra mathematical work to prove the P.S.D property, and it's hard to say whether such chained approximation(the decomposition trick itself is an approximation) would have a negative influence on the performance of the model.

3.3.3 $K_{\alpha\beta}$

$$TrK_{\alpha\beta}(x_1,x_2) = \begin{cases} \Sigma_i \alpha_i k_i(x_1,x_2)\ if\ x_1\ \&\ x_2\ from\ the\ same\ domain \\ \Sigma_i \beta_i k_i(x_1,x_2)\ if\ x_1\ \&\ x_2\ from\ varied\ domains \end{cases} \tag{11}$$

$K_{\alpha\beta}$ is another transferable kernel proposed by paper [24], originally named $TrK\alpha\beta$. Compared to K_{MS}, the essential feature of $K_{\alpha\beta}$ is, instead of trying to parameterize the domain information to treat each domain differently, it only separates the same-domain and cross-domain cases. When inputs are from the same domain, α measures how much cross-domain knowledge can be transferred to the same-domain case, and β accounts for the cross domain cases.

The reasons to include this kernel in experiments are two: Firstly, this kernel contains much simpler and straightforward architecture, where each parameter represents a very specific and explicit meaning and a free choice of sub-kernel(adding kernel linearly does not compromise P.S.D property). This makes this kernel an ideal starting point if any modified kernel were to be constructed. Secondly, as stated before, the kernel follows an entirely different approach to model domain information, and the experiment would be more comprehensive to include both approaches to compare their performance in the DRP field.

The implementation of this kernel, though not given in the original paper, is simple and straightforward. Following the consistent idea of minimizing matrix concatenation, the covariance matrices from each sub-kernel are calculated in the first step. These matrices are stacked together and multiplied by α and β as a vector with a length of number of domains, to obtain mat_α and mat_β. Next, 2 masks containing only 0 and 1 are generated by the domain information of input, with the same size as the final output covariance. One masks the location of same-domain cases, and the other masks the contrary.

Eventually, mat_α and mat_β do element-wise multiplication with the mask and add together to get the final covariance matrix as the output of the kernel.

3.3.4 The K_{MS} with Adaptive Feature Scaling

$$K_{MS-AFS}(x_1, x_2) = K_{MS}(sx_1, sx2) \tag{12}$$

K_{MS} with Adaptive Feature Scaling(K_{MS-AFS}) is the final result of an attempt to combine K_{MS} with the quadratic kernel used in the paper[2]. The s is a fitted vector acting as a weight, element-wise multiplied to the original x_1 and x_2. It's a generalized form of the quadratic RBF kernel in paper[2].

The quadratic kernel[2] is expressed as:

$$K_{quad}(x,x') = K_{mu}(x_{mu},x'_{mu})K_{met}(x_{met},x'_{met})K_{cn}(x_{cn},x'_{cn})K_{dc}(x_{dc},x'_{dc}) \tag{13}$$

where K_{mu}, K_{met}, K_{cn}, K_{dc} are RBF kernels, and x_{mu},x_{met},x_{cn},x_{dc} are sub-components of the input x describing mutation, methylation, copy number, and drug compound features respectively. Importantly, the experiments conducted on this kernel found that the MOGP model with this kernel does not perform well when trying to transfer knowledge across domains (cancer type or drug compound), and in most of the experiments, the K_{dc} will always return one as the data used does not have more variants than the only kind of drug compound.

In the first place, a quadratic transferable kernel is constructed as follows:

$$K_{quad}(x,x')^{D_i,D_j}(x_1,x_2) = \begin{cases} \lambda_{(D_i,D_j)} K_{SE-quad}^{(D_i,D_j)}(X_1,X_2),\ x_1 \in D_i \ \&\ x_2 \in D_j \ \&\ i \neq j \\ K_{SE-quad}^{(D_i,D_j)}(X_1,X_2), x_1 \in D_i \ \&\ x_2 \in D_j \ \&\ i = j \end{cases} \tag{14}$$

and

$$K_{SE-quad}(x,x') = K_{SE-mu}(x_{mu},x'_{mu})K_{SE-met}(x_{met},x'_{met})K_{SE-cn}(x_{cn},x'_{cn})K_{SE-dc}(x_{dc},x'_{dc}) \tag{15}$$

where K_{SE-mu} is a K_SE but treats only the mutation features component, and others follow the same analogy.

This is not a naturally legal extension and the P.S.D property cannot be assumed. As is previously mentioned, the K_{MS} implementation used in this research is merely an instance that satisfies a framework proposed in [3]. To guarantee the P.S.D. property, this framework demands the sub-kernel to satisfy the following property:

Given a stationary kernel $K^{(D_i,D_j)}(x,x')$, it is considered a valid component of K_{MS} if it can be written in the following form:

$$K^{(D_i,D_j)}(x,x') = \int_{-\infty}^{\infty} f_{D_i}(x-u) f_{D_j}(x'-u) du \tag{16}$$

As the quadratic kernel is based on the RBF kernel, the stationary property is self-explanatory. To prove RBF kernel and the quadratic kernel can be written in the mentioned form, first we assume $f_{D_i}(x-u)$ follows a Gaussian form:

$$f_{D_i}(x-u) = N(\ \mu_i \mid \delta_i^2) \tag{17}$$

Then the equation 16 can be written as:

$$K^{(D_i,D_j)}(x,x') = \int_{-\infty}^{\infty} N(\ x-\mu,\ P_i^{-1}) N(\ x'-\mu\ , P_j^{-1}) d\mu \tag{18}$$

where μ_i is the multi-variate mean and P_i^{-1} is the covariance matrix of the Gaussian. P_i is also known as the precision matrix, which is the invert of covariance. Here we replaced u with μ to respect default Gaussian notations.

As multiplication of Gaussian follows:

$$N(x-\mu_1, P_1^{-1}) N(x-\mu_2, P_2^{-1}) = N(\mu_1-\mu_2, P_1^{-1}+P_2^{-1}) N(x-\mu_c, P_c^{-1}) \tag{19}$$

where $\mu_c = (P_1+P_2)^{-1}(P_1\mu_1 + P_2\mu_2)$ and $P_c^{-1} = (P_1+P_2)^{-1}$. Notice that on the right-hand side of the equation, the first term is composed purely of constants.

Substituting 19 into 18, we get:

$$K^{(D_i,D_j)}(x,x') = \int_{-\infty}^{\infty} N(x-x', P_1^{-1}+P_2^{-1}) N(\mu-\mu_c, P_c^{-1}) d\mu \tag{20}$$

Notice that during the substitution, the x and x' as kernel input correspond to μ_1 and μ_2 in the Gaussian multiplication. Since the first term can be considered constant with respect to the integral, we can then extract it to outside of it:

$$K^{(D_i,D_j)}(x,x') = N(x-x', P_1^{-1}+P_2^{-1}) \int_{-\infty}^{\infty} N(\mu-\mu_c, P_c^{-1}) d\mu \tag{21}$$

As the interal of the Gaussian PDF is always one, the original conditon to satisfy becomes:

$$K^{(D_i,D_j)}(x,x') = N(x-x', P_1^{-1}+P_2^{-1}) \tag{22}$$

This suggests that a sufficient not necessary condition of a constructed kernel to be a valid sub-component of K_{MS}, is for it to follow a Gaussian form. This validates RBF kernel as a K_{MS} sub-component. The serial product of an RBF-like kernel(if following a Gaussian form) can be validated by following a similar transform, which would validate the original quadratic kernel and quadratic K_{SE}. Because the proof of these are highly similar to the provided single RBF case, they will not be repeated in this paper.

However, the quadratic transferable kernel shown very similar performance on the tested 2-shots dataset to a regular K_{MS} kernel(reported in detail in section 4). A similar phenomenon also happened in some preliminary explorations of MOGP, where the original quadratic kernel shown very similar performance to a regular RBF kernel. The cause of such fall-back is unclear and could be potentially explained by a lot of reasons, as this research doesn't use a GDSC subset of the same size as the paper [2], there could be minor discrepancies in model structure and most importantly, the paper [2] covered only for specific drugs and specific cancer types, and pattern uncovered on those may not be universal. The exact cause of such phenomenon is unknown and determining those is not the purpose of this research. Yet this gives sufficient reason to reconsider the structure of the kernel proposed by this research and lead to the K_{MS} with Adaptive Feature Scaling.(K_{MS-AFS})

The original quadratic kernel can be re-expressed in the following way:

$$\begin{aligned}
K_{quad}(x,x') &= K_{mu}(x_{mu},x'_{mu})K_{met}(x_{met},x'_{met})K_{cn}(x_{cn},x'_{cn})K_{dc}(x_{dc},x'_{dc}) \\
&= exp(\frac{(x_{mu}-x'_{mu})^2}{l_{met}^2})exp(\frac{(x_{met}-x'_{met})^2}{l_{cn}^2})exp(\frac{(x_{cn}-x'_{cn})^2}{l_{dc}^2})exp(\frac{(x_{dc}-x'_{mu})^2}{l_{dc}^2}) \\
&= exp(\frac{(x_{mu}-x'_{mu})^2}{l_{met}^2}+\frac{(x_{met}-x'_{met})^2}{l_{cn}^2}+\frac{(x_{cn}-x'_{cn})^2}{l_{dc}^2}+\frac{(x_{dc}-x'_{dc})^2}{l_{dc}^2}) \\
&= exp((x-x')^2 L_s)
\end{aligned}$$

where L_s is a diagonal matrix that is expressed as follows:

$$L_s = diag(\underbrace{\frac{1}{l_{mu}^2},\frac{1}{l_{mu}^2}\dots,\frac{1}{l_{mu}^2}}_{Number\ of\ Mutation\ features},\frac{1}{l_{met}^2},\dots\frac{1}{l_{met}^2},\frac{1}{l_{cn}^2},\dots\frac{1}{l_{cn}^2},\frac{1}{l_{dc}^2},\dots\frac{1}{l_{dc}^2}) \tag{23}$$

This shows the original quadratic MOGP kernel is essentially a single RBF but with its input scaled on 4 parts of different features respectively. More importantly, this explains why in specific scenarios (as in this research), this kernel can potentially fall back to RBF kernel when l_{mu},l_{met},l_{cn} and l_{dc} take similar value.(This can also greatly simplify the K_{MS} validation of the quadratic RBF kernel)

While the exact reason for such fallback is unclear, what is implied by this phenomenon, is the assumption that input features are varied by the 4 types of them(which leads to the motivation to construct the quadratic kernel), may not hold in the dataset used in this research. Intuitively, if assigning such a pattern to scale is inappropriate for the dataset, possibly we can instead let the kernel itself learn about which features is more important and which are not. This brings the K_{MS-AFS}, which is mathematically represented as follows:

$$K_{MS-AFS} = exp((x-x')^2 L_A) \tag{24}$$
$$L_A = diag(\underbrace{\frac{s_1}{l^2}, \frac{s_2}{l^2}, ..., \frac{s_n}{l^2}}_{Number\ of\ all\ features}) \tag{25}$$

which is mathematically equivalent to the form 12. Instead of assuming each type of input feature should share the same scaling term, this kernel treat every single input feature as a independent type and learn the scaling term during the training.

The implementation of this kernel is simple and trivial and, therefore not covered in this paper.

3.4 Other Implementation Details

This section covers certain details that would become too scattered if discussed in other sections yet important to this research. The purpose of this section is to address those details together.

3.4.1 Reparameterization

Reparameterization refers to a technique to substitute certain parameters in a model with function value, usually for the purpose of solving gradient propagation issues. During this research, it was discovered that non-P.S.D. problem occurs widely across different models with different kernels. This wouldn't be surprising for some of the kernels that are not guaranteed to be P.S.D., but unexpectedly this occurs also for some of the kernels that are supposed to be P.S.D. It was later discovered that more reasons can lead to P.S.D. problems other than the kernel itself. In this research, it seems that P.S.D. is caused by some instability in gradient back-propagation. They are more likely to occur when given a large learning rate or slow decay of it. Efforts are made by searching for a suitable "learning rate channel" for the model to carry out gradient descent, yet very unsuccessful since this limits the performance of the model very badly, especially when the learning rate is throttled at very low level.

Different models with different kernels seem to react differently to the P.S.D. problem. E.g. RBF kernels included in the library rarely have such issue, yet self-defined kernels such as K_{MS} suffered from the problem badly. A correlation is found between the number of self-defined parameters(especially those required to be limited in a specific interval) and sensitivity to learning rate-caused non-P.S.D. issue. This leads to the hypothesis, that parameters with interval constraints are correlated with non-P.S.D. errors, and therefore this research is motivated to seek for substitution for interval constraints, eventually reaching the solution of reparameterization.

In paper [3], it is mentioned that the domain relateness coefficient a_i is reparameterized with a hyperbolic tangent function to limit its output at [-1,1]. A similar technique is tested and applied to the kernels in this research having constrained parameters, whether they have the same constraints as the domain relateness coefficient or different. Most of the parameters are reparameterized by hyperbolic tangent function with few exceptions. Provided below is a chart of all parameters being reparameterized.

3.4.2 Work-rounds for matrices concatenation

Transferable kernels need to handle inputs with domain information varied on each sample. This would require samples from different domains to undergo different math processes. For instance, the kernel K_{MS} requires the variance to be calculated using specific K_{SE} selected according to the domain information of samples. While it looks natural to use the divide-and-conquer strategy, to slice the inputs according to their domains, calculate the variance and put them in the correct place in the final covariance matrix. This, unfortunately, turned out to be a very inefficient solution.

In early preliminary explorations, it was found that matrices concatenation damages the efficiency of the kernel very badly. The kernel would take so long to calculate that any work based on them, even on 2-shots dataset, becomes extremely slow and therefore impractical. This is because matrix concatenation disrupts the parallelized matrix mathematics by breaking them into much smaller subparts.

The fix to this issue, in this research, is to convert concatenation into mask-and-add operation. Taking K_{MS} again as the example, the original logic, according to equation 10 and 8, is to use different parameters s_i and l_i for variance calculation according to the domain information input. This needs to be avoided because once these parameters participate directly in the calculation, and, for instance, are multiplied to the individual input distance $||x_1 - x_2||$, the concatenation will be unavoidable from this point since they become necessary to connect these individual input distance together.

The solution of this research is to postpone the timing when domain-specific parameters interact with non-domain-specific ones, to calculate them prior to the vector distance using vector and matrix arithmetics, and to assemble them with the vector distance component $||x_1 - x_2||$ eventually. In the K_{MS}'s case, this means the following steps

1. Vectorize domain-specific parameter according to domain information input. This means to replace each domain-representing digit with the corresponding s_i and l_i. This can be easily implemented with a mapping operation that is common across Python libraries that operates on vectors.
2. Divide-and-conquer but in a different way. The sub-kernel K_{SE} is separated into parts: $\sqrt{s_i^2, s_j^2}$, $2l_il_j$ and $l_i^2 + l_j^2$, which are the major domain-specific components involved during the calculation of it.

3. Convert them into matrix arithmetics. In this case, addition between vectorized parameters p_1 and p_2 can be represented by First left multiplying them respectively to an identity matrix and doing matrix addition with the result. Multiplication can be represented by outer product.
4. Assemble all components with element-wise product. Notice all intermediate result obtained in the previous steps should produce matrices of same size as the final covariance matrix. For multiplication between components, use again the element-wise multiplication. For functions, the implementations of them, e.g. square root and exponential, are usually compatible with element-wise operation on matrix and therefore can be used directly.

By the previously described procedure, matrix concatenation can be saved and the kernel calculation can benefit more from matrix arithmetic acceleration mechanisms.

3.5 Experiments Design

As introduced in earlier sections, there are a few different datasets prepared for different purposes in this research. In this section, this will be discussed in more detail about how exactly the experiments' configurations are and how they are evaluated.

3.5.1 Datasets

Quantitative tests are conducted on models with 2-shot and 8-shot datasets. A grid scan is conducted at first to identify potentially outstanding models, then these models go through a 10-fold validation to obtain a more accurate representation of their performances.

The grid search scans hyperparameters that are critical to performance of the model, including learning rate, the learning rate decaying parameter γ, and number of epochs that the decay is triggered. A combination of the following chart is covered in the grid search.

Model	LMC	MTGP			
Kernel	K_{MS-AFS}	$K_{\alpha\beta}$	K_{MS}	$K_{\alpha\beta}$	RBFKernel
Learning rate(LR)	0.007	0.178	0.349	0.520	0.691
Step size(SS)	0.1	30	0.8	-0.9646	
γ	0.2	0.35	0.65	-0.80	

Table 1. Grid search results on full-shot dataset

The 10-fold validation tests the models on the same dataset but with 10 train-tests split with different random seeds. The mean and standard deviation of the model are extracted for further analysis.

A train-test split of 80%/20% is used, and all parameter combinations share the same random seed accounted for the split.

Important to know, that while the burden of the test is relatively easier on the 2-shot subset, when the test moved on to 8-shots, it slows dramatically and access to CSF is vital for this research for all experiments to be completed in reasonable amount of time.

The full-shots subset merely serves the purpose of a pressure test. Grid search can't be applied to this one as for 8-shots dataset the amount of needed time has raised to 80 hours in serial, and for the full-shots a rough estimation for the time consumption is near 2 weeks. For this reason, no quantitative tests are conducted on this subset but only for specific model-kernel combination with outstanding performance obtained on 8-shots subset. The initial hyperparameter choice is inherent from the result of 8-shot subset, and small turbulence is added to each hyperparameter manually for better performance. It is not guaranteed to deliver a near-optimal hyperparameter combination in this way, and therefore comparison made on this subset is for reference only.

Additionally, an extra test is added on the 8-shots dataset to evaluate how the domain relateness may affect the transferable model. Heat maps will be used to represent the domain relateness pairs($a_i a_j$ in K_{MS} and α and β in $K_{\alpha\beta}$) If they do help for transferable learning, then they should share some similarity in their pattern when in each fold of the cross-validation. Otherwise, it may suggest either the parameters don't help in an explicit and straightforward way, or they may not be ideally effective.

3.5.2 Training

All models trained are given 500 epochs at maximum and a learning-rate stepping mechanism is introduced, such that after every S number of epochs the learning rate is multiplied by γ. An early cutoff mechanism is also introduced, such that for every time the learning rate steps, the training function check for the average loss in the latest S epochs and another S epochs earlier. If the average loss does not differ more than 1%, the early-cutoff mechanism will kick in and terminate the training. Notice that this does not necessarily prevents overfitting but instead reduce the time consumption when the model has already converged.

3.5.3 Evaluation metric

For the evaluation metrics, negative log-likelihood will be used to measure the extent of matching between the model and data. Compared to other metrics such as mean square error, the advantage of NLL is that it evaluates not only the mean but also the variance. This is critical as the essence of GP models is they provide predictions with the degree of uncertainty.

4 Result and discussion

In this chapter, the experiments results carried out on 2-shot, 8-shot and full-shot will be reported and evaluated.

4.1 2-shot

Model	Kernel	LR	SS	γ	NLL
MTGP	RBF	0.18	55	0.8	-1.1935
LMC	RBF	0.69	32	0.5	-1.1955
MTGP	K_{MS}	0.18	77	0.65	-1.2147
LMC	K_{MS}	0.18	77	0.65	-1.2431
MTGP	$K_{\alpha\beta}$	0.18	32	0.2	-1.1767
LMC	$K_{\alpha\beta}$	0.69	77	0.2	-1.2061
MTGP	K_{MS-AFS}	0.18	77.5	0.35	-1.1789
LMC	K_{MS-AFS}	0.18	33	0.35	-1.1922

Table 2. Grid search results on a 2-shot dataset

The grid search results on the 2-shot dataset are presented in the table 2 (LR: learning rate; SS: step size; γ: learning rate decay; NLL: Negative log-likelihood). This takes around 8-12 hours to complete in serial on CPU. Moving the task to GPU doesn't seem to improve the efficiency significantly but starting to trigger certain numerical errors. On the other hand, the time consumption can be greatly reduced if the hyperparameter combinations to several sub-parts and runs all of them in parallel. For these reasons, it is considered better to run the task on the CPU with multiple tasks parallelization if possible.

Notice that the hyperparameter combinations presented here are not always the best but those can finish a 10-fold cross-validation. It turns out that it would be quite rare for certain model-kernel combinations to complete 10-fold cross-validation, therefore in practice, a cross-validation result is considered acceptable as long as the model-kernel combination completed 8 out of 10 folds. For some hyperparameter combinations, the learning rate sloping strategy is too aggressive leading to the models throwing P.S.D. errors sometimes. (As discussed earlier, a relatively too high learning rate in all stages potentially risks P.S.D. error) Combinations experienced this issue are marked with " *". Coincidentally, all combinations accepted with such compromise happens to complete exact 8 folds while failing in different folds respectively. While this research does not find this either interesting or relevant to investigate, this may be a hint to identify the actual cause of the problem in the future. (a possible hypothesis would be, e.g. certain corrupted samples allocated to the training set)

These hyperparameter combinations are used in the 10-fold cross-validation, the result of which is shown in table 2:

Model	Kernel	*NLL* ± *Standard deviation*
MTGP	RBF	-0.77 ± 0.47
LMC	RBF	-0.87 ± 0.23
MTGP	K_{MS}	$-0.76 \pm 0.347*$
LMC	K_{MS}	$-0.57 \pm 0.35*$
MTGP	$K_{\alpha\beta}$	-0.71 ± 0.26
LMC	$K_{\alpha\beta}$	-1.00 ± 0.26
MTGP	K_{MS-AFS}	-0.98 ± 0.37
LMC	K_{MS-AFS}	-0.92 ± 0.28

Table 3. Cross validation results on 2-shot dataset

The NLL here is merely for ranking the quality of hyperparameter combinations, not for measuring the performance of the model. To obtain more reliable metrics of models, the 10-fold cross-validation is performed, the result of which is presented as table 3 and figure 4

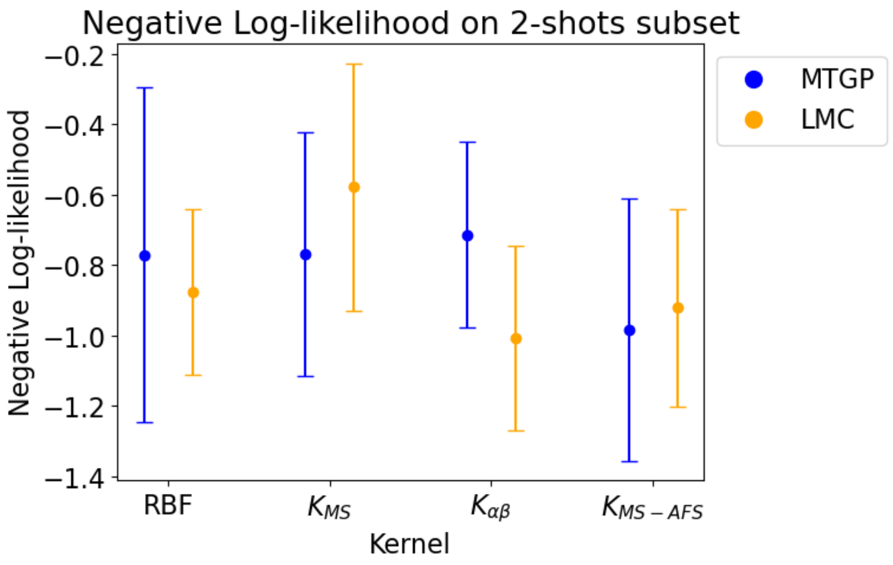

Fig. 4. Negative Log-likelihoods on 2-shots subset

It appears that, compared to RBF kernel, most transferable kernels only shown marginal advantage while K_{MS} even do slightly worse than RBF. As discussed earlier, LMC is also capable of modeling potential domain differences, though not in an explicit way. In most of the combinations, LMC brings an improvement in the deviation, which corresponds to the stability of model's performance. It's hard to say in this stage that which kernel is necessarily better for transfer learning task, as none of them has shown a fundamental difference than regular RBF. To some extent, this is understandable, as 2-shot subsets are not really complicated tasks, and because the domain label is also an input to RBF, it's possible for RBF to partially adapt to the domain difference, despite being impossible to model them well as its simple structure forbids.

However, with the growth of complexity, we should expect to see a drop in RBF's performance, while the transferable kernels' performances should at least maintain or even improve. More domains lead to more complicated transfer learning tasks and more subtle relationships between domains. This is further shown in the following section.

4.2 8-shot

The grid search results on the 8-shot dataset are presented in the table above. A rough estimation of time consumption to run the task on Google Colab with TPU v2 is around 80 hours. This has become impractical until access to UoM CSF is approved. The grid search was split into 4 pieces and ran in parallel. The grid search was eventually completed within 24 hours.

Model	Kernel	LR	SS	γ	NLL
MTGP	RBF	0.18	10	0.8	-0.9009
LMC	RBF	0.18	10	0.8	-0.8276
MTGP	K_{MS}	0.18	55	0.65	-1.0937
LMC	K_{MS}	0.35	10	0.5	-0.9083
MTGP	$K_{\alpha\beta}$	0.18	100	0.5	-1.1963
LMC	$K_{\alpha\beta}$	0.69	33	0.2	-1.174
MTGP	K_{MS-AFS}	0.18	55	0.65	-1.211
LMC	K_{MS-AFS}	0.18	33	0.35	-1.1706

Table 4. Grid search results on 8-shot dataset

During 8-shot grid search, extra measure are deployed to keep the training going by skipping those epochs that produces a P.S.D. error. Yet this doesn't seems to help, as once the P.S.D. error is produced it seems to damage the kernel in some way and produce more P.S.D. error despite the countermeasure. A quick analysis points out, that P.S.D. error on a given epoch is not caused by the input of this epoch but the result of gradient descend of the last epoch. This makes the P.S.D. error unavoidable as it's impossible to know whether to skip a given epoch based on if a P.S.D. error is going to be produced in the future.

From the grid search result, it looks like the previous expectation is confirmed on how the performance of kernels would change corresponding to the growing complexity of the task. For the best NLLs alone, all transferable kernels seems to at least roughly maintain the performance facing task of transfer learning between 8 different domains, except for LMC with K_{MS}, which sees a significant cutback in performance. Additionally. a partial slight improvement is spotted on K_{MS-AFS}. However, these observations is soon proved to be partial.

Based on the grid search, the 10-fold cross validation returns surprising result, shown in table 5 and figure 5. For most of the transferable models, the result of cross validation greatly differs from that of grid search. This is due to their performance do not maintain well across different folds. Most transferable models shown a significantly larger fluctuation of performance compared to those of RBF models. There are 3 different models that outperform RBF models, which is MTGP with K_{MS}, LMC with $K_{\alpha\beta}$ and MTGP with K_{MS-AFS}. These 3 kernels have similar NLL means but with different variance. Among all model-kernel combinations, MTGP-K_{MS-AFS} has the best NLL mean and around second best variance.

Model	Kernel	$NLL \pm Standard\ deviation$
MTGP	RBF	-0.79 ± 0.06
LMC	RBF	-0.72 ± 0.09
MTGP	K_{MS}	-0.95 ± 0.29
LMC	K_{MS}	-0.45 ± 0.29*
MTGP	$K_{\alpha\beta}$	1.22 ± 1.38
LMC	$K_{\alpha\beta}$	-0.99 ± 0.20
MTGP	K_{MS-AFS}	-1.13 ± 0.30
LMC	K_{MS-AFS}	-0.65 ± 0.39

Table 5. Cross validation results on 8-shot dataset

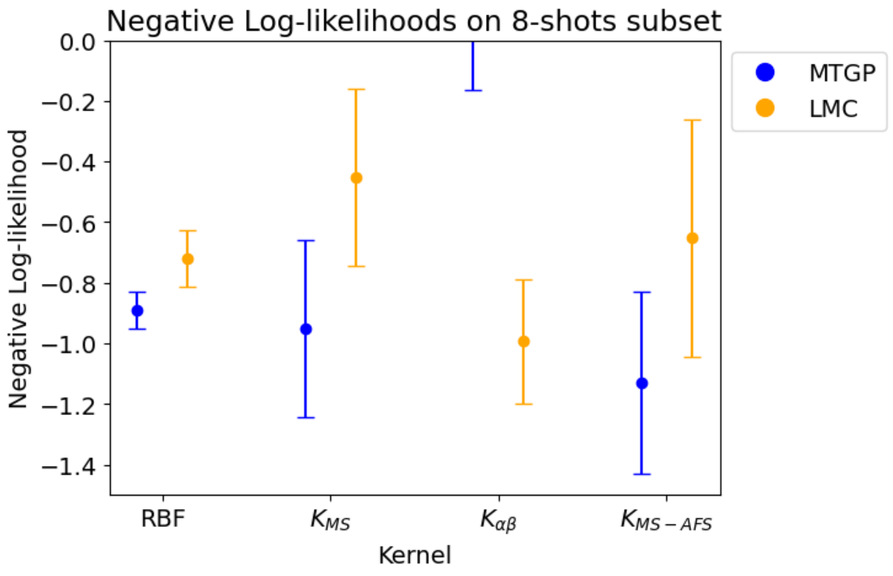

Fig. 5. Negative Log-likelihoods on 8-shot subset

Interestingly, many transferable kernels show a large difference when accompanied by different models. K_{MS} and K_{MS-AFS} shares the same structure of domain modeling, while $K_{\alpha\beta}$ is different to the others. This is inherent when K_{MS} and K_{MS-AFS} both show a preference for the MTGP model, while $K_{\alpha\beta}$ prefers the LMC model. This is likely because of the LMC model's implicit capacity to model domain differences with latent GPs. Both K_{MS} and K_{MS-AFS} have better structure modeling the domain difference, which competes with those of LMCs. Latent GP is in the downstream of transferable kernels, which would lead to an earlier position when back-propagating. The gradient information that domain difference incurred may be "absorbed" partially prior to it arriving at transferable kernels, leading to part of the domain difference being modeled by less capable components.

While in the $K_{\alpha\beta}$'s case, it does not model domain difference directly but measures how much knowledge in each domain can be transferred to all others. This, compared to the previous case, does not incur competition between components, which may explain why it prefers LMC model. The terrible performance of MTGP-$K_{\alpha\beta}$ can be partially explained by the same reason, where neither MTGP nor $K_{\alpha\beta}$ effectively models domain difference and unable to deal with domains that are dramatically different from each other. Yet this alone doesn't explain why MTGP-$K_{\alpha\beta}$ ended up significantly worse than RBF models. Another potential reason could be numerical instability, as it is indeed shown MTGP-$K_{\alpha\beta}$ only passed 8(coincidentally again) out of 10 folds.

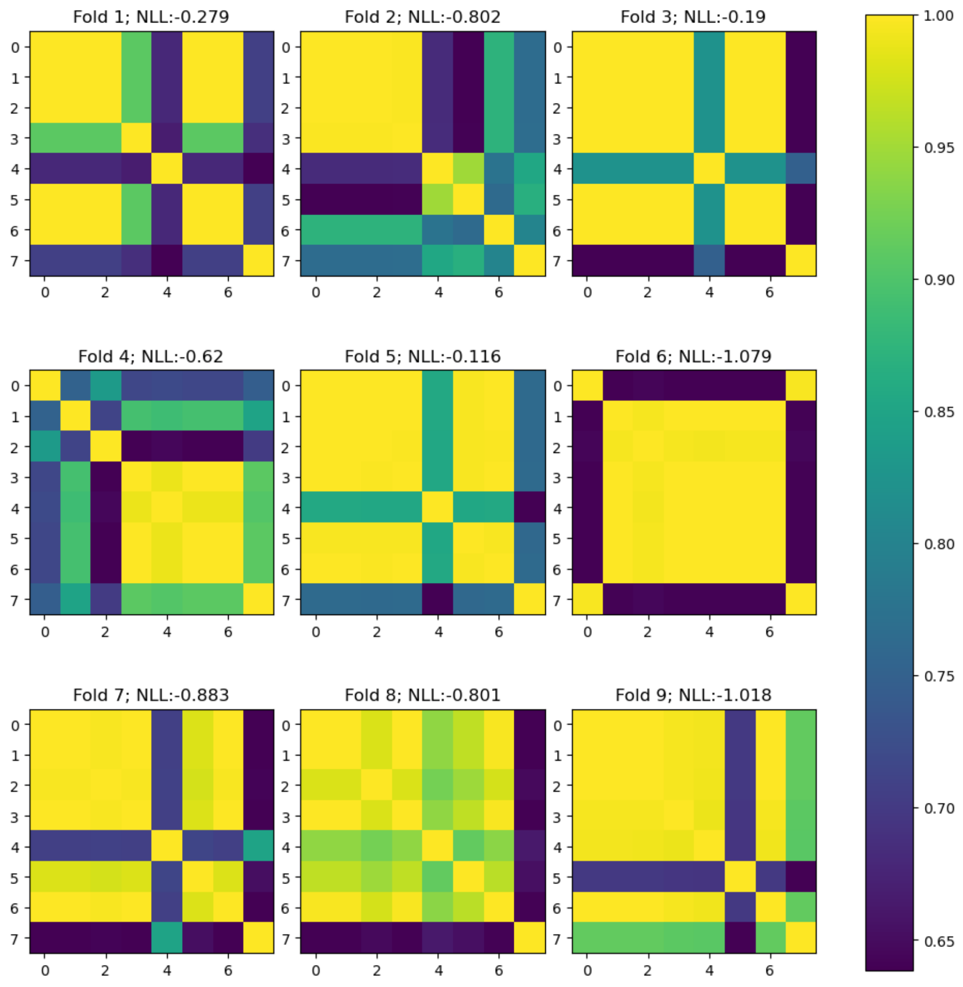

Fig. 6. Heat map of K_{MS}'s domain relateness

A supplementary experiments is conducted to research on if there would be anything interesting in domain relateness. Figure 6 shows the domain relateness term a_i after training on 8-shot dataset on the first 9 folds(the 10-th abandoned for spacing). It appears that, whether these domain relateness gives a regular speadout doesn't seem to relate to the actual performance of the model, which is more or less counter-intuitive. It's very clear that the 4-th domain appears to be influential to other domains throughout many folds. The folds 1,2,3,5,7,8 all show such a tendency, even fold 4 has a faint pattern of such. Yet being regular as they are, their performance varies from some of the worst to the best. Notice that for fold 6, the fold ends up with the best NLL, the domain relateness is so polarized that it concludes the 1-st and 8-th domain to be entirely irrelevant to the rest, and the rest being entirely equal.

On the other hand, the result returned from $K_{\alpha\beta}$ is far from satisfactory. Different than the case of K_{MS}, $K_{\alpha\beta}$'s parameter, α and β don't show any correlation with any domain. The figure 7 and 8 is heat maps for the parameters. In heat map7, each alpha in the kernels of all latent GPs, as column vectors, are stacked horizontally, and these results are extracted from the first 9 folds out of 10 folds for convenience of the report's layout. Ideally, if any pattern were to be found, what should be expected is horizontal stripes with similar colours. However, the α's result returns no better than random noise. When it comes to beta, the heat map is arguably even worse, since we are seeing vertical stripes instead of horizontal ones. Because β accounts for cross-domain similarities, this means for

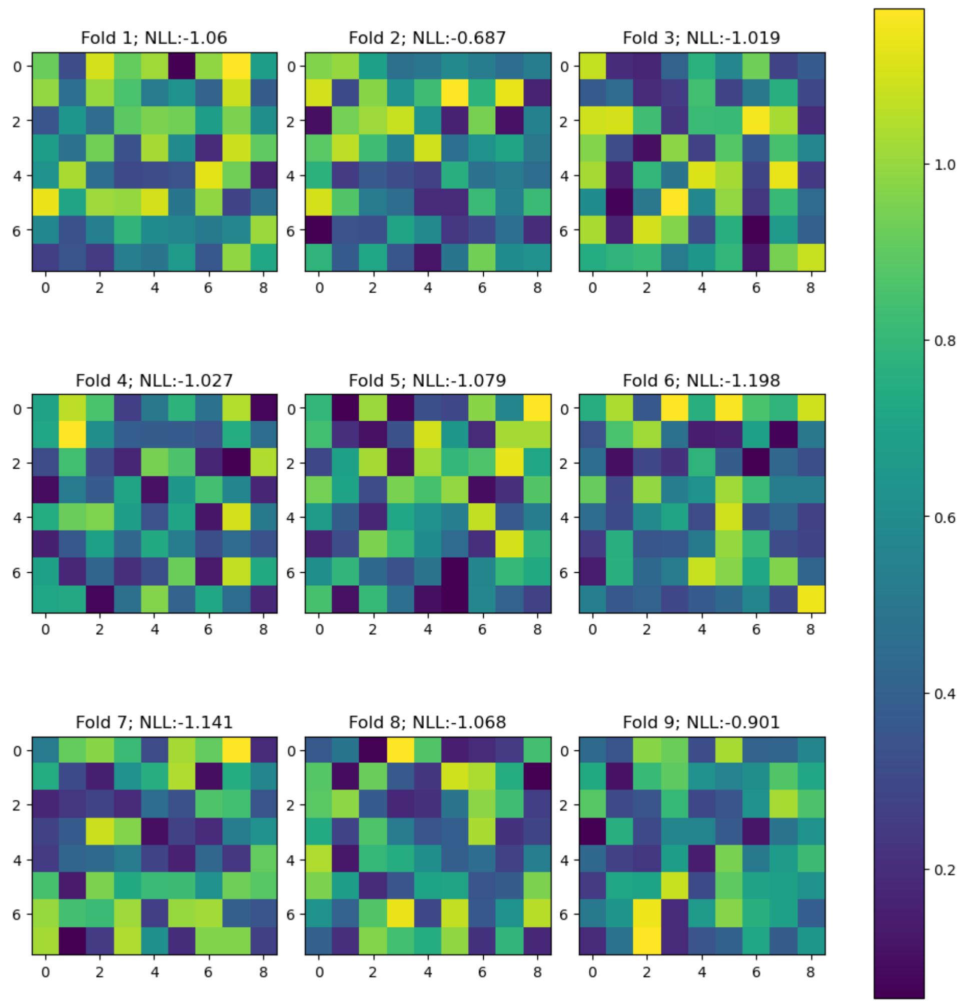

Fig. 7. Heat map of $K_{\alpha\beta}$'s α

some latent GPs, the kernel considers there is no knowledge to transfer and completely turns cross-domain knowledge transfer off. In short conclusion, it seems the domain relateness from K_{MS} can show a certain degree of explicit distribution pattern after training, suggesting the kernel offers more interpretable domain relateness. For kernel $K_{\alpha\beta}$, at least such a pattern is much more implicit and not directly understandable.

4.3 full-shot

Eventually, 3 outstanding models, LMC-K_MS, MTGP-$K_{\alpha\beta}$ and LMC-K_{MS-AFS}. Important to know that the comparison made on this dataset is for reference only. This is because massive grid search has become extremely hard at this stage, and therefore the grid search is modified to focus on a much narrower range with much less frequency. Specifically, the grid search points for learning rate are (0.1,0.2,0.3), (0.2,0.5,0.8) for γ and (30,60,90) for step size. Additionally, MTGP-RBF and LMC-RBF are also trained in a similar way as a benchmark.

The result of the grid search is presented in table 6, that of cross-validation is shown in table 7 and further depicted in figure 9. Similar to the case in the 8-shot experiment, sometimes advantageous

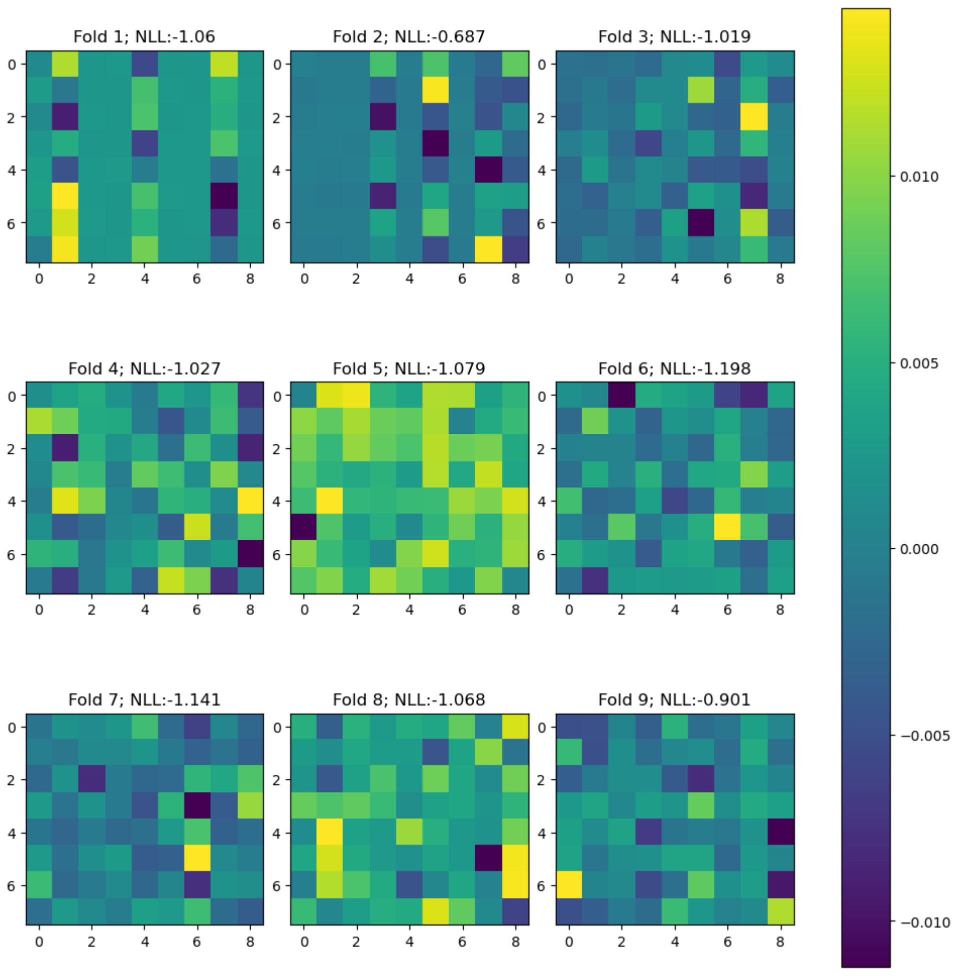

Fig. 8. Heat map of $K_{\alpha\beta}$'s β

Model	Kernel	LR	SS	γ	NLL
MTGP	RBF	0.1	30	0.8	-0.9646
LMC	RBF	0.1	60	0.8	-0.8951
MTGP	K_{MS}	0.1	90	0.8	-1.0313
LMC	$K_{\alpha\beta}$	0.2	90	0.5	-1.0313
MTGP	K_{MS-AFS}	0.1	90	0.5	-1.0435

Table 6. Grid search results on full-shot dataset

models during the grid search don't give stable performance across all folds of cross-validation. In this subset, both of the models using LMC see a significant cutback in performance. This may have a correlation with some numerical error since at this stage the model starts to throw numeric warnings indicating precision lost. Considering LMC's complexity, there is good reason to believe its performance

Model	Kernel	$NLL \pm Standard\ deviation$
MTGP	RBF	-0.89 ± 0.09
LMC	RBF	0.39 ± 0.58
MTGP	K_{MS}	-0.91 ± 0.18
LMC	$K_{\alpha\beta}$	0.13 ± 0.73
MTGP	K_{MS-AFS}	-0.94 ± 0.10

Table 7. Cross validation results on full-shot dataset

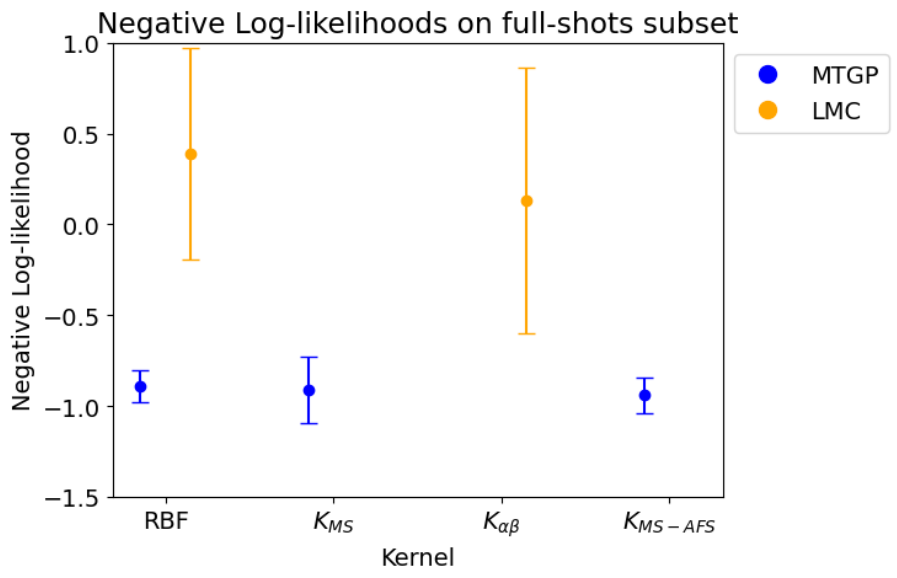

Fig. 9. Negative Log-likelihoods on full-shot subset

may have been damaged in this way. Besides, LMCs in this research use the number of domains as number of latent GPs. It is possible that too much latent GPs(30) is leading to serious overfitting in full-shot dataset. Further exploration in this regard is unfortunately impossible as including number of GP's into grid search will multiply the searching workload on the current basis, which is already near unbearable.

the MTGP-RBF model reaches the third among all outstanding models. This is surprising yet understandable. In previous tests, MTGP-RBF has shown very steady performance, probably due to its robustness in model structure. This also implies in the given subset, the variance between domain knowledge may not be dramatic. K_{MS} and its variant K_{MS-AFS} rank the second and the first in all models, while K_{MS-AFS} achieves the second steady performance, next to the MTGP-RBF.

5 Conclusion

5.1 Findings

This research has investigated the performance of domain knowledge transfer of many MOGP models and kernel combinations. The major findings are concluded as follows.

5.1.1 Model's transfer learning potential

The regular MTGP model and LMC model both show potential for transfer learning but from different approaches. LMC allows for modelling more complex and variable data, yet its capability of such seems to compete with the transfer kernels and usually leads to worse performance in the cases covered by this research. It is shown, however, that kernels don't model domain variance directly(e.g. $K_{\alpha\beta}$), it can, at least in cases, reach an ideal performance. MTGP models work better with common transferable kernels, which would react differently and produce varied covariance when handling input from varied domains.

5.1.2 Efficiency of kernels' implementation

One of the major challenges this research has faced was the calculating efficiency of self-defined kernels. Compared to traditional kernels, implementations of which are provided by the library, these kernels require dedicated performance optimization and carefully designed workflow in order to reach the equivalent efficiency of traditional ones. Further more, too intensive mathematical operations of the kernel seem to relate to the loss of precision and therefore numerical warning.

To address this issue, techniques of reparameterization are applied and the workflow of kernels is redesigned to avoid matrix concatenation. The former improved the stability of the kernel when working with the optimizer, reducing significantly the cases of non-P.S.D. error. The latter improved the time consumption for calculation and therefore made further test on these kernels possible.

5.1.3 Kernel with feature scaling component

The research is inspired by the quadratic kernel given in paper [2], which uses a serial quadratic multiplication of 4 RBF kernels operating on different input sub-components. Yet for reasons unknown, it seems to perform similarly to regular RBF on the datasets in this research. This research mathematically proved is equivalent to a single RBF but with a weight term multiplied by the inputs. Additionally, it is also proved that the quadratic kernel is possible to fall back to regular RBF if the weight term is near all-one.

Based on these findings, this research further proposed a method that extends the weight term from 4 free components to free components at the number of input features. Experiments have shown this method improves the performance of K_{MS} on all tested datasets.

5.2 Limitations

There are also issues known to this research yet improvements or fixes are impossible or impractical due to all sorts of reasons. This section discusses the major ones them, which, if known earlier, given abundant time or access to more resources, could have benefitted the outcome of this research better.

5.2.1 Datasets

This research uses a small subset of GDSC, given and pre-processed in the paper [22]. The shortcomings of this is that all findings obtained by this research are subjected to a single drug with around 250 cancer cell lines. The experiments taken, the results obtained and the conclusion drawn are naturally limited to this dataset and the ubiquity of those are not proved and unknown in all other circumstances.

To include more datasets, however, would meanwhile multiply the workload of grid search and cross-validation and add to the extra need for an extra pipeline to pre-process a new dataset. Considering the time frame given for this research, these explorations are never put on the agenda.

5.2.2 Scattered point output

An essential feature of GP is, being equivalent to an infinite-variate Gaussian distribution, it's able to produce predictions of smooth curves.(function output) This is unfortunately not achieved in this research as the model structure proposed in paper [2] uses the number of outputs to represent number of dose concentrations. This utilizes the MOGP structure yet forbids the model from giving output over a continuous range.

In the paper[2], Piecewise Cubic Hermite Interpolating Polynomial is used to generate continuous curve prediction, which is a method that connects scattered point based on gradient information with cubic curves. Applying this would allow for continuous curve prediction.

5.3 Future work

5.3.1 LMC latent GPs

All LMC models in this research use the number of outputs as the number of latent GPs. which is the maximum allowed in GPytorch. Some preliminary experiments on the 8-shot dataset indicate more latent GPs seem to improve the performance but marginally. Before large-scale parameter search, further exploration on this matter was done and results suggest similar. Eventually, the cross-validation finds out most LMC models have relatively high fluctuation of performance. While being an extended model from MOGP, LMC doesn't seem to perform absolutely better. It's hard to conclude for this research alone whether this instability of performance may be correlated with the aggressive choice of the number of latent GPs. This research speculates LMC, compared to regular MOGP, may be better at adapting to varied domain input,(though still cannot effectively react to it) yet it is indeed interesting to study in detail how transfer kernel learning is affected by LMC with different hyperparameters.

Research [25] has proposed a method determining the number of latent GPs using a variational inference-based approach, that formulates model inference as an optimization problem compatible with federated settings. Other common optimization solutions can also be considered, such as the EM algorithm, Bayesian optimization, etc.

5.3.2 Transferable kernel with neuron networks

The $K_\alpha\beta$, proposed in paper [24], is not the core of the presentation but a kernel to compare to for another kernel, which introduced neuron-network into kernel function. The idea of K_{MS-AFS} also comes partially from this, where the weight vector applied to both of the inputs is essentially a linear layer, and the kernel function can be understood as some sort of activation function.

It would be interesting to, either try the neuron network kernel(referred as Trk_ω) or to extend the K_{MS-AFS} to multiple linear layers. In the former idea, the neuron network in Trk_ω does not entirely follow the common structure of neuron networks, which may bring challenges in implementation. The latter idea can also enable dimension reduction by creating a bottleneck in the neuron network, which is helpful as input matrix sparsity is also a primary challenge in the DRP field.

References

[1] W. Yang, J. Soares, P. Greninger, et al., “Genomics of Drug Sensitivity in Cancer (GDSC): a resource for therapeutic biomarker discovery in cancer cells,” Nucleic Acids Research, vol. 41, no. D1, pp. D955–D961, Nov. 2012, ISSN: 0305-1048. DOI: 10.1093/nar/gks1111. eprint: https://academic.oup.com/nar/article-pdf/41/D1/D955/3626591/gks1111.pdf. [Online]. Available: https://doi.org/10.1093/nar/gks1111 (cited on pp. 7, 9).

[2] D. Wang, J.-J. G. Gutierrez, E. Lau, et al., “Multi-output prediction of dose-response curves enable drug repositioning and biomarker discovery,” 2023 (cited on pp. 7, 9, 12, 13, 17, 21, 23, 36, 37).

[3] P. Wei, T. V. Vo, X. Qu, Y. S. Ong, and Z. Ma, “Transfer Kernel Learning for Multi-Source Transfer Gaussian Process Regression,” en, IEEE Transactions on Pattern Analysis and Machine Intelligence, pp. 1–15, 2022, ISSN: 0162-8828, 2160-9292, 1939-3539. DOI: 10.1109/TPAMI.2022.3184696. [Online]. Available: https://ieeexplore.ieee.org/document/9802749/ (visited on 08/19/2024) (cited on pp. 7, 18, 19, 21, 25).

[4] P. Wei, R. Sagarna, Y. Ke, Y.-S. Ong, and C.-K. Goh, “Source-target similarity modelings for multi-source transfer Gaussian process regression,” in Proceedings of the 34th International Conference on Machine Learning, D. Precup and Y. W. Teh, Eds., ser. Proceedings of Machine Learning Research, vol. 70, PMLR, Jun. 2017, pp. 3722–3731. [Online]. Available: https://proceedings.mlr.press/v70/wei17a.html (cited on pp. 7, 9, 10, 13, 17).

[5] K. Qiu, J. Lee, H. Kim, S. Yoon, and K. Kang, “Machine learning based anti-cancer drug response prediction and search for predictor genes using cancer cell line gene expression,” Genomics & informatics, vol. 19, no. 1, 2021 (cited on p. 9).

[6] X. Xie, F. Wang, G. Wang, W. Zhu, X. Du, and H. Wang, “Learning the cellular activity representation based on gene regulatory networks for prediction of tumor response to drugs,” Artificial Intelligence in Medicine, p. 102 864, 2024 (cited on p. 9).

[7] M. E. Abbasnejad, D. Ramachandram, and R. Mandava, “Optimizing kernel functions using transfer learning from unlabeled data,” in 2009 Second International Conference on Machine Vision, IEEE, 2009, pp. 111–117 (cited on p. 9).

[8] F. Zhuang, Z. Qi, K. Duan, et al., “A Comprehensive Survey on Transfer Learning,” en, Proceedings of the IEEE, vol. 109, no. 1, pp. 43–76, Jan. 2021, ISSN: 0018-9219, 1558-2256. DOI: 10.1109/JPROC.2020.3004555. [Online]. Available: https://ieeexplore.ieee.org/document/9134370/ (visited on 09/05/2024) (cited on p. 9).

[9] K. Weiss, T. M. Khoshgoftaar, and D. Wang, “A survey of transfer learning,” en, Journal of Big Data, vol. 3, no. 1, p. 9, Dec. 2016, ISSN: 2196-1115. DOI: 10.1186/s40537-016-0043-6. [Online]. Available: http://journalofbigdata.springeropen.com/articles/10.1186/s40537-016-0043-6 (visited on 09/05/2024) (cited on p. 9).

[10] P. Wei, T. V. Vo, X. Qu, Y. S. Ong, and Z. Ma, “Transfer kernel learning for multi-source transfer gaussian process regression,” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 45, no. 3, pp. 3862–3876, 2023. DOI: 10.1109/TPAMI.2022.3184696 (cited on pp. 9, 10, 13, 14, 17).

[11] E. Schulz, M. Speekenbrink, and A. Krause, “A tutorial on gaussian process regression: Modelling, exploring, and exploiting functions,” Journal of Mathematical Psychology, vol. 85, pp. 1–16, 2018 (cited on p. 10).

[12] O. Babak and C. V. Deutsch, “An intrinsic model of coregionalization that solves variance inflation in collocated cokriging,” Computers & geosciences, vol. 35, no. 3, pp. 603–614, 2009 (cited on p. 11).

[13] Y. W. Teh, M. Seeger, and M. I. Jordan, “Semiparametric Latent Factor Models,” en, (cited on p. 11).

[14] M. A. Alvarez, L. Rosasco, and N. D. Lawrence, Kernels for vector-valued functions: A review, 2012. arXiv: 1106.6251 [stat.ML] (cited on p. 11).

[15] M. Goulard and M. Voltz, “Linear coregionalization model: Tools for estimation and choice of cross-variogram matrix,” Mathematical Geology, vol. 24, pp. 269–286, 1992 (cited on p. 11).

[16] Z.-H. Zhou, Machine learning. Springer nature, 2021 (cited on p. 12).

[17] T. Hofmann, B. Schölkopf, and A. J. Smola, “Kernel methods in machine learning,” 2008 (cited on p. 13).

[18] B. Schölkopf and A. J. Smola, Learning with kernels: support vector machines, regularization, optimization, and beyond. MIT press, 2002 (cited on p. 13).

[19] E. V. Bonilla, K. Chai, and C. Williams, “Multi-task gaussian process prediction,” Advances in neural information processing systems, vol. 20, 2007 (cited on p. 14).

[20] R. Dürichen, M. A. Pimentel, L. Clifton, A. Schweikard, and D. A. Clifton, “Multitask gaussian processes for multivariate physiological time-series analysis,” IEEE Transactions on Biomedical Engineering, vol. 62, no. 1, pp. 314–322, 2014 (cited on p. 14).

[21] K. Chai, “Generalization errors and learning curves for regression with multi-task gaussian processes,” Advances in neural information processing systems, vol. 22, 2009 (cited on p. 14).

[22] D. Jacob and D. M. Alvarez, “University of Manchester School of Computer Science Project Report 2022-23,” en, (cited on pp. 15, 37).

[23] J. R. Gardner, G. Pleiss, D. Bindel, K. Q. Weinberger, and A. G. Wilson, GPyTorch: Blackbox Matrix-Matrix Gaussian Process Inference with GPU Acceleration, en, arXiv:1809.11165 [cs, stat], Jun. 2021. [Online]. Available: http://arxiv.org/abs/1809.11165 (visited on 09/06/2024) (cited on p. 16).

[24] P. Wei, Y. Ke, Y.-S. Ong, and Z. Ma, "Adaptive Transfer Kernel Learning for Transfer Gaussian Process Regression," en, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 45, no. 6, pp. 7142–7156, Jun. 2023, ISSN: 0162-8828, 2160-9292, 1939-3539. DOI: 10.1109/TPAMI.2022.3219121. [Online]. Available: https://ieeexplore.ieee.org/document/9937157/ (visited on 08/15/2024) (cited on pp. 20, 38).

[25] J. Gao and S. Chung, Federated automatic latent variable selection in multi-output gaussian processes, 2024. arXiv: 2407.16935 [stat.ML]. [Online]. Available: https://arxiv.org/abs/2407.16935 (cited on p. 38).