Training Data for Machine Learning to Enhance Patient-Centered Outcomes Research (PCOR) Data Infrastructure — A Case Study in Tuberculosis Drug Resistance

Implementation Guide

National Library of Medicine

National Institute of Allergy and Infectious Diseases

Training Data for Machine Learning to Enhance Patient-Centered Outcomes Research (PCOR) Data Infrastructure — A Case Study in Tuberculosis Drug Resistance

Implementation Guide

Manohar Karki[1], Karthik Kantipudi[2], Babak Haghighi[1], Vy Bui[1], Feng Yang[1], Hang Yu[1], Michael Harris[2], Yasmin M. Kassim[1], Darrell E. Hurt[2], Alex Rosenthal[2], Ziv Yaniv[2], and Stefan Jaeger [1]

[1]Lister Hill National Center for Biomedical Communications
National Library of Medicine
National Institutes of Health
8600 Rockville Pike, Bethesda, MD 20894

[2]Office of Cyber Infrastructure and Computational Biology
National Institute of Allergy and Infectious Diseases
National Institutes of Health
8600 Rockville Pike, Bethesda, MD 20894

March 7, 2023

Contents

1 Introduction

This implementation guide describes research conducted by the National Library of Medicine (NLM) in collaboration with the National Institute of Allergy and Infectious Diseases (NIAID) at the National Institutes of Health (NIH). The Office of the Secretary Patient-Centered Outcomes Research Trust Fund (OS-PCORTF), coordinated by the Assistant Secretary for Planning and Evaluation (ASPE), supported this research under Interagency Agreement #750119PE080057, during the period from August 1, 2019, to September 30, 2022. This work was also supported in part by the Intramural Research Program of the National Library of Medicine, National Institutes of Health, and had been funded in part with federal funds from the National Institute of Allergy and Infectious Diseases under BCBB Support Services Contract HHSN316201300006W/HHSN27200002. Furthermore, this research was supported in part by an appointment to the National Library of Medicine Research Participation Program administered by the Oak Ridge Institute for Science and Education (ORISE) through an interagency agreement between the U.S. Department of Energy (DOE) and the National Library of Medicine. ORISE is managed by Oak Ridge Associated Universities (ORAU) under DOE contract number DE-SC0014664.

All opinions, findings, and conclusions expressed in this guide are the authors' and do not necessarily reflect the policies and views of NIH, NLM, DOE, or ORAU/ORISE. Therefore, no statement in this report should be construed as an official position of NLM/NIH or the U.S. Department of Health and Human Services.

None of the investigators have affiliations or financial involvement that conflicts with the material presented in this guide. This guide is not intended to be a substitute for the application of clinical judgment. Anyone who makes decisions concerning the provision of clinical care should consider this guide in the same way as any medical reference and in conjunction with all other pertinent information, i.e., in the context of available resources and circumstances presented by individual patients.

NLM/NIH or U.S. Department of Health and Human Services endorsement of any derivative products developed from this report, such as clinical practice guidelines, other quality enhancement tools, or reimbursement or coverage policies, may not be stated or implied.

Suggested citation: Karki et al. Training Data for Machine Learning to Enhance Patient-Centered Outcomes Research (PCOR) Data Infrastructure — A Case Study in

Tuberculosis Drug Resistance. Implementation Guide. National Library of Medicine, November 2022.

2 Datasets

For this project, multiple publicly available datasets comprising de-identified images and clinical data were used to train machine classifiers. This section lists the main sets used.

2.1 Chest X-ray images

Frontal chest X-ray (CXR) images for pulmonary tuberculosis (TB) were obtained from the following sources.

2.1.1 TB portals

Under the leadership of the National Institute of Allergy and Infectious Diseases (NIAID), the TB Portals program is designed to unite radiological, genomic, clinical, laboratory, and socioeconomic/geographic data from prospective and retrospective TB cases and their associated clinical samples and to freely share these curated data, including powerful and user-friendly analytical tools, with researchers and health care specialists throughout the world. This mission is accomplished through (i) international collaboration with clinical research sites and academic research organizations in countries with a heavy burden of drug-resistant TB and (ii) the creation, support, expansion, and promotion of the multifactor repository of anonymized clinical data from patients with drug-resistant forms of TB.

The TB Portals program is still in the data acquisition phase and includes both data acquired before and during this PCOR project. It should be noted that TB Portals data differs in two main aspects from other data sources. First, it only contains data for patients with TB. Second, it is a much richer data source. For each patient, it includes clinical and socioeconomic information (e.g., treatment regimens, outcome, education level, etc.), imaging studies with manual and AI-generated labels (CXR and possibly CT), and finally, the pathogen's genomic information and the type of TB in terms of drug susceptibility (drug-sensitive (DS) or specific type of drug-resistant (DR) TB

such as Mono DR, Poly DR, and XDR). The TB Portals data acquired to support this project since 2020 is listed in Table 1.

	CXR	CT
2022	204(DS), 262(DR)	1(DS), 3(DR)
2021	955(DS), 1891(DR)	2(DS), 24(DR)
2020	527(DS), 1108(DR)	56(DS), 92(DR)
Total	1686(DS), 3261(DR)	59(DS), 119(DR)

Table 1: TB Portals imaging data acquired in support of this work. Data represents patients with drug-sensitive (DS) and drug-resistant (DR) TB.

Access to TB Portals data is managed. To access the data, one must sign a data use agreement, describe the research project and agree not to distribute the data or attempt to identify patients. Once approved, the data is available in two forms: (a) download from a server using a dedicated protocol for the transfer of big data, which is more appropriate for images, or (b) use of a RESTful API to retrieve the tabular data (all data except images and genomic information). The former is updated quarterly and the latter weekly.

As of now, TB Portals [7] contains images from hospitals in 16 countries. A sample CXR from the TB Portals collection is shown in Figure 1.

For a more detailed overview of TB Portals data sharing and usage examples, see: https://datasharing.tbportals.niaid.nih.gov/.

Figure 1: Sample frontal CXR from the TB Portals dataset.

2.1.2 External datasets

Three external CXR datasets predate this project: Montgomery County, Shenzhen, and TBX11K. Their data acquisition phase was completed, and they had been finalized before this project started. Each of the sets comprises CXRs from patients with and without TB. The project work assumed that the TB cases found in these datasets are drug-sensitive as this is the prevalent TB type found in the corresponding geographical locations. These datasets were directly downloaded from their respective websites.

1. Montgomery County [3], 138 images (58 TB, 80 Non-TB). These images were acquired from the tuberculosis control program of the Department of Health and Human Services of Montgomery County, MD, USA. The set contains 138 frontal CXRs, of which 80 are normal cases, and 58 are cases with manifestations of TB. The CXRs were captured with a Eureka stationary X-ray machine and are provided in Portable Network Graphics (PNG) format as 12-bit gray-level images. They can also be made available in DICOM format upon request.

2. Shenzhen [3], 662 images (336 TB, 326 Non-TB). The Shenzhen dataset was collected in collaboration with Shenzhen No.3 People's Hospital, Guangdong Medical College, Shenzhen, China. The chest X-rays are from outpatient clinics and were captured as part of the daily hospital routine within one month, mainly in September 2012, using a Philips DR Digital Diagnost system. The set contains 662 frontal CXRs, of which 326 are normal cases and 336 are cases with manifestations of TB, including pediatric X-rays (AP). The X-rays are provided in PNG format. All image file names follow the same template: CHNCXR_####_X.png, where #### represents a 4-digit numerical identifier, and X is either 0 for a normal X-ray or 1 for an abnormal X-ray. The clinical reading for each X-ray is saved in a text file following the same format, except that the ending ".png" is replaced with ".txt." Each reading contains the patient's age, gender, and abnormality seen in the lung if any.

3. TBX11K [5], 8400 images (800 TB, 7600 Non-TB). The images were acquired from hospitals in China. The dataset includes 3800 normal, 3800 abnormal but non-TB, and 800 TB CXR images publicly available. All images are in png format with a size of 512x512 pixels and an 8-bit gray-level intensity scale.

2.2 CT images

Sample CT slices from the TB Portals collection are shown in Figure 2. In addition, thoracic CT datasets were obtained from the following sources, all except one do not deal with TB:

1. Lung CT Segmentation Challenge (LCTSC) 2017 [12], 60 CTs and corresponding lung masks (RTSTRUCT) from 60 patients were acquired from three institutions (20 each). Datasets were divided into three groups stratified per institution: 36 training datasets, 12 off-site test datasets, and 12 live test datasets.

2. NSCLC-Radiomics [1], 422 CTs and corresponding lung segmentations from non-small cell lung cancer (NSCLC) patients.

3. Lung Nodule Analysis (LUNA) 2016 [9], 888 CTs and corresponding lung segmentations from LIDC/IDRI database. Scans with a slice thickness greater than 2.5 mm were excluded.

4. MDPI Bioengineering COVID19 Dataset [13]. This set comprises data from 81 COVID-19-positive patients. The CT volumes are provided with an automatic lung tissue annotation and scoring provided by an expert radiologist, ranging from 0 (clinically healthy) to 5 (pathological).

5. COVID-19 CT Lung and Infection Segmentation Dataset [4], 20 labeled COVID-19 CT scans. All cases are COVID-19 infections, with the proportion of infected lung areas ranging from 0.01% to 59%.

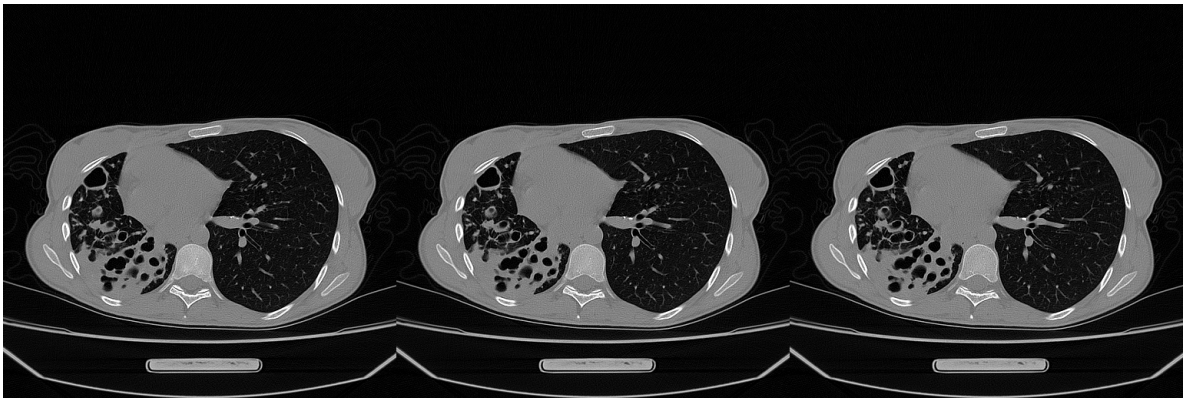

Figure 2: Axial CT slices from the TB Portals dataset.

6. VESsel SEgmentation in the Lung 2012 (VESSEL12) [8], 20 CTs and corresponding lung masks. The challenge was organized in conjunction with the IEEE International Symposium on Biomedical Imaging (ISBI 2012).

In the list above, the datasets that contain diffused abnormalities are COVID-19 [4] and MDPI [13]. Datasets that contain focal abnormalities are VESSEL12 [8], LCTSC [12], NSCLC-Radiomics [1] and LUNA [9].

2.3 Genomic data in TB portals

As mentioned in Section 2.1.1, one of the permanent goals of TB Portals is to collect, study, and make public a body of information about drug-resistant TB. A significant part of this effort has been the establishment of the TB Portals database, an anonymized patient-centric resource containing multi-domain metadata, including patient images (chest X-rays and CT scans) and pathogen single-nucleotide polymorphisms (SNP) associated with drug resistance, which are collected, curated, and made available to the research community. As part of this international collaboration, the TB Portals program has collected and processed genome sequences for more than 2200 Mycobacterium tuberculosis (MTB) samples [10].

3 Tools and frameworks

The algorithms in this project were primarily implemented in Python 3.8, Anaconda distribution 2022.10.

For the tasks of discriminating DR-TB and DS-TB in CXRs, including lung segmentation (Section 4 and Section 7), the following frameworks were used: Tensorflow 2.4.1, Keras with TensorFlow backend, and scikit-learn. For lung segmentation in CTs, Section 6, nnunet was used.

All deep learning training was performed on a graphical processing unit (GPU). For this project, we used the NVIDIA Tesla K80 GPU.

The analysis of DR-TB and DS-TB on clinical data was implemented in Matlab 2020b, using the statistics and machine learning toolboxes; see Section 5.

The radiological and genomic data analysis in Section 9 was implemented in Python 3.8, Anaconda distribution 2022.10., and the scikit-learn framework.

4 Discriminating between drug-resistant TB and drug-sensitive TB using chest X-rays

CXRs from NIAID TB Portals were used to train models for the classification of DR-TB and DS-TB. External datasets such as Montgomery County [3],Shenzhen [3], and TBX11K [5] were also used. CXRs from these sets were assumed to be DS-TB, as DR-TB is less common in the countries where these sets were acquired.

4.1 Setting up a virtual environment

A YML file is provided to install the necessary packages. Use the terminal or an Anaconda prompt for the following steps:

(1) Create the environment from the environment.yml file:

```
conda env create -f environment.yml
```

(2) Activate the new environment:

```
conda activate dr_ds_tb
```

(3) Verify that the new environment was installed correctly:

```
conda env list
```

4.2 Preprocessing

4.2.1 Running the preprocessing function for CXRs

```
python code/generate_data_file.py input_csv_path.csv  \
trained_lung_segmentation_model_path.h5 output_binary_masks.h5
```

4.2.2 Input arguments for preprocessing

The user needs to run the above command line to crop a CXR and extract the lung region. The input to the preprocessing function is as follows:

(1) [required] input_csv_path: (CSV Path) CSV file path that contains at least a single column named 'image_file,' which contains the file path of a CXR.

(2) [required] trained_lung_segmentation_model_path: (Pathlib path) Path for the trained lung segmentation model (see lung segmentation directory).

(3) [required] output_binary_masks.h5: (Pathlib path) Output file name provided with an extension as .h5 file. This file contains the preprocessed cropped lung arrays and corresponding labels (1 for DR-TB / 0 for DS-TB).

4.2.3 Description of preprocessing function

Before cropping the lung regions, the *get_image* method in *generate_data_file.py* initially resamples all the original images to (256,256), scales them, and performs histogram equalization on each image. After this preprocessing step, the loaded lung segmentation model computes the lung masks on the preprocessed arrays. The bounding box coordinates of the binary lung masks are used to crop lung regions from the normalized (mean=0 and standard deviation=1) original images to the size of *output_image_size*. Cropping of these images to their lung regions defined by the binary masks happens in *_segmented_lung_2_tb_cnn* in the *crop_input.py* file.

```
def get_image(filepath, model):
    """
    Reads, normalizes, and crops images with lung regions,
    generates a lung mask for each image
    Input Params:
          filepath(string): Image file path
          model(keras.Model): Trained lung segmentation model
    Returns:
      out[...,0](np.ndarray): Cropped image array
      conf(string):  Segmentation confidence
    """
    original_img, resampled_for_seg = \

        _resample_cxr_for_lung_segmentation_cnn(
                            SEGMENTATION_INPUT_SIZE=(256,256),
                            gaussian_sigma=0.5, filepath)

    img = sitk.GetArrayFromImage(resampled_for_seg)
    img = img - np.min(img)
    img = img / np.max(img)
    img = exposure.equalize_adapthist(img)

    if len(img.shape) == 2:
        c = np.expand_dims(img, axis=0)
        img = np.stack((c, c, c), axis=3)

    pred = model.predict(img, batch_size=1, verbose=0)[..., 0]
    """
```

4.3 Training

4.3.1 Running the training function

```
python dr_ds_train.py input_csv_path model_output_filename
--lung_segmentation_model_path
 ../../lung_segmentation/weights/Segmentation_resnet50_UNet.h5
 --gpu_id 0 --batch_size 32 --epochs 100 --lr 0.0001
 --random_seed  42  --augment_images
```

4.3.2 Input arguments for the training step

To train the DR-TB vs. DS-TB detection model, the user has to run the above command line with the following input:

(1) [required] input_csv_path: (CSV Path) CSV file path that contains at least a single column with column name 'image_file,' which contains the file path of each image.

(2) [required] trained_lung_segmentation_model_path: (Pathlib path) Path for the trained lung segmentation model (see lung segmentation directory).

(3) [required] output_binary_masks.h5: (Pathlib path) Output file name should be provided as .h5 file. This file contains the preprocessed cropped lung arrays and their corresponding labels (1 for DR-TB / 0 for DS-TB)

(4) [optional] gpu_id: (int) GPU Device Number. The default is 0.

(5) [optional] batch_size: (int) Batch Size to train the model. Users can change this value depending on the GPU memory available.

(6) [optional] lr: (float) Learning Rate for the Adam optimizer.

4.3.3 Building a DR-TB vs. DS-TB model

Prepare the model architecture to be trained for DR-TB vs. DS-TB using the *build_model_pretrained* function as shown in the code below. Users can specify model architecture names such as "vgg16", "resnet50", "densenet121", or "xception" in the *MODEL_NAME* variable of the *global_constants.py* file. The default architecture name in the *global_constants.py* file is 'inceptionv3'. This function extracts the pre-trained 'imagenet' weights from the desired model. It attaches the Global AveragePooling2D

layer, Dense layer with 16 neurons, Dropout layer, and final Dense layer to classify DR-TB vs. DS-TB using the 'sigmoid' activation function.

```
def build_model_pretrained():
    """
    Load pretrained model with pretrained weights

    Args:
           ---

    Returns:
       model(keras.Model): Loaded model with 'pretrained' imagenet
       weights.
    """
    classifier, preprocess_input = Classifiers.get(MODEL_NAME)
    inputs = Input(shape=(IMAGE_WIDTH, IMAGE_HEIGHT, 1),
                   name='image')
    # pretrained model requires three channels
    img_conc = Concatenate()([inputs, inputs, inputs])  #
    input_model = Model(inputs=inputs, outputs=img_conc)
    base_model = classifier(input_tensor=input_model.output,
                        weights='imagenet',
                        input_shape=(IMAGE_WIDTH, IMAGE_HEIGHT, 3),
                        include_top=False)
    """
    """
```

4.3.4 Preparing data for model training

After generating the cropped image array in the preprocessing step, the *load_data* function in the *dr_ds_train.py* file prepares the data to train the model for DR-TB vs. DS-TB. This function stratifies the loaded data to balance labels for DR-TB and DS-TB. After stratifying the data, this function performs histogram equalization and splits the data into training and validation sets based on the validation ratio. The obtained training set is further augmented using the *augment_images* function in the *augment.py*

file.

```
def load_data(data_path,augment_images=True,random_seed=42,
              validation_ratio=0.2):
    """
    Load the preprocessed array containing cropped lung regions
    by shuffling and stratifying the labels

    Args:
        data_path: (str) Path to data file
        augment_images: (bool) Augmentation flag
        random_seed: (int) Seed to ensure consistency of the data
                           between training and validation sets
        validation_ratio(float): Ratio to divide training and
                                 validation sets
    Returns:
       train_X,train_Y,val_X,val_Y:  array of training data,
                                     array of training labels,
                                     array of validation data, and
                                     array of validation labels
    """
    # Read cropped X-ray image data and labels
    data = h5py.File(data_path, 'r')  # Load preprocessed data
    """
    """
    ....
```

4.3.5 Description of training function

After the model and the data are loaded, the script *dr_ds_train.py* invokes the train function described below to train the model with Adam optimizer. The model training stops whenever there is no improvement in the validation loss for seven epochs.

```
def train(model, epochs, batch_size, data, model_output_filename,
        lr):
    """
    Trains classification model, saves periodically and stops
    the training early if the validation loss does not improve for
    seven epochs.

    Args:
      model: (keras.Model) Prepared model from the
      build_model_pretrained function.
      epochs: (int) Max number of epochs for training
      batch_size: (int) Number of samples in a batch
      data: (list) List containing arrays of training data,
                  training labels, validation data, and
                  validation labels
      model_output_filename: (string) Filename for saving the
      output model
      lr: (float) Learning rate
    Returns:
       model: (keras.Model) Loaded model with 'pretrained'
              imagenet weights.
    """
    train_X = data[0]
    train_Y = data[1]
    val_X = data[2]
    val_Y = data[3]
   .....
```

4.4 Inference

4.4.1 Running the inference function

```
python dr_ds_inference.py input_filenames.csv output_predictions.csv
--threshold 0.6323408
```

4.4.2 Input arguments for the inference function

The inputs for the inference function are as follows:

(1) [required] input_csv_path: (CSV Path) CSV file path with a column named 'image_file,' which contains the file path of the image.

(2) [required] model_output_filename: (Pathlib path) Output file path to save the model.

(3) [optional] lung_segmentation_model_path: (Pathlib path) Lung model path to segment the lung region in a CXR. The default threshold is 0.6323408. This threshold is computed based on Youden's index. More adjustments are possible by changing the global_constants.py file.

4.4.3 Loading the DR-TB vs. DS-TB model

The inference script takes the DR-TB/DS-TB model path from the command line and loads the model for the 'inference' function.

```
model = load_model(dr_ds_model_path)
```

4.4.4 Description of the inference script

The CSV file containing image file paths is fed as input to the *prep_inference_data* function (shown below) within the 'inference' function along with the lung segmentation model path and the *output_image_size*, which is inferred from the loaded DR-TB/DS-TB model; see step above. The *prep_inference_data* function, which is imported from *generate_data_file.py*, preprocesses the CXRs by cropping each of the original CXR images to their lung regions, respectively.

```
  filenames,test_X = \
      prep_inference_data(csv_path,lung_seg_model_path,
                          output_img_size =
                          model.layers[0].input_shape[0][1:3])
```

```
def prep_inference_data(csv_path, lung_seg_model_path,
                        output_img_size):
    """
    Preprocesses segmentation model, crops images, and generates
    lung masks
    Args:
          csv_path: (CSV Path) CSV file with image path names.
          lung_segment_model_path: (pathlib.Path) Pretrained
                                   lung segmentation model path.
          output_img_size: Output image size of the cropped
                           lung region image.
                           This size is calculated from
                           the DR-TB/DS-TB model
                           as the script can predict only on the
                           size that the DR-TB/DS-TB model has been
                           trained on.
    Outputs:
        filenames(list): Filenames obtained from the
                         'image_file' column.
        X(np.ndarray): Preprocessed array with output_img_size
                       containing lung regions from the original
                       images
    """
    df = pd.read_csv(csv_path)
    filenames = df['image_file']
    samples = df.shape[0]
    X = np.zeros((samples, img_size[0], img_size[1]))
```

4.4.5 Predicting on preprocessed CXR data using the loaded model

After the DR-TB/DS-TB model is loaded and the data from *input_csv_path* is preprocessed, the *inference* function in the *dr_ds_inference.py* script invokes the prediction step as shown below.

```
y_pred = model.predict(test_X)
```

4.4.6 Discriminating between DR-TB or DS-TB

After the predictions are generated from the model on the preprocessed cropped lung image arrays, the decision between DR-TB or DS-TB is made based on whether the prediction values are greater than a threshold derived from Youden's index.

```
labels = ['DR-TB' if val == 1 else 'DS-TB'
          for val in y_pred > threshold]
```

5 Discriminating between drug-resistant TB and drug-sensitive TB using clinical data

The functions described in this section are to investigate the relevance of clinical features and features derived from CXRs in DR-TB prediction.

5.1 Software environment

Experiments in this section are based on Matlab 2020b. The statistics and machine learning toolboxes are required.

5.2 Radiological and clinical features

The number of lung sextants affected by abnormalities such as nodules, cavities, and infiltrates is used as a radiological feature. Other radiological features based on sextants include lung cavity size, nodule size, infiltrate density, mediastinal lymph nodes,

calcified nodules, and others. In addition, the size of abnormal lung volume and pleural effusion are used as features. In total, 25 radiological findings are recorded in the clinical spreadsheet of TB Portals represented by the following columns:

```
overall_percent_of_abnormal_volume
pleural_effusion_percent_of_hemithorax_involved
ispleuraleffusionbilateral
other_non_tb_abnormalities
are_mediastinal_lymphnodes_present
collapse
smallcavities
mediumcavities
largecavities
isanylargecavitybelongtoamultisextantcavity
canmultiplecavitiesbeseen
infiltrate_lowgroundglassdensity
infiltrate_mediumdensity
infiltrate_highdensity
smallnodules
mediumnodules
largenodules
hugenodules
isanycalcifiedorpartiallycalcifiednoduleexist
isanynoncalcifiednoduleexist
isanyclusterednoduleexists
aremultiplenoduleexists
lowgroundglassdensityactivefreshnodules
mediumdensitystabalizedfibroticnodules
highdensitycalcifiedtypicallysequella
```

Specifically, the following three clinical features have been used in the experiments.

```
age_of_onset
gender
case_definition
```

5.2.1 Running the preprocessing function

```
run step1_preprocessing.m
```

5.2.2 Preprocessing function and parameters

The preprocessing role is to: 1) combine features, 2) extract radiological features from a CXR, and 3) save features and their corresponding labels for model training and testing. There are two options for the number of features: NumFeat=25 or NumFeat=29 (Line 8). The default number is set to 25. More details can be found in [11].

5.2.3 Input arguments for the preprocessing step

To run the preprocessing function, the following input files are required:

(1) [required]: "TB Portals Patient Cases_20201021.csv" under the folder "TB Portals Published data_20201021".

The output files are as follows:

(1) Extracted features: "25FeaturesfromCXR_Oct2022.mat". Each row represents a patient, and each column shows a feature.

(2) Corresponding labels for each patient: "Labelsfor25FeaturesfromCXR_Oct2022.mat".

5.3 Training and inference

5.3.1 Running the training and inference function

```
run step2_TrainingandInference_SVM.m
```

5.3.2 Input arguments for running the training and inference steps

To run the training and inference function, two files are required:

(1) [required] Feature file: "25FeaturesfromCXR_Oct2022.mat"

(2) [required] Label file: "Labelsfor25FeaturesfromCXR_Oct2022.mat"

The output of this function includes:

(1) trained classifier (SVM or Random Forest): if SVM is chosen for classification, then the output file is: "SVMModel_Oct2022.mat"; otherwise, Random Forest is chosen for classification, and the output file is: "RandomForestModel_Oct2022.mat."

(2) ROC curves based on ten-fold cross-validation.

(3) Testing results, including the average AUC, accuracy, sensitivity, specificity, precision, and F1 score, with their respective standard deviations, are saved in the file: "TestingResults_TenFold_2022.xlsx."

5.3.3 Description of training and inference function

This function uses a ten-fold cross-validation strategy for training and testing. SMOTE is used to balance datasets; see details in [11]. Users can change the number of folds (k) for cross-validation. The default training model is SVM, but users can change it to Random Forest by changing the value of HN in Line 13: HN=1 indicates "SVM" and HN=2 indicates "Random forest."

6 Lung segmentation in CTs

This section describes software to segment lung regions in CT scans based on deep learning.

6.1 Datasets and input arguments for preprocessing

For this section, datasets from the following sources were used: [7], [12], [1], [9], [13], [4], [8]; see also Section 2. Not all of these datasets contain ground truth segmentation masks for lungs. After downloading these datasets, the user has to run the *standardize_datasets.py* script so that all ground truth masks get standardized to a binary lung mask to train the lung segmentation model. This script standardizes all CTs and their corresponding masks to NIFTI format for consistency.

```
python -m segment_lung_ct.preparation_code.standardize_inputs  \
   data/inputs/dataset_configuration.json  standardized_masks  \
   standardized_cts \ lung_segmentation_files.csv
```

The input arguments to the above script are as follows:

(1) [required] data/inputs/dataset_configuration.json: (Pathlib path) JSON path that contains keys and values of each dataset and their corresponding subdirectories of CTs and masks, respectively.

(2) [required] standardized_masks: (string) Directory name to save standardized masks. This directory will contain sub-directories (generated by the script) referenced by the above configuration file. The masks will be saved in a .nii.gz file.

(3) [required] standardized_cts: (string) Directory name to save standardized CTs. This directory will contain sub-directories (generated by the script) referenced by the above configuration file. The CTs will be saved in a .nii.gz file.

(4) [required] lung_segmentation_files.csv : (string) Output CSV filename containing column names such as 'ct_file' , 'seg_file' and 'dataset', representing CT paths and the corresponding binary lung masks and datasets.

6.2 Organizing the data for training

After generating the CSV file containing lung segmentation filenames from the above command, the user must now create the data structure for training nnUNet as follows:

```
nnUNet
├── nnUNet_raw_data_base
│   └── nnUNet_raw_data
├── nnUNet_trained_models
└── nnUNet_preprocessed
```

After creating the above structure, the user must set up the paths below

```
export nnUNet_raw_data_base="/path/to/nnUNet/nnUNet_raw_data_base"
export nnUNet_preprocessed="/path/to/nnUNet/nnUNet_preprocessed"
export RESULTS_FOLDER='/path/to/nnUNet/nnUNet_trained_models'
```

After setting up these paths, the user can run the command line below to create the folders of the preprocessing step for nnUNet models.

```
python -m segment_lung_ct.data.preparation_code.data_prep
 data/inputs/lung_segmentation_files.csv \
   data/inputs/sampling_factors_for_lung_segmentation.json 0
  /path/to/nnUNet/nnUNet_raw_data_base/nnUNet_raw_data/ \
  Task001_lungsegmentation
```

After running the above command line, the user will see the following structure, which is necessary to run the preprocessing step for nnUNet.

```
nnUNet
├── nnUNet_raw_data_base
│   └── nnUNet_raw_data
│       └── Task001_lungsegmentation
│           ├── imagesTr
│           │   ├── a_0000.nii.gz
│           │   ├── b_0000.nii.gz
│           │   └── c_0000.nii.gz
│           ├── labelsTr
│           │   ├── a.nii.gz
│           │   ├── b.nii.gz
│           │   └── c.nii.gz
│           ├── imagesTs
│           │   ├── d_0000.nii.gz
│           │   ├── e_0000.nii.gz
│           │   └── f_0000.nii.gz
│           ├── labelsTs
│           │   ├── d.nii.gz
│           │   ├── e.nii.gz
│           │   └── f.nii.gz
│           └── dataset.json
├── nnUNet_trained_models
└── nnUNet_preprocessed
```

The inputs for the data preparation script are as follows:

(1) [required] input_csv_path : (Pathlib path) CSV file path with column names 'ct_file' and 'ref_seg_file' representing CT paths and their respective lung segmentations.

(2) [required] sampling_factors_for_lung_segmentation.json:(Pathlib path) JSON file path that contains a dictionary of 'train,' 'val,' and 'test' keys for each dataset.

(3) [required] fold_num: (int) Fold number in the k-fold cross validation

(4) [required] folder_name: (Pathlib path) Output folder to save files in "imagesTr," "imagesTs," "labelsTr," "labelsTs" subfolders created as part of the script. These subfolders are required as per nnUNet training.

6.3 Preprocessing CTs

6.3.1 Running the preprocessing step

```
python preprocess_data_nnunet.py -task_number 001
```

6.3.2 Input arguments for the preprocessing step

After setting up the data structure, the user needs to preprocess the CTs by executing the above command line.

[required] task_number : (int) Task number (XXX). A folder name is stored in nnUNet_raw_data_base / nnUNet_raw_data / TaskXXX_MYTASK format.

6.4 Training a lung segmentation model with nnUNet

6.4.1 Running the training step

```
python  code/train_lung_ct.py --task_number 001 --cv_fold_number 0
```

6.4.2 Input arguments for the training step

The inputs for the above command line are as follows:

(1) [required] task_number : (int) Task number (XXX), where the folder name is stored in nnUNet.

(2) [required] cv_fold_number: (int) cross-validation number in [0,1,2,3,4] to train the model on a given fold. The user can also specify the value 'all' to train the model on all available images instead of a fold-based partitioned set.

After training the lung and lesion segmentation models separately, the generated models will be saved in the following directory *nnUNet_trained_models*. For more info on this, see the following repository: https://github.com/MIC-DKFZ/nnUNet.

6.5 Inference

6.5.1 Running the inference from a lung segmentation model

```
python code/inference_lung_ct.py \
sample_volume.nii.gz weights/lung_segment.model \
sample_volume_lung_preds.nii.gz --post_process
```

6.5.2 Input arguments for the inference step

The input arguments for the above command line are as follows:

(1) [required] sample_volume.nii.gz: (Pathlib path) Path of the lung CT volume to be segmented

(2) [required] lung_segment.model: (Pathlib path) Path of the trained lung segmentation model

(3) [required] sample_volume_lung_preds.nii.gz: (str) Output prediction filename

(4) [optional] –post_process: By default, this argument will post-process the combined prediction label from the segmentation models. If the user does not wish to post-process the final prediction label, then –post_process can be removed from the command line.

6.5.3 Reading the input CT volume

The inference script first reads the input path for the original CT volume provided in the command line

```
img = sitk.ReadImage(ct_path)
```

6.5.4 Loading the segmentation model

The inference script takes the lung and lesion segmentation model paths from the command line and loads them in the *_get_dataset_properties_and_model* function. This outputs the loaded model and training data properties, which are used to preprocess the input CT volume.

```
        (
            lung_segmentation_model,
            lung_training_dataset_properties,
        ) = \
        _get_dataset_properties_and_model(args.model_path_for_lung)
```

6.5.5 Segmenting CT volumes using the loaded model

After the loaded model and dataset properties are obtained from the model path, they are provided as inputs to the *get_prediction_label* function, which generates the prediction labels. This method first preprocesses the loaded CT images using the *preprocess_image* function and then predicts the segmentation on the preprocessed CT array using the *predict_segmentation* function. The *preprocess_image* function crops the original CT volume so that it contains non-zero values. The CT volume is then resampled to a size containing the same median spacing obtained from all the spacings of all the CT volumes the model has been trained on. After the volume is resampled, it is normalized to mean=0 and std=1 before clipping the intensity values of the resampled image to the 0.5 and 99.5 percentiles of the training dataset intensities. These percentile values are read from the *lung_segment.model.pkl* file.

```
    pred_label = get_prediction_label(
        lung_segmentation_model, lung_training_dataset_properties,
        img )
```

```
def preprocess_image(training_dataset_properties, img):
    """
    Args:
        training_dataset_properties: (dict) Dictionary containing
              dataset properties that the model has trained on.
        img: (SimpleITK Image) Loaded SimpleITK image.
    Returns:
          data: (np.ndarray) Final preprocessed array of the
             CT volume.
          test_image_properties: (dict) Dictionary with keys like
              "itk_origin", "itk_spacing" , "itk_direction,"
              representing the origin, spacing, and direction
              of the CT volume.
          bbox: (list) [[minzidx, maxzidx], [minxidx, maxxidx],
             [minyidx, maxyidx]]. List of lists containing
             bounding box coordinates.
    """
    """
```

After the input CT volume array is prepared for inference, the loaded model is used to predict on the preprocessed array to get the prediction label. This label is in the size of the preprocessed image and not in the original image size, so it is resized back to the original size of the input CT volume using the *sitk.Resample* filter.

```
def predict_segmentation(
    model, d, dct, bbox, mixed_precision=True, all_in_gpu=None,
    step_size=0.5,
    do_tta=False
):

    """
    This function predicts the lung segmentation in CTs by a single
    model.
    Args:
        model: (nnUNetTrainerV2) Loaded nnUNet model.
        d: (np.ndarray) preprocessed numpy array
        dct: (dict) properties of the preprocessed array of the
                    image.
        bbox: (list)
           [[minzidx, maxzidx], [minxidx, maxxidx],
            [minyidx, maxyidx]]
           List of lists containing bounding box coordinates.
        mixed_precision: (bool) If None, then no action is taken.
                                If True/False, then overwrite
                                what the model has in its init

    """
    """
    """
```

6.5.6 Postprocessing the prediction label

After obtaining the prediction label from the step above, the script will execute the postprocessing function if the user wishes to do so in the command line (–post_process).

```
    final_pred_label = postprocess_label(final_pred_label)
```

The postprocessing function removes objects in the prediction label whenever the

objects are below the threshold limit of 549 ml. This limit is obtained from [2].

```
def postprocess_label(pred_label, threshold=549000):
    """
    Post-process the predicted binary mask by filtering and
    dilating. The filter is determined based on the lower limit
    of the left and right lung volumes, 864 +/- 315 ml and
    1035 +/- 280 ml, respectively.
    Reference:https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4118261/
    Args:
        pred_label: (SimpleITK Image) Binary mask predicted by
                                      lung model
        threshold: (integer) Threshold (in mm3) for filtering lung
                             volumes
    Returns:
       postprocessed_image: (SimpleITK Image) Post-processed image
                                              after filtering
    """
    """
    """
```

7 Lung segmentation in chest X-rays

7.1 Dataset

The lung segmentation model for CXRs was trained on Shenzhen CXR images, which can be downloaded here: Shenzhen dataset.

7.1.1 Dataset structure

The file directory tree of the project is designed as below, with folders for code, data, and network weights.

```
lung_segmentation
├── code
├── data
│   ├── images
│   └── labels
└── weights
```

The lung masks for the Shenzhen CXR images are provided in the following folder: GitHub repository. Samples of CSV files for training and inference are also provided. Users can modify them based on their datasets.

Note: For any new dataset, copy images and labels (masks) to the data folder, create the related CSV files, and perform the training or inference.

7.2 Preprocessing

7.2.1 Setting up a virtual environment

A YML file is provided to install necessary packages for users. Use the terminal or an Anaconda prompt for the following steps:

(1) Create the environment from the environment.yml file:

```
conda env create -f environment.yml
```

(2) Activate the new environment:

```
conda activate Lung-segmentation
```

(3) Verify that the new environment was installed correctly:

```
conda env list
```

For more information, please see https://docs.conda.io/projects/conda/en/latest/user-guide/tasks/manage-environments.html.

7.2.2 Importing necessary python packages

Keras's deep learning framework and the following Python packages were used for lung segmentation:

```
import argparse
import os
import h5py
import cv2
import numpy as np
import pandas as pd
import segmentation_models as sm
from skimage import exposure
from tensorflow.keras import optimizers
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from global_constants import *
import pathlib
```

7.3 Training

7.3.1 Training function for CXR segmentation

The training for lung segmentation is performed by the following function:

```
def train(input_csv_path, lung_segmentation_model_output, EPOCHS,
BATCH_SIZE, LEARNING_RATE):
    '''
    Parameters
    ----------
    input_csv_path: (file_path) Path to CSV file with
    columns 'images' and 'masks' for the training step
    lung_segmentation_model_output: (str) The output path to save the
    lung segmentation model (weights)
    EPOCHS: (positive_int) Epochs for training
    BATCH_SIZE: (positive_int) Batch size for training
    LEARNING_RATE: (float) Learning rate for training
    -------
    '''
    # reading input csv file
    df = pd.read_csv(input_csv_path)
```

```
# number of samples
samples = df.shape[0]
data = np.zeros((samples, SAVE_IMAGE_SIZE[0], SAVE_IMAGE_SIZE[1], 2))
ctr = 0
# reading images and mask with OpenCV (cv2)
for index, row in df.iterrows():
    print(ctr)
    try:
        img = cv2.imread(row["images"], 0)
        print ("reading images -->", row["images"])
        lbl = cv2.imread(row["masks"], 0)
        print ("reading masks -->", row["masks"])
        img = cv2.resize(img, SAVE_IMAGE_SIZE)
        lbl = cv2.resize(lbl, SAVE_IMAGE_SIZE,
                         interpolation=cv2.INTER_NEAREST)
        data[ctr, :, :, 0] = img
        data[ctr, :, :, 1] = lbl
        ctr += 1
    except:
        print ("check this image and its mask -->", row["images"])
# data normalization
x = data[..., 0] / 255.
y = (data[..., 1] / 255.)
# histogram Equalization
for k in range(x.shape[0]):
    x[k, ...] = exposure.equalize_adapthist(x[k, ...])
x = np.stack((x, x, x), axis=3)
x = preprocess_input(255. * x) / 255.
# train-val split
split = int(y.shape[0] * .1)
val_X = x[:split, ...]
val_Y = y[:split, :]
train_X, train_Y = x[split:], y[split:]
```

7.3.2 Input arguments for the training step

Training requires an input CSV file containing column names 'images' and 'masks' with file paths of images and masks for training.

(1) [required] input_csv_path: the path to a CSV file with columns 'images' and 'masks' containing the file paths of images and masks for training.

(2) [optional] –lung_segmentation_model_output: the path to save the lung segmentation model (weights). If not specified, the default path is used: '../weights/Segmentation_resnet50_UNet.h5'.

(3) [optional] –gpu_id: GPU ID on which to train the model

(4) [optional] –BATCH_SIZE: (int) Batch size for training

(5) [optional] –EPOCHS: (int) Number of epochs for training

(6) [optional] –LEARNING_RATE: (float) Learning rate

The default hyper-parameters for training are

```
BATCH_SIZE=16
EPOCHS=100
LEARNING_RATE=0.0001
```

A user can change and train a model with different hyperparameter values.

The weight file (*Segmentation_resnet50_UNet.h5*) is available in the weights folder of this GitHub repository.

7.3.3 Deep learning models for training

Deep learning models are built to work together with Keras and TensorFlow Keras frameworks. The Python library *segmentation_models* was used for image segmentation.

```
import segmentation_models as sm
# Segmentation Models: using `keras` framework.
```

The segmentation model is an instance of Unet Model, which can be built as follows:

```
model = sm.Unet()
```

A user can change the network architecture depending on the task by using different backbones and pre-trained weights for initialization:

```
model = sm.Unet(BACKBONE, encoder_weights=WEIGHTS,
                          encoder_freeze=False)
```

A pre-trained UNet with a ResNet50 backbone was used for this work. The initial weights for the backbone were obtained from this GitHub repository. The deep learning model can be optimized and compiled within the training process as follows

```
# defining the optimizer
opt = optimizers.Adam(learning_rate=LEARNING_RATE)
# compiling the model
model.compile(optimizer=opt,
              loss=sm.losses.bce_jaccard_loss,
              metrics=[sm.metrics.iou_score],
              )
# scheduler and callpoint for saving the output model
earlystopper = EarlyStopping(monitor='val_loss', patience=7, verbose=1,
            restore_best_weights=True)
modelSaver = ModelCheckpoint(lung_segmentation_model_output,
                             monitor='val_loss',
                             verbose=0,
                             save_best_only=True,
                             save_weights_only=False, mode='auto',
                             period=1)
# fitting the model with training data
model.fit(train_X, train_Y, epochs=EPOCHS, batch_size=BATCH_SIZE,
          validation_data=(val_X, val_Y), callbacks=[earlystopper,
                                                     modelSaver])
```

7.3.4 Running the training function for CXR segmentation

Example of how to run a command line for the training function with the required options:

```
python train.py input_csv_path
             --lung_segmentation_model_output  model_output_filename
```

7.4 Inference

7.4.1 Inference function for CXR segmentation

The inference (prediction) process is performed by the following function:

```
def inference(input_csv_path, segmentation_model_path,
output_prediction_directory, threshold):
    '''
    Parameters
    ----------
    input_csv_path: (file_path) Path to a CSV file that contains a
                    column for 'images'
    segmentation_model_path: (file_path) Path for the saved weights
                             from the training step to run the
                             inference
    output_prediction_directory: (str) Folder for saving the
                                 predicted binary images
    threshold: (float) Threshold for the inference prediction
    -------
    '''
    # reading input csv file
    df = pd.read_csv(input_csv_path)
    # Getting the number of images
    samples = df.shape[0]
    data = np.zeros((samples, SAVE_IMAGE_SIZE[0], SAVE_IMAGE_SIZE[1]))
    # reading the images and resizing
    ctr = 0
    list_image_names = list()
    for index, row in df.iterrows():
        print(ctr)
        try:
```

```
            img = cv2.imread(row["images"], 0)
            image_name = os.path.basename(row["images"])
            list_image_names.append(image_name)
            print ("reading images for inference -->", row["images"])
            img = cv2.resize(img, SAVE_IMAGE_SIZE)
            data[ctr, :, :] = img
            ctr += 1
        except:
            print ("check this image -->", row["images"])

    print (list_image_names)
    # normalization
    x = data / 255.
    print (data.shape, "inference mode...")
    print('Data Normalized!')

    # histogram Equalization
    for k in range(x.shape[0]):
        x[k, ...] = exposure.equalize_adapthist(x[k, ...])
    # preprocessing the input
    x = np.stack((x, x, x), axis=3)
    x = preprocess_input(255. * x) / 255.
```

Then, loading the weights derived from the training process allows making predictions and saving the results as follows

```
# load segmentation model
model.load_weights(segmentation_model_path)
# model prediction
seg = model.predict(x)
seg = np.squeeze(seg)
print (seg.shape)
# saving the predicted mask
for i in range(seg.shape[0]):
    img_path = os.path.join(output_prediction_directory,
                            list_image_names[i])
```

```
    print (img_path)
    plt.imsave(img_path, seg[i, :, :])

    originalImage = cv2.imread(img_path)
    grayImage = cv2.cvtColor(originalImage, cv2.COLOR_BGR2GRAY)
    (thresh, blackAndWhiteImage) = cv2.threshold(grayImage, threshold,
                                                 255,
                                                 cv2.THRESH_BINARY)
    cv2.imwrite(img_path, blackAndWhiteImage)
    print ("inference images saved...")
```

7.4.2 Input arguments for the inference step

Inference requires an input CSV file that contains a column 'images' specifying images for inference and an output_path to save binary images (segmented images):

(1) [required] input_csv_path: Path to a CSV file that contains a column of input 'images' for inference

(2) [required] output_prediction_directory: Folder for the predicted binary output images

(3) [optional] –segmnetation_model_path: Path for the saved weights from the training step to run the inference; if not specified, the default path is used: '../weights/Segmentation_resnet50_UNet.h5'

7.4.3 Running the inference function for CXR segmentation

Example of how to run a command line for the inference function with the required options:

```
python inference.py input_csv_path output_prediction_directory
                        --segmnetation_model_path model_path
```

The segmentation result for a sample CXR image is shown in Figure 3.

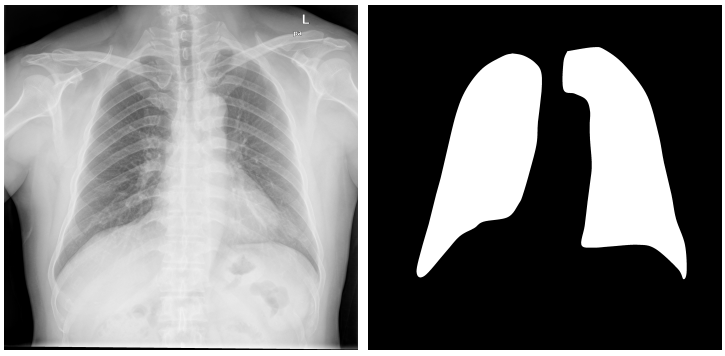

Figure 3: An example CXR (left) and its segmentation (right).

8 Discriminating between TB and NOT-TB using CXRs

8.1 Dataset

The classification model was trained on Shenzhen CXR images, which can be downloaded here Shenzhen Dataset.

8.1.1 Dataset structure

The file directory tree of the project is designed as below, with folders for code, data, and network weights:

```
TB_classification
├── code
├── data
│   └── images
│       ├── test
│       │   ├── NORMAL
│       │   └── TB
│       └── train
│           ├── NORMAL
│           └── TB
└── weights
```

The data folder provides some sample images and inference examples for users. Samples of CSV files for training and inference are also provided. Users can modify them based on their dataset. Users can copy images to the data folder, create the CSV file, and perform the training or inference for a new dataset.

8.2 Preprocessing

8.2.1 Setting up a virtual environment

A YML file is provided to install necessary packages for users. Use the terminal or an Anaconda prompt for the following steps:

(1) Create the environment from the environment.yml file:

```
conda env create -f environment.yml
```

(2) Activate the new environment:

```
conda activate TB-Classification
```

(3) Verify that the new environment was installed correctly:

```
conda env list
```

For more information, please see https://docs.conda.io/projects/conda/en/latest/-user-guide/tasks/manage-environments.html.

8.2.2 Importing necessary python packages

Keras's deep learning framework and necessary Python packages were used for this TB classification task.

```
import argparse
import os
import pathlib
import tensorflow
from tensorflow.keras.preprocessing import image
from tensorflow.keras.models import Model, load_model
from tensorflow.keras.callbacks import ModelCheckpoint, ReduceLROnPlateau
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D, Dropout
from tensorflow.keras.layers import BatchNormalization, Activation
from tensorflow.keras.optimizers import Adam, RMSprop
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
from np import loadtxt
from PIL import Image, ImageOps
from sklearn.model_selection import train_test_split
```

8.3 Training

8.3.1 Training function for TB and NOT-TB classification

The training process can be performed with the training function as follows:

```
def train(images_labels_csv, IMAGE_SIZE, BATCH_SIZE, EPOCHS,
    model_output_path):
  '''
  Parameters
  ----------
  input_csv_path: (file_path) Input CSV has columns for image filenames
                  and their labels
  EPOCHS: (positive_int) Epochs for training
  BATCH_SIZE: (positive_int) Batch size for training
  IMAGE_SIZE: (positive_int) Image size after resizing when training
  model_output_path: (str) Path for saving the model
  ---------
  '''
  # reading the csv file including the images paths and labels
  train_data = pd.read_csv(images_labels_csv)
  train_data['label'] = train_data['label'].astype(str)
  # number of samples
  n_samples = train_data.shape[0]
  Y = train_data[['label']].astype(str)
  # splitting into training and test set
  train_df, val_df = train_test_split(train_data, test_size=0.1)
```

```
input_shape = (IMAGE_SIZE, IMAGE_SIZE, 3)
# train generator
train_datagen = ImageDataGenerator(rescale=1. / 255,
                  height_shift_range= 0.02,
                  width_shift_range=0.02,
                  rotation_range=0.02,
                  shear_range = 0.01,
                  fill_mode='nearest',
                  zoom_range=0.01)
# reading from csv and converting to a data frame
train_generator = train_datagen.flow_from_dataframe(train_df,
                  directory = None,
                  x_col = "filename", y_col = "label",
                  target_size=(IMAGE_SIZE, IMAGE_SIZE),
                  batch_size=BATCH_SIZE,
                  class_mode = "categorical", shuffle = True)
```

8.3.2 Input and output for the training function

Training requires an input CSV file containing columns for 'images' and their 'labels.' A pre-trained model with an Xception backbone was used for classification. The initial weights for the backbone were obtained from the Keras application module.

(1) [required] input_csv_path: Path of a CSV file with columns for 'images' and 'labels' for training.

(2) [optional] –model_output_path: Path to save the classification model (weights). If not specified, the default path is '../weights/inceptionv3_fine_tuned.h5.'

(3) [optional] –IMAGE_SIZE: (int) image size when training

(4) [optional] –BATCH_SIZE: (int) Batch size for training

(5) [optional] –EPOCHS: (int) Number of epochs for training

(6) [optional] LEARNING_RATE: (float) Learning rate

The default hyper-parameters for training are

```
IMAGE_SIZE=299
BATCH_SIZE=16
EPOCHS=40
LEARNING_RATE=0.0001
```

A user can change and train a model with different hyperparameter values.

8.3.3 Deep learning models for training

First, the train generator was used and then fed to the *Xception* network.

```
# using train generator for scaling and other processing
train_datagen = ImageDataGenerator(rescale=1. / 255,
                height_shift_range= 0.02,
                width_shift_range=0.02,
                rotation_range=0.02,
                shear_range = 0.01,
                fill_mode='nearest',
                zoom_range=0.01)
# reading images using train generator
train_generator = train_datagen.flow_from_dataframe(train_df,
                directory = None,
                x_col = "filename", y_col = "label",
                target_size=(IMAGE_SIZE, IMAGE_SIZE),
                batch_size=BATCH_SIZE,
                class_mode = "categorical", shuffle = True)
```

```
# defining and reading a pre-trained Xception model
pretrained_model = tf.keras.applications.Xception(weights='imagenet',
                                                  include_top=False)
pretrained_model.trainable = False
# transfer learning from the pre-trained model
x = pretrained_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(512)(x)
x = BatchNormalization()(x)
x = Activation('relu')(x)
x = Dropout(0.5)(x)
# prediction
predictions = Dense(2, activation='softmax')(x)
model = Model(inputs=pretrained_model.input, outputs=predictions)
# compiling the model
model.compile(loss='categorical_crossentropy',
          optimizer='adam',
          metrics=["accuracy"])
```

8.3.4 Running the training function for TB and NOT-TB classification

Example of how to run a command line for the train function:

```
python train.py images_labels_csv --model_output_path
                                    model_output_filename
```

8.4 Inference

8.4.1 Inference function for TB and NOT-TB classification

The inference (prediction) process is performed by the inference function, as described below.

```
def inference(images_labels_csv, IMAGE_SIZE, model_path,
              prediction_csv):
  '''
    Parameters
    ----------
    images_labels_csv: (file_path) CSV with image filenames
                       for prediction
    IMAGE_SIZE: (positive_int) Image size after resizing
                when training
    model_path: (str) Path for loading the trained model
    prediction_csv: (file_path) Output CSV file for predicted
                    labels
    -------
    '''
    # reading input image labels from CSV file
    test_df = pd.read_csv(images_labels_csv)
    n_samples = test_df.shape[0]
    image_size = IMAGE_SIZE
    batch_size = 1
    input_shape = (IMAGE_SIZE, IMAGE_SIZE, 3)
    # creating a test data generator
    test_datagen = ImageDataGenerator(rescale=1. / 255)
    test_generator = test_datagen.flow_from_dataframe(test_df,
                    directory = None,
                    x_col = "filename",
                    target_size=(image_size, image_size),
                    batch_size=1, class_mode=None)
```

A necessary step is to load the weights from the training process before making the predictions and saving them to a CSV file.

```
# loading the saved model for the inference process
model = tf.keras.models.load_model(model_path)
# prediction
pred=model.predict_generator(test_generator, verbose=1)
predicted_class_indices=np.argmax(pred,axis=1)
# saving the predicted class to a CSV file
filenames=test_generator.filenames
results=pd.DataFrame({"Filename":filenames,
                      "Predictions":predicted_class_indices})
results.to_csv(prediction_csv,index=False)
```

8.4.2 Input and output for the inference function

Inference requires an input CSV file with an 'images' column for inference and an output_path for saving output labels.

(1) [required] input_csv_path: Path to a CSV file with a column for 'images.'

(2) [required] prediction_csv: CSV file path for saving the predicted labels

(3) [optional] –model_path: Path for reading the saved weights from the training step; if not specified, the default path is '../weights/inceptionv3_fine_tuned.h5.'

8.4.3 Running the inference function for TB and NOT-TB classification

Example of how to run a command line for the inference function with the required input parameters:

```
python inference.py input_csv_path prediction_csv
                                    --model_path model_path
```

9 Classifying radiological and genomic data

This section describes software for classification experiments to identify the potential relationship between radiological and genomic data in TB Portals. Radiological information in TB Portals includes manual CXR annotations by radiologists, such as

lung cavity size, nodule size, infiltrate density, mediastinal lymph nodes, and calcified nodules. Genomic information includes a detailed breakdown of the pathogen genomic variance, such as gene mutation variants.

The first objective is to investigate the relationship between radiological and genomic features for drug-resistant TB (DR-TB) and drug-sensitive TB (DS-TB) prediction. This includes the performance accuracy of machine learning algorithms when identifying DR-TB using these features. The second objective is to predict the first successful drug regimen treatment period (TP) using radiological and genomic features.

The following sections describe how to set up the software environment and run the codes for each of the aforementioned objectives.

9.1 Project structure

The following diagram shows the file directory tree for the files and folders.

```
TB_genomics
├── data
│   ├── TB_Portals_Patient_Cases_<month>_<year>.csv
│   ├── TB_Portals_Genomics_<month>_<year>.csv
│   ├── TB_Portals_Regimens_<month>_<year>.csv
│   ├── TB_Portals_DST_<month>_<year>.csv
│   ├── TB_Portals_Sequenced_DST_<month>_<year>.csv
│   ├── TB_Portals_Specimen_<month>_<year>.csv
│   └── TB2258-variantDetail.csv
├── code
│   ├── rg_drds_classification.py
│   ├── rg_tp_regression.py
│   └── helpers.py
├── results
├── environment.yml
└── README.md
```

The *data* folder contains CSV files from TB Portals and TB Profiler [6]. The source codes are placed in the *code* folder. The result files are generated in the *results* folder.

9.2 Setting up a virtual environment

A YML file is provided to install necessary packages for users. Use the terminal or an Anaconda prompt for the following steps:

(1) Create the environment from the environment.yml file:

```
conda env create -f environment.yml
```

(2) Activate the new environment:

```
conda activate radgentb
```

(3) Verify that the new environment was installed correctly:

```
conda env list
```

9.3 Investigating the relationship between radiological and genomic features

This section uses radiological and genomic features to train a Random Forest classifier to predict DR-TB and DS-TB. Sections 9.3.1-9.3.3 describe the source code, preprocessing of CSV files, and encoding of the categorical features. Sections 9.3.4 and 9.3.5 analyze the association between radiological and genomic features using correlation and statistical tests.

9.3.1 Classifying DS-TB/DR-TB with radiological and genomic features

(1) Getting help

```
python rg_drds_classification.py  -h
```

```
usage: rg_drds_classification.py [-h]
        [-rad Add radiological features]
        [-gen Add genomic features]
        [-same Use the same samples]
        [-rp Radiological csv file location]
        [-gp Genomic csv file location]
        [-tp TB Profiler csv file location]
        [-resultp Results path]
```

```
DS-TB/DR-TB classification using radiological & genomic features
optional arguments:
  -h, --help  show this help message & exit
  -rad Add radiological features
  -gen Add genomic features
  -same Specifies the use of samples having both
        radiological & genomic information
  -rp Radiological CSV file location
        Default:../data/TB_Portals_Patient_Cases.csv
  -gp Genomic CSV file location
        Default:../data/TB_Portals_Genomics.csv
  -tp TB Profiler CSV file location
        Default:../data/TB2258-variantDetail.csv
  -resultp result path
        Default:../results/
```

(2) Example of performing classification using both radiological and genomic features

```
python rg_drds_classification.py -rad 1 -gen 1 -same 1
```

(3) Performing classification using only radiological features on the same sample size as in Example (2). Without setting the *-same* flag, the program uses all samples with radiological information in the entire dataset.

```
python rg_drds_classification.py -rad 1 -same 1
```

(4) Performing classification using only genomic features on the same sample size as in Example (2). Without setting the *-same* flag, the program uses all samples with genomic information in the entire dataset.

```
python rg_drds_classification.py -gen 1 -same 1
```

(5) Pointing to the radiological CSV file in */data/TB_Portals_Patient_Cases.csv* by setting up the *-rp* flag. Similarly, users can point to the genomic or TB Profiler data locations using the *-gp* and *-tp* flags.

```
python rg_drds_classification.py -rad 1 -gen 1 -same 1
            -rp '/data/TB_Portals_Patient_Cases_<month>_<year>.csv'
```

9.3.2 Preprocessing radiological and genomic CSV files

The software uses 25 columns with radiological features from the clinical spreadsheet (TB_Portals_Patient_Cases.csv). The definition of each column is explained on the following TB Portals webpage: https://datasharing.tbportals.niaid.nih.gov/#/about-the-data.

```
overall_percent_of_abnormal_volume,
pleural_effusion_percent_of_hemithorax_involved,
ispleuraleffusionbilateral, other_non_tb_abnormalities,
are_mediastinal_lymphnodes_present, collapse,
smallcavities, mediumcavities, largecavities,
isanylargecavitybelongtoamultisextantcavity,
canmultiplecavitiesbeseen, infiltrate_lowgroundglassdensity,
infiltrate_mediumdensity, infiltrate_highdensity,
smallnodules, mediumnodules, largenodules, hugenodules,
isanycalcifiedorpartiallycalcifiednoduleexist,
isanynoncalcifiednoduleexist, isanyclusterednoduleexists,
aremultiplenoduleexists, lowgroundglassdensityactivefreshnodules,
mediumdensitystabalizedfibroticnodules,
highdensitycalcifiedtypicallysequella
```

The genomic column in the genomic spreadsheet (TB_Portals_Genomics.csv) used by the software is listed below. This column contains nine variants from the TB Portals genomic spreadsheet.

```
gene_snp_mutations
```

All columns used as genomic features by the software in the TB Profiler spreadsheet (TB2258-variantDetail.csv) are listed below. The spreadsheet has 2218 columns; however, only 34 variant names are shown. These names are followed by sub-names (marked as *), which are the nucleotide or amino acid changes in the gene.

```
ahpC*, ald*, alr*, ddn*, eis*, embA*, embB*, embC*, embR*,
ethA*, fabG1*, fbiA*, folC*, gid*, gyrA*, gyrB*, ethR*,
inhA*, kasA*, katG*, mmpR5*, panD*, pncA*, ribD*, rplC*,
rpoB*, rpoC*, rpsA*, rpsL*, rrl*, rrs*, thyA*, thyX*, tlyA*
```

The unused columns are grouped into the following fields, which will be removed in the *preprocess_csv* function.

```
CT_fields, CXR_fields, CLINICAL_merged_fields, CLINICAL_fields,
TCS_fields, GENOMIC_fields, QURE_fields, TBP_variants_fields,
DST_fields
```

The following column is used as a label in the regression. This column can be found in the clinical spreadsheet (TB_Portals_Patient_Cases.csv).

```
type_of_resistance
```

The *preprocess_csv* function merges the TB Portals clinical and genomic CSV files, with Poly DR and Pre-XDR removed. The *Not Reported* rows are also removed.

```
def preprocess_csv(args, r_df, g_df, t_df, out_folder):
    """
    This function preprocesses the CSV files. It merges the
    CSV rows and removes the unused columns.
    Args:
        args: (parser.parse_args()) Input from the command line
        r_df, g_df, t_df: (pandas.DataFrame) Read from clinical,
        genomic, TB Profiler spreadsheets
        out_folder: (string) Path to output the results
    Returns:
        df: (pandas.DataFrame) Returned Pandas data frame
    """

# How to call 'preprocess_csv' in the 'main' function
df = preprocess_csv(args, r_df, g_df, t_df, out_folder)
```

9.3.3 Categorical feature encoding

The *encoding* function converts the categorical data to a numeric format before passing the data to the classification model. Because the categorical features used are all nominal, one-hot encoding is used.

To encode the radiological features, columns such as small/medium/large cavities, low ground glass density/medium density/high density infiltrates, small/medi-

um/large/huge nodules are grouped into cavities, infiltrates, and nodules. Therefore, there are 18 radiological features after encoding.

To encode the *gene_snp_mutations* column, the variants are comma-delimited in this column. A total of nine unique variants are observed after encoding. To encode the genomic features in the TB Profiler spreadsheet, 2218 columns are grouped into 34 features by combining the main variant names. Therefore, a total of 43 genomic features are used to train.

```
def encoding(args, df, r_df, g_df, t_df, out_folder):
    """
    This function converts categorical features to a numeric type.
    Args:
        args: (parser.parse_args()) Input from the command line
        df, r_df, g_df, t_df: (pandas.DataFrame) Pandas data frames
        out_folder (string): Output path for the results
    Returns:
        df (pandas.DataFrame): Returned Pandas data frame
    """

# How to call 'encoding' in the 'main' function
df = encoding(args, df, r_df, g_df, t_df, out_folder)
```

9.3.4 Computing the correlation between radiological and genomic features

The function below computes the correlation between each feature in the radiological and genomic data.

```
def correlation(df, out_folder):
    """
    Compute pairwise correlations of columns in a Pandas data frame,
    excluding NA/null values.
    Args:
        df: (pandas.DataFrame) Pandas data frame with features
                               to calculate the correlation
        out_folder: (string) Output path for the results
    Returns:
        None
    """

# How to call 'correlation' in the 'main' function
correlation(df, out_folder)
```

9.3.5 Computing the statistical significance of radiological and genomic features

Pearson's chi-squared test measures the statistical significance of radiological and genomic features regarding the resistance type. Null hypothesis: Features are independent of the resistance type. A p-value of less than 0.05 is considered statistically significant.

```
def chi2(df, TOR, out_folder):
    """
    This function computes the chi-square statistic and p-value
    for the hypothesis test of independence.
    Args:
        df: (pandas.DataFrame) Pandas data frame with features
                               to calculate the chi2 test
        TOR: (string) Variable containing the column name for
                      the resistance type
        out_folder: (string) Output path for the results
    Returns:
        None
    """

# How to call 'chi2' in the 'main' function
chi2(df, TOR, out_folder)
```

9.3.6 Selecting balanced data for classification

The following code ensures a balanced training set of DR-TB and DS-TB cases.

```
"""Select balanced data"""
TYPE = 'balanced'
if TYPE == 'balanced':
    S = df.loc[df[TOR] == 'Sensitive']
    DR = df.loc[df[TOR] == 'DR']
    DR = DR[:S.shape[0]]
    #DR = DR[S.shape[0]*3:S.shape[0]*4+1]
    df = pd.concat([S, DR])
    df.to_csv(out_folder + '/balanced.csv', index=True)
else:
    df.to_csv(out_folder + '/unbalanced.csv', index=True)
```

9.3.7 Classifying DR-TB/DS-TB based on radiological and genomic features

The *classifier* function uses Random Forest to classify DR-TB and DS-TB using a five-fold cross-validation scheme.

```
def classifier(df, out_folder):
    """
    This function predicts drug-resistant and drug-sensitive TB
    using a Random Forest classifier technique.
    Five-fold cross-validation is used.
    ROC and accuracy are computed to evaluate the model
    Args:
        df: (pandas.DataFrame) Pandas data frame to use
        out_folder: (string) Output path for the results
    Returns:
        None
    """

# How to call 'classifier' in the 'main' function
classifier(df, out_folder)
```

9.4 Investigating the relationship between the radiological and genomic features with the treatment period

A Gradient Boosting regression model is used to predict the length of the first successful drug regimen using both radiological and genomic features. The software also predicts the treatment period of a specific single drug regimen for the first treatment. A Gradient Boosting regression model is trained on the most popular drug combination in the TB Portals data.

9.4.1 Running the TB treatment period regression model using radiological and genomic features

(1) Getting help

```
python rg_tp_regression.py -h
```

```
usage: rg_tp_regression.py [-h]
        [-rad Add radiological features]
        [-gen Add genomic features]
        [-same Use the same samples]
        [-pdrug Predict treatment period of
            specific popular drug combination]
        [-rp Radiological csv file location]
        [-gp Genomic csv file location]
        [-tp TB Profiler csv file location]
        [-dstp DST csv file location]
        [-regimenp Regimen csv file location]
        [-resultp results path]
```

```
Predicting the treatment period using radiological & genomic
features

optional arguments:
  -h, --help  show this help message & exit
  -rad Add radiological features
  -gen Add genomic features
  -same Specifies the use of samples having both
        radiological & genomic information.
  -pdrug Predict the treatment period of a drug combination
  -rp Radiological CSV file location
        Default:../data/TB_Portals_Patient_Cases.csv
  -gp Genomic CSV file location
        Default:../data/TB_Portals_Genomics.csv
  -tp TB Profiler CSV file location
        Default:../data/TB2258-variantDetail.csv
  -dstp DST CSV file location
        Default:../data/TB_Portals_DST.csv
  -regimenp Regimen CSV file location
        Default: ../data/TB_Portals_Regimens.csv
  -resultp Result path
        Default:../results/
```

(2) The following example shows how to perform regression using both radiological and genomic features to predict the treatment period of all drug combinations (*-pdrug* flag is 0).

```
python rg_tp_regression.py -rad 1 -gen 1 -same 1 -pdrug 0
```

(3) This example shows how to use both radiological and genomic features to predict the treatment period of the most popular drug combination (*-pdrug* flag is 1).

```
python rg_tp_regression.py -rad 1 -gen 1 -same 1 -pdrug 1
```

(4) Using only radiological features on the same sample size as in Example (2):

Without setting the *-same* flag, the program uses all samples with radiological information in the entire dataset.

```
python rg_tp_regression.py -rad 1 -same 1 -pdrug 1
```

(5) Similarly, the following command line can be used when performing regression using only genomic features on the same sample size as in Example (2). Without setting the *-same* flag, the program uses all the samples with genomic information in the entire dataset.

```
python rg_tp_regression.py -gen 1 -same 1 -pdrug 1
```

9.4.2 Preprocessing radiological, genomic, DST, and regimen CSV files

The software uses 25 columns with radiological features from the clinical spreadsheet (TB_Portals_Patient_Cases.csv). The definition of each column is explained on the following TB Portals webpage: https://datasharing.tbportals.niaid.nih.gov/#/about-the-data.

```
overall_percent_of_abnormal_volume,
pleural_effusion_percent_of_hemithorax_involved,
ispleuraleffusionbilateral, other_non_tb_abnormalities,
are_mediastinal_lymphnodes_present, collapse,
smallcavities, mediumcavities, largecavities,
isanylargecavitybelongtoamultisextantcavity,
canmultiplecavitiesbeseen, infiltrate_lowgroundglassdensity,
infiltrate_mediumdensity, infiltrate_highdensity,
smallnodules, mediumnodules, largenodules, hugenodules,
isanycalcifiedorpartiallycalcifiednoduleexist,
isanynoncalcifiednoduleexist, isanyclusterednoduleexists,
aremultiplenoduleexists, lowgroundglassdensityactivefreshnodules,
mediumdensitystabalizedfibroticnodules,
highdensitycalcifiedtypicallysequella
```

The genomic column in the genomic spreadsheet (TB_Portals_Genomics.csv) used

by the software is listed below. This column contains nine variants from the TB Portals genomic spreadsheet.

```
gene_snp_mutations
```

All columns used as genomic features by the software in the TB Profiler spreadsheet (TB2258-variantDetail.csv) are listed below. The spreadsheet has 2218 columns; however, only 34 variant names are shown. These names are followed by sub-names (marked as *), which are the nucleotide or amino acid changes in the gene.

```
ahpC*, ald*, alr*, ddn*, eis*, embA*, embB*, embC*, embR*, ethA*,
fabG1*, fbiA*, folC*, gid*, gyrA*, gyrB*, ethR*, inhA*, kasA*,
katG*, mmpR5*, panD*, pncA*, ribD*, rplC*, rpoB*, rpoC*, rpsA*,
rpsL*, rrl*, rrs*, thyA*, thyX*, tlyA*
```

The unused columns are grouped into the following fields, which will be removed in the *preprocess_csv* function.

```
CT_fields, CXR_fields, TCS_fields, GENOMIC_fields, QURE_fields,
TBP_variants_fields, DST_merged_fields, RAD_merged_fields,
CLINICAL_fields, RAD_fields, DST_fields, REGIMEN_merged_fields,
REGIMEN_fields
```

The following column is used as a label in the regression. This column can be found in the clinical spreadsheet (TB_Portals_Patient_Cases.csv).

```
period_span
```

The *preprocess_csv* function merges the TB Portals clinical and genomic CSV files, with Poly DR and Pre-XDR removed. Unused columns and *Not Reported* rows are also removed.

```
def preprocess_csv(args,r_df,g_df,t_df,dst_df,regi_df,out_folder):
    """
    This function preprocesses the CSV files. It merges the
    CSV rows and removes the unused columns.
    Args:
        args: (parser.parse_args()) Input from the command line
        r_df, g_df, t_df, dst_df, regi_df: (pandas.DataFrame)
        Read from clinical, genomic, TB Profiler, DST profile,
        and Regimen CSV spreadsheet.
        out_folder: (string) Path to output the results
    Returns:
        df: (pandas.DataFrame) Returned Pandas data frame
    """

# How to call 'preprocess_csv' in the 'main' function
df = preprocess_csv(args,r_df,g_df,t_df,dst_df,regi_df,out_folder)
```

9.4.3 Popular drug combinations

The *popular_drug* function finds the most popular drug combinations and computes the conditional probabilities between the outcome, regimen_count, and regimen_drug. To predict the treatment period of a specific drug combination, *-pdrug* must be set to 1.

```
def popular_drug(args, df, out_folder):
    """
    This function finds the top 20 most popular drug combinations
    and computes the conditional probabilities between the outcome,
    regimen_count, and regimen_drug
    Args:
        args: (parser.parse_args()) Input from the command line
        df: (pandas.DataFrame) Pandas data frame to use
        out_folder: (string) Path to output the results
    Returns:
        df (pandas.DataFrame): Returned Pandas DataFrame
    """

# How to call 'popular_drug' in the 'main' function
if args.pdrug[0] == 1:
    df = popular_drug(args, df, out_folder)
```

9.4.4 Categorical feature encoding

The *encoding* function converts the categorical data to a numeric format before passing the data to the classification model. Because the categorical features used are all nominal, one-hot encoding is used.

To encode the radiological features, columns such as small/medium/large cavities, low ground glass density/medium density/high density infiltrates, and small/medium/large/huge nodules are grouped into cavities, infiltrates, and nodules. Therefore, there are 18 radiological features after encoding.

To encode the *gene_snp_mutations* column, the variants are comma-delimited in this column. A total of nine unique variants are observed after encoding. To encode the genomic features in the TB Profiler spreadsheet, 2218 columns are grouped into 34 features by combining the main variant names.

```
def encoding(args, df, r_df, g_df, t_df, out_folder):
    """
    This function converts categorical features to a numeric
    type.
    Args:
        args: (parser.parse_args()) Input from the command line
        df, r_df, g_df, t_df: (pandas.DataFrame) Pandas data frame
        to use
        out_folder: (string) Output path for the results
    Returns:
        df: (pandas.DataFrame) Returned Pandas data frame
    """

# How to call 'encoding' in the 'main' function
df = encoding(args, df, r_df, g_df, t_df, out_folder)
```

9.4.5 Computing the correlation between radiological and genomic features

To compute the feature correlations, refer to Section 9.3.4.

9.4.6 Computing the statistical significance of radiological and genomic features regarding treatment period

Pearson's chi-squared test measures the statistical significance of radiological and genomic features regarding the treatment period. Null hypothesis: Features are independent of the treatment period. A p-value of less than 0.05 is considered statistically significant. To perform Pearson's chi-squared test, refer to Section 9.3.5.

9.4.7 Predicting the TB treatment period based on radiological and genomic features

The *regressor* function uses a Gradient Boosting Regressor to predict the treatment period. The model is trained with five-fold cross-validation. The mean, median, and relative errors are used to evaluate the model.

```
def regressor(df, out_folder):
    """
    This function predicts the treatment period using the
    Gradient Boosting Regression technique. Five-fold
    cross-validation for mean, median, and relative errors
    are used to evaluate the model.
    Args:
        df: (pandas.DataFrame) Pandas data frame to use
        out_folder: (string) Path to output the results
    Returns:
        None
    """

# How to call 'regressor' in the 'main' function
regressor(df, out_folder)
```

References

[1] H. J. Aerts, E. R. Velazquez, R. T. Leijenaar, C. Parmar, P. Grossmann, S. Carvalho, J. Bussink, R. Monshouwer, B. Haibe-Kains, D. Rietveld, et al. Decoding tumour phenotype by noninvasive imaging using a quantitative radiomics approach. *Nature communications*, 5(1):1–9, 2014.

[2] J. Fleming, J. Conway, C. Majoral, M. Bennett, G. Caillibotte, S. Montesantos, and I. Katz. Determination of regional lung air volume distribution at mid-tidal breathing from computed tomography: a retrospective study of normal variability and reproducibility. *BMC Medical Imaging*, 14(1):1–14, 2014.

[3] S. Jaeger, S. Candemir, S. Antani, Y.-X. J. Wang, P.-X. Lu, and G. Thoma. Two public chest X-ray datasets for computer-aided screening of pulmonary diseases. *Quant Imaging Med Surg*, 4(6):475–477, 2014.

[4] M. Jun, G. Cheng, W. Yixin, A. Xingle, G. Jiantao, Y. Ziqi, Z. Minqing, L. Xin, D. Xueyuan, C. Shucheng, et al. COVID-19 CT Lung and Infection Segmentation Dataset. Apr. 2020.

[5] Y. Liu, Y.-H. Wu, Y. Ban, H. Wang, and M.-M. Cheng. Rethinking computer-aided tuberculosis diagnosis. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2643–2652, 2020.

[6] J. E. Phelan, D. M. O'Sullivan, D. Machado, J. Ramos, Y. E. Oppong, S. Campino, and J. O'Grady. Integrating informatics tools and portable sequencing technology for rapid detection of resistance to anti-tuberculous drugs. *Genome medicine*, 11:1–7, 2019.

[7] A. Rosenthal, A. Gabrielian, E. Engle, D. E. Hurt, S. Alexandru, V. Crudu, E. Sergueev, V. Kirichenko, V. Lapitskii, E. Snezhko, et al. The TB Portals: an open-access, web-based platform for global drug-resistant-tuberculosis data sharing and analysis. *Journal of Clinical Microbiology*, 55(11):3267–3282, 2017.

[8] R. D. Rudyanto, S. Kerkstra, E. M. Van Rikxoort, C. Fetita, P.-Y. Brillet, C. Lefevre, W. Xue, X. Zhu, J. Liang, I. Öksüz, et al. Comparing algorithms for automated vessel segmentation in computed tomography scans of the lung: the VESSEL12 study. *Medical image analysis*, 18(7):1217–1232, 2014.

[9] A. A. A. Setio, A. Traverso, T. De Bel, M. S. Berens, C. Van Den Bogaard, P. Cerello, H. Chen, Q. Dou, M. E. Fantacci, B. Geurts, et al. Validation, comparison, and combination of algorithms for automatic detection of pulmonary nodules in computed tomography images: the LUNA16 challenge. *Medical image analysis*, 42:1–13, 2017.

[10] K. Wollenberg, B. Jeffrey, M. Harris, A. Gabrielian, D. Hurt, and A. Rosenthal. Patterns of genomic interrelatedness of publicly available samples in the TB Portals database. *Tuberculosis*, 133:102171, 2022.

[11] F. Yang, H. Yu, K. Kantipudi, M. Karki, Y. M. Kassim, A. Rosenthal, D. E. Hurt, Z. Yaniv, and S. Jaeger. Differentiating between drug-sensitive and drug-resistant tuberculosis with machine learning for clinical and radiological features. *Quant Imaging Med Surg*, 12(1):675–687, 2022.

[12] J. Yang, H. Veeraraghavan, W. van Elmpt, A. Dekker, M. Gooding, and G. Sharp. CT images with expert manual contours of thoracic cancer for benchmarking auto-segmentation accuracy. *Medical physics*, 47(7):3250–3255, 2020.

[13] P. Zaffino, A. Marzullo, S. Moccia, F. Calimeri, E. De Momi, B. Bertucci, P. P. Arcuri, and M. F. Spadea. An open-source COVID-19 CT dataset with automatic lung tissue classification for radiomics. *Bioengineering*, 8(2):26, 2021.