Data in Brief 42 (2022) 108113

Contents lists available at ScienceDirect

Data in Brief

journal homepage: www.elsevier.com/locate/dib

Data Article

Synthetic data for design and evaluation of binary classifiers in the context of Bayesian transfer learning

Omar Maddouri[a], Xiaoning Qian[a,b], Francis J. Alexander[b], Edward R. Dougherty[a], Byung-Jun Yoon[a,b,*]

[a] Department of Electrical and Computer Engineering, Texas A&M University, College Station TX 77843, USA
[b] Computational Science Initiative, Brookhaven National Laboratory, Upton NY 11973, USA

ARTICLE INFO

Article history:
Received 7 December 2021
Revised 21 January 2022
Accepted 28 March 2022
Available online 2 April 2022

Dataset link: Synthetic Data for Design and Evaluation of Binary Classifiers in the Context of Bayesian Transfer Learning (Original data)

Keywords:
Bayesian transfer learning
Binary classification
Classifier design
Error estimation

ABSTRACT

Transfer learning (TL) techniques can enable effective learning in data scarce domains by allowing one to re-purpose data or scientific knowledge available in relevant source domains for predictive tasks in a target domain of interest. In this Data in Brief article, we present a synthetic dataset for binary classification in the context of Bayesian transfer learning, which can be used for the design and evaluation of TL-based classifiers. For this purpose, we consider numerous combinations of classification settings, based on which we simulate a diverse set of feature-label distributions with varying learning complexity. For each set of model parameters, we provide a pair of target and source datasets that have been jointly sampled from the underlying feature-label distributions in the target and source domains, respectively. For both target and source domains, the data in a given class and domain are normally distributed, where the distributions across domains are related to each other through a joint prior. To ensure the consistency of the classification complexity across the provided datasets, we have controlled the Bayes error such that it is maintained within a range of predefined values that mimic realistic classification scenarios across different relatedness levels. The provided datasets

DOI of original article: 10.1016/j.patter.2021.100428
* Corresponding author.
E-mail address: bjyoon@ece.tamu.edu (B.-J. Yoon).

https://doi.org/10.1016/j.dib.2022.108113

may serve as useful resources for designing and benchmarking transfer learning schemes for binary classification as well as the estimation of classification error.

Specifications Table

Subject	Applied Machine Learning
Specific subject area	Bayesian transfer learning
Type of data	Binary Matlab files Matlab source code Shell script
How data were acquired	Matlab simulations
Data format	Binary Matlab files (*.mat) Matlab scripts (*.m) Shell script (*.sh)
Parameters for data collection	Three feature dimensions ($d \in \{2, 3, 5\}$) were considered for the studied feature spaces. Four classification complexity levels (Bayes error $\in \{0.1, 0.2, 0.3, 0.4\}$) were considered for each dimension. Six relatedness levels ($\|\alpha\| \in \{0.1, 0.3, 0.5, 0.7, 0.9, 0.99\}$) were used to model the relatedness between source and target domains.
Description of data collection	Source and target datasets have been generated by Matlab simulations. The feature-label distributions in source and target domains were assumed to be multivariate Gaussian distributions. Both domains were related to each other through a joint prior: i.e., Wishart distribution of the precision matrices of the underlying Gaussian feature-label distributions. The classification complexity has been modeled by the Bayes error that has been determined via the true classification error of an optimal quadratic discriminant analysis (QDA) classifier.
Data source location	Institution: Texas A&M University City/Town/Region: College Station, TX 77843 Country: USA
Data accessibility	Repository name: Synthetic Data for Design and Evaluation of Binary Classifiers in the Context of Bayesian Transfer Learning [1] DOI: 10.17632/fn33cknmfx.1 Direct URL to data: https://data.mendeley.com/datasets/fn33cknmfx/1
Related research article	O. Maddouri, X. Qian, F. J. Alexander, E. R. Dougherty, B.-J. Yoon, Robust importance sampling for error estimation in the context of optimal Bayesian transfer learning, Patterns 3 (3) (2022) 100428. https://doi.org/10.1016/j.patter.2021.100428.

Value of the Data

- The data here provide useful resources for studying binary classification and error estimation problems from a transfer learning perspective. The relatedness across domains has been mathematically modeled as in [2] through a joint Wishart distribution over the model parameters. This enables rigorous quantification of the relevance across the source and target domains. The selective sampling of the model parameters in the source and target domains based on the classification complexity (Bayes error) makes the comparison of the evaluation results across different dimensions and relatedness levels possible, as it preserves the simulation conditions across different experiments. Without these stringent conditions, drawing statistically meaningful conclusions from empirical analysis would be practically difficult.
- The provided data are of practical values to any data-driven machine learning approach that employs transfer learning to solve binary classification problems. More specifically, the dataset can be used to design novel classifiers in the target domain based on additional data

from the source domain. The large size of the provided dataset (for each configuration, there are 10^5 data points per class for each domain) will facilitate the design, validation, and evaluation of new algorithms. The wide range of values for the feature space dimensions, Bayes errors, and relatedness levels will enable a comprehensive performance assessment of new classification and error estimation methods under diverse classification settings.

- In many scientific or clinical settings, training data are typically limited in the target domain (e.g., due to high data acquisition cost), which impedes the design and evaluation of accurate classifiers. Transfer learning can improve the learning outcome in the target domain by incorporating data from relevant source domain(s). From this perspective, the optimal setting to use the provided data is to consider only a few data points in the target domain to develop new machine learning methods (*e.g.,* classifier design [2], classification error estimation [3]), and to leverage a relatively larger amount of source data to improve the machine learning task in the target domain. The substantial part of the remaining target data that are provided in the dataset should be mainly used to estimate the ground-truth metrics (*i.e.,* true classification error) and not as training data.
- The provided simulation source code can be used to simulate other classification scenarios for higher feature space dimensions and/or different classification complexity levels.
- The detailed description of the simulation setup that was used to generate the current dataset can provide a solid guideline on how the experimental setup should be configured to study classification problems from a transfer learning perspective. As the transfer learning aspect involves various factors affecting the classification and error estimation performance, especially due to the heterogeneity of the data characteristics across domains, it is critical to maintain uniformity of the experimental conditions across all the simulations to enable interpretations of the obtained results that are accurate, valid, and statistically meaningful.

1. Data Description

As illustrated in Fig. 1, the main folder **Synthetic_Data_Classification_Bayesian_Transfer_Learning** contains three data sub-folders (**d_2, d_3**, and **d_5**) that correspond to dimensions 2, 3, and 5, respectively. The remaining sub-folder **generation_source_code** contains the Matlab source code.

In every data sub-folder (**d_2, d_3**, or **d_5**) there are 24 binary Matlab files with names encoded as follows: `Data_d_x_Bayes_x.x_n_t_x_n_s_x_alpha_x.x_nu_x.mat`, where:

- `d_x`: refers to the dimension of the feature space where x takes values 2, 3, or 5.
- `Bayes_x.x`: designates the classification complexity level (0.1, 0.2, 0.3, or 0.4).
- `n_t_x`: indicates the number of target data points per class (*i.e.:* for 10^5, this string is set to `n_t_100000`)
- `n_s_x`: indicates the number of source data points per class (*i.e.:* for 10^5, this string is set to `n_s_100000`)

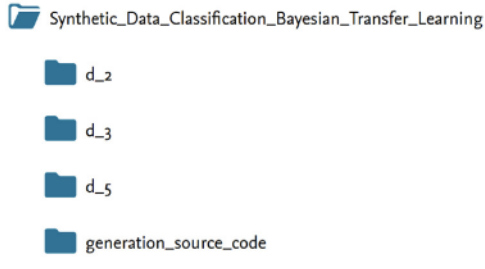

Fig. 1. Hierarchy of the main data repository.

- `alpha_x.x`: indicates the relatedness level (0.1, 0.3, 0.5, 0.7, 0.9, or 0.99).
- `nu_x`: specifies the value of a hyperparameter ν that corresponds to the degrees of freedom used to model the joint Wishart distribution (in our simulations, we set $\nu = d + 20$).

In every Matlab binary file among the aforementioned files, there are 4 indexed data containers called cell arrays that also contain, each, two cell containers. These data containers are described as follows:

- `D_s`: source dataset of dimension ($10^5 \times d$) per class.
- `D_t`: target dataset of dimension ($10^5 \times d$) per class.
- `param_s`: parameter vector of the source domain that specifies the means and the precision matrices of the multivariate Gaussian distributions that underlie the data classes and has the following cell parameters:
  1. `mu_s`: contains a d-dimensional real-valued vector per class.
  2. `Lambda_s`: contains a $(d \times d)$ positive definite precision matrix per class.
- `param_t`: parameter vector of the target domain that specifies the means and the precision matrices of the multivariate Gaussian distributions that underlie the data classes and has the following parameters:
  1. `mu_t`: contains a d-dimensional real-valued vector per class.
  2. `Lambda_t`: contains a $(d \times d)$ positive definite precision matrix per class.

The Matlab source code directory **generation_source_code**[1] is structured as illustrated in Fig. 2. For each dimension ($d \in \{2,\ 3,\ 5\}$), we include a list of sub-folders **Bayes_x.x** that correspond to different classification complexity levels that have been used in the evaluation experiments. Under each **Bayes_x.x** folder, we dedicate a sub-folder **alpha_x.x** for each relatedness level.

Fig. 3 shows the content of each relatedness level directory. Under each relatedness level folder, there is a main script file named **simulate_Data.m** that includes the simulation settings relevant to the parameter values as specified by the architecture of the parent directories. The remaining files (**generate_Data.m, setup_parameters.m, test_error_QDA.m**, and **train_QDA.m**)[2] are shared across all the experiments and are duplicated in the different folders for distributed execution purposes.

2. Experimental Design Materials and Methods

2.1. Bayesian transfer learning framework for binary classification

To model the synthetic data we consider a binary classification problem in the context of supervised transfer learning where there are two classes in each domain. Let $\mathcal{D}_s$ and $\mathcal{D}_t$ be two labeled datasets from the source and target domains with sizes N_s and N_t, respectively. Let $\mathcal{D}_s^y = \left\{\mathbf{x}_{s,1}^y, \mathbf{x}_{s,2}^y, \cdots, \mathbf{x}_{s,n_s}^y\right\}$, $y \in \{0, 1\}$, where $n_s^y = 10^5$ denotes the size of source data in class y. Likewise, let $\mathcal{D}_t^y = \left\{\mathbf{x}_{t,1}^y, \mathbf{x}_{t,2}^y, \cdots, \mathbf{x}_{t,n_t}^y\right\}$, $y \in \{0, 1\}$, where $n_t^y = 10^5$ denotes the size of target data in class y. With the assumption of disjoint datasets across classes, we have $\mathcal{D}_s = \mathcal{D}_s^0 \cup \mathcal{D}_s^1$ and $\mathcal{D}_t = \mathcal{D}_t^0 \cup \mathcal{D}_t^1$ with $N_s = n_s^0 + n_s^1 = 2 \times 10^5$ and $N_t = n_t^0 + n_t^1 = 2 \times 10^5$.

We consider a d-dimensional homogeneous transfer learning scenario where $\mathcal{D}_s$ and $\mathcal{D}_t$ are normally distributed and separately sampled from the source and target domains, respectively:

$$\mathbf{x}_z^y \sim \mathcal{N}\left(\mu_z^y, \left(\Lambda_z^y\right)^{-1}\right),\ y \in \{0, 1\}, \tag{1}$$

[1] The **submit_jobs.sh** file is optional and is dedicated to submitting all the simulation scenarios as parallel jobs on high performance computing resources.

[2] The **text_progress_bar.m** file is optional and is used to show the progress when the heuristic search for the model parameters is ongoing.

Fig. 2. Structure of the Matlab source code repository.

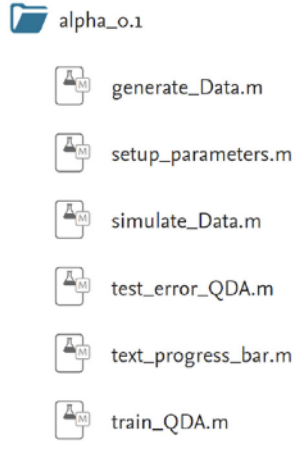

Fig. 3. Matlab script files.

where $z \in \{s,t\}$, μ_z^y is a $(d \times 1)$ mean vector in domain z for class y, Λ_z^y is a $(d \times d)$ matrix that denotes the precision matrix (inverse of covariance) in domain z for label y, and $\mathcal{N}(\cdot,\cdot)$ denotes the multivariate normal distribution. An augmented feature vector $\mathbf{x}^y = \begin{bmatrix} \mathbf{x}_t^y \\ \mathbf{x}_s^y \end{bmatrix}$ is a joint sample point from two related source and target domains given by

$$\mathbf{x}^y \sim \mathcal{N}\left(\mu^y, \left(\Lambda^y\right)^{-1}\right), \; y \in \{0,1\}, \tag{2}$$

with

$$\mu^y = \begin{bmatrix} \mu_t^y \\ \mu_s^y \end{bmatrix}, \Lambda^y = \begin{bmatrix} \Lambda_t^y & \Lambda_{ts}^y \\ \Lambda_{ts}^{y\,T} & \Lambda_s^y \end{bmatrix}, \tag{3}$$

where X^T denotes the transpose of matrix X. Using a Gaussian-Wishart distribution as the joint prior for mean and precision matrices, the joint model factorizes as

$$p\left(\mu_s^y, \mu_t^y, \Lambda_s^y, \Lambda_t^y\right) = p\left(\mu_s^y, \mu_t^y | \Lambda_s^y, \Lambda_t^y\right) \; p\left(\Lambda_s^y, \Lambda_t^y\right). \tag{4}$$

For conditionally independent mean vectors given the covariances, the joint prior in (4) further expands to

$$p\left(\mu_s^y, \mu_t^y, \Lambda_s^y, \Lambda_t^y\right) = p\left(\mu_s^y | \Lambda_s^y\right) \; p\left(\mu_t^y | \Lambda_t^y\right) \; p\left(\Lambda_s^y, \Lambda_t^y\right). \tag{5}$$

The block diagonal precision matrices Λ_z^y for $z \in \{t,s\}$ are obtained after sampling Λ^y from a predefined joint Wishart distribution as defined in [2] such that $\Lambda^y \sim W_{2d}(\mathbf{M}^y, \nu^y)$, where ν^y is a hyperparameter for the degrees of freedom that satisfies $\nu^y \geq 2d$ and $\mathbf{M}^y$ is a $(2d \times 2d)$ positive definite scale matrix of the form $\mathbf{M}^y = \begin{pmatrix} \mathbf{M}_t^y & \mathbf{M}_{ts}^y \\ \mathbf{M}_{ts}^{y\,T} & \mathbf{M}_s^y \end{pmatrix}$. $\mathbf{M}_t^y$ and $\mathbf{M}_s^y$ are also positive definite scale matrices and $\mathbf{M}_{ts}$ denotes the off-diagonal component that models the interaction between source and target domains. Given Λ_z^y, and assuming normally distributed mean vectors we get

$$\mu_z^y \sim \mathcal{N}\left(\mathbf{m}_z^y, \left(\kappa_z^y \; \Lambda_z^y\right)^{-1}\right), \; z \in \{s,t\} \text{ and } y \in \{0,1\}, \tag{6}$$

where $\mathbf{m}_z^y$ is the $(d \times 1)$ mean vector of the mean parameter μ_z^y and κ_z^y is a positive scalar hyperparameter.

2.2. Synthetic datasets

In order to generate the synthetic data, we consider feature space dimensions 2, 3, and 5. In the simulated datasets, we set up the data distributions as follows:

$\nu = \nu^y = d + 20$, $\kappa_t = \kappa_t^y = 100$, $\kappa_s = \kappa_s^y = 100$, $\mathbf{m}_t^0 = \mathbf{0}_d$, $\mathbf{m}_t^1 = \vartheta \times \mathbf{1}_d$, $\mathbf{m}_s^0 = \mathbf{m}_t^0 + 10 \times \mathbf{1}_d$, $\mathbf{m}_s^1 = \mathbf{m}_t^1 + 10 \times \mathbf{1}_d$, where ϑ is an adjustable scalar used to control the Bayes error in the target domain, and $\mathbf{0}_d$ and $\mathbf{1}_d$ are $d \times 1$ all-zero and all-one vectors, respectively.

For the scale matrices of Wishart distributions we set $\mathbf{M}_t^y = k_t \mathbf{I}_d$, $\mathbf{M}_s^y = k_s \mathbf{I}_d$, and $\mathbf{M}_{ts}^y = k_{ts} \mathbf{I}_d$ where $\mathbf{I}_d$ is the identity matrix of rank d.

To ensure that the joint scale matrix $\mathbf{M}^y = \begin{pmatrix} \mathbf{M}_t^y & \mathbf{M}_{ts}^y \\ \mathbf{M}_{ts}^{y\,T} & \mathbf{M}_s^y \end{pmatrix}$ is positive definite $\forall y \in \{0,1\}$, we set $k_{ts} = \alpha\sqrt{k_t k_s}$ with $k_t > 0$, $k_s > 0$, and $|\alpha| < 1$. As in [2], the value of $|\alpha|$ controls the amount of relatedness between the source and target domains.

To fully control the level of relatedness by adjusting only $|\alpha|$ and without involving other confounding factors, we set $k_t = k_s = 1$ such that $\mathbf{M}_{ts}^y = \alpha \; \mathbf{I}_d$. In this setting, the correlation between the features across source and target domains are governed by $|\alpha|$, where small values of $|\alpha|$ correspond to poor relatedness between source and target domains while larger values imply stronger relatedness.

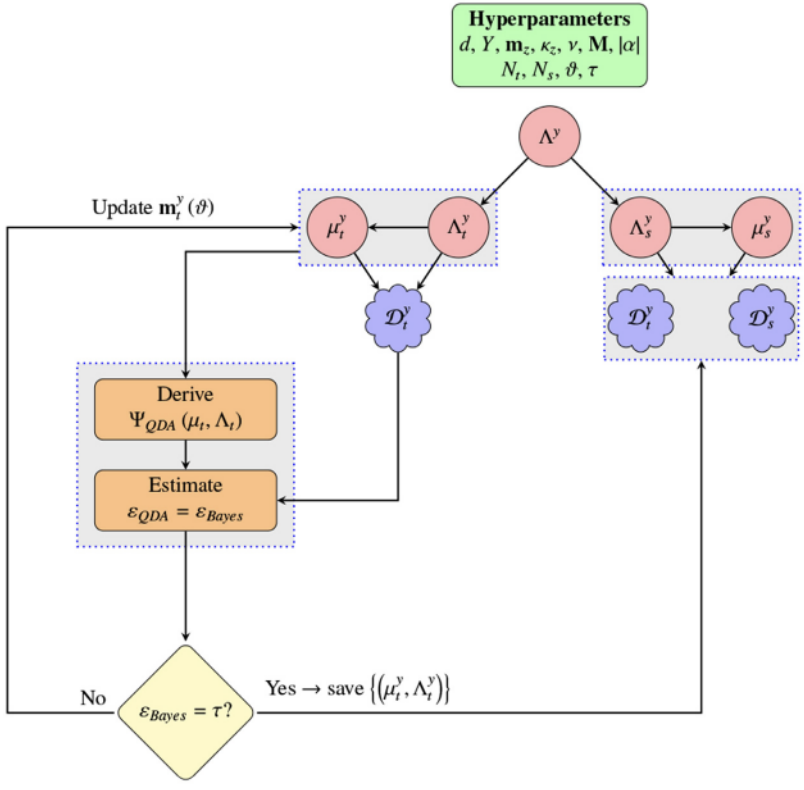

Fig. 4. Flow chart illustrating the simulation set-up for generating the synthetic datasets in this paper.

To sample from the joint prior, we first sample from a non-singular Wishart distribution $W_{2d}(\mathbf{M}^y, \nu)$ to get a block partitioned sample of the form $\mathbf{\Lambda}^y = \begin{pmatrix} \mathbf{\Lambda}_t^y & \mathbf{\Lambda}_{ts}^y \\ \mathbf{\Lambda}_{ts}^{y\,T} & \mathbf{\Lambda}_s^y \end{pmatrix}$, from which we extract $\left(\mathbf{\Lambda}_t^y, \mathbf{\Lambda}_s^y\right)$. Afterwards, we sample $\mu_z^y \sim \mathcal{N}\left(\mathbf{m}_z^y, \left(\kappa_z^y \mathbf{\Lambda}_z^y\right)^{-1}\right)$ for $z \in \{s, t\}$ and $y \in \{0, 1\}$.

As illustrated in Fig. 4, we use in our simulations two types of datasets. Training datasets that contain samples from both domains and testing datasets that contain only samples from the target domain. While the training datasets are saved and stored in our data repository, the testing datasets are only aimed for simulation purposes to specify a desired level of classification complexity. In all the simulations we consider testing datasets of 10^4 data points per class and we assume equal prior probabilities for the classes. We note that for normally distributed data, the optimal classifier for the the feature-label distributions, called also Bayes classifier, is a quadratic classifier that can be determined through quadratic discriminant analysis (QDA). This Bayes classifier is defined as: $\Psi_{QDA}(\mathbf{x}) = \mathbf{x}^T\mathbf{A}\mathbf{x} + \mathbf{b}^T\mathbf{x} + c$, where

$$\begin{aligned} &\mathbf{A} = -\tfrac{1}{2}\left(\mathbf{\Lambda_t^1} - \mathbf{\Lambda_t^0}\right), \quad \mathbf{b} = \mathbf{\Lambda_t^1}\mu_t^1 - \mathbf{\Lambda_t^0}\mu_t^0, \\ &c = -\tfrac{1}{2}\left(\mu_t^{1\,T}\mathbf{\Lambda_t^1}\mu_t^1 - \mu_t^{0\,T}\mathbf{\Lambda_t^0}\mu_t^0\right) - \tfrac{1}{2}\ln\left(\frac{|\mathbf{\Lambda_t^0}|}{|\mathbf{\Lambda_t^1}|}\right). \end{aligned} \tag{7}$$

Its true error is called Bayes error.

To draw data for a specific Bayes error, we start by drawing a joint sample $\left(\mathbf{\Lambda}_t^y, \mathbf{\Lambda}_s^y\right)$ for each class $y \in \{0, 1\}$. Next, we iterate over the values of the hyperparameter ϑ to control $\mathbf{m}_t(\vartheta)$ through a dichotomic search to get a desired value τ of the Bayes error. This is achieved by

drawing a sample $\mu_t^y \sim \mathcal{N}\left(\mathbf{m}_t(\vartheta), \left(\kappa_t^y \Lambda_t^y\right)^{-1}\right)$ and then generating a test set based on the joint sample $\left(\mu_t^y, \Lambda_t^y\right)$. Using this test set, we determine the true error of the optimal QDA derived from $\left(\mu_t^y, \Lambda_t^y\right)$. If the desired Bayes error (true error of the designed QDA) is attained then the iteration stops, otherwise we update ϑ and reiterate.

2.3. Matlab script files

2.3.1. simulate_Data.m:

This is the main simulation file that generates and saves the data into binary Matlab files (`*.mat`). First, the constants and the model hyperparameters are set. In this example we show configurations for $d = 2$, `Bayes_error`$=0.1$, and a relatedness level $|\alpha| = 0.1$.

```
%% Define constants
d = 2; %Number of features
L = 2; %Number of classes
n_t = 10^(5); %Target data per class
n_s = 10^(5); %Source data per class
N_test = 10^(4); %Size of test set per class (drawn from target domain)
alpha = 0.1; %Relatedness coefficient

complexity_Bayes_error = 0.1; %Desired classification complexity
eps_error = 10^(-6); %Error tolerance
nu = 22; %Degrees of freedom
mean_shift = 1; %Intitial shift between the means of target domain
```

Next, we draw from the joint model an initial sample for the model parameters in the source and target domains to initiate the heuristic search for the model parameters that correspond to the desired `Bayes_error`.

```
%% Sample New Parameters
param = setup_parameters(d, L, mean_shift, nu, alpha);
mu_t = cell(1, L);
mu_s = cell(1, L);
Lambda = cell(1, L);
Lambda_t = cell(1, L);
Lambda_s = cell(1, L);
Sigma_t = cell(1, L);
for l = 1:L
    Lambda{l} = wishrnd(param.M{l},param.nu{l});
    Lambda_t{l} = Lambda{l}(1:d,1:d);
    Sigma_t{l} = Lambda_t{l}^(-1);
    Lambda_s{l} = Lambda{l}((d+1):2*d,(d+1):2*d);
end
```

Afterwards, we loop over different realizations of model parameters until we obtain the desired `Bayes_error`. To do so, we start with the joint sample $\left(\Lambda_t^y, \Lambda_s^y\right)$ for each class $y \in \{0, 1\}$. Next, we iterate over the values of the hyperparameter ϑ, referred to in the source code by `mean_shift`, to control $\mathbf{m}_t(\vartheta)$ through a dichotomic search to get a desired value τ (`complexity_Bayes_error` in the code) of the Bayes error. This is achieved by drawing a sample $\mu_t^y \sim \mathcal{N}\left(\mathbf{m}_t(\vartheta), \left(\kappa_t^y \Lambda_t^y\right)^{-1}\right)$ and then generating a test set based on the joint sample $\left(\mu_t^y, \Lambda_t^y\right)$. Using this test set, we determine the true error of the optimal QDA derived from $\left(\mu_t^y, \Lambda_t^y\right)$ (lines 16 and 19). If the desired Bayes error (true error of the designed QDA) is attained then the iteration stops, otherwise we update the `mean_shift` variable and reiterate.

```
%% Heuristic search for the model parameters that correspond to the ...
    desired classification complexity level.
text_progress_bar(sprintf('Search model parameters for complexity(Bayes ...
    error = %g), d=%1d, alpha=%g:', complexity_Bayes_error, d, alpha));
Bayes_error=0;
i=0;

while true
    text_progress_bar(i);
    i=i+1;

    for l = 1:L
        mu_t{l} = mvnrnd(param.m_t{l},(param.kappa_t{l} * ...
            Lambda_t{l})^(-1));
        mu_s{l} = mvnrnd(param.m_s{l},(param.kappa_s{l} * ...
            Lambda_s{l})^(-1));
    end

    %% Determine the optimal QDA classifier (Bayes classifier)
    QDA_Bayes = train_QDA(mu_t, Lambda_t, L);

    %% Compute Bayes Error
    Bayes_error = test_error_QDA(QDA_Bayes, param, mu_t, Lambda_t, N_test);

    if abs(Bayes_error - complexity_Bayes_error) < eps_error
        break
    end
    if Bayes_error > complexity_Bayes_error
        mean_shift = (mean_shift * 2) * (1 + rand);
    elseif Bayes_error < complexity_Bayes_error
        mean_shift = (mean_shift / 2) * rand;
    end

    if mean_shift==0 && Bayes_error < complexity_Bayes_error
        for l = 1:L
            Lambda{l} = wishrnd(param.M{l},param.nu{l});
            Lambda_t{l} = Lambda{l}(1:d,1:d);
            Sigma_t{l} = Lambda_t{l}^(-1);
            Lambda_s{l} = Lambda{l}((d+1):2*d,(d+1):2*d);
        end
        mean_shift = rand;
    elseif mean_shift==0 && Bayes_error > complexity_Bayes_error
        mean_shift = rand;
    end

    param = setup_parameters(d, L, mean_shift, nu, alpha);
end

text_progress_bar(sprintf('\nSearch for model parameters converged ...
    (Bayes error = %g)\n', Bayes_error));
```

Once a realization of model parameters satisfies the desired `Bayes_error`, target and source datasets are generated and stored into binary Matlab files.

```
%% Generate source and target datasets
[D_full, ~, ~] = generate_Data(mu_t, mu_s, Lambda_t, Lambda_s, L, n_t, ...
    n_s, N_test);
%% Target dataset + True model parameters
D_t = D_full.t;
param_t.Lambda_t = Lambda_t;
param_t.mu_t = mu_t;
%% Source dataset + True model parameters
D_s = D_full.s;
param_s.Lambda_s = Lambda_s;
param_s.mu_s = mu_s;
%% Save datasets
save_name = ...
    sprintf('Data_d_%1d_Bayes_%g_n_t_%1d_n_s_%1d_alpha_%g_nu_%1d.mat', ...
    d, complexity_Bayes_error, n_t, n_s, alpha, nu);
save(save_name, 'D_t', 'param_t', 'D_s', 'param_s');
```

2.3.2. *setup_parameters.m:*

This function takes as input the model hyperparameters that change their values across the simulated datasets, and uses the shared values of the remaining hyperparameters to fully characterize the feature-label distributions in source and target domains.

```
function param = setup_parameters(d, L, mean_shift, nu, alpha)
    param.d = d;
    param.L = L;
    for i = 1:L
        %Distance between the means of the target classes to fix the ...
            Bayes error
        dist_class_means = mean_shift;
        % nu:
        param.nu{i} = nu;
        % kappa:
        param.kappa_t{i} = 100;
        param.kappa_s{i} = 100;
        % m:
        param.m_t{i} = dist_class_means * (i-1) * ones(1,d);
        param.m_s{i} = param.m_t{i} + 10*ones(1,d);
        % M:
        k_t = 1;
        k_s = 1;
        param.k_t = k_t;
        param.k_s = k_s;
        param.alpha = alpha;
        param.M_t{i} = k_t * diag(ones(1,d));
        param.M_s{i} = k_s * diag(ones(1,d));
        param.M_ts{i} = param.alpha * sqrt(k_t * k_s) * eye(d);
        param.M{i} = [param.M_t{i} param.M_ts{i}; param.M_ts{i}' ...
            param.M_s{i}];
        param.Sigma{i} = param.M{i}^-1;
        param.M_t_inv{i} = param.M_t{i}^-1;
    end
end
```

2.3.3. *generate_Data.m:*

As indicated by its name, this function takes as input a specified set of model parameters of source and target domains and generates synthetic training and testing datasets.

```
function [X, S, Label] = generate_Data(mu_t, mu_s, Lambda_t, Lambda_s, ...
    nc, n_t, n_s, n_test)

    X.test = cell(1, nc);
    X.t = cell(1, nc);
    X.s = cell(1, nc);
    X.t_mean = cell(1, nc);
    X.s_mean = cell(1, nc);
    S.t = cell(1, nc);
    S.s = cell(1, nc);
    X.test_all = [];
    Label.test_all = [];

    for i = 1:nc
        % Test (for target)
        X.test{i} = mvnrnd(mu_t{i},Lambda_t{i}^(-1),n_test);
        X.test_all = [X.test_all; X.test{i}];
        Label.test{i} = i * ones(1,n_test);
        Label.test_all = [Label.test_all Label.test{i}];

        % Train (for both target and source)
        if n_t > 0
            X.t{i} = mvnrnd(mu_t{i},Lambda_t{i}^(-1),n_t);
        else
            X.t{i} = 0;
        end
        Label.t{i} = i * ones(1,n_t);
        if n_s > 0
            X.s{i} = mvnrnd(mu_s{i},Lambda_s{i}^(-1),n_s);
        else
            X.s{i} = 0;
        end
        Label.s{i} = i * ones(1,n_s);

        % Sample mean and covariance
        if n_t ~= 1
            X.t_mean{i} = mean(X.t{i});
        else
            X.t_mean{i} = X.t{i};
        end
        if n_s ~= 1
            X.s_mean{i} = mean(X.s{i});
        else
            X.s_mean{i} = X.s{i};
        end

        S.t{i} = (X.t{i} - repmat(X.t_mean{i},n_t,1))' * (X.t{i} - ...
            repmat(X.t_mean{i},n_t,1));
        S.s{i} = (X.s{i} - repmat(X.s_mean{i},n_s,1))' * (X.s{i} - ...
            repmat(X.s_mean{i},n_s,1));
    end
end
```

2.4. train_QDA.m:

This function permits to identify the Bayes classifier in the target domain. It implements the definition of a QDA classifier designed based on a predefined set of model parameters for a binary classification problem when the two classes are *a-priori* equally likely.

```
function QDA = train_QDA(mu, Lambda, L)

    for ll = 1:L
        mu{ll} = reshape(mu{ll},[],1);
    end

    c = 1/2;%Class-0 prior probability
    A = (-1/2) * (Lambda{2} - Lambda{1});
    a = (Lambda{2} * mu{2} - Lambda{1} * mu{1});
    b = -(1/2) * ( (mu{2}' * Lambda{2} * mu{2}) - (mu{1}' * Lambda{1} * ...
        mu{1}) )...
        - (1/2) * log(det(Lambda{1})/det(Lambda{2}))...
        - log((1-c)/c);

    QDA.A = A;
    QDA.a = a;
    QDA.b = b;
end
```

2.5. test_error_QDA.m:

This function allows to approximate the true classification error of a QDA classifier based on a given test set. In our simulations, this function is called to determine the `Bayes_error` that corresponds to the true classification error of a QDA classifier that has been designed based on the true model parameters in the target domain.

```
function error = test_error_QDA(QDA, param, mu_t, Lambda_t, N_test)

    L = param.L;
    [X, ~, Label] = generate_Data(mu_t, 0, Lambda_t, 0, L, 0, 0, N_test);

    X_test = X.test_all;
    Label_test = Label.test_all;
    n_test = length(Label_test);

    Label_pred = zeros(1,n_test);
    for i = 1:n_test
        x = X_test(i,:);
        x = reshape(x,[],1);
        Label_pred(i) = x' * QDA.A * x + x' * QDA.a + QDA.b;
    end

    Label_pred(Label_pred>0) = 2;
    Label_pred(Label_pred<=0) = 1;
    accuracy = sum(Label_pred == Label_test)/n_test;
    error = 1 - accuracy;
end
```

Ethics Statement

The work did not involve any human or animal subjects, nor data from social media platforms.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships which have, or could be perceived to have, influenced the work reported in this article.

Data Availability

Synthetic Data for Design and Evaluation of Binary Classifiers in the Context of Bayesian Transfer Learning (Original data) (Mendeley Data).

CRediT Author Statement

Omar Maddouri: Conceptualization, Data curation, Formal analysis, Investigation, Methodology, Software, Validation, Visualization, Writing – original draft, Writing – review & editing; **Xiaoning Qian:** Conceptualization, Formal analysis, Supervision, Validation, Writing – review & editing; **Francis J. Alexander:** Conceptualization, Formal analysis, Validation, Writing – review & editing; **Edward R. Dougherty:** Conceptualization, Formal analysis, Funding acquisition, Project administration, Supervision, Validation, Writing – review & editing; **Byung-Jun Yoon:** Conceptualization, Formal analysis, Funding acquisition, Methodology, Project administration, Resources, Supervision, Validation, Writing – review & editing.

Acknowledgments

This work was supported in part by the Department of Energy (DOE) under Award DE-SC0019303 and the National Science Foundation (NSF) award 1835690.

Portions of the simulations were conducted with the advanced computing resources provided by Texas A&M High Performance Research Computing.

Supplementary Materials

Supplementary material associated with this article can be found in the online version at doi:10.1016/j.dib.2022.108113.

References

[1] O. Maddouri, X. Qian, F.J. Alexander, E.R. Dougherty, B.-J. Yoon, Synthetic data for design and evaluation of binary classifiers in the context of Bayesian transfer learning, Mendeley Data v1 (2021). https://data.mendeley.com/datasets/fn33cknmfx/1

[2] A. Karbalayghareh, X. Qian, E.R. Dougherty, Optimal Bayesian transfer learning, IEEE Trans. Signal Process 66 (14) (2018) 3724–3739.

[3] O. Maddouri, X. Qian, F.J. Alexander, E.R. Dougherty, B.-J. Yoon, Robust importance sampling for error estimation in the context of optimal Bayesian transfer learning, Patterns 3 (3) (2022) 100428.