TRANSFER LEARNING IN PROCESS ENGINEERING

By

Manoj Chandra Machavolu, MEng student (Toronto)

Daniela Galatro, Assistant Professor, PhD (Toronto)

A project submitted in conformity with the requirements
for the degree of Master's in Engineering Chemical Engineering
Department of Chemical Engineering and Applied Chemistry
University of Toronto

Table of Contents

1 Transfer Learning Definition:

A technique of deep learning (and machine learning) called transfer learning involves moving knowledge from one model to another. By applying transfer learning, we can use all or a portion of a model that has already been trained for another task to complete a specific task. Let's take the hypothetical case where Model A has successfully been trained to solve source task Ta utilizing a sizable dataset Da. However, Model B cannot train well because the dataset Db for the target task Tb is small. We, therefore, employ a portion of Model A to forecast the outcomes of task Tb. This is termed transfer learning.[1]

Humans have always had the capacity to apply knowledge across tasks. We apply the knowledge we get while learning about one activity to solve related tasks in the same manner. We can transfer or use our knowledge across tasks more readily if they are closely related to one another. Up until now, classical machine learning and deep learning algorithms have been created to operate independently. These algorithms have been honed to complete particular jobs. As soon as the feature-space distribution changes, the models must be recreated from scratch. The concept of transfer learning is to get past the isolated learning paradigm and use the knowledge you've learned to address one problem to solve others that are similar. The situation where what has been learned in one setting is exploited to improve generalization in another setting is called transfer learning.[2]

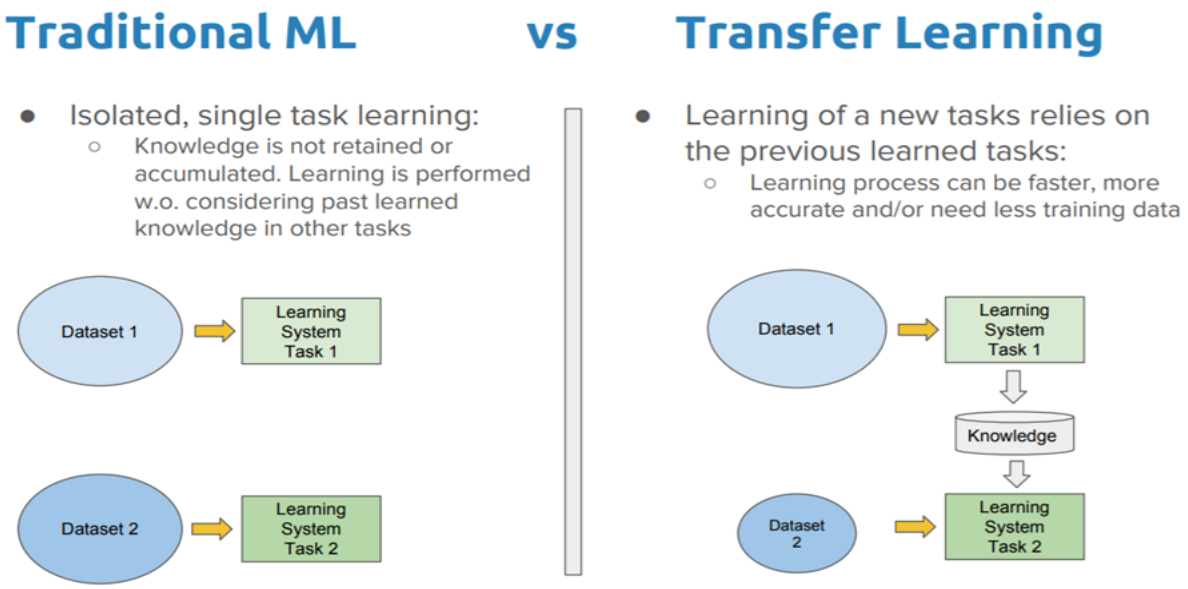

Figure 1 Comparison of Traditional ML with Transfer Learning

Transfer learning, used in machine learning, is the reuse of a pre-trained model on a new problem. In transfer learning, a machine exploits the knowledge gained from a previous task to improve generalization about another. Most researchers, developers and companies that work on using Deep Learning for their projects and products in specific domains and uses are often faced with 2 common challenges [3]. They are:

a) Insufficient Data: For most custom use cases, it is hard to find thousands of training data needed to create a deep learning model that will achieve desired accuracy in practical applications.

b) Cost of Training: Using powerful compute resources like NVIDIA GPUs to train can be very expensive due to the need to train for hours or days depending on tasks involved.

These challenges are solved by implementing transfer learning in training deep learning models. Transfer learning is the process of employing a model that has already been trained (pre-trained) to help train a new model to execute a new task. Some of the advantages of transfer learning are mentioned below:

a) Training very accurate models with limited training data.

b) Reducing training cost and time

2 Applications of Transfer Learning

Transfer learning techniques have been mainly applied to small scale applications with a limited variety, such as sensor-network-based localization, text classification, and image classification problems. Transfer learning methods will be heavily utilized in the future to address various complex applications, including video categorization, social network analysis, and logical reasoning.[4] Xiang Yu et al. [5] performed comprehensive survey of transfer learning on medical image analysis.

Liu Yang et al. [6] further explored that transfer learning can benefit a wide range of real-world applications, including computer vision, natural language processing, cognitive science (e.g., fMRI brain state classification), and speech recognition. Ling Shao [7] addresses the application of transfer learning to visual categorization and efficiently solved view divergence in action recognition tasks and concept drifting in image classification tasks.

Chenjing Cai et al. [8] provides applications of transfer learning applied to drug discovery and adopts deep transfer learning technique to leverage knowledge from related tasks. The research

paper provides outlooks on the future development of transfer learning for drug discovery and provides an overview of deep transfer learning network.

3 Transfer Learning under High-Dimensional Generalized Linear Models

3.1 Introduction:

A transfer learning algorithm is selected and used to predict corrosion rates in chemical process industries, thereby initiating studies of transfer learning in chemical engineering domain. The aim of the algorithm is to enhance the fit on target data by taking information from useful source data.[9] Sometimes the training data can be limited which prevents most of the machine learning models to perform well. However, in many cases some related datasets may be available in addition to the limited data for the original task. How to use these additional data to help with the original target task motivates the concept of transfer learning. Ye-Tian and Yang Feng aimed to transfer some useful information from similar tasks (sources) to the original task (target), in order to boost the performance on the target. We use the same concept here to predict corrosion rates using the methyldietanolamine (MDEA) data where target is the dataset of interest which is smaller in size. The source data can be any related dataset that has similar characteristics to the target MDEA dataset. However, for convenience and due to data shortage, we use MDEA dataset as both target and source i.e., the data is divided into required number of partitions. A generalised linear model (GLM)-based transfer learning algorithm is used in this work, and compare its L1/L2-estimation error bounds and constraint for a prediction error measure. Theoretical research demonstrates that these constraints could be superior than those of the traditional penalized estimator using only target data under moderate conditions where the target and source are sufficiently close to one another. [9] The detection consistency is checked based on the high-dimensional GLM transfer learning setting. Thus, which sources to transfer is estimated by calculating the error bounds. The sources with least error bounds are transferred to fit the target data.

Finally, after proposing a transfer learning algorithm, we fit the model and make predictions on the held out test dataset to determine the corrosion rates. In total, four models are developed (3 transfer learning models, 1 naïve LASSO regression model) and we compare different models

using error metrics such as Mean Squared Error (MSE) and Mean Absolute Error (MAE). The feasible model obtained has a lesser error metric value and is chosen as the best model to predict the corrosion rates. We implement the GLM transfer learning algorithm which is already implemented in R using the glmtrans package (available on CRAN) to our dataset.

3.2 Literature Review:

There are a few studies exploring transfer learning under the high-dimensional setting. Bastani (2021) [10] researched the single-source scenario in which the source data size is sufficiently larger than the dimension and the target data originates from a high-dimensional GLM with a small sample size. A two-step transfer learning algorithm was developed, and the estimation error bound was derived when the contrast between target and source coefficients is L_o - sparse. Li et al (2021) [11] looked into the multi-source high-dimensional linear regression problem in more detail. In this case, the source and target samples are both very high dimensional. The L_2 - estimation error bound under L_q - regularization ($q \in [0; 1]$) was derived and proved to be minimax optimal under some conditions. The analysis was expanded to include Gaussian graphical models with false discovery rate control in Li et al. (2020) [12]. The non-parametric classification model [13] , [14] and the analysis under generic functional classes via transfer exponents [15] are two other studies on transfer learning with theoretical assurance. Additionally, over the past few years, various relevant studies on parameter sharing in the context of regression have been conducted. In order to study the attributes of the oracle tuned estimator with a quadratic penalty, Chen et al [16] and Zheng et al [17] devised the so-called "data enriched model" for linear and logistic regression in a single-source context. Gross et al [18] and Ollier et al [19] investigated the so-called "data sharing Lasso" in the context of multi-task learning, where all contrasts L_1 penalties are taken into account. In this work, we offer three views that contribute to transfer learning in a high dimensional situation. First, we apply the results and algorithms of Ye Tian and Yang Feng (2022) i.e., applying multi-source transfer learning algorithms on Industrial plant data to predict the corrosion rates. Second, we extend the results of Bastani (2021) and Li et al (2021), by proposing multi-source transfer learning algorithms on generalized linear models (GLMs) and we assume both target and source to be high dimensional. We assume the contrast between target and source coefficients to be L_1 - sparse, which differs from the L_o – sparsity considered in Bastani (2021). According to the theoretical study, under favourable conditions, the estimate error bound of target coefficients could be better than that of the traditional penalized estimator when the target and source are sufficiently

close to one another. Moreover, the error rate is shown to be minimax optimal under certain conditions. To the best of my knowledge, this is the first study of multisource transfer learning under high-dimensional setting applied to chemical engineering. Third, transferring sources that are close to the target can bring benefits. However, some sources might be far away from the target, and transferring them can be harmful. This phenomenon is often called negative transfer learning in literature Torrey et al [20]. To avoid this issue we develop an algorithm-free transferable source detection algorithm, which can help identify informative sources (Ye-Tian and Yang Feng (2022)). With certain conditions satisfied, the algorithm is shown to be able to distinguish useful sources from useless ones.

3.3 Case Study 1 Transfer learning under high-dimensional GLM's using R programming:

3.3.1 Objective:

The aim of this study is to propose a strong foundation for predicting corrosion rates in amine gas treating units using transfer learning algorithms written in R programming language. The reason for predicting the corrosion rates is to evaluate the lifetime or duration, equipment performance without showing any depletion in the efficiency of the process. Corrosion is considered a vital aspect in chemical process industries, and optimal corrosion management practices can help in maximizing the efficiency, ensuring safe and environmentally compliant operations and reducing costs. The data for analysis is obtained from either industries or through process simulation software.

3.3.2 Process Description:

Amine gas treatment process is mainly carried out to remove impurities such as carbon dioxide (CO_2) and hydrogen sulfide (H_2S) from gas process streams using solvents. The list of solvents used for this process may include primary, secondary and tertiary amines, mixed amines and formulated amines. The equiment used for this process are an absorber, supporting equipments and a regenerator. To give a brief working of the process, the lean solvent removes the acid gases when the gas stream passes through the absorber. In the regenerator, the rich solvent stream leaving the absorber from the bottom is entered and the acid gases such as H_2S, CO_2 are removed and sent to either a flare or sulfur recovery unit.

3.3.3 Dataset:

The solvent selected for removing the acid gas is methyldiethanolamine (MDEA) which is a tertiary amine. The dataset consists of several factors that influence the corrosion rate, namely acid gas loading, heat-stable amine salts (HSAS), temperature, and velocity. Since this is a regression problem, the column of interest is the corrosion rate.

3.4 Algorithm:

The MDEA dataset is divided into three sets which are target, source and test. The test dataset is set aside for corrosion rate prediction and is used in the end. Since this is a multi-source transfer learning problem, the source dataset is further divided into five sets. The target data is fitted by borrowing useful information from source dataset. The dataset is fitted on all the 4 models and predicted on test dataset to estimate the corrosion rate. The best model is concluded based on evaluating the MSE and MAE. The model having the least error metric value is finalized for predicting the corrosion rates.

The mathematics and explanation behind the transfer learning logic is detailed as follows:

(The transfer learning algorithm/syntax is referred from "A demonstration of GLM trans package" by Ye-Tian and Yang Feng (2022)). The link can be found here.

Generalized linear models (GLMs):

Given the predictor x, if response y follows the generalized linear models (GLMs), then its distribution satisfies

$y|\boldsymbol{x} \sim \mathrm{P}(y|\boldsymbol{x}) = \rho(y)\exp\{y\boldsymbol{x}^T\boldsymbol{w} - \psi(\boldsymbol{x}^T\boldsymbol{w})\}$, where $\psi'(\boldsymbol{x}^T\boldsymbol{w}) = \mathrm{E}(y|\boldsymbol{x})$ is called the *inverse link function* [21]. Another important property is that $\mathrm{Var}(y|\boldsymbol{x}) = \psi''(\boldsymbol{x}^T\boldsymbol{w})$, which is derived from the exponential family property. It is ψ which characterizes different GLMs. For example, in Gaussian model, we have the continuous response y and $\psi(u) = 1/2u^2$; in the logistic model, y is binary and $\psi(u) = \log(1+ e^u)$; and in Poisson model, we have the integral response y and $\psi(u) = e^u$.

Two-step transfer learning algorithms:

Consider the multi-source transfer learning problem. Suppose we have the *target* data $(X^{(0)},Y^{(0)}) = \{\boldsymbol{x}_i^{(0)},y_i^{(0)}\}_{i=1}^{n_0}$ and *source* data $\{(X^{(k)},Y^{(k)})\}_{k=1}^{K} = \{\{(\boldsymbol{x}_i^{(k)},y_i^{(k)})\}_{i=1}^{n_k}\}_{k=1}^{K}$ for $k = 1,...,K$. Denote the target coefficient $\boldsymbol{\beta} = \boldsymbol{w}^{(0)}$. Suppose target and source data follow the GLM as

$Y^{(k)}|\boldsymbol{x}^{(k)} \sim \mathrm{P}(y^{(k)}|\boldsymbol{x}^{(k)}) = \rho(y^{(k)})\exp\{y^{(k)}(\boldsymbol{x}^{(k)})^T\boldsymbol{w}^{(k)} - \psi((\boldsymbol{x}^{(k)})T\boldsymbol{w}^{(k)}\}$. To borrow information from transferable sources, Bastani (2020) and Li, Cai, and Li (2020) developed *two-step transfer learning algorithms* for high-dimensional linear models. In the first step, an approximate estimator

is achieved via the information from the target data and useful source data. In the second step, the target data is used to de-bias the estimator obtained from the first step, leading to the final estimator. Tian and Feng (2021) [22] extend the idea into GLM and propose the corresponding *oracle* algorithm, which can be easily applied when transferable sources are known. It is proved to enjoy a sharper bound of L_2-estimation error when the transferable source and target data are sufficiently similar.

Transferable source detection:

The transferable source detection algorithm is run automatically in glmtrans once transfor.source.id is set to "auto". For the algorithm of source_detection refer the documentation of Ye-Tian and Yang Feng's "Package Glmtrans". Sometimes the source datasets may share similarities with the target dataset and this can mislead the fitted model. This phenomenon is called negative transfer (Pan and Yang (2009), Torrey and Shavlik (2010), Weiss, Khoshgoftaar, and Wang (2016)) [23].

To detect the transferable sources, Tian and Feng (2021) developed an *algorithm-free* detection approach. It computes the gain for transferring each source and compares it with the baseline where only the target data is used. The sources having significant performance gain compared with the baseline are regarded as transferable ones. Tian and Feng (2021) also proved the detection consistency property for this method under the high-dimensional GLM setting.

Implementation:

The package used for the transfer learning code is glmtrans and glmnet which can be found in CRAN website. We use the argument offset provided by the function glmnet and cv.glmnet to implement our two-step algorithms. Besides Lasso [24] , this package can adapt the elastic net type penalty [25].

Installation:

glmtrans is now available on CRAN and can be easily installed by one-line code. install.packages("glmtrans", repos = "http://cran.us.r-project.org"). Then we can load the package: library(glmtrans)

glmtrans usage:

glmtrans(

target,

source = NULL,

family = c("gaussian", "binomial", "poisson"),

transfer.source.id = "auto",

alpha = 1,

standardize = TRUE,

intercept = TRUE,

nfolds = 10,

cores = 1,

valid.proportion = NULL,

valid.nfolds = 3,

lambda = c(transfer = "lambda.1se", debias = "lambda.min", detection = "lambda.1se"),

detection.info = TRUE,

target.weights = NULL,

source.weights = NULL,

C0 = 2,

...

)

Arguments for Glmtrans function:

target	target data. Should be a list with elements x and y, where x indicates a predictor matrix with each row/column as a(n) observation/variable, and y indicates the response vector.
source	source data. Should be a list with some sublists, where each of the sublist is a source data set, having elements x and y with the same meaning as in target data.
family	response type. Can be "gaussian", "binomial" or "poisson". Default = "gaussian". • "gaussian": Gaussian distribution. • "binomial": logistic distribution. When family = "binomial", the input response in both target and source should be 0/1. • "poisson": poisson distribution. When family = "poisson", the input response

	in both target and source should be non-negative.
transfer.source.id	transferable source indices. Can be either a subset of {1,...,length(source)}, "all" or "auto". Default = "auto". • a subset of {1,...,length(source)}: only transfer sources with the specific indices. • "all": transfer all sources. • "auto": run transferable source detection algorithm to automatically detect which sources to transfer. For the algorithm, refer to the documentation of function source_detection.
alpha	the elasticnet mixing parameter, with $0 \leq \alpha \leq 1$. The penalty is defined as $(1-\alpha)/2\|\|\beta\|\|^2_2+\alpha\|\|\beta\|\|_1$ alpha = 1 encodes the lasso penalty while alpha = 0 encodes the ridge penalty. Default = 1.
standardize	the logical flag for x variable standardization, prior to fitting the model sequence. The coefficients are always returned on the original scale. Default is TRUE.
intercept	the logical indicator of whether the intercept should be fitted or not. Default = TRUE.
nfolds	the number of folds. Used in the cross-validation for GLM elastic net fitting procedure. Default = 10. Smallest value allowable is nfolds = 3.
Cores	the number of cores used for parallel computing. Default = 1
Valid.proportion	the proportion of target data to be used as validation data when detecting transferable sources. Useful only when transfer.source.id = "auto". Default = NULL, meaning that the cross-validation will be applied.

Valid.nfolds	the number of folds used in cross-validation procedure when detecting transferable sources. Useful only when transfer.source.id = "auto" and valid.proportion = NULL. Default = 3.
Lambda	a vector indicating the choice of lambdas in transferring, debiasing and detection steps. Should be a vector with names "transfer", "debias", and "detection", each component of which can be either "lambda.min" or "lambda.1se". Component transfer is the lambda (the penalty parameter) used in transferrring step. Component debias is the lambda used in debiasing step. Component detection is the lambda used in the transferable source detection algorithm. Default choice of lambda.transfer and lambda.detection are "lambda.1se", while default lambda.debias = "lambda.min". If the user wants to change the default setting, input a vector with corresponding lambda.transfer/lambda.debias/lambda.detection
Detection.info	the logistic flag indicating whether to print detection information or not. Useful only when transfer.source.id = "auto". Default = TURE
Target.weights	weight vector for each target instance. Should be a vector with the same length of target response. Default = NULL, which makes all instances equal-weighted
Source.weights	a list of weight vectors for the instances from each source. Should be a list with the same length of the number of sources. Default = NULL, which makes all instances equal-weighted.

Example of glmtrans:

```
# fit a linear regression model
D.training <- models("gaussian", type = "all", n.target = 100, K = 2, p = 500)
D.test <- models("gaussian", type = "target", n.target = 100, p = 500)
fit.gaussian <- glmtrans(D.training$target, D.training$source)
y.pred.glmtrans <- predict(fit.gaussian, D.test$target$x)
# compare the test MSE with classical Lasso fitted on target data
library(glmnet)
fit.lasso <- cv.glmnet(x = D.training$target$x, y = D.training$target$y)
y.pred.lasso <- predict(fit.lasso, D.test$target$x)
mean((y.pred.glmtrans - D.test$target$y)^2)
mean((y.pred.lasso - D.test$target$y)^2)
```

models usage:

```
models(
family = c("gaussian", "binomial", "poisson"),
type = c("all", "source", "target"),
cov.type = 1,
h = 5,
K = 5,
n.target = 200,
n.source = rep(100, K),
s = 5,
p = 500,
Ka = K
)
```

Arguments for models function:

family	response type. Can be "gaussian", "binomial" or "poisson". Default = "gaussian". • "gaussian": Gaussian distribution. • "binomial": logistic distribution. When family = "binomial", the input response in both target and source should be 0/1. • "poisson": poisson distribution. When family = "poisson", the input response in both target and source should be non-negative
type	the type of generated data. Can be "all", "source" or "target".
cov.type	the type of covariates. Can be 1 or 2 (numerical). If it equals to 1, the predictors will be generated from the distribution used in Section 4.1.1 (Ah-Trans-GLM) in the latest version of Tian, Y. and Feng, Y., 2021. If it equals to 2, the predictors will be generated from the distribution used in Section 4.1.2 (When transferable sources are unknown). • "all": generate a list with a target data set of size n.target and K source data set of size n.source. • "source": generate a list with K source data set of size n.source. • "target": generate a list with a target data set of size n.target.
h	measures the deviation (l1-norm) of transferable source coefficient from the target coefficient.
K	the number of source data sets. Default = 5.
n.target	the sample size of target data. Should be a positive integer. Default = 100.
n.source	the sample size of each source data. Should be a vector of length K. Default is a K-vector with all elements 150.
s	how many components in the target coefficient are non-zero, which controls the sparsity of target problem. Default = 15.
p	the dimension of data. Default = 1000.

Ka	the number of transferable sources. Should be an integer between 0 and K. Default = K.

Transfer Learning models with syntax:

1) Orcale-Trans-GLM:

Let 'A' be the number of sources to be transferred. Then suppose we know A, let's fit an "oracle" GLM transfer learning model on the target data and source data in A by the oracle algorithm. We denote this procedure as Oralce-Trans-GLM.

```
D.training <- models(family = "binomial", type = "all", cov.type = 2, Ka = 3,
K = 5, s = 10, n.target = 100, n.source = rep(100, 5))
fit.oracle <- glmtrans(target = D.training$target, source = D.training$source,
family = "binomial", transfer.source.id = 1:3, cores = 2)
```

Notice that we set the argument transfer.source.id equal to A = {1, 2, 3} to transfer only the first three sources.

2) Trans-GLM:

Then suppose we do not know A, let's set transfer.source.id = "auto" to apply the transferable source detection algorithm to get estimate A . After that, glmtrans will automatically run the oracle algorithm on A to fit the model. We denote the approach as Trans-GLM.

```
fit.detection <- glmtrans(target = D.training$target, source = D.training$source,
family = "binomial", transfer.source.id = "auto", cores = 2)
```

Output

```
## Loss difference between source data and the threshold: (negative to be transferable)
## Source 1: -0.046214
## Source 2: -0.064628
## Source 3: -0.105844
## Source 4: 0.113911
## Source 5: 0.211745
```

Source data set(s) 1, 2, 3 are transferable

From the results, we could see that A = {1, 2, 3} is successfully detected via the detection algorithm. In some cases when all the loss difference values are negative, then the sources with high negative values are transferred like the case in predicting corrosion rates.

3) Lasso & Pooled-Trans-GLM

Next, to demonstrate the effectiveness of GLM transfer learning algorithm and the transferable source detection algorithm, we also fit the naive Lasso on target data (Lasso) and transfer learning model using all source data (Pooled-Trans-GLM) as baselines.

```
library(glmnet)
fit.lasso <- cv.glmnet(x = D.training$target$x, y = D.training$target$y,
family = "binomial")
fit.pooled <- glmtrans(target = D.training$target, source = D.training$source,
family = "binomial", transfer.source.id = "all", cores = 2)
```

Finally, we compare the different models using error metrics such as the mean squared error (MSE) and mean absolute error (MAE) to determine which model is suitable for predicting corrosion rates.

3.5 Implementation:

Code for predicting corrosion rates using R programming with glmtrans, glmnet packages:

```
install.packages("glmtrans", repos = "http://cran.us.r-project.org")
library(glmtrans)
install.packages("readxl")
library("readxl")
D.training <- read_excel(file.choose())
n = 100
target_train <- D.training[row.names(D.training) %in% 1:n, ]
target_source <- D.training[row.names(D.training) %in% (n+1):(6*n), ]
target_test <- D.training[row.names(D.training) %in% (6*n+1):nrow(D.training), ]
target_source_new_1 <- target_source[row.names(target_source) %in% 1:100, ]
target_source_new_2 <- target_source[row.names(target_source) %in% 101:200, ]
target_source_new_3 <- target_source[row.names(target_source) %in% 201:300, ]
target_source_new_4 <- target_source[row.names(target_source) %in% 301:400, ]
target_source_new_5 <- target_source[row.names(target_source) %in% 401:500, ]
```

```
train_target_x <- matrix(c(target_train$'Acid gas', target_train$HSAS, target_train$Velocity, target_train$Temperature), nrow=100)
train_target_y <- matrix(c(target_train$'Corrosion Rate'), nrow=100)
target_source_new_1_x<-matrix(c(target_source_new_1$'Acid gas',
target_source_new_1$HSAS, target_source_new_1$Velocity,
target_source_new_1$Temperature), nrow=100)
target_source_new_1_y <- matrix(c(target_source_new_1$'Corrosion Rate'), nrow=100)
target_source_new_2_x<-matrix(c(target_source_new_2$'Acid gas',
target_source_new_2$HSAS, target_source_new_2$Velocity,
target_source_new_2$Temperature), nrow=100)
target_source_new_2_y <- matrix(c(target_source_new_2$'Corrosion Rate'), nrow=100)
target_source_new_3_x <- matrix(c(target_source_new_3$'Acid gas',
target_source_new_3$HSAS, target_source_new_3$Velocity,
target_source_new_3$Temperature), nrow=100)
target_source_new_3_y <- matrix(c(target_source_new_3$'Corrosion Rate'), nrow=100)
target_source_new_4_x <- matrix(c(target_source_new_4$'Acid gas',
target_source_new_4$HSAS, target_source_new_4$Velocity,
target_source_new_4$Temperature), nrow=100)
target_source_new_4_y <- matrix(c(target_source_new_4$'Corrosion Rate'), nrow=100)
target_source_new_5_x <- matrix(c(target_source_new_5$'Acid gas',
target_source_new_5$HSAS, target_source_new_5$Velocity,
target_source_new_5$Temperature), nrow=100)
target_source_new_5_y <- matrix(c(target_source_new_5$'Corrosion Rate'), nrow=100)
D.training <- list(target=list(x = train_target_x, y = train_target_y), source=list(list(x1 = target_source_new_1_x, y1 = target_source_new_1_y),list(x2 = target_source_new_2_x, y2 = target_source_new_2_y),list(x3 = target_source_new_3_x, y3 = target_source_new_3_y),list(x4 = target_source_new_4_x, y4 = target_source_new_4_y),list(x5 = target_source_new_5_x, y5 = target_source_new_5_y)))
fit.gaussian <- glmtrans(D.training$target, D.training$source)
test_target_x <- matrix(c(target_test$'Acid gas', target_test$HSAS, target_test$Velocity, target_test$Temperature), nrow=156)
```

```
test_target_y <- matrix(c(target_test$'Corrosion Rate'), nrow=156)
D.test <- list(target=list(x = test_target_x, y = test_target_y))
y.pred.glmtrans <- predict(fit.gaussian, D.test$target$x)
mean((y.pred.glmtrans - D.test$target$y)^2)
library(glmnet)
fit.oracle <- glmtrans(target = D.training$target, source = D.training$source, transfer.source.id = 1:2, cores=2)
fit.detection <- glmtrans(target = D.training$target, source = D.training$source, transfer.source.id = "auto", cores=2)
fit.lasso <- cv.glmnet(x = D.training$target$x, y = D.training$target$y)
fit.pooled <- glmtrans(target = D.training$target, source = D.training$source, transfer.source.id = "all", cores=2)
y.pred.oracle <- predict(fit.oracle, D.test$target$x)
y.pred.detection <- predict(fit.detection, D.test$target$x)
y.pred.lasso <- predict(fit.lasso, D.test$target$x)
y.pred.pooled <- predict(fit.pooled, D.test$target$x)
mean((y.pred.oracle - D.test$target$y)^2)
mean((y.pred.detection - D.test$target$y)^2)
mean((y.pred.lasso - D.test$target$y)^2)
mean((y.pred.pooled - D.test$target$y)^2)
install.packages("Metrics", repos = "http://cran.us.r-project.org")
library(Metrics)
mae(D.test$target$y, y.pred.oracle)
mae(D.test$target$y, y.pred.detection)
mae(D.test$target$y, y.pred.lasso)
mae(D.test$target$y, y.pred.pooled)
```

The transfer learning code in R programming file format can be found here.

3.6 Results:

```
> fit.detection <- glmtrans(target = D.training$target, source = D.training$$
Loss difference between source data and the threshold: (negative to be trans$
Source 1: -0.093193
Source 2: -0.127194
Source 3: -0.026111
Source 4: -0.034823
Source 5: -0.052484
```

Mean Squared Error (MSE) metrics for the models: Output shown from R code

```
> mean((y.pred.oracle - D.test$target$y)^2)
[1] 0.7321382
> mean((y.pred.detection - D.test$target$y)^2)
[1] 0.8107981
> mean((y.pred.lasso - D.test$target$y)^2)
[1] 1.335645
> mean((y.pred.pooled - D.test$target$y)^2)
[1] 0.7979572
```

Model Name	**MSE Value**
Oracle-Trans-GLM	0.7321
Trans-GLM	0.8108
Lasso	1.3356
Pooled-Trans-GLM	0.798

Mean Absolute Error (MAE) metrics for the models:

```
> mae(D.test$target$y, y.pred.oracle)
[1] 0.6627138
> mae(D.test$target$y, y.pred.detection)
[1] 0.6871994
> mae(D.test$target$y, y.pred.lasso)
[1] 0.8612621
> mae(D.test$target$y, y.pred.pooled)
[1] 0.6827152
```

Model Name	**MAE Value**
Oracle-Trans-GLM	0.6627
Trans-GLM	0.6872
Lasso	0.8613
Pooled-Trans-GLM	0.6827

From both the MSE and MAE metrics, we can verify that **Oracle-Trans-GLM** transfer learning model has the least value and shows better corrosion rate prediction capabilities compared to other models. In this model, we transfer the first two sources, since it is highly negative. We can also conclude that all the transfer learning models outperform the classical LASSO fitted on target data. Due to negative transfer, there is a dip in the performance of Pooled-Trans-GLM than Oracle-Trans-GLM. The values of the corrosion rates for every transfer learning model can be viewed by printing their corresponding y values namely y.pred.oracle, y.pred.detection, y.pred.lasso, y.pred.pooled.

4 Modular Gaussian Process for Transfer Learning

4.1 Introduction:

We modified a transfer learning code based on modular variational gaussian processes to predict corrosion rates. Similar to the first case study, we estimate the model using the metrics, Mean Absolute Error (MAE), Root Mean Squared Error (RMSE) and Negative Log Predictive Density (NLPD). A dictionary of well-fitted GP's are developed from which ensemble GP models are built without revisiting any data. Every model has its own characteristics such as hyperparamerers, pseudo-inputs and their corresponding posterior densities. This method avoids data centralization, reduces computational costs and allows the transfer of learned uncertainty metrics after training. [26]

Basic Introduction to Gausssian Process:

In regression analysis, given a set of observed data points, we want to fit a function to represent these data points. This function can then be used to make predictions at new data points. From a set of observed data, there can be infinite possible functions that fit these data points. In Gaussian Process Regression (GPR), the Gaussian processes conduct regression by defining a distribution over these infinite number of functions.

The Gaussian process model is a probabilistic supervised machine learning framework that has been widely used for regression and classification tasks [27]. A GPR model can make predictions incorporating prior knowledge (kernels) and provide uncertainty measures over predictions [28].

Some important technical terms used in gaussian processes are prior, likelihood and posterior distribution.

Prior refers to the probability distribution representing knowledge or uncertainty of a data object prior or before observing it.

Likelihood means probability of falling under a specific category of class.

Posterior is the conditional probability distribution representing what parameters are likely after observing the data object. Prior probability represents what is originally believed before new evidence is introduced, and posterior probability takes this new information into account.

In simple steps:

1) The Gaussian Process model defines the Gaussian Process prior and the likelihood, and explains the model parameters in the prior and the likelihood.

2) Posterior is obtained by taking into account the prior and likelihood and describes how to make predictions using the posterior.

3) The Parameter learning finds optimal concrete values for the model parameters. The posterior calculated is a symbolic expression that mentions model parameters. After we found concrete values for model parameters, we can evaluate the posterior into a concrete number [29].

The GP prior:

The Gaussian process prior is a multivariate Gaussian distribution over random variable vectors f(X) and f(X*). Where f(X) is a random variable vector of length n and represents possible values of the regression function at training locations X. Similarly, f(X*) is a random variable vector of length n^* and represents possible values of the regression function at testing locations X^*.

This multivariate Gaussian distribution has zero mean and uses kernel function 'k' to estimate the covariance matrix [30]. We use squared exponential function for k.

$$k(x, x') = \sigma^2 \exp(-\frac{(x - x')^2}{2l^2})$$

(1)

Where, σ signal variance and l length scale are model parameters.

The Likelihood:

It is represented as p(y). It originates from multivariate Gaussian distribution with mean f(X), and covariance $\eta^2 I_n$, where η^2 is noise variance (model parameter) and I_n is nxn identity matrix since we assume independent observation noise. Training data Y is modeled as samples of the random variable vector y(X). Since y(X) depends on f(X), we also denote the likelihood as p(y(X)|f(X)). For convenience, y(X) and f(X) are considered as y and f respectively.

As per research, since both f and y are multivariate Gaussian random variables, and f is the mean of y, we can rewrite y as a linear transformation from f:

$$y = f + \in \quad where \ \in \sim N(0, \eta^2 I_n)$$

(2)

The probability density function for y is given by:

$$p(y) = \frac{1}{(2\Pi)^{n/2} \det(K + \eta^2 I_n)^{\frac{1}{2}}} \exp(-\frac{1}{2} y^T (K + \eta^2 I_n)^{-1} y)$$

(3)

Where p(y) is the marginal likelihood.

Parameter Learning:

Now the GPR model has three parameters namely, length scale l, signal variance σ^2, and observation variance η^2 . We use parameter learning to estimate values for model parameters. We need to find optimal values which the model can best explain the training data. To estimate optimal values, parameter learning uses gradient descent algorithm to maximize the objective function log(p(y)) with respect to model parameters.

The objective function is given by:

$$\log\big(p(y)\big) = -\frac{1}{2}\log(\det(K+\eta^2 I_n)) - \frac{1}{2}y^T(K+\eta^2 I_n)^{-1}y - \frac{n}{2}\log(2\Pi) \quad (4)$$

Once the optimal values for model parameters are determined, predictions are made using the posterior distribution. The posterior distribution is given by $p(f_*|y)$:

$$p(f_*|y) = N(K_{*X}(K+\eta^2 I_n)^{-1}y, K_{**} - K_{*X}(K+\eta^2 I_n)^{-1}K_{*X}^T \quad (5)$$

For full derivation of posterior distribution, refer [33]

In GPR model, both the objective function for parameter learning and the posterior distribution contains the matrix inversion term $(K+\eta^2 I_n)^{-1}$ which is computationally expensive.

One solution to the above problem is to summarize the training data which means choosing less data points from the dataset that summarizes the entire training data.

Sparse and Variational Gaussian Process model (SVGP):

Let $f(X_s)$ be a new set of random variables at location X_s. X_s is a vector of scalars of length n_s. Subscript '$_s$' stands for sparse. In order to summarize training data, n_s should be lesser than n. Let X_s be the inducing locations and f_s , inducing random variables.

SVGP prior:

In SVGP prior, we use f_s at inducing locations X_s to explain f at training locations X. The covariance matrix K_{XS} defines the correlation between f and f_s. The kernel function k defines each entry inside K_{XS}.

In GPR prior, we use f at training locations X to explain f_* at testing locations X_*. The covariance K_{*X} defines the correlation between f_* and f. The same kernel function k defines each entry of K_{*X}.

Significance of Inducing variables:

The main difference between GPR and SVGP prior is that, SVGP summarizes the training data using inducing variables. To summarise is to succinctly state the most crucial details about anything. In SVGP, the number of inducing variables n_s must be less than the number of training data points n. Due to this, we refer to the SVGP model as sparse. We want to use a small number of inducing variables at pivotal inducing locations to explain a much larger number of random variables at training locations. The value of number of inducing locations is initialized and not a model parameter. However the location of inducing variables is a model parameter.

Evaluating the extent to which our model explains the training data:

Marginal likelihood p(y) is used to measure how well the inducing variables summarize the training data. The simplified final equation for p(y) using SVGP is given by:

$$p(y) = N(y; 0, K + \eta^2 I_n) \quad (6)$$

When deriving p(y), the SVGP prior defines p(y) as an expectation with regard to the random variables f and f_s . Therefore, using the weights p(f, f_s), p(y) represents the average likelihood of the data y while accounting for all potential values of f and f_s . We are stating that since f and f_s are random variables, they may take on various values with various probability. The likelihood p(y|f, f_s) determines the data probability for each different value of f and f_s . Therefore, we use the weighted sum of those likelihood probabilities, or p(y), to assess how well on average our model generates the training data.

From the above equation of p(y), we can infer that there are two problems associated which are:

1) The matrix inversion still persists, so therefore it is computationally expensive.

2) Only the lengthscale l, the signal variance σ^2, and the observation noise variance η^2 are mentioned in this formula. The inducing locations X_s, another model parameter, are not mentioned. All model parameters must be included in an objective function that is valid for parameter learning. As a result, this equation is an invalid objective function.

We need to consider an alternative method to compute the posterior without computing p(y) since using the Bayes rule to get the posterior entails computing the marginal likelihood p(y), and computing p(y) results in the problems listed above.

Variational Inference:

To solve the above problem, variational inference technique directly approximates the posterior $p(f, f_s|y)$ instead of using the Bayes rule. As a result, it does not require calculating the marginal likelihood p(y). To understand more about variational inference on a Gaussian process model, please refer [31]. In order to approach the correct posterior $p(f, f_s|y)$, variational inference employs a new distribution $q(f, f_s)$, also known as the variational distribution. The ELBO formula is provided to us by the variational inference method. We arrive to a $q(f, f_s)$ that closely resembles the genuine posterior $p(f, f_s|y)$ by maximising the ELBO with respect to the model parameters. Our model can handle both Gaussian and non-Gaussian likelihood because we are utilising the variational distribution q(f, fs) to estimate the posterior $p(f, f_s|y)$. This is why variational inference was added to the Gaussian Process. The variational distribution is given by:

$$q(f, f_s) = p(f|f_s)q(f_s) \tag{7}$$

Where, $p(f|f_s) = N(f; Af_s, B)$ with $A = K_{XS} K_{SS}^{-1}$ and $B = K - K_{XS} K_{SS}^{-1} K^T_{XS}$

For detailed steps of the derivation, refer the research paper mentioned before. It is significant to note that only the inverse of K_{SS}, a smaller n_s x n_s matrix, is mentioned in A and B. The inverse of K, which is a larger n x n matrix, is not mentioned. The size of the matrix K_{XS} in A and B is n x n_s, which can be big because n itself can be large. However, since we don't need to invert K_{XS}, its size is not an issue.

The factorization of the variational distribution $q(f, f_s)$ contains a component called $q(f_s)$. Because the joint $q(f, f_s)$ only specifies one of the two random variable vectors, we refer to it as a marginal variational distribution which is given by.

$$q(f_s) = N(f_s; \mu, \Sigma) \tag{8}$$

The ELBO

We want to maximise the marginal likelihood p(y) with regard to the model parameters because, it represents how well the SVGP model explains the training data. Typically, we maximise log p(y) to make computations easier. The variational inference technique uses the ELBO formula as an alternate maximisation objective function because log p(y) is challenging to compute (here to see why).

The final expression for ELBO is given as:

$$\log\big(p(y)\big) = \iint \log\big(p(y|f)\big)\, q(f, f_s) df df_s - KL(q(f, f_s)||p(f, f_s))$$

(9)

The first term in the equation represents the likelihood term and the second represents KL term. An analytical expression for both the terms are derived so that a gradient descent algorithm is used to compute the gradient of the analytical ELBO with respect to the model parameters.

The likelihood term:

The likelihood term is further simplified as $\int \log\big(p(y|f)\big)\, q(f) df$. The expression for q(f) is given by.

$$q(f) = N(f; A\mu, A\Sigma A^T + B)$$

(10)

Therefore the likelihhod term in the ELBO now becomes

Likelihood term = $\int \log\big(p(y|f)\big) . N(f; A\mu, A\Sigma A^T + B) df$

(11)

The KL term is further simplified as $KL(q(f_s)||p(f_s))$. For entire derivation, please refer the literature.

Parameter Learning:

The analytical expression for the ELBO is a function with all the model parameters as arguments. To estimate the best values for those model parameters, we maximise the ELBO using the gradient descent method.

The model parameters are listed as follows:

1) Legthscale l, from the kernel function (singular scalar)

2) Signal variance σ^2, from the kernel function (singular scalar)

3) Observation noise variance η^2 from the Gaussian likelihood (singular scalar)

4) The mean vector μ from the marginal variational distribution $q(f_s)$ has length n_s (n_s scalars)

5) The covariance vector Σ from the marginal variational distribution $q(f_s)$ ($1/2 n_s(n_s+1)$ scalars)

The number of inducing locations n_s is decided by us. More the number of inducing locations, better is the model in summarizing the training data. Use as many triggering variables as your budget, including time and memory, allows. This is a good rule of thumb. For instance, you can begin with fewer inciting places and gradually increase n_s to see if you can obtain a larger ELBO while staying within your means.

Stochastic Gradient Descent (SGD)

Gradient descent performs the following 3 tasks at every optimization step.

a) Uses the analytical expression of the ELBO to create partial gradient expressions for each trainable scalar. The resulting formulas for the analytical partial gradient are $3+1/2n_s(n_s+1)$. Gradient descent only ever computes these analytical formulas once in reality. Here, for the sake of comprehension, we presume that the gradient expressions are computed at each step.

b) In order to evaluate those analytical partial gradient expressions into concrete gradients, which is a float vector of length $3+1/2n_s(n_s+1)$, the training data (X, Y) and all current values for the $3+1/2n_s(n_s+1)$ scalars are then plugged in.

c) The values of the trainable scalars are then updated using this float gradient vector.

The task becomes expensive when the training data is large.

$$ELBO = \int \log\big(p(f|y)\big)\, q(f) df - \int \log\left(\frac{q(f_s)}{p(f_s)}\right) q(f_s) df_s$$

(12)

In the above equation, on the right side, the first term is likelihood and the second term is KL.

Both $q(f_s)$ and $p(f_s)$ in KL term does not mentions training data, $p(f_s)$ contains K_{SS} where K_{SS} mentions X_s not X. So the cost of computing the KL term is a function of the number of inducing variables n_s that we have complete control over. To put it another way, we can pick a n_s that is small enough to evaluate the gradient for the KL term inexpensive.

The likelihood term mentions both X and Y that makes the computation expensive. Log(p(f|y)) mentions Y through random variable y and q(f) mentions X through the matrix A and B.

A variation of the gradient descent algorithm is stochastic gradient descent (SGD). By merely calculating the gradient of the objective function evaluated on a portion of the training data at each optimization step, it avoids the issue of expensive gradient evaluation. The phrase stochastic gradient descent is used when this subset of training data contains a single data point. Mini-batch stochastic gradient descent is utilised when this subset comprises several training data points (the subset is referred to as a mini-batch).

Sometimes you can cut the time required for parameter learning from a day to just an hour by employing stochastic gradient descent.

The SGD does the following:

1) Sample a batch of m data points from the training set at random and uniformly, with replacement; m is the batch size.

2) Then proceed according to the original gradient descent approach, treating the batch like the entire training set of data.

The cost of the gradient evaluation at each optimization step can be adjusted by adjusting the batch size m. Gradient descent updates the values of the model parameters in a way that makes intuitive sense. Gradient descent uses gradients computed throughout the whole training set. On the other side, stochastic gradient descent uses stochastic gradients calculated on a portion of training data. Therefore, it is highly unlikely that a single stochastic gradient points in the same direction as the gradient as a whole. However, it is correct if, on average, the stochastic gradients (averaged across a number of batches) point in the same direction as the complete gradient does.

Making predictions:

The posterior distribution is used in a Bayesian model to make predictions. The predictive distribution $p(f_*|y)$ can be derived from a set of testing sites X_*. With $p(f_*|y)$, we can make predictions. A single value for l and σ^2 is discovered using parameter learning. This means that the multivariate Gaussian conditional correlation structure will be used by the model to explain:

1) Training data f (through the likelihood $p(y|f)$) at training location X from inducing variables f_s at inducing locations X_s.

2) f_* at testing locations X_* from inducing variables f_s at inducing locations X_s.

We presume that the training and testing sets of data were produced using the same methodology. If this supposition is incorrect, there won't be a model that can take lessons from the past and forecast the future. According to this presumption, the kernel function with the parameter values discovered by gradient descent should enable us to make reasonable predictions for f_* at testing locations at X_* from the same set of inducing variables if it can summarise the training data using inducing variables (in the high marginal likelihood p(y) sense, or alternatively, in the high ELBO sense). The SVGP model can thus make predictions because of this.

The predictive distribution $p(f_*|y)$ is given by

$$p(f_*|y) = \int p(f_*|f_s)q(f_s)df_s \qquad (13)$$

The predictive distribution only depends on the inducing variables f_s , according to the equation above, and is independent of the random variable f at training sites. The model no longer needs the training data after parameter learning. This demonstrates how accurately the inducing variables reflect the training data. Unlike the Gaussian Process regression model, which requires training data for predictions, this one does not.

4.2 Case Study 2 Transfer Learning under Modular Gaussian Process using Python:

4.2.1 Objective:

The aim of this study is to propose a strong foundation for predicting corrosion rates in amine gas treating units using transfer learning algorithms written in Python programming language. We create a module-based strategy that allows one to create ensemble GP models without revisiting any data if they have a dictionary of well-fit GPs.

4.2.2 Theory:

Learning algorithms are forced to develop new capabilities simply due to the availability of data being limited. Examples include spreading data for federated learning [32] , continuously observing streaming samples for constant learning [33] , and restricting data exchange for privately owned models [34]. The concept of model recycling, or applying the previously fitted parameters for another task without reviewing any data, is a recurring one. Uncertainty is more difficult to utilize than parameters of functions, according to the probabilistic interpretation of this concept. At this stage, Gaussian processes come into play.

Due to high computational costs of GPs as we discussed, most of the comtemporary works are focused on parallelizing inference. Parallelizing means to split the data into several partitions and processing each partitions parallelly on multiple devices. Each module is considered as a spare variational GP containing only variational parameters and hyperparameters. Our key contribution consists on maintaining such modules and creating meta-GPs without using any previous observations. This lowers the computational cost, making it competitive with large-dataset inference approaches. The augmentation of integrals, which may be explained in terms of the Kullback-Leibler (KL) divergence between stochastic processes, is the fundamental idea [35].

Our approach is in line with the trend of moving away from data-driven systems and toward module-driven ones, where a user of Python gives a list of models as input: models = { $model_1$, $model_2$,...., $model_K$ }. This is called the dictionary of modules.

Data modules:

The dataset for example D is partitioned into arbitraty number of k subsets or modules which are observed and independently processed i.e, $\{D_1, D_2,\ldots D_k\}$. We take the same MDEA dataset for corrosion prediction and split into 3 subsets. Instead of performing SVGP on the entire training data, we perform SVGP on modules/subsets of training data to bring out more efficient results in short span of time.

4.2.3 Sparse variational for independent modules:

We use the sparse variational GP technique for each data module, where the posterior mean vectors and covariance matrices are generated as parameters [36]. We create a k^{th} variational distribution $q_k(f)$ for each data module D_k in order to get the independent approximation to the posterior distribution $p(f|D_k)$ of the GP. Similar to what we have discussed in the previous sections, to fit the GP modules and their variational distributions, we build lower bounds L_k on the marginal log likelihood (ELBO) of every dataset D_k. Then, gradient-based optimization method is used to maximize the K objective functions L_k, one per module.

Dictionary of modules:

As per Pablo Moreno-Muñoz et al, the principal goal here is to obtain a dictionary, containing the already fitted GP modules, for their later use. This dictionary consists of a list of modules. $\{M_1, M_2,\ldots M_k\}$ where each M_k consists of variational and hyper parameters as described in the previous sections.

Module-driven lower bound:

Now, specifically, our objective is to maximise a lower bound L_M under the log-marginal density log p(D) in order to produce an approximate posterior distribution q(f) = p(f|D). This bound should only revisit the objects in the dictionary of modules that are models and not any other data. Be aware that the GP model will no longer be data-driven but rather module-driven. We create a GP meta-model, also known as a meta-GP, by building new models out of existing ones.

We implement augment and reduce strategy to maximize the lower bounds. The steps are:

i) we augment the model to be conditioned on the high-dimensional index set of the stochastic process

ii) we apply properties of Gaussian marginals to reduce the integral operators to a finite amount of GP function values of interest.

The derivation of obtaining the bound can be refered in [30] from equation 2.

Derivations are also provided for multi-output GP i.e, in cases where we have both discrete and continuous data in the dataset. Since the dataset MDEA we are dealing with is regression, we focus only on the continuous function.

Computational costs:

From, Pablo Moreno-Muñoz et al, the computational cost of training modules is $O(N_k M_k^2)$, while the meta GP reduces it to $O((\Sigma_k M_k)M^2)$ and $O(M^2)$ in training and memory respectively.

4.2.4 Distributed GP models:

Distributing the computations using independent local "expert" models, which act on portions of training data, is an alternative to sparse approximations or SVGP. [37] Scaling up computations is made possible by distributed GP models. There are certain hyper-parameters for each local model that need to be optimized. PoEs, or product-of-GP-experts models, avoid the issue with mixture models weight distribution. Since PoEs multiply predictions given by independent GP experts, the input of each expert is inherently weighted in the final prediction. However, the model tends to be overconfident [38]. Cao and Fleet [39] recently proposed a generalised PoE-GP model in which the contribution of an expert in the overall prediction can weighted individually. This model is often too conservative, i.e., it over-estimates variances. Tresp's Bayesian Committee Machine (BCM) [40] can be considered a PoE-GP model, which provides a consistent framework for combining independent estimators within the Bayesian framework, but it suffers from weak experts.

Robust BCM (RBCM) models, is a new family of hierarchical PoE-GP models that (i) includes the BCM and to some degree the generalised PoE-GP. (ii) provides consistent approximations of a full GP. (iii) scales to arbitrarily large data sets by parallelisation. Unlike sparse GPs, RBCM operates on the full data set but distributes the computational and memory load amongst a large set of independent computational units. The RBCM recursively recombines these independent computations to form an efficient distributed GP inference/training framework.

Every distributed GP model has the same salient features of SVGP model such as calculating kernels and likelihood and computing the posterior probability to make predictions.

4.3 Algorithm:

a) Read the methyl diethanolamine (MDEA) dataset and understand the features.

b) Mean normalization is done to bring down all the features to a common scale. Normalization makes all variables contribute equally.

c) Principal Component Analysis (PCA) is performed on the dataset to reduce the dimensionality of the data and increase its interpretability and at the same time minimizing the information loss.

d) Separate the MDEA dataset to train and test.

e) Divide the train dataset into 'n' tasks or subsets.

f) Apply SVGP for every subsets of training data.

g) Tune the variational and hyperparameters for every subset/module. (We use the fitted module for training new data and not the entire data)

h) Apply distributed GP for every subset of training data

i) Optimize the hyperparameters of every module and store each module in a list "models_dist"

j) Using ensemble inference, combine all the SVGP modules (Ensemble GP). Optimize the hyperparameters of the new model and estimate the model using performance metrics.

k) Similarly we apply the concept of ensemble inference in Distributed GP models using "models_dist" and estimate the best distributed GP model using metrics.

l) The performance metrics used are Negative Log Predictive Density (NLPD), Root Mean Square Error (RMSE) and Mean Absolute Error (MAE)

The transfer learning code for Modular Gaussian process can be found here.

4.4 Results:

We are comparing the Sparse Variational Gaussian Process (SVGP) with distributed GP models and providing an alternative to speed up computations for large datasets.

MODEL	**NLPD**	**RMSE**	**MAE**
SVGP	1.4307	0.6969	0.5915
POE	0.9953	0.3785	0.3148
GenPOE	1.0044	0.3785	0.3148
RBCM	0.9951	0.3789	0.3151

From the above results, on comparing SVGP with the distributed GP models such as POE, GenPOE and RBCM, we can verify that the distributed GP models indicate a better fit since they yield better performance metrics. Lesser the error metrics, better is the model fit. The MAE and

RMSE values of Distributed GP are less which means the predicted corrosion values are closer to actual values.

All the three distributed GP models shows no difference in the error metrics which means either one of the 3 models can be used for computation. Considering RBCM's ability to scale Gaussian process to large datasets and many other benefitsas listed above, RBCM can be chosen for corrosion prediction.

5 Conclusion

In this section we compare both the case studies i.e, Transfer Learning under High-dimensional Generalized Linear Models using R programming (Case study 1) and Modular Gaussian Process for transfer learning using Python programming (Case study 2).

Some of the advantages of adopting Case study 1 in corrosion predictions is that:

1) The transfer learning logic using R programming is less complex and easy to understand compared to that in python programming.

2) In R programming, there is a separate package for transfer learning available in CRAN namely Glmtrans. Adopting such package reduces several lines of code and keeps the work space simple.

3) Apart from calculating the error metrics such as MAE, the GLM transfer learning algorithm also provides the predicted corrosion rates values for every transfer learning model.

4) Oracle-Trans-GLM is estimated to be the best model for corrosion rate prediction due to its least MAE value.

5) Hyperparameters of all the models are automatically tuned using the transfer learning library glmtrans, hence the computational time in running the code is subsequently reduced.

Some of the advantages of adopting Case study 2 in corrosion predictions is that:

1) All the Distributed GP models have less MAE compared to SVGP model and the models from case study 1. This is due to the usage of ensemble modeling which improves the predictions by considering the predictions of all the models.

2) Distributed GP can be scaled to large datasets and RBCM is chosen as the best model.

Some of the disadvantages of adopting Case study 2 in corrosion predictions is that:

1) The transfer learning code for modular Gaussian process written in python is highly complicated and there is no pre-installed library for transfer learning in python for structured/tabulated data (like glmtrans in R programming).

2) Hyperparametrs are not automatically tuned, hence the computational time is expensive due to optimization of model parameters using mini batch gradient descent algorithm.

3) This code only outputs the error metrics such as NLPD, RMSE and MAE and does not predict any corrosion rate values.

5.1 Comparison of case studies

Case study 1 - Transfer Learning under High-dimensional Generalized Linear Models using R programming	**Case study 2 - Modular Gaussian Process for transfer learning using Python programming**
Transfer learning logic using R programming is less complex and easy to understand	The code is highly complicated and requires computation of prior, likelihood and posterior to give predictions
A separate package called "Glmtrans" is available in CRAN for transfer learning	No separate package for transfer learning to deal with structured/tabular data
Provides predicted corrosion values	Does not provide predicted corrosion values
The hyperparameters of all the transfer learning models are tuned and optimized automatically	Manual tuning of hyperparameters are required
The time taken for computation is less	Due to the tuning of model parameters, the computation is time consuming
The work space is simple and transfer learning algorithm in R is reduced to few lines of code due to availability of glmtrans package	Due to highly complicated logic with tuning the hyperparameters for every distributed GP model, the algorithm in python is enormously huge.
The best model is chosen to be Oracle-Trans-GLM	The best model is chosen to be RBCM – Distributed GP model
The value of MAE is 0.6627	The value of MAE is 0.3151

Thus comparing, both the case studies, Case Study 1 - Transfer Learning under High-dimensional Generalized Linear Models using R programming is chosen as the best method to predict corrosion rates since its advantages outnumbers the advantages of case study 2.

6 Bibliography

[1] Simeon Kostadinov (2019). What is Deep Transfer Learning and why is it becoming so popular?. Towards Data Science, November 16, 2019, available at https://towardsdatascience.com

[2] Dipanjan (DJ) Sarkar (2018). A comprehensive Hands-on-Guide to Transfer Learning with Real-World Applications in Deep Learning. Towards Data Science, November 14, 2018, available at https://towardsdatascience.com

[3] Moses Olafenwa (2019). Transfer Learning with 5 lines of code. DeepQuestAI, June 28, 2019, available at https://medium.com

[4] Sinno Jialin Pan and Qiang Yang (2010). A Survey on Transfer Learning. IEEE Transactions on Knowledge and Data Engineering, vol. 22, No. 10, October 2010.

[5] Xiang Yu, Jian Wang, Qing-Qi Hong, Raja Teku, Shui-Hua Wang, Yu-Dong Zhang. (2022). Transfer Learning for Medical Image Analyses: A Survey. Neurocomputing Volume 489, 7 June 2022, Pages 230-254.

[6] Yang, L., Hanneke, S. & Carbonell (2013). J. A theory of transfer learning with applications to active learning. Mach Learn 90, 161–189. https://doi.org/10.1007/s10994-012-5310-y

[7] Ling Shao, Fan Zhu (2015). Transfer Learning for Visual Categorization: A Survey. IEEE Transactions on Neural Networks and Learning Systems, vol. 26, No. 5, May 2015.

[8] Chenjing Cai, Shiwei Wang, Youjun Xu, Weilin Zhang, Ke Tang, Qi Ouyang, Luhua Lai, and Jianfeng Pei (2020). Journal of Medicinal Chemistry 2020 63 (16), 8683-8694

[9] Ye Tian and Yang Feng. Transfer Learning under High-dimensional Generalized Linear Models, arXiv:2105.14328v4 [stat.ML], 2022.

[10] Bastani, H. (2021). Predicting with proxies: Transfer learning in high dimension. Management
Science, 67(5):2964-2984.

[11] Li, S., Cai, T. T., and Li, H. (2021). Transfer learning for high-dimensional linear regression: Prediction, estimation and minimax optimality. Journal of the Royal Statistical Society: Series B (Statistical Methodology).

[12] Li, S., Cai, T. T., and Li, H. (2020). Transfer learning in large-scale gaussian graphical models with false discovery rate control. arXiv preprint arXiv:2010.11037.
[13] Cai, T. T. and Wei, H. (2021). Transfer learning for nonparametric classification: Minimax rate and adaptive classifier. The Annals of Statistics, 49(1):100-128.
[14] Reeve, H. W., Cannings, T. I., and Samworth, R. J. (2021). Adaptive transfer learning. The Annals of Statistics, 49(6):3618-3649.
[15] Hanneke, S. and Kpotufe, S. (2020a). A no-free-lunch theorem for multitask learning. arXiv preprint arXiv:2006.15785.
[16] Chen, A., Owen, A. B., Shi, M., et al. (2015). Data enriched linear regression. Electronic Journal of Statistics, 9(1):1078-1112.
[17] Zheng, C., Dasgupta, S., Xie, Y., Haris, A., and Chen, Y. Q. (2019). On data enriched logistic regression. arXiv preprint arXiv:1911.06380.
[18] Gross, S. M. and Tibshirani, R. (2016). Data shared lasso: A novel tool to discover uplift. Computational Statistics & Data Analysis, 101:226-235.
[19] Ollier, E. and Viallon, V. (2017). Regression modelling on stratified data with the lasso. Biometrika, 104(1):83-96.
[20] Torrey, L. and Shavlik, J. (2010). Transfer learning. In Handbook of research on machine learning applications and trends: algorithms, methods, and techniques, pages 242-264. IGI global.
[21] McCullagh, P, and John A Nelder. 1989. Generalized Linear Models. Vol. 37. CRC Press.
[22] Tian, Ye, and Yang Feng. (2021). Transfer Learning with High-Dimensional Generalized Linear Models.
[23] Weiss, Karl, Taghi M Khoshgoftaar, and DingDing Wang. (2016). “A Survey of Transfer Learning.” Journal of Big Data 3 (1): 1–40.
[24] Tibshirani, Robert. (1996). Regression Shrinkage and Selection via the Lasso. Journal of the Royal Statistical Society: Series B (Methodological) 58 (1): 267–88.
[25] Zou, Hui, and Trevor Hastie. (2005). Regularization and Variable Selection via the Elastic Net. Journal of the Royal Statistical Society: Series B (Statistical Methodology) 67 (2): 301–20.
[26] Pablo Moreno-Muñoz, Antonio Artés-Rodríguez, and Mauricio A. Álvarez. Modular Gaussian Processes for Transfer Learning. (2021) 35th Conference on Neural Information Processing Systems (NeurIPS 2021).

[27] Jie Wang. (2022). An Intuitive Tutorial to Gaussian Process Regression. arXiv:2009.10862v4 [stat.ML] 18 Apr 2022

[28] C. E. Rasmussen, C. K. I. Williams. (2006) Gaussian Processes for Machine Learning, The MIT Press.

[29] Wei Yi. (2019). Understanding Gaussian Process, the Socratic way. Towards Data Science, December 1, 2019, available at https://towardsdatascience.com

[30] Wei Yi. (2020). Sparse and Variational Gaussian Process (SVGP) – What to do when data is large. Towards Data Science, June 26, 2020, available at https://towardsdatascience.com

[31] Wei Yi. (2020). Variational Gaussian Process (VGP) – What to do when things are not Gaussian. Towards Data Science, June 5, 2020, available at https://towardsdatascience.com

[32] V. Smith, C.-K. Chiang, M. Sanjabi, and A. S. Talwalkar. (2017). Federated multi-task learning. In Advances in Neural Information Processing Systems (NIPS), pages 4424–4434.

[33] I. J. Goodfellow, M. Mirza, D. Xiao, A. Courville, and Y. Bengio. (2014). An empirical investigation of catastrophic forgetting in gradient-based neural networks. In International Conference on Learning Representations (ICLR).

[34] D. Peterson, P. Kanani, and V. J. Marathe. (2019) Private federated learning with domain adaptation. Workshop on Federated Learning for Data Privacy and Confidentiality at NeurIPS.

[35] A. G. d. G. Matthews, J. Hensman, R. Turner, and Z. Ghahramani. (2016). On sparse variational methods and the Kullback-Leibler divergence between stochastic processes. In Artificial Intelligence and Statistics (AISTATS), pages 231–239.

[36] M. K. Titsias. (2009). Variational model selection for sparse Gaussian process regression. Technical Report, University of Manchester, 2009b.

[37] Marc Peter Deisenroth, Jun Wei Ng. Distributed Gaussian Processes. Imperial College London, United Kingdom

[38] Ng, Jun W. and Deisenroth, Marc P. (2014). Hierarchical Mixtureof-Experts Model for Large-Scale Gaussian Process Regression. http://arxiv.org/abs/1412.3078, December 2014.

[39] Cao, Yanshuai and Fleet, David J. (2014). Generalized Product of Experts for Automatic and Principled Fusion of Gaussian Process Predictions. http://arxiv.org/abs/1410.7827, October 2014.

[40] Tresp, Volker. A Bayesian Committee Machine. Neural Computation, 12(11):2719–2741, 2000. URL http://www.dbs.ifi.lmu.de/˜tresp/papers/bcm6.pdf.