AusCAT: Distributed Machine Learning Framework

Matthew Field[a,b,c]

[a] *University of New South Wales, Sydney, Australia*
[b] *Ingham Institute of Applied Medical Research, Sydney, Australia*
[c] *Liverpool and Macarthur Cancer Therapy Centres, Australia*

Abstract

This document outlines the components in a proposed distributed machine learning framework for the development and validation of prediction models in the context of radiation oncology. We describe the components that are installed in each institution, how they interact with the local databases to produce standardized data sets and how they participate in distributed learning with a network of other equivalently configured institutions. A usage guide, installation instructions and example algorithms are presented for reference.

Contents

1. Introduction

Distributed machine learning is an approach for developing and validating predictive models across data kept in different physical locations. In the context of medical databases, this is to address patient confidentiality, privacy, administrative and data management challenges with centralising patient data, particularly when this involves large volumes of medical imaging. Many predictive models can be reformulated to solve the required computations in parallel, sequentially or by an iterative consensus over local models [1]. The steps are all constructed such that no individual patient data needs to be transferred between institutes. Instead the messages transmitted across the network generally consist of model parameters, aggregates over patient groups for constructing probability distributions, or histogram bins.

Previous work in implementing distributed learning systems in a radiation oncology setting have employed an infrastructure co-developed by MAASTRO Clinic and Varian Medical Systems (Palo Alto, CA) [2, 3]. The software presented here is devised as an additional open-source project towards enabling the independent administration of data analysis projects across research groups in a secure, modular and concise framework.

2. Intended Usage

The distributed learning web framework is intended for communication purposes. It serves the deployment of machine learning algorithms which read only the approved datasets into the local data centre memory for statistical computations necessary for learning. It also serves as a subsequent medium for communication of messages such as derived model parameters, aggregated statistics over local data, sufficient statistics for probability distributions, normalisation constants, algorithm settings or model hyperparameters. It is

not intended for the transmission of individualised data. If this system, or derivatives thereof, are used for research, we ask that the appropriate white papers describing this system are acknowledged or referenced in the research output. This software is released under MIT license.

3. Infrastructure Overview

The distributed learning system presented here consists of a network of nodes connected in a star layout. A central communication server is connected to each node that has direct access to data. This communication server acts as a gateway to provide a mechanism serving two purposes, to securely send algorithms and to send messages summarising the data in the form of statistics or models of the data. Each data node has a research data governance structure ensuring that only de-identified data is analysed by the algorithm. Another central terminal is used to coordinate the algorithms and collect model messages from the nodes via the communication server.

An overview of the system is illustrated in Figure 1. On the communication server a web service ('ServerNode') hosts the status of the network and each data centre function ('DataNode') polls this service for a submitted algorithm or an indicator to activate an algorithm. A user at the central terminal accesses a web page ('WebClient') to upload an algorithm and coordinate the execution across the network by selecting the relevant participants. An additional web service is available as a Message Passing Interface (MPI), which used by the algorithm to send aggregates, statistics or parameters relating to model development and validation. Once a local algorithm is deployed to each node a central algorithm is executed on the central terminal to broadcast model information to the nodes and to receive iterative updates from the nodes through the MPI system. This central algorithm collates the statistics received from the participating clinics and produces a model. The messaging protocol through the MPI is chosen in designing the algorithm and caution is taken in designing the scope of messages that can be sent from each node.

All communications are encrypted with HTTPS/TLS packets which requires both server and client authentication. Certificates are requested from clients upon connection and only approved clients or nodes can communicate to the central server since it hosts additional capabilities to control the network operation and information about the connected data centres.

Information transferred between each of the processes in the network can be categorised into:

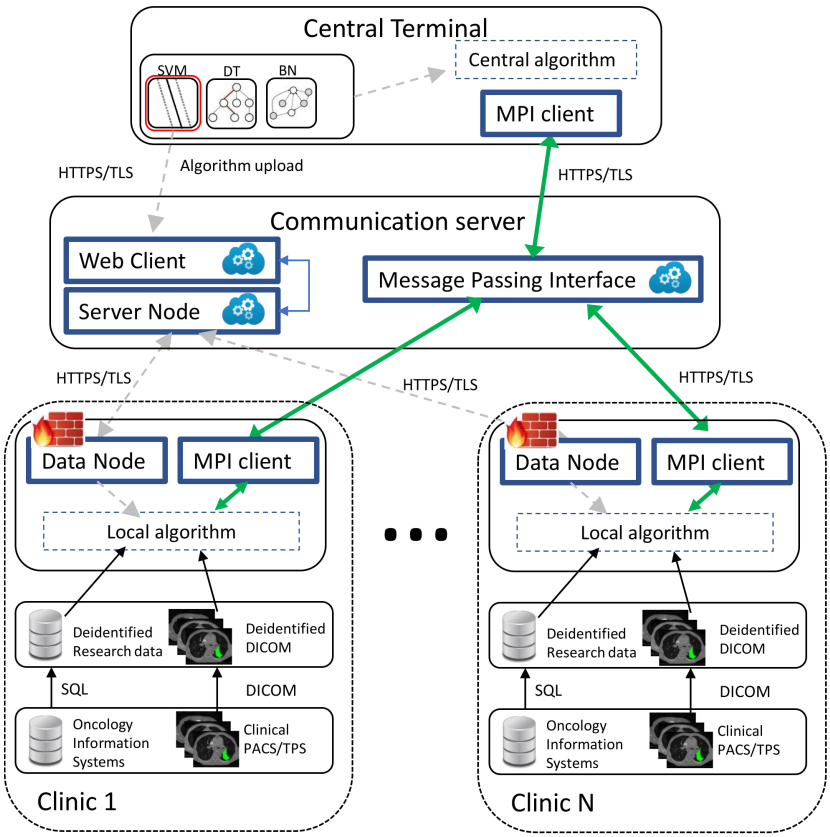

Figure 1: Overview of Computer Assisted Theragnostics (CAT) distributed machine learning infrastructure. (a) Each clinic has data extraction systems installed. (b) The learning system links each of the clinical datasets by passing parameters through an MPI and produces a consensus model in the central terminal.

- **Status of the network**. Includes status flags if the node is active, currently executing an algorithm, sends the permissions to contribute and to be listed as node on the network. The server sends confirmation, the allocated index in the network, the status of server such as whether an algorithm is currently running and names/signatures of current algorithm or previously executed algorithms.

- **Algorithm code**. The communication server receives a compiled algorithm from the central terminal through the web page upload function. The data centre downloads the algorithm binary and can trigger the execution. A flag is set on the communication server through the web

page to enable execution of the algorithm.

- **Model messages**. Set up parameters of the algorithm such as model validation settings, features selected, normalisation constants, imputation model parameters, hyperparameters defining structure of the model, state of the algorithm defining which messages to send. Model related aggregated statistics, model parameters, or descriptive statistics describing patient cohort data or sub cohorts.

Each of the coordinating functions are run as web applications within a Apache Tomcat container. The local algorithm executable is assumed to be a compiled data analysis software that can execute locally with the appropriate libraries in Java, Matlab, python, R or other programming languages installed in advance.

3.1. Data centre

A data centre connected to the CAT system will contain data extractions from the local Oncology Information Systems (OIS), medical imaging and radiotherapy treatment planning data stored in a picture archiving and communication system (PACS), and linked external data sources (for example, obtained through state/national registry linkage). The oncology data is deidentified and stored in various formatted databases: a relational database, an ontology-based resource description framework (RDF) storage, and temporary files for distributed learning such as comma separated variable files. The medical imaging is deidentified with an RSNA CTP service and stored in a local PACS and also other binary formats for efficient analysis. An identifier database is maintained separately for re-identification purposes in local data quality assurance projects [4].

Two processes communicate externally to the distributed learning communications server. A data node function is periodically polling the server for submitted algorithms. An MPI client web service hosted locally delivers messages to and from the server MPI. This function produces a log of messages sent from the system and encrypts all packets. Both processes are modularised such that multiple projects can be coordinated in parallel or pointed to different project specific services as explained in 4.4.

3.2. Central server

To coordinate the algorithm execution the central terminal is given authorisation to access a web page displaying the network status and mechanisms

for uploading local algorithms. A central algorithm is executed controlling the sequence of requests from each data node through the MPI to develop or validate a model.

This central program can be coordinated from one of the connected nodes in the network or from a separate system that is given appropriate credentials and presents a valid certificate to connect to each of the web services on the server.

3.3. Communications server

The communications server is designed to be administered by each data analysis project coordinator. A list of project participants is defined in the configuration of each server node web service. If multiple projects are hosted on the server, there will be dedicated web page per project with the appropriate permissions.

4. Usage Guide

4.1. Set up

An installation of Java JDK 1.8 (64-bit) and Apache Tomcat 9 (64-bit). To use the example algorithm code described in this document either a MATLAB installation or MATLAB Compiler Runtime for version R2017b is required. Executables provided are currently compiled and tested under R2017b.

Download the archive containing the following files from `https://bitbucket.org/swscsmedphys/distributedlearning/downloads/`

- `DLDataNode.war`
- `MPIClient.war`
- `generateImportCertificates.bat`
- `DLDataNode.properties`
- `MPIClient.properties`
- `algorithm.properties`
- `server-public.cer`

Inside the batch script, `generateImportCertificates.bat`, enter the data centre name and password to be used for key storage. Once ensuring the Java 1.8 64-bit `bin` folder is in the system environment path variable, run the batch file `generateImportCertificates.bat` to generate private and public keys for the data centre.

Create a directory for the configuration of the system in `*.properties` (eg. C:\DistributedLearning\properties). To set this as the base directory open <Tomcat directory>\bin\tomcat9w.exe, navigate to the Java tab and insert the newly created directory in the Java options with '-Dproperty.home= C:\DistributedLearning\properties\'. Create a directory to store the private keys and trusted certificates (eg. <directory>\keystore\). Enter this path into `*.properties` along with the key store password, details on data centre name, location of the data sets, WSDL links for the project (provided by server administrator).

Deploy the two web application files (`DLDataNode.war` and `MPIClient.war`) to the installed Tomcat service. This can be achieved on the standard Tomcat manager page interface, `http://localhost/manager/html`.

Import server and client certificates into a web browser. Use the `*.p12` file generated by the batch script as the client certificate. Contact provider of `server-public.cer` and send the data centre public key to be imported and accepted by the server.

4.2. Operating algorithm sender

Visit the web site `https://<IP_address>/<project_name>/DLWebClient` and navigate to the manager page. This manager page should display a list of data centres connecting to the server as shown below in Figure 2.

To deploy an algorithm the target centres should be selected with the check box on the first column. Initially the data centres display an inactive status but once selected will indicate the system is waiting for the deployment of a task. Above the table list, browse for an algorithm file (eg. `LocalGaussian.exe`) and select upload. Once the file is received the corresponding data centre will display "Ready to start". To remotely trigger an execution of the local algorithms that have been deployed select the start algorithm button. The console output of each local algorithm is displayed in the last column.

If data centres cannot establish connection they will not appear on the list. In this case the server can be refreshed or if there is still no connection

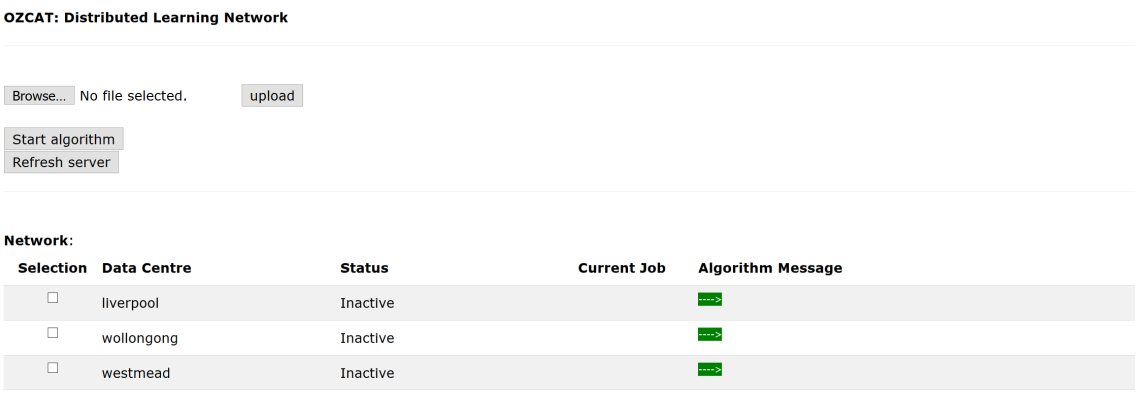

Figure 2: Display of the list of nodes connected to the distributed learning network.

the `DLDataNode` web application in the local Tomcat install may need to be reloaded.

4.3. Operating message passing interface - algorithm execution

To start the system the coordinating function on the central server needs to be executed. Examples of central algorithms are presented in 5. Each are programmed with MATLAB, however if this is not available compiled executables are supplied for the MATLAB Compiler Runtime R2017b. If the local algorithms have been deployed and started, as in the previous section, each will be attempting to establish a connection to the server MPI.

In the current framework, the recommended practice is to start the central algorithm first to initialise the MPI status then trigger the local algorithm to begin. If not the local algorithms may retrieve the last available message on the MPI. The operation of the algorithm can be tracked through the project web site. The final column of the manager table can be expanded for each data node to observe the output, including any error messages, of each local algorithm.

4.4. Parallel projects

Each project is allocated a unique WSDL link on the communication server. For example, the WSDL for a project server node will be `https://<IP_address>/DL_<project_name>/ServerNode?wsdl` and for the corresponding MPI, `https://<IP_address>/MPI_<project_name>/MPI?wsdl`. The project that the local 'DLDataNode' and 'MPIClient' functions are

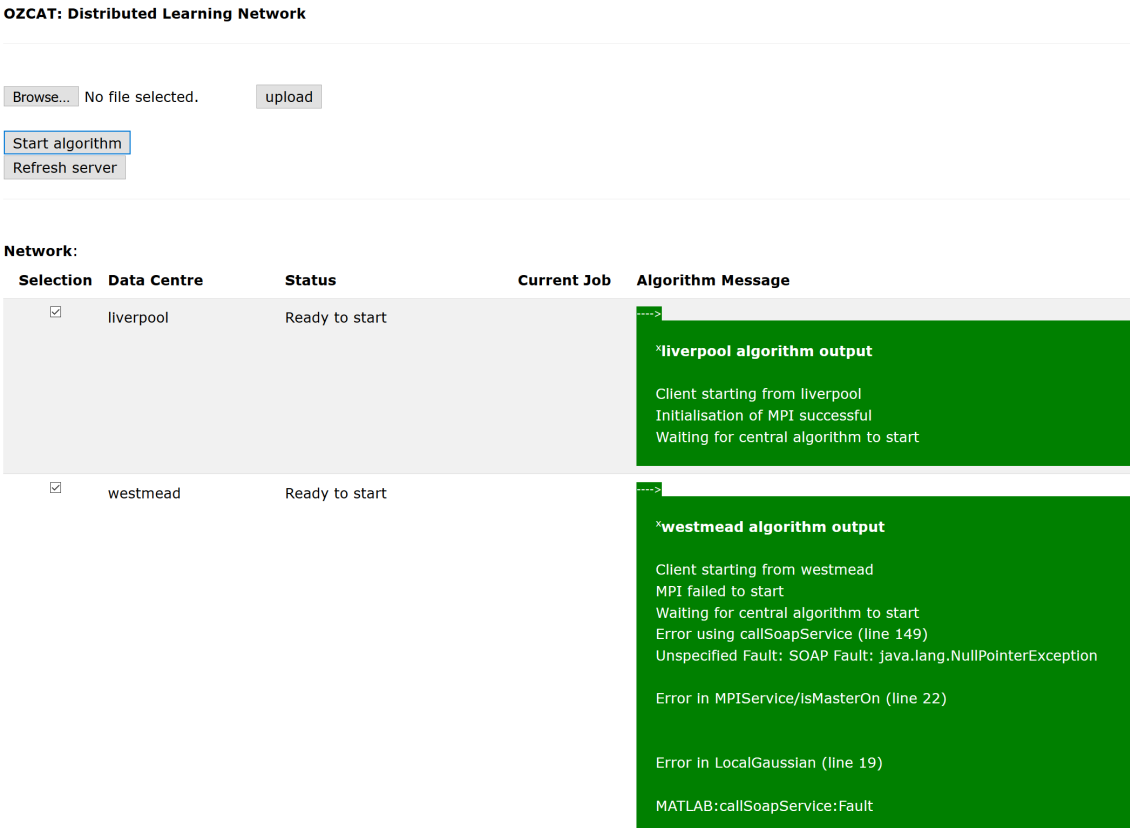

Figure 3: Once algorithms are downloaded by each node and started the progress is displayed in the last expandable column. Here, one system failed to establish connection to the MPI.

linked to can be set in the corresponding `*.properties` file. Furthermore, additional 'DLDataNode' and 'MPIClient' functions can be deployed in the Apache Tomcat container by changing the file name and creating a configuration file for each process. Each pair of functions should communicate to one project exclusively therefore multiple equivalent processes from the same node should not communicate on the same project.

As an example, create a copies of `DLDataNode.war` and `MPIClient.war` to be named 'DLDataNode1' and 'DLDataNode2', and 'MPIClient1' and 'MPIClient2'. Create configuration files for each of these as `DLDataNode1.properties` where the WSDL in the file is set to the appropriate project, that is `https://<IP_address>/DL_<project1>/ServerNode?wsdl`. Deploy each of the *.war files into the local Tomcat container. Each project can be admin-

istered independently through separate URLs designated for the project `https://<IP_address>/<project1>/DLWebClient`.

5. Example Algorithms

5.1. Summary statistics - sample mean and covariance

As a minimal example we implement the collection of summary statistics from distributed data sets. In this case, a Gaussian distribution defined by mean and covariance $\{\boldsymbol{\mu}, \boldsymbol{\Sigma}\}$ is computed across two data sets.

Let $\mathbf{X} = [\mathbf{x}_1, ..., \mathbf{x}_i, ..., \mathbf{x}_N]$ denote a data matrix where each $\mathbf{x}_i \in \mathbb{R}^d$ is a vector of values for a total of N data points. Firstly, the mean $\boldsymbol{\mu}$ is computed by passing the sum over local $\mathbf{x}$ and the number of data points per data set to the central algorithm,

$$\boldsymbol{\mu} = \frac{1}{N_1 + N_2}\Bigg(\overbrace{\sum_i^{N_1} \mathbf{x}_{1i}}^{\text{center 1}} + \overbrace{\sum_i^{N_2} \mathbf{x}_{2i}}^{\text{center 2}}\Bigg), \tag{1}$$

where $\mathbf{x}_{1i}$ is the ith vector and N_1 the number of vectors in data set 1. Similarly, to compute covariance, the combined mean is first broadcast to the each data center to collect local covariance statistics which are then in turn sent back to be combined centrally,

$$\boldsymbol{\Sigma} = \frac{1}{N_1 + N_2}\Bigg(\sum_i^{N_1} (\mathbf{x}_{1i} - \boldsymbol{\mu})(\mathbf{x}_{1i} - \boldsymbol{\mu})^T + \sum_i^{N_2} (\mathbf{x}_{2i} - \boldsymbol{\mu})(\mathbf{x}_{2i} - \boldsymbol{\mu})^T\Bigg). \tag{2}$$

An implementation of this computation is provided in the example repository with local and central algorithms in `LocalGauss.m` and `CentralGauss.m` respectively.

5.2. Logistic regression

For the next example, logistic regression, the conjugate gradient ascent approach is implemented with the following notation [5]. The target predictor is denoted y and the model parameters or weights of the target logistic function is denoted $\mathbf{w}$, relating the data vector, $\mathbf{x}$, to the outcome variable,

$$p(y = \pm 1 | \mathbf{x}, \mathbf{w}) = \sigma(y\mathbf{w}^T\mathbf{x}) = \frac{1}{1 + \exp(-y\mathbf{w}^T\mathbf{x})}. \tag{3}$$

With the weights drawn from a Gaussian distribution $p(\mathbf{w}) \sim \mathcal{N}(0, \lambda^{-1}\mathbf{I})$, scaled by regularization term, λ, the likelihood function can be defined as

$$\mathcal{L} = -\sum_i \log(1 + \exp(-y_i \mathbf{w}^T \mathbf{x}_i)) - \frac{\lambda}{2}\mathbf{w}^T\mathbf{w}. \tag{4}$$

Optimisation of the function is performed via gradient ascent where the gradient is

$$\mathbf{g} = \nabla_{\mathbf{w}}\mathcal{L} = \sum_i (1 - \sigma(y_i \mathbf{w}^T \mathbf{x}_i)) y_i \mathbf{x}_i - \lambda \mathbf{w} \tag{5}$$

and the Hessian matrix,

$$\mathbf{H} = \nabla^2_{\mathbf{w}}\mathcal{L} = -\sum_i \sigma(y_i \mathbf{w}^T \mathbf{x}_i)(1 - \sigma(y_i \mathbf{w}^T \mathbf{x}_i))\mathbf{x}_i \mathbf{x}_i^T - \lambda \mathbf{I}. \tag{6}$$

$$a_{ii} = \sigma(y_i \mathbf{w}^T \mathbf{x}_i)(1 - \sigma(y_i \mathbf{w}^T \mathbf{x}_i))) \tag{7}$$

$$\mathbf{H} = -\mathbf{X}\mathbf{A}\mathbf{X}^T - \lambda \mathbf{I} \tag{8}$$

$$\mathbf{w} = \hat{\mathbf{w}} - \frac{\mathbf{g}^T\mathbf{u}}{\mathbf{u}^T\mathbf{H}\mathbf{u}}\mathbf{u} \tag{9}$$

$$-\mathbf{u}^T\mathbf{H}\mathbf{u} = \lambda \mathbf{u}^T\mathbf{u} \sum_i a_{ii}(\mathbf{u}^T\mathbf{x}_i)^2 \tag{10}$$

The search line is updated along the gradient by a factor of β,

$$\mathbf{u} = \mathbf{g} - \hat{\mathbf{u}}\beta, \tag{11}$$

where $\hat{\mathbf{u}}$ is the prior value of $\mathbf{u}$ and

$$\beta = \frac{\mathbf{g}^T(\mathbf{g} - \hat{\mathbf{g}})}{\hat{\mathbf{u}}^T(\mathbf{g} - \hat{\mathbf{g}})}, \tag{12}$$

where $\hat{\mathbf{g}}$ is the gradient from the previous iteration. The equation for β comes from Hestenes-Stiefel [6].

Implementing this algorithm in the distributed learning framework requires the transfer of model parameters, search direction and local statistics for the gradient, Hessian and likelihood as shown in algorithm pseudo code, Algorithm 1. Code for this algorithm is provided in `CentralLogReg.m` and `LocalLogReg.m`

Algorithm 1: Distributed logistic regression - conjugate gradient

$\hat{\mathcal{L}} = -\text{tol}$, $\mathcal{L} = 0$
Normalize all features with $\frac{\mathbf{X} - \min(\mathbf{X})}{\max(\mathbf{X}) - \min(\mathbf{X})}$
while $\mathcal{L} - \hat{\mathcal{L}} \geq tol$ **do**
    Broadcast $\{\mathbf{w}, \mathbf{u}\}$
    foreach $k \in K$ **do**
        Compute local $\{\mathbf{g}_k, \mathbf{H}_k, \mathcal{L}_k\}$: Eqns 5, 10, 4
    end
    Collect and combine $\{\mathbf{g}, \mathbf{H}, \mathcal{L}\}$
    Update $\{\mathbf{w}, \beta, \mathbf{u}\}$: Eqns 9, 12, 11
end

6. Control server

6.1. Installation steps

Note: Installation instructions are for Windows Server 2012. No known limitations for installing on other Windows versions or on Unix.

Install Java JDK 8, 64-bit (or a compatible OpenJDK version) and download Apache Tomcat 9 64-bit version in a zip file. Refer to the following batch script for these installation steps in Windows (equivalent steps will need to followed for Unix). Add the Java directories to environment variables as per the first lines of the script, enter the appropriate directory to install Tomcat and the location of the Tomcat zip file. Extract the distributed learning template package to the location chosen when using the script below. The example here is using Apache-Tomcat 9.0.4.

```
setx /M PATH "%PATH%;<java bin directory>"
setx /M JAVA_HOME "<java folder>"
set DIST_LEARNING_BASE_DIR=<full path to base directory>
set TOMCAT_BASE_DIR=<full path to Tomcat location>\Tomcat9
set TOMCAT_ZIP=apache-tomcat-9.0.4-windows-x64.zip
%TOMCAT_DIR:~0,2%
cd %TOMCAT_BASE_DIR%
jar xf %TOMCAT_ZIP%
cd %TOMCAT_BASE_DIR%\apache-tomcat-9.0.4\bin
install service
```

```
tomcat9 //US//Tomcat9
    ++JvmOptions=-Dproperties.home="%DIST_LEARNING_BASE_DIR%\properties"
net start Tomcat9
```

Generate a new certificate for the server and have it signed by a certificate authority. This can be achieved by generating a self-signed certificate and a certificate signing request (CSR). When creating the certificates one will also be created for the web client to connect to the web service. The supplied script (`CreateNewServerCertificates.bat`) can be used create these certificates and to add each of the data center public certificates to the trusted keys file.

For the Apache Tomcat 9 installation the `server.xml` file should be modified to operate with TLS/SSL on port 443. Remove the following section from the file.

```
    <Connector port="80" protocol="HTTP/1.1"
        connectionTimeout="20000"
        redirectPort="8443"
      address="0.0.0.0" />
```

Enter the following section into the file with the location of the private key file generated and associated password as well as the trusted key location and password.

```
<Connector clientAuth="true" port="443" minSpareThreads="5"
    maxSpareThreads="75"
   enableLookups="true" disableUploadTimeout="true"
   acceptCount="100" maxThreads="200"
   scheme="https" secure="true" SSLEnabled="true"
   keystoreFile="<path>\server_key.jks"
   keystoreType="JKS" keystorePass="changeme"
   truststoreFile="<path>\trusted_keys.jks"
   truststoreType="JKS" truststorePass="changeme"
   SSLVerifyClient="require" SSLEngine="on" SSLVerifyDepth="2"
      sslProtocol="TLS"/>
```

The following web applications are to be deployed into Apache Tomcat. Deployment can be handled through the Tomcat manager web page. To use the manager page a `manager-gui` user and role should be added to `tomcat-users.xml`. Before deploying these applications a configuration

file should be created per web application and the configuration files should match the name of the each application.

- DLServerNode.war
- MPI.war
- DLWebClient.war

The deployment service, DLServerNode and MPI must have a configuration including the list of data centres connected and the specific password for each as well as the password for the central algorithm connection. Note that currently the client name string cannot contain an underscore character.

```
projectsClients=client1,client2 // list of data centre names
client1_pw=changeme // password for data centre: client1
client2_pw=changeme // password for data centre: client2
ws_password=changeme // password for central algorithm
```

The configuration for the DLWebClient is the DLServerNode WSDL link and the key file created for the client.

Configure firewall on the server to have in-bound connection allowance to tomcat9.exe on port 443.

To observe the list of data centres connecting to the deployment service examine the web page served by DLWebClient.war, `https://<control_server_IP>/DLServerNode/DLWebClient/index.html` and navigate to the manager page. Once a group of data centres are configured to poll the control server then the data centre network table should be populated.

To operate an MPI system a central controlling function is needed to coordinate the distributed learning function. Create a copy of the MPIClient web application renaming it to CentralMPI. Modify the associated properties file to include the password for the central algorithm as chosen in the server. Deploy the `CentralMPI.war` into the Apache Tomcat container on the system where the central algorithm will be executed, this could be one of the data centres, the central server itself or another designated server.

6.2. Usage

To use the distributed learning network to transmit an algorithm, firstly a compiled function is required to be broadcast on the network. Currently, this system is configured to work with Matlab compiled executable files with

the requirement that the appropriate Matlab Compiler Runtime version in installed at each data centre in the network. Once this is prepared open an internet browser with the client certificate available to present to the server and visit `https://<serverIP>/DLServerNode/DLWebClient/manager.jsp`. The central model is controlled through a CentralMPI web application on a data centre system (or appropriately configured system) with credentials to connect as a centralised modelling application. Ensure that access to the server web page and the CentralMPI are accessible on the same system.

The process for running a modelling job is:

- Select the data centres that will participate via the web interface.
- Browse for a compiled application and upload via the web interface
- Wait for signal on the web interface that each centre has downloaded the application and that the algorithm is ready.
- Run the central algorithm wherever it is hosted.
- Select the start algorithm button on the web interface.

The progress of the local algorithm at each centre is tracked through the web interface and the central algorithm will provide the final results.

6.2.1. Parallel projects

To create parallel projects make a copy of the DLServerNode.war and rename it to a new project name, `DLproject1.war`, the associated configuration file should have update project participants and passwords for connecting. The link to the manager webpage will then require the project name `https://<control_server_IP>/DLProject1/DLWebClient/index.html`

References

[1] S. Boyd, N. Parikh, E. Chu, B. Peleato, J. Eckstein, Distributed optimization and statistical learning via the alternating direction method of multipliers, Foundations and Trends in Machine Learning 3 (1) (2011) 1–122. doi:10.1561/2200000016.

[2] T. M. Deist, A. Jochems, J. van Soest, G. Nalbantov, C. Oberije, S. Walsh, M. Eble, P. Bulens, P. Coucke, W. Dries, A. Dekker,

P. Lambin, Infrastructure and distributed learning methodology for privacy-preserving multi-centric rapid learning health care: euroCAT, Clinical and Translational Radiation Oncology 4 (Supplement C) (2017) 24–31. doi:10.1016/j.ctro.2016.12.004.
URL http://www.sciencedirect.com/science/article/pii/S2405630816300271

[3] A. Jochems, T. M. Deist, J. van Soest, M. Eble, P. Bulens, P. Coucke, W. Dries, P. Lambin, A. Dekker, Distributed learning: Developing a predictive model based on data from multiple hospitals without data leaving the hospital A real life proof of concept, Radiotherapy and Oncology 121 (3) (2016) 459–467. doi:10.1016/j.radonc.2016.10.002.
URL http://linkinghub.elsevier.com/retrieve/pii/S0167814016343365

[4] E. Roelofs, A. Dekker, E. Meldolesi, R. G. van Stiphout, V. Valentini, P. Lambin, International data-sharing for radiotherapy research: An open-source based infrastructure for multicentric clinical data mining, Radiotherapy and Oncology 110 (2) (2014) 370–374. doi:10.1016/j.radonc.2013.11.001.

[5] T. P. Minka, A comparison of numerical optimizers for logistic regression 18.
URL http://research.microsoft.com/~minka/papers/logreg

[6] M. R. Hestenes, E. Stiefel, Methods of conjugate gradients for solving linear systems, Journal of Research of the National Bureau of Standards 49 (6) (1952) 409–436. doi:10.6028/jres.049.044.