Master's thesis
Theoretical and Computational Methods

Federated Learning for Mortality Prediction in Intensive Care Units

Hannu Pelttari

September 7, 2020

Supervisor(s): Antti Honkela, Jyri Kivinen, Timo Petäjä

Examiner(s): Antti Honkela
Jyri Kivinen

University of Helsinki
Faculty of Science

PL 64 (Gustaf Hällströmin katu 2a)
00014 Helsingin yliopisto

HELSINGIN YLIOPISTO — HELSINGFORS UNIVERSITET — UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty Faculty of Science	Koulutusohjelma — Utbildningsprogram — Degree programme Theoretical and Computational Methods

Tekijä — Författare — Author Hannu Pelttari
Työn nimi — Arbetets titel — Title Federated Learning for Mortality Prediction in Intensive Care Units

Työn laji — Arbetets art — Level Master's thesis	Aika — Datum — Month and year September 7, 2020	Sivumäärä — Sidantal — Number of pages 67

Tiivistelmä — Referat — Abstract

Federated learning is a method to train a machine learning model on multiple remote datasets without the need to gather the data from the remote sites to a central location. In healthcare, gathering the data from different hospitals into a central location can be a difficult and time-consuming task, due to privacy concerns and regulations regarding the use of sensitive data, making federated learning an attractive alternative to more traditional methods.

This thesis adapted an existing federated gradient boosting model and developed a new federated random forest model and applied them to mortality prediction in intensive care units. The results were then compared to the centralized counterparts of the models.

The results showed that while the federated models did not perform as well as the centralized models on a similar sized dataset, the federated random forest model can achieve superior performance when trained on multiple hospitals' data compared to centralized models trained on a single hospital. In scenarios where the centalized models had data from multiple hospitals the federated models could not perform as well as the centralized models. It was also found that the performance of the centralized models could not be improved with further federated training. In addition to practical advantages such as possibility of parallel or asynchronous training without modifications to the algorithm, the federated random forest performed better in all scenarios compared to the federated gradient boosting. The performance of the federated random forest was also found to be more consistent over different scenarios than the performance of federated gradient boosting, which was highly dependent on factors such as the order with the hospitals were traversed.

Avainsanat — Nyckelord — Keywords

Federated Learning, Random Forest, Gradient Boosting, Mortality Prediction

Säilytyspaikka — Förvaringsställe — Where deposited

Muita tietoja — Övriga uppgifter — Additional information

Acknowledgements

I want to thank GE Healthcare and Chief Scientist Hanna Viertiö-Oja for providing the topic and an opportunity to write this thesis. Special thanks also to my thesis supervisors Jyri Kivinen and Timo Petäjä at GE Healthcare as well as Antti Honkela, my thesis supervisor at University of Helsinki. Lastly, I would like to thank the data science team at GE Healthcare, and my friends and family.

September 7, 2020
Hannu Pelttari

Contents

Acronyms

AUPRC Area under the precision-recall curve.

AUROC Area under the receiver operator characteristic curve.

BPdiast Diastolic blood pressure.

BPmean Mean blood pressure.

BPsyst Systolic blood pressure.

BUN Blood urea nitrogen.

CRP C-reactive protein.

CV Cross-validation.

CVP Central venous pressure.

EMR Electronic medical records.

FADL Federated autonomous deep learning.

FedGB Federated gradient boosting.

FedRF Federated random forest.

$\mathbf{FiO_2}$ Fraction of inspired oxygen.

FPR False positive rate.

GB Gradient boosting.

GCS Glasgow coma scale.

GDPR General data protection regulation.

HIPAA Health insurance portability and accountability act.

ICU Intensive care unit.

MCH Mean corpuscular hemoglobin.

MCHC Mean corpuscular hemoglobin concentration.

MCV Mean corpuscular volume.

MPV Mean platelet volume.

MSE Mean squared error.

$PaCO_2$ Arterial carbon dioxide partial pressure.

PaO_2 Arterial oxygen partial pressure.

PEEP Positive end-expiratory pressure.

RDW Red cell distribution width.

RF Random forest.

ROC Receiver operator characteristic.

SpO_2 Peripheral capillary oxygen saturation.

TPR True positive rate.

1. Introduction

In the recent years, machine learning has gained popularity in solving various problems in many industries, including healthcare. During the last decade, the number of articles published focused on machine learning in healthcare has increased manyfold [1].

Machine learning methods usually require that the data used in training the models is gathered to a single database on which the models are trained on. Collecting data from e.g. multiple hospitals to a central location poses a challenge in healthcare, since the data usually contains sensitive information and privacy must be guaranteed for everyone included in the dataset. Hence, there are many regulations, such as the Health Insurance Portability and Accountability Act (HIPAA) in the United States and the General Data Protection Regulation (GDPR) in the European Union, that must be followed when sensitive data are used [2] [3]. In addition to privacy concerns, the amount of data might be very large which makes copying and transferring the data from one location to another more difficult. The data may also be spread out to many different locations which, in combination to the two previous factors, makes the data collection efforts a time-consuming and expensive task.

One solution to these problems is a method called federated learning [4]. In federated learning, a centralized server distributes the model parameters to the remote sites, which then train the model and update the parameters using their own data and send the updated parameters back to the central server which aggregates the parameters and updates the global model on the server. This is in contrast to more traditional methods where usually a a single model is trained on a single centralized dataset.

The objective of this thesis is to study federated learning methods in mortality prediction in the intensive care unit (ICU) and their performance compared to traditional machine learning methods where the data is collected centrally. The patients in intensive care units are generally very ill and early identification of high risk patients may save lives and help doctors to target their attention and efforts where it is most impactful. In the experiments of this thesis, the models make hourly predictions for each patient of whether the patient is going to die within the next 8 hours or not. These kinds of predictions can be used for example to alert doctors in advance if the patient's state is about to deteriorate.

The main research question for this thesis is that can federated learning achieve similar performance compared to centralized learning? This question is studied by training both the federated and the more traditional, centralized models on the same data. The centralized models keep the dataset as a single dataset and the federated models split the dataset into multiple smaller datasets, each containing data from a single hospital.

Further research questions are whether the federated models can achieve superior performance compared to the centralized models if the federated models have access to more data, and if the performance of the centralized models can be improved if they are trained centrally on a smaller dataset and then additional training is carried out in a federated manner. The former question is studied by training the centralized models on datasets with data from 1 to 3 largest hospitals and the federated models on datasets with data from 5 to 20 largest hospitals. The latter question is studied by having data from 2 or 3 largest hospitals centralized and data from the rest of the hospitals as separate datasets. The models are then trained on this partly federated, partly centralized dataset and the performance is compared to the centralized models having data from the 2 or 3 largest hospitals.

Chapter 2 contains a brief introduction to some background information on machine learning necessary for understanding this thesis. Chapter 3 introduces the concept of federated learning and the federated gradient boosting model used in this thesis as well as a brief discussion of privacy protection and some relevant previous work on federated learning. Chapter 4 presents a novel federated random forest method developed for this thesis. Then we move on to Chapter 5, which describes the data used in this thesis as well as the experiments that are carried out.Chapter 6 presents the results of the experiments. A discussion of the results is contained in Chapter 7, along with the limitations related to the data, experiments and analysis. Finally, Chapter 8 contains a summary and conclusions from the results of the thesis.

2. Background

This chapter contains a short introduction to some machine learning concepts necessary for understanding the methods and experiments used in this thesis. The chapter starts with a brief introduction to machine learning, then provides an overview of decicion trees, gradient boosting, random forests and neural networks, and ends with a review of some general concepts used for evaluating and training machine learning models.

2.1 Machine Learning

The book *Machine Learning: A Probabilistic Perspective* [5] defines machine learning as "a set of methods that can automatically detect patterns in data". These learned patterns can then be used to perform different kinds of tasks. Some of the most common tasks that machine learning is used are clustering [6] where similar items are grouped together, anomaly detection [7] where anomalies are detected in a signal, regression [8] where the value of an outcome variable is predicted based on one or more independent variables and classification [9] where items are classified into one or more classes.

From another point of view, machine learning can be categorized into supervised and unsupervised learning. In unsupervised learning the computer is not given any examples of what is the correct end result of performing the task. Clustering for example is usually unsupervised since the machine learning algorithm gets some data as input and performs the clustering without any instructions on which samples should belong to which clusters. A simple example of supervised learning is a classification task where we have some training data for which we know the true class of each observation, and based on that data, we try to learn how to assign a correct class to observations for which we do not know the true label beforehand. The classification task can then be separated into a training phase and a prediction phase. Model training can also be called model fitting, especially when considering models such as linear regression [10] where a curve is fitted to the data. Instead of prediction, the term inference can be used and it can be more appropriate in some cases since not all machine learning is focussed on prediction. In the training phase the classification algorithm gets as input not only the data it is supposed to classify but also the correct classification labels for each data point. The classification algorithm uses the training data and the correct labels to learn from the data how to classify each data point. After the training phase, the algorithm can be used to classify new data for which the correct labels are not known. These are not all of the possible categorizations and types of machine learning, and multiple other categories exist, such as semi-supervised learning [11] and reinforcement learning [12].

The machine learning tasks in this thesis are all binary classification tasks, meaning that there exists two distinct classes to one of which each data point belongs to. Many different classification algorithms have been developed and the basics of the ones relevant to this thesis are presented in the following sections.

2.2 Decision Trees

Decision trees are one of the simplest machine learning methods and they can be used for both regression and classification tasks [13]. This thesis is only concerned with classification trees and this section will provide an introduction to them.

The training phase for classification decision trees work by splitting the feature space recursively into two regions and classifying a sample to the majority class in the region the sample is located in. Decision trees are binary trees meaning that each node have at most two child nodes. The first node in the tree, which has no incoming edges, is called the root node, the final nodes, which have no outgoing edges, are called leaf nodes, and the nodes between the root and the leaves are called internal nodes [14].

To keep things simple, let us consider the case where there are two classes and all the features are binary, i.e. have two possible values, for example 0 and 1. The first stage in building the tree is to find the best feature tho make the first split on. For this, all the features are considered and the split is executed for the feature that results in the smallest impurity, determined by some impurity measure. The root node now splits the data so that all data for which the chosen features value is 0 goes into one branch and all the data for which the value is 1 goes to the other branch. In the next stage the features which result in the lowest impurity need to be found for the two resulting nodes. This search for splits is continued until some stopping criterion is reached or until there are no more features to split on.

Since at each stage, the split that is chosen is determined by the lowest impurity in the current stage, the method is considered to be greedy, i.e. it chooses the best split for the current situation, and does not consider how it affects the performance further down the line. It is not guaranteed that this produces the optimal tree. [15]

To classify a sample to a class in the prediction phase, the tree is traversed downwards. For example, if the first split is made for feature f_1, the value of the sample for f_1 is examined, and depending on the value of f_1 and the split criterion, the traversing continues either to the left or the right branch,. This is done until one of the leaf nodes is reached. At the leaf node, the sample is classified to the majority class in that region.

So far, only binary features have been considered. Using continuous features complicates the matter only slightly. If the features are continuous, we need to consider not only what feature to split on but also at what value to make the split on. In practice, when searching for the optimal split for a feature, the values of the feature are sorted and the averages of each adjacent values are considered as the potential split points. The split is chosen by calculating the impurity for each point and choosing the one resulting in the lowest impurity.

2.2.1 Impurity Measures

A simple way to measure impurity would be to use the misclassification rate of the samples

$$\frac{1}{N}\sum_i (1-\delta_{y_i k}) = 1 - \text{accuracy}, \tag{2.1}$$

where N is the number of samples, k is the class label, y_i is the class that the classfier assigned for sample i and $\delta_{y_i k}$ is the Kronecker delta, which is defined as

$$\delta_{i,j} = \begin{cases} 1, \text{if } i = j \\ 0, \text{if } i \neq j. \end{cases} \tag{2.2}$$

Accuracy can be defined as

$$\text{accuracy} = \frac{1}{N}\sum_i \delta_{y_i k}. \tag{2.3}$$

Although simple, since misclassification rate is not differentiable and not as suitable for numerical optimization, it is usually not used as an impurity measure for classification trees. [16]

Two common impurity measures used in classification decision trees are the Gini index and entropy. The formula for the Gini index is

$$\begin{aligned} \mathrm{I_G}(p) &= \sum_{i=1}^{J} p_i \sum_{k\neq i} p_k \\ &= 1 - \sum_{i=1}^{J} p_i^2, \end{aligned} \tag{2.4}$$

where J is the number of classes and p_i is the fraction of items labeled to class i in the data set. Considering the simple case where $J = 2$, i.e. we have two distinct classes, we can see that the impurity is clearly minimized when all the items belong to the same class and maximized when there is equal number of items in both classes. This can be verified by plugging in the numbers:

$$\mathrm{I_G}(p) = 1 - \sum_{i=1}^{2} p_i^2 = 1 - 0^2 + 1^2 = 0, \tag{2.5}$$

which is the impurity when all the items belong to one of the classes and

$$\mathrm{I_G}(p) = 1 - \sum_{i=1}^{2} p_i^2 = 1 - 0.5^2 + 0.5^2 = 0.5, \tag{2.6}$$

which is the impurity when both classes have equal number of items in them.

The formula for entropy does not differ greatly from the Gini index:

$$\mathrm{I_E}(p) = -\sum_{i=1}^{J} p_i \log(p_i). \tag{2.7}$$

Since $0\log(0)$ and $1\log(1)$ both are equal to zero, we can see that entropy too is minimized when all items belong to the same class.

Since both the Gini index and entropy are minimized when the region contains only one class and maximized when the distribution is equal to all classes, either one of them can be used as a criterion for choosing a split. The Gini index and entropy impurity measures are better than misclassification rate in that they are more sensitive to changes in class probability and favour pure splits that only contain one class [5]. Many other impurity measures also exist, such as likelihood ratio chi-squared statistic, DKM criterion and gain ratio [14]. The details of these other impurity measures will not be presented here however, since only the Gini index and entropy are considered in the experiments of this thesis.

2.2.2 Advantages and Disadvantages

One of the greatest advantages of decision trees compared to other model types is the easy interpretability and intuitivity of decision trees [15]. If one has access to the tree structure, it is easy to manually traverse through the tree to see why a sample was classified the way it was.

Another advantage is the handling of missing values. Many other models cannot handle missing values, hence they need to be imputed. For decision trees there are a couple of options. For categorical variables one can either create a new category "missing" or construct surrogate variables, where one first chooses a best feature and split point for the primary split and then creates a list of surrogate predictors and splits to use if the primary is missing [16]. Decision trees also perform automatic variable selection and scale well to large datasets [5].

The downside of decision trees is that they have high variance, so changing only a small part of the data can lead to a very different tree structure. This is due to the fact that a change in the top split propagates its effect downwards to further splits [16].

2.3 Boosting and Ensembles

This section will present methods for combining multiple decision trees into a larger model to overcome some of the weaknesses of decision trees. Two different ways of combining trees are considered, namely boosting and ensemble learning. Although it is possible to use boosting or ensembles with other model types than decision trees, this section will consider boosting and ensemble learning only for decision trees.

2.3.1 Boosting

Boosting is a way to combine multiple base models that perform only slightly better than random, sometimes called *weak learners* [17], into a single model that performs better than the individual weak learners. Boosting in general can be viewed as way to fit an additive expansion to a set of elementary "basis" functions [16]. The basis function expansion can be described as

$$f(x) = \sum_{m=1}^{M} \beta_m b(x; \gamma_m), \tag{2.8}$$

where β_m are the expansion coefficients and $b(x;\gamma)$ are the basis functions. The basis functions considered in this chapter are the individual decision tree classifiers, although they can in principle be any simple functions.

The basic idea of boosting is to give more weight to samples that were misclassified as more weak learners are added to the model [5]. Next we will show how this can be done with a method called gradient boosting.

Gradient Boosting

Gradient boosting was first introduced by Jerome Friedman in 2001 [18]. Function approximation in gradient boosting is done in function space as opposed to parameter space. Friedman introduced the following algorithm for gradient boosting:

Algorithm 1 Gradient Boosting

```
1: $F_0(\mathbf{x}) = \text{argmin}_\rho \Sigma_{i=1}^N L(y_i, \rho)$
2: for m=1 to M do:
3:     $\tilde{y_i} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)}, i = 1, \ldots, N$
4:     $\mathbf{a}_m = \text{argmin}_{\mathbf{a},\beta} \Sigma_{i=1}^N \left[\tilde{y_i} - \beta h(\mathbf{x}_i; \mathbf{a})\right]^2$
5:     $\rho_m = \text{argmin}_\rho \Sigma_{i=1}^N \mathcal{L}(y_i, F_{m-1}(\mathbf{x}_i) + \rho h(\mathbf{x}_i; \mathbf{a}_m))$
6:     $F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \rho_m h(\mathbf{x}; \mathbf{a}_m)$
7: end for
```

In Algorithm 1 [18], $\mathcal{L}(y_i, \rho)$ can be any differentiable loss function and $\mathbf{a}$ is a parameter characterizing a weak learner $h(\mathbf{x}; \mathbf{a})$, usually a decision tree.

In practice, at each stage, gradient boosting adds to the ensemble a weak learner that tries to predict the residuals $\tilde{y}$, calculated as the negative gradient of the loss function.

In line 1 of Algorithm 1, the model is initialized with a constant value by minimizing the loss function with respect to ρ. In the loop, starting from line 2, M trees are built. The negative gradient of the loss function is calculated in line 3, which gives us the values for the residuals. A weak learner (a decision tree in this thesis) is fitted on the residuals on line 4. The parameter ρ_m is then optimized in line 5 and the model is updated in line 6. This method results in gradient descent in function space.

2.3.2 Ensemble Learning

Ensemble learning is another way of combining base models into a larger model. In principle, neural networks and boosting can also be considered as ensemble learning [5] but for clarity, these will be considered as separate concepts in this chapter.

A simple way to perform ensemble learning is to train multiple models and then average the predictions of the models. If all models are trained on the same data, this will result in multiple identical or very similar models in most cases. One way to introduce variability is to use *bootstrap* and train the models on bootstrap datasets [17]. The basic idea is to randomly draw samples with replacement from the original dataset until a same size (bootstrap) dataset is produced, and then train the base models on different bootstrap datasets [16].

Next, random forests will be introduced as a way to introduce even more variability between the models.

Random Forests

Decision trees are noisy [16] and prone to *overfitting* [19], i.e. they have good prediction performance on the data they were trained on but poor performance on other data.

This means that decision trees generalize poorly to unseen data. Random forests try to increase the generalization performance of individual trees by using *bootstrap aggregation* or *bagging* as it is often called. The general idea is to build many de-correlated trees and average the predictions of the trees to get the final prediction. For classification, each tree in the forest cast a unit vote and the prediction given by the forest is taken to be the mode of the votes.

Since fully grown trees usually have low bias but high variance, the generalization performance can be increased by reducing the variance. The variance of B independently distributed random variables is given by

$$\rho\sigma^2 + \frac{1-\rho}{B}\sigma^2, \tag{2.9}$$

where ρ is the positive pairwise correlation and σ^2 is the variance. Random forests decrease the variance by increasing the number of trees B and decreasing ρ. [16]

The decrease of ρ in random forests is achieved with building de-correlated trees by randomly selecting the input variables. When building a tree, a bootstrap sample of size N is drawn and for each node, m out of p input variables are randomly selected to be considered for the split. The best split is then selected from these m variables. If no random sampling of variables would be used and the training set contains one very strong predictor, then almost all trees would use the same strong predictors in their first split, resulting in highly correlated trees [15]. Hence, random sampling of variables results in de-correlated trees, decreasing ρ and therefore reducing the variance of the average.

Another advantage of random forests is that they are robust to overfitting, and although there is a limit to the generalization error, overfitting does not happen when the number of trees B is increased [20].

2.4 Neural Networks

Neural networks are statistical models inspired by the structure and information processing of the brain, but since the goal is to develop an efficient machine learning algorithm and not model the brain, biological accuracy is unnecessary [17] [21]. This section provides a brief introduction to neural networks, necessary to understand federated deep learning, which will be presented later in the thesis. The terms *neural networks* and *deep learning* are used interchangeably in this thesis.

2.4.1 Feed-Forward Networks

The purpose of feed-forward networks is to approximate some function f and define a mapping $\boldsymbol{y} = f(\boldsymbol{x}; \boldsymbol{\theta})$ [22]. Here $\boldsymbol{x}$ is the input to the network, $\boldsymbol{y}$ is the output of the network and $\boldsymbol{\theta}$ are the parameters that the network learns in order to approximate the original function f as well as possible. Typically, feed-forward networks are composed of multiple layers by chaining together multiple functions: $f(\boldsymbol{x}) = f^{(3)}(f^{(2)}(f^{(1)}(\mathbf{x})))$. They are called feed-forward networks because information flows forward from the first layer $f^{(1)}$, called input layer, through intermediate layers to the last, output layer $f^{(3)}$ and there are no closed directed cycles [17]. The layers between the input layer and the output layer are called hidden layers. There are different conventions in literature regarding how many layers a network is said to have, depending if the input and output layers are counted as the layers of the network [17].

Without applying any activation function, the output of each linear layer is $\boldsymbol{z} = \boldsymbol{W}^\top \boldsymbol{h} + \boldsymbol{b}$, where $\boldsymbol{W}$ are the weights of the layer, $\boldsymbol{b}$ is the bias and $\boldsymbol{h}$ is the output from the previous layer. For the first hidden layer, $\boldsymbol{h} = \boldsymbol{x}$, i.e. the input to the network. The optimal values of the weights $\boldsymbol{W}$ and biases $\boldsymbol{b}$ for each layer are the parameters values that the network is trying to learn.

Since $\boldsymbol{z} = \boldsymbol{W}^\top \boldsymbol{h} + \boldsymbol{b}$ is just a linear transformation applied to vector $\boldsymbol{h}$, and can not capture non-linear relationships, typically a non-linear activation function is applied to each layer. A common choice for an activation function for hidden layers is the rectified linear unit (ReLU), which is defined as $f(\boldsymbol{x}) = \max(0, \boldsymbol{x})$, where $\boldsymbol{x}$ is the input vector. For the output layer, the problem definition determines what kind of an activation function is suitable. For example, the sigmoid function, $f(\boldsymbol{x}) = \frac{e^{\boldsymbol{x}}}{e^{\boldsymbol{x}}+1}$ is a suitable activation function for binary classification since it outputs values between 0 and 1, and the softmax function, $f(\boldsymbol{x})_i = \frac{e^{\boldsymbol{x}_i}}{\sum_j e^{\boldsymbol{x}_j}}$ is suitable for multiclass classification since it normalizes the output's sum to 1.

Next, we will consider how the values for the weights and biases of neural networks are optimized during the training phase via *backpropagation*.

2.4.2 Backpropagation

As noted before, the training phase consists of optimizing the values of the weights and biases and the way this is done is called backpropagation. The way that this optimization works is that first some input is fed to the first layer of network which then feeds its output to the next layer and so on, until the final layer is reached. This phase is called *forward propagation*. The output of the final layer and the true value (the true label in the case of classification) of the datum is fed to a *loss function*, such as the $\mathcal{L}_1$ and $\mathcal{L}_2$ [5] or cross-entropy [17] losses. For each problem type, there is typically a natural output activation function and loss function pair, such as sigmoid output activation and cross-entropy loss for binary classification, or softmax output activation and multiclass cross-entropy loss for multiclass classification [17]. The objective is then to find the weights and biases that minimize the loss. The local minimum of the loss function can be found at the point where the gradient is zero. Using this knowledge, the values of weights and biases can be adjusted towards the minimum with gradient descent [17]. The gradient is calculated for each layer, starting from the last layer, and the error is propagated backwards, hence the name backpropagation. Let us now present this concept in a more formal way.

Let $y = g(x)$ and $z = f(g(x)) = f(y)$. The gradient of z with respect to x can now be computed with the chain rule [22]:

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}. \tag{2.10}$$

For vectors, this can be written in vector notation:

$$\nabla_{\boldsymbol{x}} \boldsymbol{z} = \left(\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}}\right)^T \nabla_{\boldsymbol{y}} \boldsymbol{z}. \tag{2.11}$$

From Equation (2.11) we can see that the gradient of $\boldsymbol{x}$ is equivalent to multiplying the Jacobian matrix $\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}}$ with the gradient $\nabla_{\boldsymbol{y}} \boldsymbol{z}$ [22].

To sum up, backpropagation is an algorithm to calculate the derivatives of all the nodes in a neural network by recursively applying the chain rule and can be used in neural networks to find a minimum of the loss function. An important consideration that has been neglected so far however, is that there might be multiple local minima of the loss function where the gradient is zero or very small, but we would like to find the global minimum, i.e. the smallest value of the loss function for any weight and bias vector. There is no reliable and general way to find the global minimum but luckily, a sufficiently good solution can in many cases be found with a local minimum [17].

2.5 Model Evaluation and Tuning

2.5.1 Training, Test and Validation Sets

The data used in training the model is called the training set. The goal of the training stage is to fit the model to the training data as well as possible, i.e. to minimize the training error. However, usually we are not really interested in what the training error is but we are more concerned with the model's performance on new, unseen data. Focusing only on minimizing the training error can lead to overfitting [19]. This is why the data set is usually divided into separate training and test sets. The test set is necessary, since the model with the lowest training error is not necessarily the same model as the one that has the lowest test error because the model can, for example, fit to the random noise on the training data, which is not present in the test data. When the data has been split into a training set and a test set, the model is first trained on the training data and once the training is finished, the performance is evaluated on the test data, which was not used in the training stage. This way it can be estimated how well the model performs on unseen data [15].

A new problem arises if we are not satisfied with the model's performance on the test data. We could adjust the values of the hyperparameters of the model, fit the model on the training data again with the new hyperparameters, evaluate the performance again on the test set and continue this hyperparameter tuning, training and testing loop until we are satisfied with the performance on the test set. Doing so would introduce bias to the model since we have tuned the hyperparameters based on the test set performance. This means that the test set data is not new and unseen for the model, since information about the test set performance has now been used to train the model, and the test error does not reflect well the performance of the model on new data.

The problem can be solved by dividing the data into three sets, called training, validation and test sets. Now the model can be trained on the training set, performance evaluated on the validation set and hyperparameters tuned based on the performance on the validation set. This can be continued until we are satisfied with the performance of the model on the validation set. Only when all the hyperparameter tuning is done and we are satisfied with the model, is the performance of the model evaluated on the test set. Since the model has not seen the test data before and the hyperparameters are not tuned based on the test set performance, the test set performance can now be used to approximate the performance of the model on new data. If we are still not satisfied with the model's performance, we must not use the same test set again for evaluating the performance, but we should somehow gather new data to be used as the test set.

2.5.2 Cross-Validation

The validation set error is an estimate to the test set error but depending on which observations were selected into the validation set, it can be highly variable and might not be a very accurate estimate [15]. This problem can be remedied by using cross-validation.

Leave-One-Out Cross-Validation

In leave-one-out cross-validation, a single sample is chosen as the validation set and the model is trained on the rest of the data. The whole data set is gone through, picking and using a single sample as the validation set each time and training the model on the rest of the data, until all the samples have been the validation set once. Now the estimate for the test set error is the average of all the validation set errors during cross-validation:

$$\text{Error}_{\text{CV}} = \frac{1}{n}\Sigma_{i=1}^{n}\text{MSE}_i. \tag{2.12}$$

Here n is the size of the data set and MSE_i is the mean squared error for sample i. Using leave-one-out cross-validation, a better estimate of the test error (compared to having a single validation set) can be achieved by taking the average of the validation set errors and, as an additional bonus, we have more data to use when fitting the model because we only leave one sample out of the training set each time, which reduces the bias of the model [15]. There is one substantial drawback on leave-one-out cross-validation however: if the data set is large or the model is complex, it may be very computationally demanding to do leave-one-out cross validation. This makes it impractical in many real world situations. What is usually done to make cross-validation more practical is to use k-fold cross-validation.

K-Fold Cross-Validation

In k-fold cross-validation, instead of leaving out one sample as the validation set, we divide the data set randomly to k equal sized parts, called folds, use each fold as the validation set and then train the model with the rest $k-1$ folds. The estimate for the test error is now

$$\text{Error}_{\text{K\_CV}} = \frac{1}{k}\Sigma_{i=1}^{k}\text{MSE}_k. \tag{2.13}$$

Instead of fitting the model n times like in leave-one-out cross validation, the model is now fit only k times. K-fold cross-validation of course is identical to leave-one-out cross-validation when $k = n$. Usually the choice of k is much smaller than n, and a typical choice is $k = 10$. There is one caveat in choosing the value for k however. Due to the bias-variance trade-off, doing k-fold cross-validation with $k < n$, we might get a more accurate estimate of the test error than when using leave-one-out cross-validation. From a bias point of view, leave-one-out cross-validation is desirable, but since the n models fitted are using almost the same data, they are highly correlated with each other which results in higher variance [15].

Multiple-Source Cross-Validation

In the federated setting, we have data from multiple sources and it would be more desirable to estimate the performance of the model on new and unseen data from a new source rather than unseen data from the same source that is used in training the model. In the healthcare setting, different sources could be for example different hospitals, and multiple-source-cross-validation would give an estimate of how a model trained on some hospitals' data generalizes to new hospitals.

Conventional cross-validation, where the data is randomly sampled to training and validation sets, would not give a good estimate for the model's performance since data from the same source could simultaneously end up in both the training set and the validation sets.

To solve this problem, cross-validation can be done so that a single source's data can only be in the training set or in the validation set in any given time, but not in both. Geras et al. called this *multiple-source cross-validation* and provided some theoretical analysis for it [23]. Multiple-source cross-validation is exactly the same as the regular cross-validation shown in Equation (2.13) but instead of randomly dividing the data to k different folds, each fold consists of the data of a single source.

Stratified Cross-Validation

Since people may visit multiple different hospitals, either because of the same condition or a completely new one, there is a possibility that there will be identical records of the same patient in multiple hospitals' datasets. In fact, even up to 15% of the records can be duplicates [24]. This might leak information if the same records are present in the training and validation sets simultaneously, therefore producing overly optimistic estimates for the model performance. For this reason, Bey et al. introduced a *stratified cross-validation* technique to alleviate the problem [25]. Stratified cross-validation makes sure that identical records do not end up in the training and validation sets simultaneously.

Stratified cross-validation works as follows:

Algorithm 2 Stratified Cross-Validation

```
1: Select a covariate $x$ that is weakly associated with other covariates and the outcome
2: Choose the number of folds k and define thresholds $\{t_0, t_1, ..., t_k\}$ that there are approximately the same amount of records fulfilling $t_i < x < t_{i+1}$ for each $i$, and associate each fold with an index $i$.
3: Group all records with the same fold index $i$ in inter-hospital folds $D_i$ and apply cross-validation on these folds.
```

Johnson et al. show that in the critical care setting, training and cross validating the model across multiple hospitals can produce a model that generalizes well to new hospitals [26].

2.5.3 Model Performance

A simple way to measure model performance is to measure the accuracy, i.e. how many samples were properly classified out of all the samples. In some cases this is sufficient but there are many situations where accuracy is not a good metric to use. In a medical setting where we are predicting if a patient is likely to die, we might be much more interested in classifying correctly all the cases where the patient is likely to die, in order to save as many lives as possible, or we might want to reduce the number of false alarms to a minimum, i.e. to not label patients who are in a good condition as ones likely to die, in order to reduce the burden and alarm fatigue for doctors.

Precision, Recall and False Positive Rate

To understand precision and recall, we first need to define the concepts of a *true positive*, a *false positive*, a *true negative* and a *false negative*. In the context of classification, a true positive is a sample that our classifier correctly labeled as a positive sample. In the case of mortality prediction where we are trying to predict if a patient dies, a positive sample is one where the patient died and a negative sample one where the patient did not die. A true positive in this case is a sample that the classifier correctly labeled as positive and where the patient actually died. A false positive on the other hand is a sample that the classifier incorrectly labeled as positive but the patient did not die. Similarly, true negative is a sample that the classifier correctly labeled negative and the patient did not die, and a false negative is one that the classifier incorrectly labeled as negative but the patient died.

The above concepts can be visually represented as a confusion matrix:

		True class	
		Positive	Negative
Predicted class	Positive	*True positive*	*False positive*
	Negative	*False negative*	*True negative*

Table 2.1: Confusion matrix

Precision can now be defined as

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}, \tag{2.14}$$

where TP is the number of true positives and FP is the number of false positives.

Recall or *True Positive Rate (TPR)* can be defined as

$$\text{Recall} = \frac{\text{TP}}{\text{P}}, \tag{2.15}$$

where P is the number of positive samples and TP is once again the number of true positives. In other words, precision is the ratio of samples correctly classified as positive out of all the samples classified as positive and recall is the ratio of samples correctly classified as positive out of all positive samples (no matter how they were classified by the classifier).

False Positive Rate (FPR) can be defined as

$$\mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{N}}, \tag{2.16}$$

where FP is the number of false positives and N is the number of negative samples.

Receiver Operator Characteristics

The *receiver operator characteristic curve* or the *ROC curve* is a curve that forms when recall (TPR) is plotted on the y-axis and false positive rate is plotted on the x-axis. With classifiers that produce a probability or a score, this can be achieved by changing the threshold where the samples are classified as positive [27]. The ROC curve can be used to determine for example the optimal threshold for the classification based on a specific use case. A ROC curve with a constant slope of 45°, going from point (0, 0) on the lower left corner of the graph to point (1, 1) on the upper right corner of the graph, is equal to random predictions, meaning that points above the 45° line are better than random predictions and points below the 45° line are worse than random predictions [27].

AUROC and AUPRC

AUROC is an abbreviation of *area under the receiver operator characteristic curve* and it can be used to assess the performance of the classifier with a single number. *AUPRC* is similarly an abbreviation of *area under the precision - recall curve* and it can also be used in model assessment. The precision-recall curve is a curve where precision is plotted on the y-axis and recall on the x-axis. The values for AUPRC in this thesis are calculated as *average precision (AP)* using Scikit-learn's [28]. Average precision can be defined as

$$\mathrm{AP} = \sum_{t=1}^{n} (\mathrm{TPR}(t) - \mathrm{TPR}(t-1))\mathrm{P}(t), \tag{2.17}$$

where $\mathrm{TPR}(t)$ is the true positive rate (recall) at threshold t and $\mathrm{P}(t)$ is the precision at threshold t. Average precision is an approximation of area under the precision-recall curve [29].

The area under the ROC-curve can be interpreted as being the probability that a random pair of a positive and a negative sample will be correctly classified [30]. Since a ROC curve with a constant slope of 45° implies performance equal to random guess, an AUROC of 0.5 implies a classifier that makes random predictions, 1.0 a perfect classifier and anything below 0.5 implies performance worse than a random classifier [30].

For AUPRC, the baseline is not fixed as it is for AUROC. The baseline where the classifier is as good as random guessing is the prevalence which is equal to the number of positives divided by the number of all samples [31]. This means that AUPRC baseline is different for data sets with differing prevalence and is only the same as it is for AUROC (0.5) when the data set is balanced.

2.5.4 Hyperparameter Tuning

In addition to the learnable parameter values that the machine learning model automatically tunes during the training stage to get the best fit, many models have other parameters called *hyperparameters* that have to be manually set before fitting the model. The hyperparameters can be, for example, the learning rate of the neural network, the number of trees in a random forest, or the loss function in gradient boosting. The other parameters that the model automatically learns can be for example the weights and biases of a neural network model. Finding the hyperparameter values that produce the best model is called *hyperparameter tuning*, or more precisely *hyperparameter value tuning* (since the values are the subject of the tuning) and sometimes *hyperparameter optimization*.

A simple way to tune the values of the hyperparameters is to choose them manually. This method can be very effective if there is some domain knowledge or extensive experience working with the model type in question. Manual selection for the values of the hyperparameters can be laborious and not very effective if there is not much knowledge of what the optimal hyperparameter combination could be. In this case, one might want to employ some automatic methods for the selection of the optimal values for the hyperparameters.

Two simple methods for automatic hyperparameter tuning are grid search and random search [32]. In grid search, one selects a range of values for all hyperparameters that should be optimized. For hyperparameters that have a contiuous range, the range has to be discretized. The grid search then automatically goes through all the hyperparameter combinations, trains and evaluates the model with all of the combinations and selects the combination of hyperparameters that results in the best fit. The advantage and the disadvantage of grid search is that it exhaustively searches the specified hyperparameter space and finds the optimal hyperparameter values from the specified ranges. At the same time, this is a disadvantage since the exhaustive search is computationally intensive and not very efficient.

An alternative to grid search is random search. Similarly to grid search, a range of hyperparameter values is specified, but instead of exhaustively trying all of the possible combinations, the hyperparameter combinations are randomly sampled from the ranges. For random search, continuous ranges do not have to be discretized but values can be sampled from a distribution. This method is not as computationally demanding since not all of the combinations are tested and usually a good combination of parameters are found with a relatively small computational cost. Bergstra et al. have demonstrated that given the same computational budget, random search finds an equally good or better hyperparameter configuration as grid search [33]. On the other hand, since the combinations are randomly sampled and not all combinations are tested, the optimal combination of parameters is not necessarily found.

Other hyperparameter optimiation methods include for example Bayesian optimization [34], gradient based optimization [35] and evolutionary optimization [36].

3. Federated Learning

This chapter contains an introduction to federated learning and a description of the models used in the experiments of this thesis, as well as a short discussion of privacy protection.

3.1 Federated Deep Learning

Federated learning is a distributed machine learning method [37] where multiple models are trained on a loose federation of remote sites and then aggregated into a single, final model. Distributed machine learning methods differ from centralized methods in that the distributed methods use multiple compute nodes instead of just one machine, and federated learning differs from other distributed learning methods in that the data need not be shared between the different compute nodes in federated learning.

In federated learning, the models trained on the remote sites can be called local models and the final aggregate model can be called the global model. The term federated learning was first coined by McMahan et al. [38] in 2015. As we can later see in Section 3.4, different federated learning techniques can achieve comparable performance compared to a centralized model which has all the data in a single location, therefore eliminating the need to collect data centrally. The minimum requirement for federated learning to be of any utility is to have better performance than a model trained with the data of a single site.

The main difference between federated learning and more traditional machine learning methods is that in the traditional setting, a model is trained on a single centralized data set, but in federated learning the model is trained on multiple distinct data sets which are generally located far away from each other. In a typical setting for federated learning, the data is distributed unevenly to multiple sites and there is a central server coordinating the learning process. Yang et al. [39] categorized federated learning into two classes: vertical federated learning and horizontal federated learning.

If we consider a dataset where rows represent different observations and columns represent features, vertical federated learning can be defined as federate learning when the data is partitioned so that all sites have the same rows but different columns. An example of a situation for vertical federated learning would be a case where the data is partitioned to two sites, for example a bank and a an insurance company. Here both of the sites could have different kinds of information about the same customers.

Horizontal federated learning on the other hand can be defined as a federated learning scenario where the data is partitioned so that all of the sites have the same columns but different rows. A typical example is in healthcare where different hospitals take the same measurements but from different patients. The data used in this thesis is partitioned horizontally so that all sites share the same feature space but have data about different patients.

3.1.1 Federated Averaging

Federated averaging was first described by McMahan et al. [4] for updating the global federated neural network model by averaging the updates of the local models. Before training with federated averaging can start, the model structure has to be defined and the model parameter values initialized in the central server. The training in the federated averaging starts with the central server distributing the model to the clients. After the clients have received the model, each client calculates the gradients of the loss function using it's own training data, updates the local model, and sends the gradient updates to the central server. Typically, the local models are updated multiple times before the update is sent to the central server. The central server then calculates the weighted average of these gradient updates and updates the global model. After the global model is updated, the new parameter values are sent to the clients, the local models are updated with the global parameter values and the training can continue.

Let us now describe federated averaging in a more formal way. Each client k calculates the gradient updates as $g_k = \nabla F_j(w_t)$ on it's current model w_t. The central server then updates the global model by aggregating these gradient updates: $w_{t+1} \leftarrow w_t - \eta \frac{1}{n} \Sigma_{k=1}^{K} n_k g_k = w_t - \eta \nabla f(w_t)$, where n_k is the number of data points in client k and η is the learning rate. If one wishes to control the computational and communication costs for each round, one can e.g. perform the computations only on a fraction of the clients on each round, perform variable number of training passes on each client before aggregating the gradients and control the minibatch size on each client.

3.1.2 Secure Aggregation

Since the gradient updates collected from the clients are used just once when updating the global model, there is no need to store the individual updates and they can be discarded as soon as the gradients are aggregated at the central server. These short-lived updates are called *ephemeral updates* [4]. Using ephemeral updates improves security, since the data is stored only as long as it is needed. The security of federated learning can still be further imporved by using *secure aggregation* [40].

The secure aggregation protocol, developed for federated learning by Bonawitz et al. [40], allows the server to aggregate the vectors collected from clients without being able to infer anything about any individual client that is not inferable from the aggregated value. In the protocol, each user u samples a vector $s_{u,v}$ uniformly from $[0, R)^k$ for each other user v, so that each pair of users agree on a matched pair of input perturbations. Users exchange the sampled vectors and compute perturbations $p_{u,v} = s_{u,v} - s_{v,u} \pmod R$ so that each user sends to the server a perturbed vector $y_u = x_u + \Sigma_v p_{u,v} \pmod R$. The server then sums the perturbed values by computing $\bar{x} = \Sigma_u y_u \pmod R$, which gives the correct aggregated value because the perturbations cancel out because $p_{u,v} = -p_{v,u}$.

The above protocol is suitable for simple cases where none of the clients drop out during the training but Bonawitz et al. [40] also proposed refinements to the protocol to make it robust to drop outs and computationally feasible. The refinements are based on encryption and secret sharing rounds so that the server can keep track of which clients have passed each round of the protocol without knowing the contents of the messages. In the refined protocol, the clients only use perturbations of the surviving clients so that the aggregate values can be computed correctly.

3.2 Federated Gradient Boosting Decision Trees

So far, in the federated setting, we have mainly been concerned with neural networks. It is now time to take a look at how to build gradient boosting decision trees in a federated manner. Since federated averaging was based on averaging the gradient updates for the model, one might think that it is possible to use the same method for training gradient boosting decision trees. This is not the case however, mainly because of one key difference in training neural networks versus training decision trees: when using federated averaging with neural networks, the model structure is decided beforehand and communicated to all the clients but in the case of decision trees, the structure of the trees is learned from the client data. This means that when using neural networks, all of the local models have the same structure, but when using decision trees, the local models might have different structure.

3.2.1 Sequential Tree Building

In 2019, Zhao et al. [41] introduced a method to build privacy-preserving gradient boosting decision trees in the horizontal federated learning setting. They presented an algorithm to build differentially private* regression trees and used them to construct a privacy

*A short explanation of differential privacy can be found in Section 3.3

preserving gradient boosting decision tree, using data from multiple sites.

Although privacy is one key motivation for conducting the research for this thesis, differential privacy is omitted here and the method of Zhao et al. is adopted using classification trees without differential privacy. Without differential privacy, the method Zhao et al. presented for building gradient boosting decision trees is simple. The algorithm loops through each site and at each site, trees from the previous sites are used to evaluate the residuals and a new tree is built. All the trees are then sent to the next site. On the first site, the first decision tree is constructed without the residuals evaluated using other trees since no trees are yet built, and on the last site, after the final tree is constructed, all the trees are sent to the central server if the training is completed, or back to the first site if multiple training rounds are performed.

Communication Costs

For sequentially built federated gradient boosting decision trees, the communication costs depend on four factors: the size T of each tree, the number of trees M built on each site at each training round, the number of clients K and the number of training rounds L. Since each site sends M trees it built, each of size T, and the trees it has received from the previous site, to the next site we get an equation for the communication costs:

$$C_{\text{total}} = \sum_{i=1}^{K} TMi, \tag{3.1}$$

and if multiple training rounds are performed (i.e $L > 1$)

$$C_{\text{total}} = \sum_{i=1}^{LK} TMi. \tag{3.2}$$

For calculating the communication cost between site i and $i+1$ we have

$$C_{i,i+1} = (L-1)K + iM. \tag{3.3}$$

Computational Costs

The computational complexity of training trees on a single site depends on the size of the dataset and the number of trees, so for a singe site the time complexity is $\mathcal{O}(MN)$ where M is the number of trees. If the number of sites is K and the number of training rounds is L we end up with a total computational cost of

$$T_{\text{GB}} = \mathcal{O}(LKMN) \tag{3.4}$$

3.2.2 SimFL

In 2019, Li et al. proposed a new method for federated gradient boosting that uses p-stable locality-sensitive hashing [42] and weighted gradients, called SimFL [43]. As advantages compared to other methods, they cited better accuracy since the method uses information from all parties to boost each tree and improved efficiency since it does not use complex cryptography that some other methods use. Instead, the method uses locality-sensitive hashing to find similar instances between sites and calculates the gradients by combining the gradients of similar instances. The method consists of a separate preprocessing stage where the similarity information is calculated, and a training stage where the boosted trees are fit.

The authors claim a computational overhead of $\mathcal{O}(NL + Nd)$ for the preprocessing stage where L is the number of hash functions and d is the number of dimensions, and an overhead of $\mathcal{O}(NT)$ for the training stage where T is the number of trees. The behavior of $\mathcal{O}(NL+Nd)$ for the preprocessing stage was not able to be reproduced for this thesis but instead the overhead was $\mathcal{O}(N^2)$. This meant that the preprocessing would have taken a too long time, so no experiments were carried out for the SimFL method.

3.3 Privacy Protection

One of the main motivation for federated learning is to protect the privacy of participants by not moving any raw data between clients. Especially in the medical setting, there might be legal considerations that prevent collecting the data to any central location, so to access data from multiple locations, federated learning might be required. Nevertheless, federated learning alone can not guarantee privacy, since there are many ways to attack machine learning models no matter how they are trained. This section provides a brief overview of some attacks against machine learning models and ways to protect the models, even though no privacy protection methods are implemented with the models used in this thesis.

One of the methods to attack machine learning models is membership inference where the attacker has access to a black-box model and tries to infer wether a specific data record was used in training the model [44].

Shokri et al. described a way to use so called *shadow models* to perform membership inference attacks [45]. They trained multiple shadow models using similar data as the model that the attack targeted used, to create models that behave similarly to the targeted model and used the shadow models to teach the attack model to distinguish between data that was used in training of the shadow models and data that was not.

Another type of attack is called a *model inversion attack*. In a model inversion attack, the attacker has access to a model trained with a data set $\boldsymbol{D_1}$ but not to the data set $\boldsymbol{D_1}$ itself. The attacker can then, having access to another dataset $\boldsymbol{D_2}$ that has some shared variables with $\boldsymbol{D_1}$, retrieve some variables from $\boldsymbol{D_1}$ for the individuals that are in both $\boldsymbol{D_1}$ and $\boldsymbol{D_2}$ [46]. Fredrikson et al. showed that given white-box access to a decision tree model, one can retrieve sensitive information about the source data with high precision [47].

One way to provide rigorous mathematical privacy-guarantees is to use a technique called differential privacy [48]. In essence, differential privacy seeks to ensure that for any given individual whose data is included in the dataset, no more information will be revealed about them than would be if their data were not included in the dataset [49]. The privacy loss associated with a randomized function can be mathematically represented with ϵ-differential privacy [49] [50]:

Definition 1. *A randomized function K gives ϵ-differential privacy if for all data sets D_1 and D_2 differing on at most one row, and all $S \subseteq$ Range(K),*

$$Pr[K(D_1) \in S] \leq \exp(\epsilon) Pr[K(D_2) \in S] \tag{3.5}$$

3.4 Related Work

In literature, many previous studies about different federated random forest and gradient boosting methods can be found. Liu et al. [51] proposed a lossless federated random forest algorithm for vertically partitioned data. In their algorithm, each tree is built jointly with all clients and each client stores the split information about the features in their nodes. In their tests, the method outperformed local models and was as accurate as the centralized model. The method is different from the federated random forest used in this thesis and it is not applicable for the experiments of this thesis since eICU data is horizontally partitioned.

Wang et al. [52] proposed a federated XGBoost method for mobile crowdsensing, which was efficient and secure to private data leakage. Another federated gradient boosting method called SecureBoost was proposed by Cheng et al. [53]. They provided a theoretical proof that SecureBoost is as accurate as a centralized gradient boosting model.

Brisimi et al. [54] developed a federated optimization scheme for support vector machines to predict future hospitalizations from EMR data.

Liu et al. [55] developed a federated autonomous deep learning (FADL) method to predict mortality during ICU stay. The dataset was the same as the one used in this thesis, the eICU dataset, but the model inputs were only the medications taken during the 24 hours of the ICU stays. FADL outperformed the original federated learning method and achieved the same AUROC as centralized learning and a higher AUPRC than centralized learning. The results can be found in Table 3.1.

Table 3.1: AUROCs and AUPRCs for federated autonomous deep learning [55]

Training method	AUROC	AUPRC
Centralized deep learning	0.79	0.21
Original federated learning	0.75	0.16
Federated autonomous deep learning (FADL)	0.79	0.23

Huang et al. [56] demonstrated that patient clustering can improve the efficiency of federated learning to predict mortality and stay time in the ICU. Their centralized model achieved an AUROC of 0.6811 and AUPRC of 0.0947 while for their federated model the same numbers were 0.6520 and 0.0871 and for their best community-based federated model 0.6628 and 0.0912.

Beaulieu-Jones et al. [57] compared centralized and federated deep learning for mortality prediction using the eICU database. They predicted mortality after 24 hours in the 5 largest hospitals. Table 3.2 shows that their federated models achieved close to the same level of AUROC as the centralized models.

Table 3.2: AUROCs for centralized and federated models in for the 5 largest hospitals [57]

Institutions	Centralized AUROC	Federated AUROC
1	0.743	0.738
2	0.760	0.762
3	0.794	0.789
4	0.804	0.797
5	0.808	0.801

In addition to the above studies that used federated learning, multiple studies have been conducted using centralized learning methods for mortality prediction. Sheikhalishahi et al. [58] compared a bidirectional long short-term memory (BiLSTM) neural network model to logistic regression, 1-layer neural network, and Acute Physiology and Chronic Health Evaluation (APACHE) [59] [60] score for predicting mortality during the first 24 and 48 hours of ICU stay, using the eICU dataset. The BiLSTM model outperformed all of the baseline models achieving an AUROC of 83.30 and AUPRC of 48.72 for the prediction for the first 24 hours and an AUROC of 86.63 and AUPRC of 55.20 for the prediction for the first 48 hours.

Ding et al. [61] developed a novel method combining just-in-time learnign (JITL) and extreme learning machine (ELM) called JITL-ELM for mortality prediction. They used data from PhysioNet [62] and achieved an AUROC of 0.8568.

Ge et al. [63] developed an interpretable LSTM model for ICU mortality prediction using data from Asan Medical Center hospital and achieved an AUROC of 0.7614.

4. A Novel Method for Federated Random Forest

Since no suitable federated random forest methods were found for the use case of this thesis from the literature, a novel federated random forest method was developed for this thesis. To the best knowledge of the author, this exact method has not been studied before. This chapter provides a description for the federated random forest algorithm developed for this thesis.

4.1 Federated Random Forest

Since the trees of a random forest are not dependent on each other and the prediction of the forest is taken to be the mode of the prediction of individual trees, the most straightforward method to create a federated random forest is to build a local random forest at each site and combine all the local forests into a single, larger global forest. The prediction of the combined forest is then taken to be the mode of all the individual trees from all of the forests built on each site.

There are many advantages to using this simple method to build a federated random forest compared to other federated methods. First one is that it is easy to understand and implement, since it does not differ much from the regular, centralized random forest. In fact, the only difference is that the bootstrap sets used to build trees are not drawn from the same data for all trees. Another advantage is that the communication costs are low since each site only needs to send the final forest to the central server, so no communication is needed between the central server and the sites or between sites during the training. The sites need not communicate anything to each other so the training can proceed in parallel at each site. Finally, it is easy to include more sites later on, since a new forest can be trained in isolation and added to the global forest in the central server. No further modifications are needed when new sites are introduced. Likewise, if the information about which trees came from which sites is stored, it is easy to discard the trees that came from a specific site, removing the influence of that site's data to the final model, if the need arises.

4.1.1 Communication Costs

For the federated random forest method, there are three factors affecting the communication costs: the size T of each tree, number of trees M built at each client and the number of clients K. If all trees are equal size and all clients build the same number of trees the total communication costs can be expressed as

$$C_{\text{total}} = TMK, \tag{4.1}$$

since the only communication needed is when the clients send the fully grown forest to the central server. For the communication cost between a single client and the central server, this equation reduces to

$$C_{\text{client}} = TM. \tag{4.2}$$

4.1.2 Computational Costs

Louppe [64] demonstrated that the worst case time complexity for random forest is

$$T_{\text{RF}} = \mathcal{O}(mN^2 \log N), \tag{4.3}$$

where m is number of input variables considered during each split and N is the size of the data set. Since the training of all local forest in the federated random forest method can be done in parallel, the time-complexity depends only on the size of the largest data set and the time-complexity becomes

$$T_{\text{RF}} = \mathcal{O}(mN_{\max}^2 \log N_{\max}), \tag{4.4}$$

where $N_{\max}$ is the size of the largest data set among the sites.

5. Dataset and Experiments

This chapter presents a description of the data and experiments used in this thesis. The eICU data set will be described along with the prediction targets, selected features and selected cohort. The chapter concludes with a description of the experiments conducted in this thesis.

5.1 The eICU Dataset

The dataset used in this thesis is the eICU Collaborative Research Database [65]. The database consists of data from 208 hospitals, collected with the Philips eICU Program, around the United States. There are 200 859 patient unit encounters from 139 367 unique patients admitted between 2014 and 2015.

The data is de-identified with measures such as removing all protected health information and randomly assigning a unique identifier for all patients to protect the privacy of the patients [65]. Before using the data set, the researcher has to complete training (course on Human Subjects Research) related to sensitive data and ethical conducting of research. The researcher also has to agree to rules, such as not sharing or trying to re-identify the data, and releasing code associated with any publication using the data.

For this thesis, the most relevant information contained in the data set are the vitals, lab results, and admission and discharge times as well as discharge status (alive or expired). In addition, the dataset of course contains information to identify distinct patients, unit stays and hospitals, such as id for the unit stay of a patient and and ids for the hospitals.

5.1.1 Targets and Prediction Windows

The binary classification task in this thesis is to predict death in the ICU. The observations where a patient is dead were labeled a positive label and the ones where the patient is alive were labeled with a negative label.

Since the eICU data set does not contain the exact time of death for each patient, but only contains the information of whether the patient has died in the ICU or not and the discharge time from the ICU, it is impossible to accurately identify the time of death for the patients. Since it is expensive to keep patients in the ICU and the number of available beds is limited, it can be expected that expired patients do not spend very long times in the ICU before they are discharged.

In the experiments for this thesis, the time of death for patients was considered to be three hours before the discharge from ICU. This is a crude approximation and most of the patients probably died closer to the discharge time and some patients probably died earlier than three hours before discharge. Making such an approximation not only makes the prediction task harder, but also makes it more difficult to measure the absolute performance of the models. The same approximation is made for all the models however, and since the main objective in this thesis is to compare the performance of the federated and centralized models, it can be assumed that the approximation affects all models in roughly the same way, so the results can be compared. If the main objective would be to produce the best predictive model possible or to examine the performance of a single model this approximation would be more problematic but since we are more interested in comparing the results of different models than the absolute performance of any given model, this will not be such a big problem.

The clinical relevance of a model that predicts patients' death in the ICU depends on the system to notify the clinician about the patient's status at the appropriate time. According to Lilly et al., the average length of stay in the ICU in USA is 3.28 days [66]. The individual variability for the length of stay is large however, and in some cases can vary even between 1 and 132 days [67]. Hence, in some cases, in theory it might be possible to make predictions very early on. Nevertheless, predictions made too early are not useful for the clinician since such an early prediction does not prompt the clinician to adjust their actions yet. On the other hand, predictions made too close to death do not leave enough time for the clinician to make actions that could save the patient. To study a clinically relevant scenario, the target in this thesis was set to predict death 8 hours in advance.

In an unprocessed data set, each patient that died would have just one observation labeled as positive at the time of death of the patient and all other observations would be labeled as negative. To enable predicting death 8 hours prior to the time of death, all the observations 8 hours prior to the (approximated) time of death for each patient were also labeled as positive during the data preprocessing stage. All observations prior to 8 hours before the time of death were labeled as negative in preprocessing. Taking into account that the true time of death is not known for the patients in the data set and the time of death was approximated to be 3 hours prior to discharge, the time-window that the observations are labeled as positive varies from 8 to 11 hours before the actual time of death, assuming that no patients died prior to 3 hours before discharge.

5.1.2 Cohort Selection

For this thesis, a specific cohort of patients was selected from the eICU database. The criteria for the selection was that for each patient there exists at least one measurement of each vital (heart rate, respiration rate, temperature, SpO2 and blood pressure), at least one measurement of the most common labs (calcium, chloride, creatinine, glucose, potassium, sodium and blood urea nitrogen) and at least one measurement of PaO2 and PaCO2. After this selection, the dataset contains a total of 20 959 distinct patients and 69 hospitals. The dataset was then further divided to a training set (51 hospitals, 11281 patients) and a test set (18 hospitals, 5484 patients).

5.1.3 Data Statistics

The target was to predict death during patients' ICU stays. The prevalence of death among the patients in the training set was 8.6% and in the test set 8.7%. However, for the whole training data set, row-wise and after the features were computed, the prevalence was only 0.89% and for the test set 0.87%. This puts the baseline for AUPRC in the training set to 0.0089 and in the test set to 0.0087.

Feature Extraction

New features were calculated from existing variables on the dataset, using different methods. A summary of which features are calculated for each type of variable are presented in Table 5.1 and the full list of all features can be found in Table A.1 in Appendix A.

Table 5.1: Features

Variable	Window	Last	First	Mean	Median	Min	Max	Slope	Std Dev	Delta	$P_{90\%}$	$P_{10\%}$
Vitals	8h	x	x	x	x	x	x	x	x	x	x	x
FiO_2	1h	x										
PEEP	1h	x										
GCS	24h	x										
Labs	48h	x										

Missing Value Imputation

Due to the nature of ICU data, a large proportion of values are missing. Typically vitals such as heart rate are measured continuously, whereas labs are ordered by doctors at different intervals, creating a situation where there are not many missing values for vitals but many missing values for labs. The status of the patients affects which labs and measurements are useful for doctors, and doctors use their own judgment in deciding which measurements to take, so the data can not be considered to be *missing at random.*

Since most observations have at least one missing value, it is not feasible to discard the observations with missing values. Decision trees in general could be able to handle missing values, but the implementations of random forest and gradient boosting from python's Scikit-learn library [28], which are used in the experiments of this thesis, do not handle missing values, the only option left is to impute the missing values.

The imputation was done using so-called forward fill, i.e. using the previous non-missing value to impute the missing values. If no previous non-missing value was found, the missing values were imputed with the median for the feature.

5.2 Experiments

Since the goal of this thesis is to study how well federated learning compares to centralized learning in the healthcare setting, a few scenarios emulating real world situations are simulated. In the first scenario, both the centralized and the federated models are trained with the full training set. In the second scenario, the centralized models have data from a few hospitals, and the federated models have data from a larger subset of hospitals from the training set. In the third scenario, data from a few hospitals is gathered centrally and the rest of the data remains remote. Federated models are then trained on the centralized plus the remote data.

The values for the hyperparameters for all models are tuned using random search with cross-validation. Hyperparameters that were tuned for the random forest were the number of trees, the impurity measure (gini or entropy) and the maximum depth of the trees. For gradient boosting the hyperparameters that were tuned were the loss function, number of trees, the learning rate and the maximum depth of the trees. The federated random forest had the same hyperparameters tuned as the centralized version but the federated gradient boosting had the number of training rounds tuned in addition to the same hyperparameters that were tuned for the centralized model.

The centralized models use standard 5-fold cross-validation and the federated models use multiple-source cross-validation. From the cross-validation results, the model with the highest AUPRC is chosen and the model is retrained on the training set without cross-validation. After retraining, the models are evaluated on the test set, which remains the same for all scenarios.

5.2.1 Experiments on Full Training Set

In the first scenario, both the centralized models and the federated models have access to the full training set. The centralized models treat the training set as a single data set while the federated models split the training set to multiple data sets, each containing data from a single hospital. For the federated random forest, this means that 51 random forests are trained, one on each hospital's data, and the trees from these forests are combined into the final and larger federated random forest. For the federated gradient boosting, it means that the method sequentially goes through each hospital's dataset, training a number of boosted trees on each dataset.

The goal of this scenario is to study the difference in performance of centralized and federated training on the same dataset. The practical implications of this experiment are that if federated training can offer similar level of performance as centralized training, costs and labour related to gathering data from different sources to a central location could be eliminated with federated training.

5.2.2 Centralized and Federated Training on a Subset of Hospitals

In the second scenario, the models are trained with subsets of the hospitals in the full training set. Three sizes of subsets are considered for the centralized models: one with only the data from the largest hospital, another with data from the two largest hospitals and third with data from the three largest hospitals. Both the centralized random forest and gradient boosting are trained on all of these subset sizes, using cross-validation to find the best models for each subset.

For the federated models, four sizes of subsets are considered: subsets with data form the 5, 10, 15 and 20 largest hospitals. Both federated model types are trained on each subset and the best models for both model types are chosen with cross-validation for each subset.

The goal of this scenario is to study whether federated training can provide superior results compared to centralized training, if one has access to more data via federated training.

5.2.3 Federated Training on Top of Centralized Training

The third scenario consists of two different subscenarios with four sizes of subsets. In the first subscenario, the federated models have data from the two largest hospitals centralized and non-centralized data from additional 3, 8, 13 and 18 hospitals. In other words, the federated models here have the same data as in the second scenario but data from the two largest hospitals is considered as a single site.

The second subscenario is otherwise identical to the first subscenario, but instead of having two largest hospitals centralized, the models have three largest hospitals centralized.

The goal of this scenario is to study whether a model trained centrally can be improved with further federated training or can the performance of federated training be improved if some of the data is gathered centrally.

6. Results

This chapter presents the results for each experiment separately. The centralized models, that treat the data from different hospitals jointly as a single dataset, are denoted as RF for random forest and GB for gradient boosting. The federated models, which treat the data from different hospitals as separate datasets, are denoted as FedRF and FedGB for federated random forest and federated gradient boosting, respectively.

6.1 Models Trained on Full Training Set

The first scenario was to train both the centralized models and the federated models with the whole training set and compare the test set results from the centralized and federated models.

Comparing models trained with data from all hospitals in the training set, Figure 6.1 and Figure 6.2 show that the centralized gradient boosting model has the best performance on the test set, both in terms of AUPRC (0.147) and AUROC (0.886). The centralized random forest performs nearly as well as the gradient boosting model, achieving an AUPRC of 0.142 and an AUROC of 0.869. Both of the federated models perform worse than the centralized models with the federated random forest achieving an AUPRC of 0.112 and AUROC of 0.846 and the federated gradient boosting performing the worst with an AUPRC of 0.098 and AUROC of 0.823.

From Figure 6.1 and Figure 6.2 we can see that all of the models performed significantly above the baseline AUPRC of 0.0087 and AUROC of 0.5. The figures also show that the cross-validation and test set performances of the models are close to each other so no significant overfitting has occured. The best centralized model (gradient boosting) achieved a 31% better AUPRC compared to the federated random forest model and 50% higher AUPRC compared to the federated gradient boosting model. For centralized random forest the same percentages are 27% and 45%. The centralized gradient boosting achieved almost 17-fold (16.9) increase to the baseline AUPRC while the federated random forest still achieved almost a 13-fold (12.9) increase to the baseline AUPRC. Table 6.1 summarizes the AUROC and AUPRC results of all models in one table.

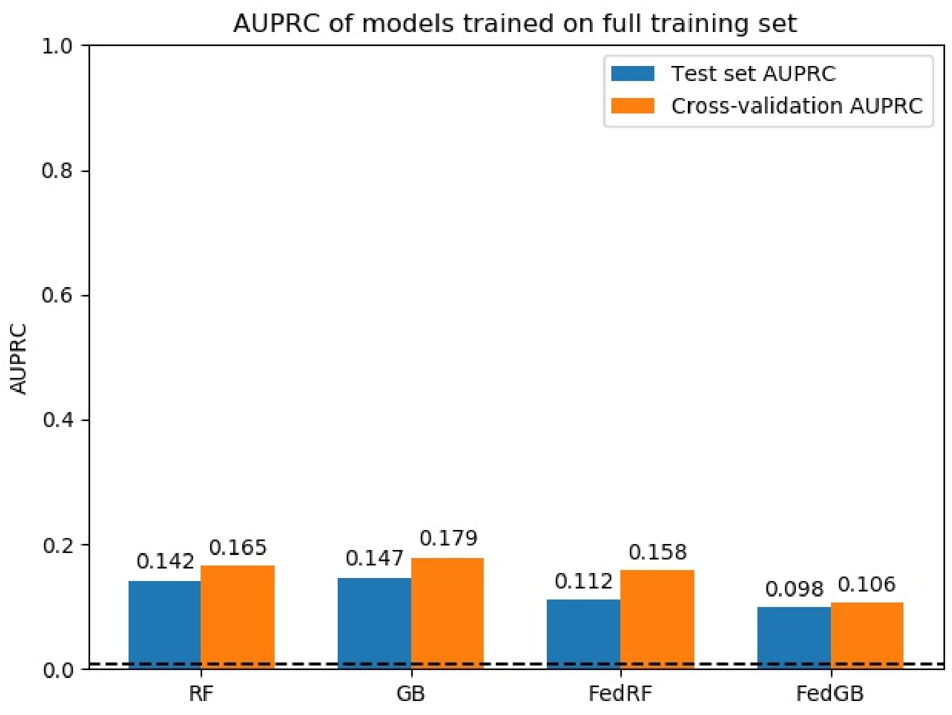

Figure 6.1: AUPRCs of centralized and federated models trained on the full training set. The baseline AUPRC is marked with a dashed line.

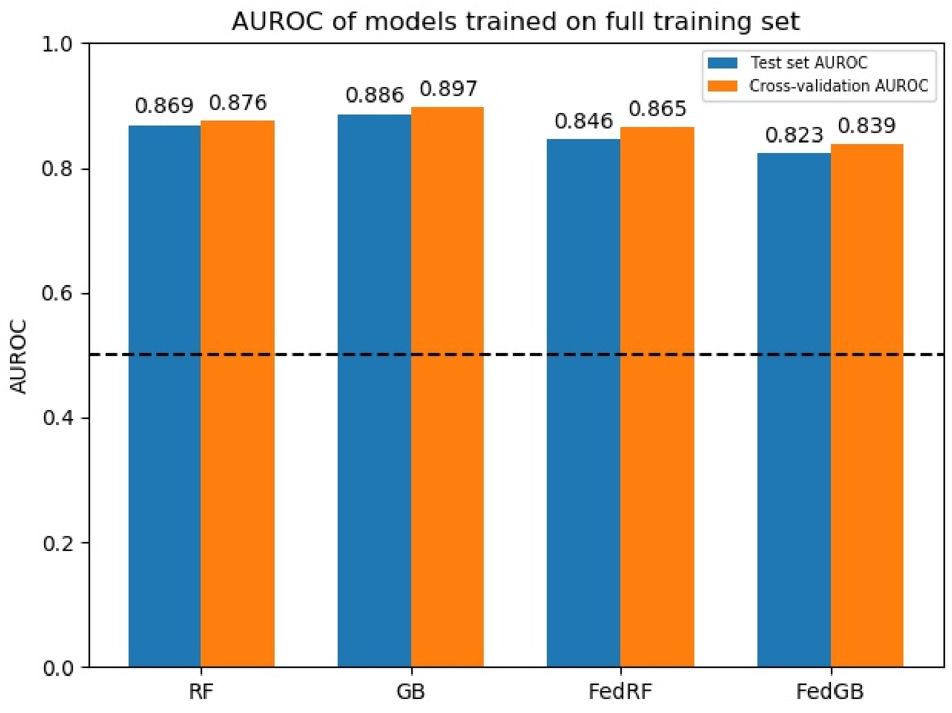

Figure 6.2: AUROCs of centralized and federated models trained on the full training set. The baseline AUROC is marked with a dashed line.

Table 6.1: Comparison of federated and centralized models trained on full data from all hospitals

	RF	GB	FedRF	FedGB
Cross-validation AUPRC	0.165	0.179	0.158	0.106
Cross-validation AUROC	0.876	0.897	0.865	0.839
Test set AUPRC	0.142	0.147	0.112	0.098
Test set AUROC	0.869	0.886	0.846	0.823

6.2 Models Trained on a Subset of Training Set Hospitals

To study the effect of having access to more data with federated training, all of the models were trained with data from differing amounts of hospitals. The centralized models were trained with three different sized data sets: data from the largest hospital, from two largest hospitals and from three largest hospitals. Similarly, the federated models were trained with data from the 5, 10, 15 and 20 largest hospitals. The test set remained the same as in the above section and all of the federated models were compared to the three centralized models.

6.2.1 Centralized Models with Data from the Largest Hospital

Figure 6.3 and Figure 6.4 show that when the centralized models were trained with data from only the largest hospital, the centralized random forest achieved an AUPRC of 0.111 and AUROC of 0.848 while the centralized gradient boosting achieved an AUPRC of 0.089 and AUROC of 0.850. All of the federated random forest models outperformed both of the centralized models and the model with data from the 15 largest hospitals performed the best with an AUPRC of 0.129 and AUROC of 0.860. None of the federated gradient boosting models outperformed the centralized random forest model and only the federated gradient boosting model with data from the 20 largest hospitals outperformed the centralized gradient boosting with an AUPRC of 0.104 while the AUROC was 0.840 and below the centralized model's AUROC.

We can also see from Figure 6.3 and Figure 6.4 that performance was still significantly above the baseline, but the difference between cross-validation and test set performance is larger than in the previous scenarios, so some slight overfitting may have started to happen. Table 6.2 has both the AUPRC and AUROC results and we can see that the federated random forest with data from 15 hospitals performed best on both metrics.

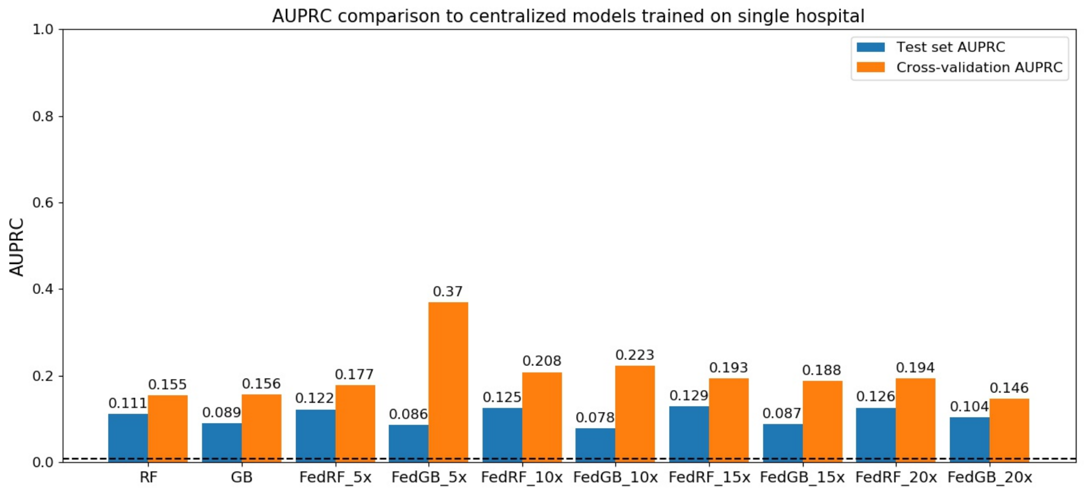

Figure 6.3: AUPRCs of centralized models trained on the data from the largest hospital and federated models trained on data from varying number of hospitals. The baseline AUPRC is marked with a dashed line.

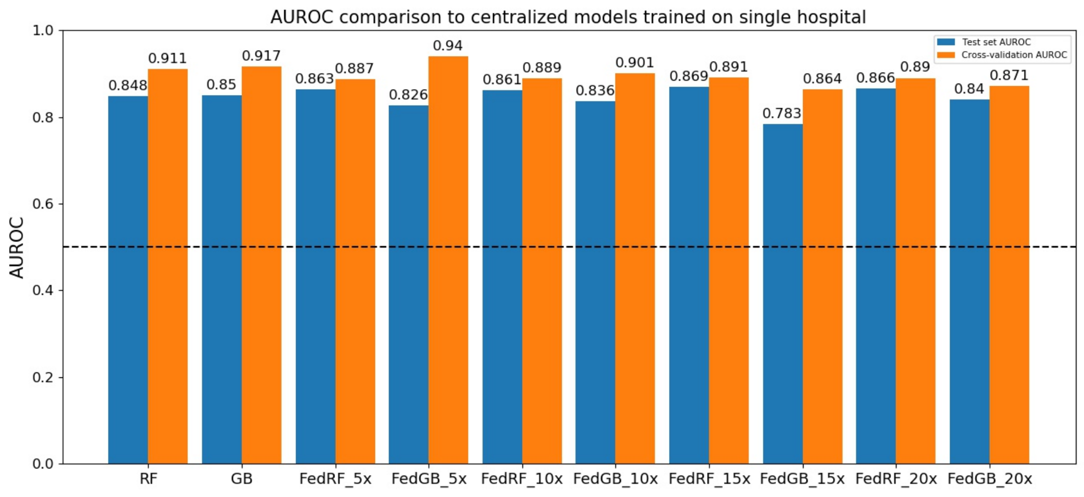

Figure 6.4: AUROCs of centralized models trained on the data from the largest hospital and federated models trained on data from varying number of hospitals. The baseline AUROC is marked with a dashed line.

Table 6.2: Comparison of federated models to centralized model with data from 1 hospital

	RF	GB	FedRF 5x	FedGB 5x	FedRF 10x	FedGB 10x	FedRF 15x	FedGB 15x	FedRF 20x	FedGB 20x
Cross-validation AUPRC	0.155	0.156	0.177	0.370	0.208	0.223	0.193	0.188	0.194	0.146
Cross-validation AUROC	0.911	0.917	0.887	0.940	0.889	0.901	0.891	0.864	0.890	0.871
Test set AUPRC	0.111	0.089	0.122	0.086	0.125	0.078	0.129	0.087	0.126	0.104
Test set AUROC	0.848	0.850	0.863	0.826	0.861	0.836	0.869	0.783	0.866	0.840

6.2.2 Centralized Models with Data from the 2 Largest Hospitals

Figure 6.5, Figure 6.6 and Table 6.3, describe the results for the case where the centralized models were trained with data from the two largest hospitals and the results for the same federated models as in Section 6.2.1. In this scenario, the gradient boosting model achieved an AUPRC of 0.135 and an AUROC of 0.863 while the random forest model performed slightly worse with an AUPRC of 0.131 and an AUROC of 0.862. Both centralized models outperformed the best federated model in terms of AUPRC (0.126) but in terms of AUROC, the best federated model performed slightly better with an AUROC of 0.869.

Figure 6.5 and Figure 6.6 also show that the centralized models with data from two hospitals still achieve above baseline performance and no significant overfitting is happening. In terms of AUPRC, the centralized models have started to outperform the federated models and the cross-validation and test set performance are closer to each other than with the federated models, suggesting lesser overfitting with the centralized models. Table 6.3, which shows both the AUPRC and AUROC results in a single table, we can see that the same model was not the best on both metrics this time and that the centralized gradient boosting was best in terms of AUPRC and the federated random forest with data from 15 hospitals was best in terms of AUROC.

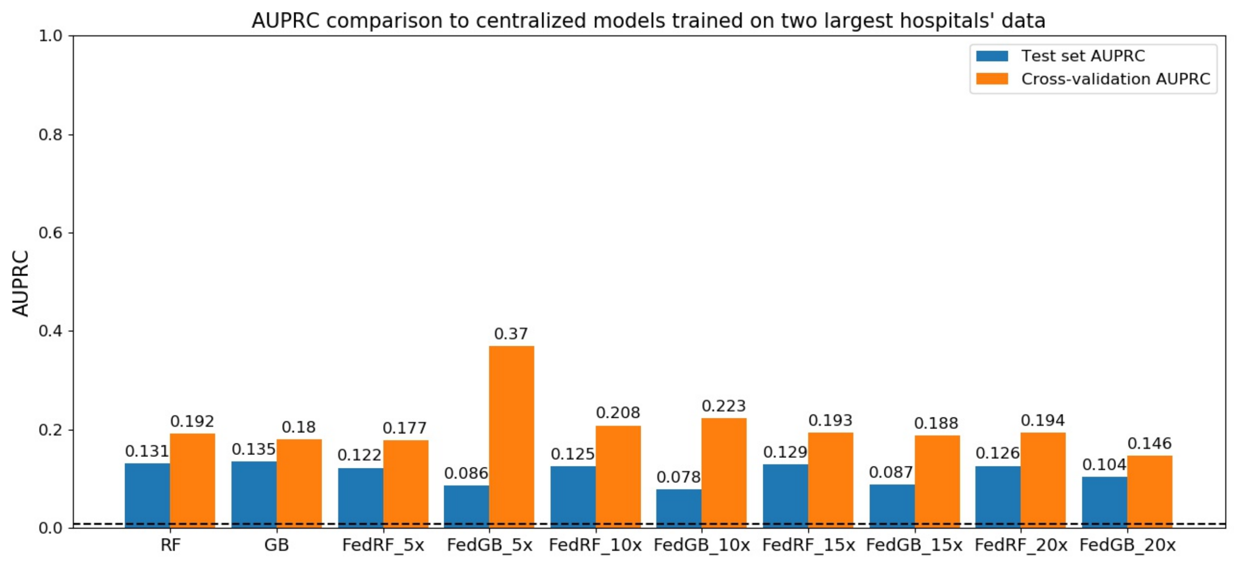

Figure 6.5: AUPRCs of centralized models trained on the data from the two largest hospitals and federated models trained on data from varying number of hospitals. The baseline AUPRC is marked with a dashed line.

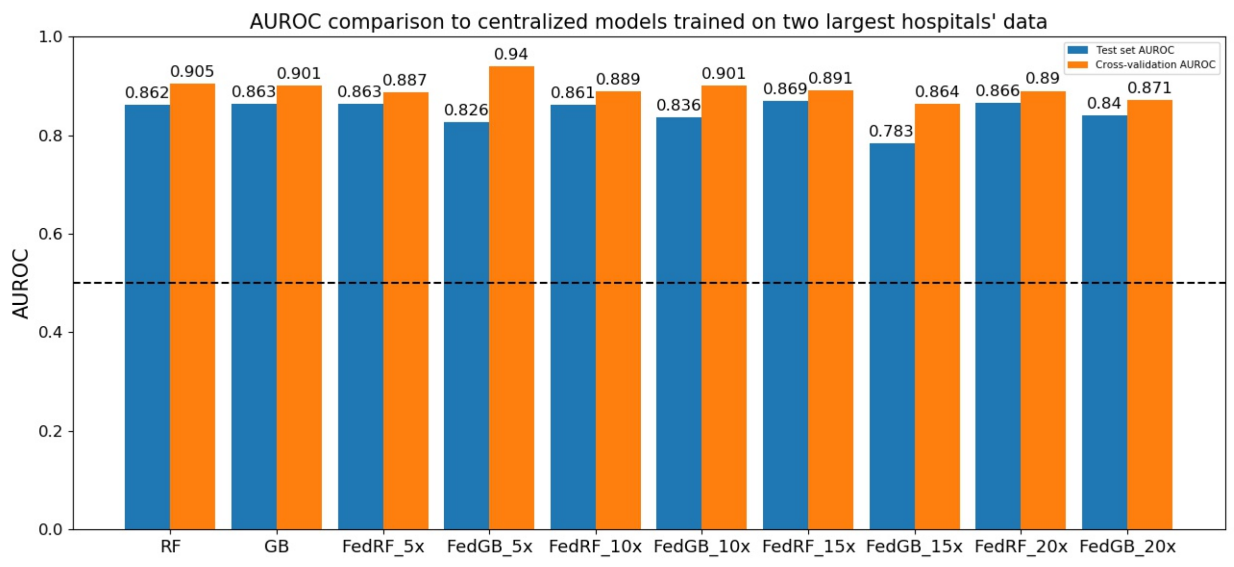

Figure 6.6: AUROCs of centralized models trained on the data from the two largest hospitals and federated models trained on data from varying number of hospitals. The baseline AUROC is marked with a dashed line.

Table 6.3: Comparison of federated models to centralized model with data from 2 hospitals

	RF	GB	FedRF 5x	FedGB 5x	FedRF 10x	FedGB 10x	FedRF 15x	FedGB 15x	FedRF 20x	FedGB 20x
Cross-validation AUPRC	0.192	0.180	0.177	0.370	0.208	0.223	0.193	0.188	0.194	0.146
Cross-validation AUROC	0.905	0.901	0.887	0.940	0.889	0.901	0.891	0.864	0.890	0.872
Test set AUPRC	0.131	0.135	0.122	0.086	0.125	0.078	0.129	0.088	0.126	0.104
Test set AUROC	0.862	0.863	0.863	0.826	0.861	0.836	0.869	0.783	0.866	0.840

6.2.3 Centralized Models with Data from the 3 Largest Hospitals

Figure 6.7, Figure 6.8 and Table 6.4 describe the results when the centralized models were trained with data from the three largest hospitals and the same results for the federated models as in the two preceding subsections. The random forest achieved an AUPRC of 0.158 and an AUROC of 0.899 while the gradient boosting model achieved an AUPRC of 0.129 and an AUROC of 0.870. The centralized random forest out performed all of the federated models in this case, and the centralized gradient boosting performed equally well as the best federated model.

We can also see from Figure 6.7 and Figure 6.8 that the performance of the centralized models has become more clearly better than the federated models' performance and that any that may have happened on the models trained with less data has decreased clearly.

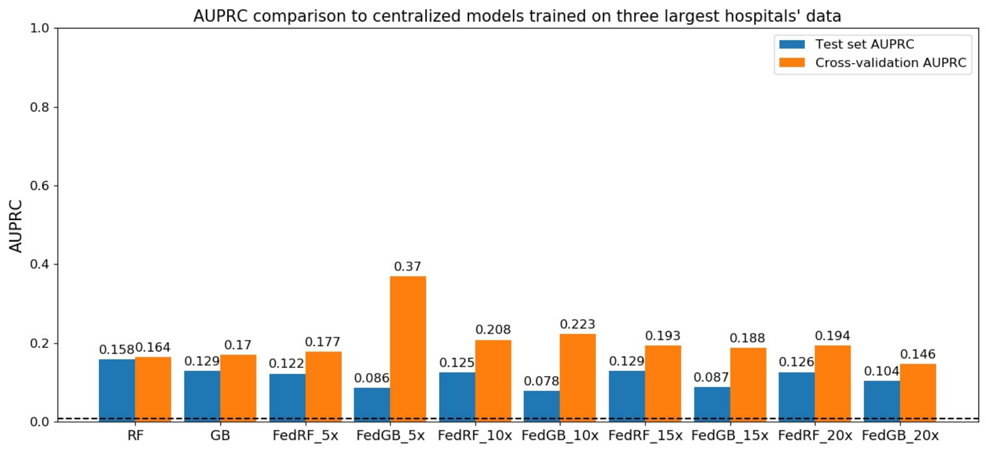

Figure 6.7: AUPRCs of centralized models trained on the data from the three largest hospitals and federated models trained on data from varying number of hospitals. The baseline AUPRC is marked with a dashed line.

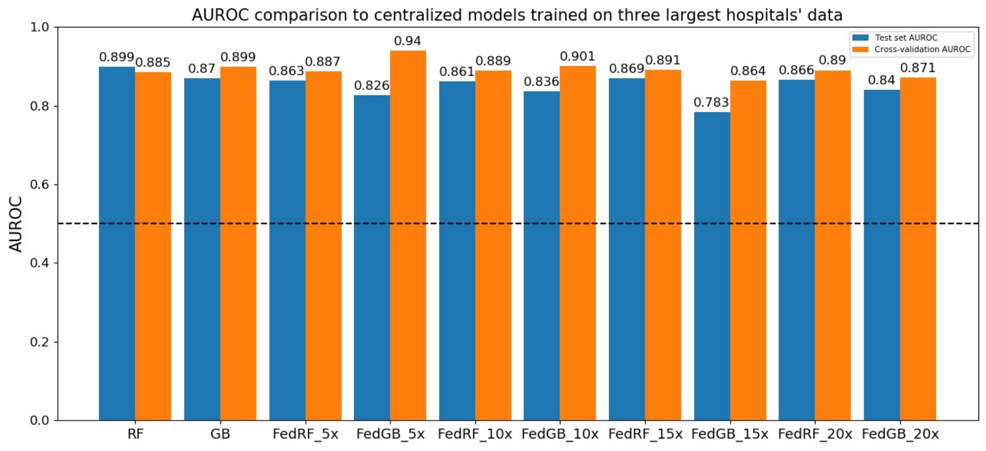

Figure 6.8: AUROCs of centralized models trained on the data from the three largest hospitals and federated models trained on data from varying number of hospitals. The baseline AUROC is marked with a dashed line.

Table 6.4 shows all the results in a single table and we can see that this time the centralized random forest was the best on both metrics.

Table 6.4: Comparison of federated models to centralized model with data from 3 hospitals

	RF	GB	FedRF 5x	FedGB 5x	FedRF 10x	FedGB 10x	FedRF 15x	FedGB 15x	FedRF 20x	FedGB 20x
Cross-validation AUPRC	0.164	0.170	0.177	0.370	0.208	0.223	0.193	0.188	0.194	0.146
Cross-validation AUROC	0.885	0.899	0.887	0.940	0.889	0.901	0.891	0.864	0.890	0.871
Test set AUPRC	0.158	0.129	0.122	0.086	0.125	0.078	0.129	0.088	0.126	0.104
Test set AUROC	0.899	0.870	0.863	0.823	0.861	0.836	0.869	0.783	0.866	0.840

6.3 Federated Training on top of Centralized Models

To study whether a centrally trained model can be improved by further training it with federated training, two scenarios were tested. In the first scenario, the centralized model has data from two of the largest hospitals and the model is further trained with data from additional hospitals so that the total number of hospitals (including the two centralized hospitals) is either 5, 10, 15 or 20. The second scenario is otherwise identical to the first but the centralized model has data from three hospitals. The results from the models with additional training are compared to the models with just the centralized data from two or three hospitals.

6.3.1 Models with 2 Centralized Hospitals

Figure 6.9, Figure 6.10 and Table 6.5 show the results for the same centralized models as in Section 6.2 but in this case the federated models had the same data centralized as the centralized models and the rest of the data as distinct (federated) datasets. Further federated training of the centralized model initially trained with data from two hospitals did not seem to improve the results, since the centralized models still perform better than the federated models. The best federated model in this case was the federated random forest trained with data from 20 largest hospitals, achieving and AUPRC of 0.127 and AUROC of 0.869 on the test set. The AUROC is slightly better than the best AUROC of the centralized model (Gradient Boosting, 0.863) and the federated random forest trained with data from 5 largest hospitals achieves the same AUROC of 0.869 but with a worse AUPRC of 0.123. The federated random forest trained with data from the 15 largest hospitals achieved the same AUPRC 0.127 but with a slightly worse AUROC of 0.867. None of the federated models achieve the AUPRC of 0.135 that the centralized gradient boosting achieved. The AUPRCs of the federated models are in fact worse than the federated random forest with fully federated training with data from the 15 largest hospitals that achieved an AUPRC of 0.129.

From Figure 6.9 and Figure 6.10 we can see that the federated models trained with data from 5 or 10 of the largest hospitals have a significant difference in cross-validation and test set AUPRC, suggesting some overfitting. The difference is not as large in the AUROC performance however.

From Table 6.5 we can see the result that in this scenario, a single model did not perform the best on both metrics but that the centralized gradient boosting perfromed the best in terms of AUPRC, and in terms of AUROC the federated random forest with data from 5 hospitals performed equally well as the federated random forest with data from 20 hospitals, outperforming the other models.

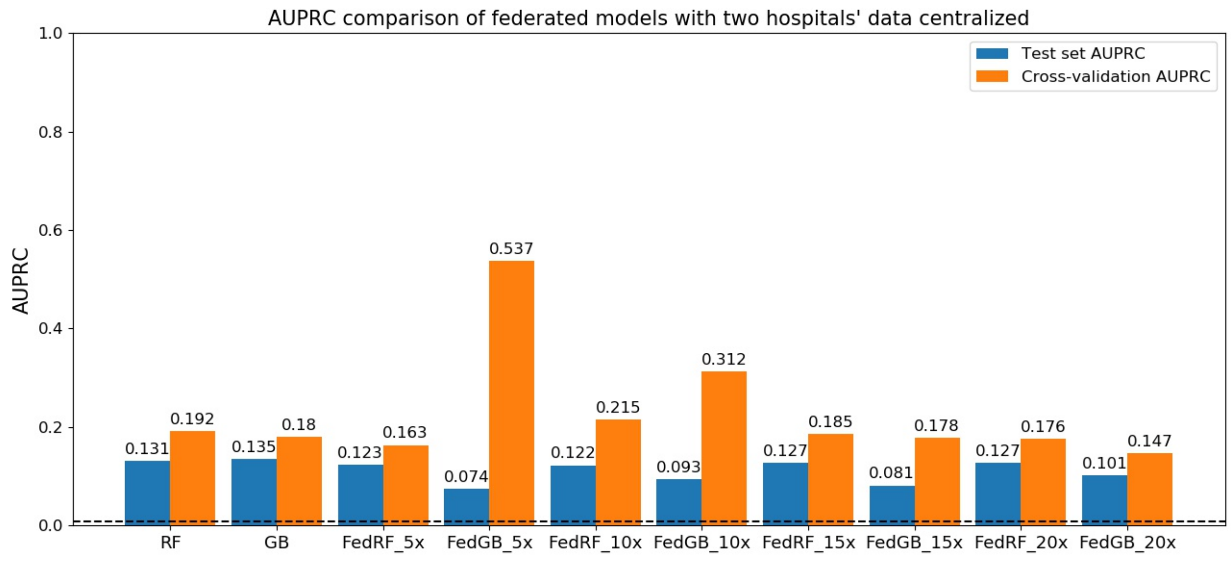

Figure 6.9: AUPRCs of centralized models trained on the data from the two largest hospitals and federated models trained data from the two largest hospitals centralized and rest of the data federated. The baseline AUPRC is marked with a dashed line.

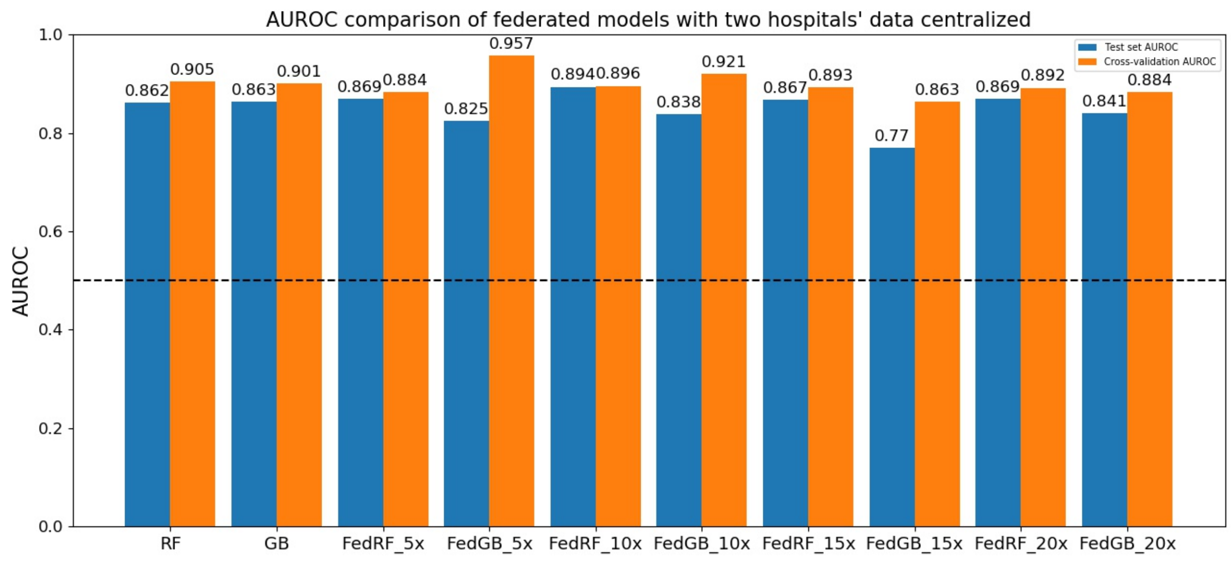

Figure 6.10: AUROCs of centralized models trained on the data from the two largest hospitals and federated models trained data from the two largest hospitals centralized and rest of the data federateds. The baseline AUROC is marked with a dashed line.

Table 6.5: Comparison of centralized models with data from 2 hospitals to models with additional federated training

	RF	GB	FedRF 5x	FedGB 5x	FedRF 10x	FedGB 10x	FedRF 15x	FedGB 15x	FedRF 20x	FedGB 20x
Cross-validation AUPRC	0.192	0.180	0.163	0.537	0.215	0.312	0.185	0.178	0.176	0.147
Cross-validation AUROC	0.905	0.901	0.884	0.957	0.896	0.921	0.893	0.863	0.892	0.884
Test set AUPRC	0.131	0.135	0.123	0.074	0.122	0.093	0.127	0.081	0.127	0.101
Test set AUROC	0.862	0.863	0.869	0.825	0.894	0.838	0.867	0.770	0.869	0.841

6.3.2 Models with 3 Centralized Hospitals

Figure 6.11, Figure 6.12 and Table 6.6 show the results for the same centralized models as before and the results for the federated models that had data from the three largest hospitals centralized. Similarly to the previous scenario in Section 6.3.1, the additional federated training does not improve results here either. The best federated model in this scenario is the federated random forest trained with data from the 20 largest hospitals, achieving an AUPRC of 0.127 and AUROC of 0.867. These are well below the centralized random forest that achieved an AUPRC of 0.158 and AUROC of 0.899. Compared to the previous scenario, the performance of the federated random forest is actually slightly worse with all training set sizes.

From Figure 6.11 and Figure 6.12 we can also see that the difference in cross-validation and test set AUPRC is still large with the federated models trained with data from 5 or 10 hospitals and that the difference in AUROC performance is once again smaller than the difference in AUPRC performance.

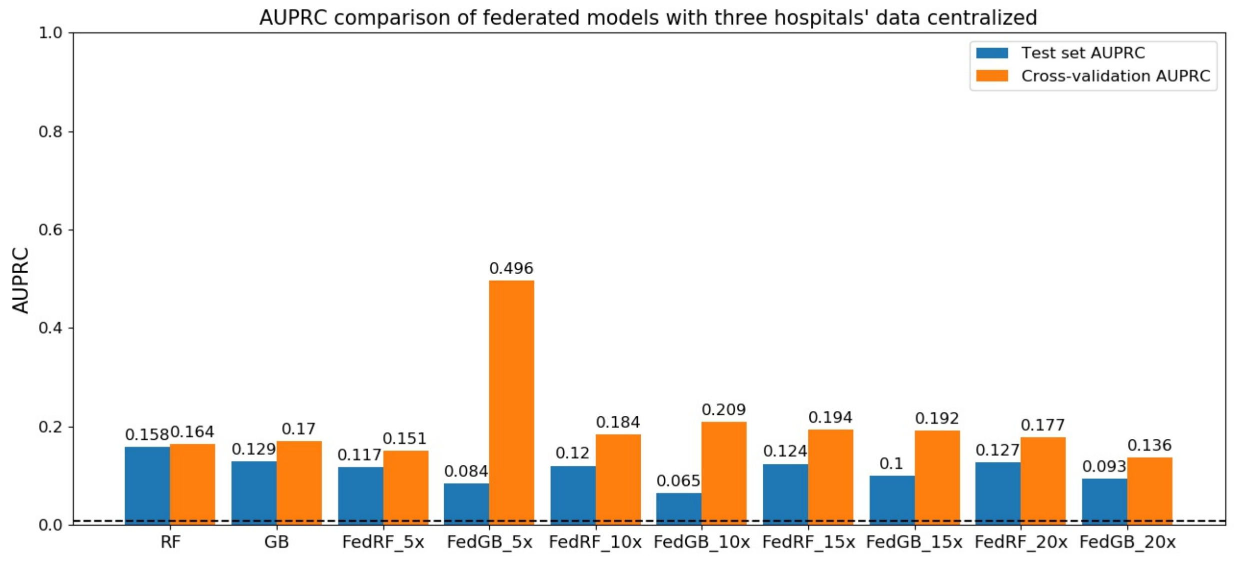

Figure 6.11: AUPRCs of centralized models trained on the data from the three largest hospitals and federated models trained data from the three largest hospitals centralized and rest of the data federated. The baseline AUPRC is marked with a dashed line.

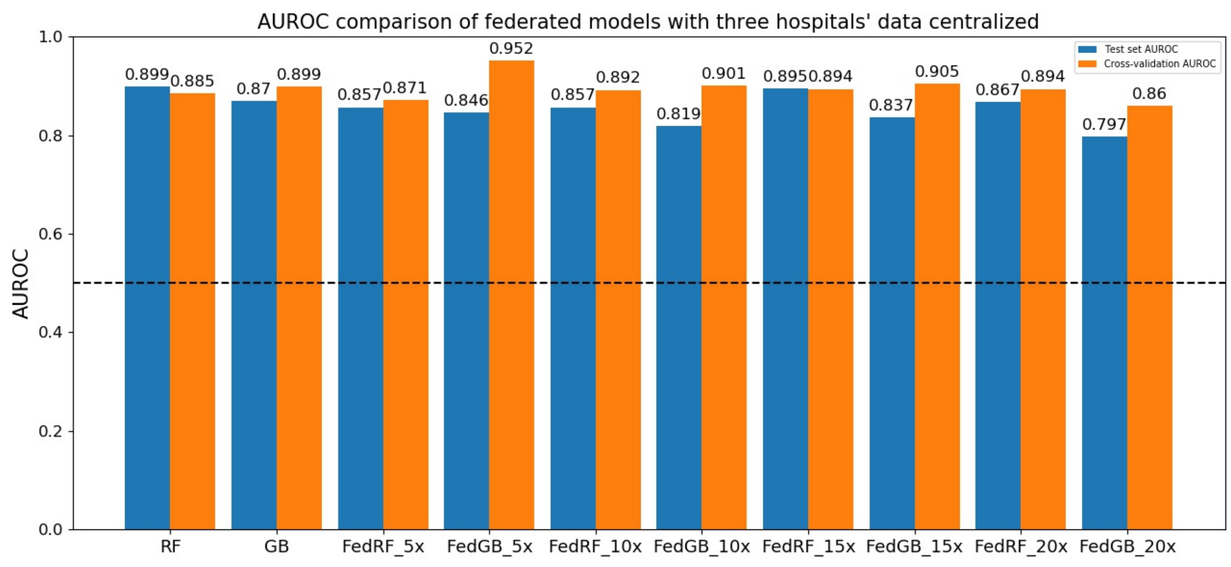

Figure 6.12: AUROCs of centralized models trained on the data from the three largest hospitals and federated models trained data from the three largest hospitals centralized and rest of the data federateds. The baseline AUROC is marked with a dashed line.

Table 6.6 shows both the AUPRC and AUROC from all the models in a single table, and we can see the result that the centralized random forest again performed the best on both metrics, as it did in the scenario of Section 6.2.3.

Table 6.6: Comparison of centralized models with data from 3 hospitals to models with additional federated training

	RF	GB	FedRF 5x	FedGB 5x	FedRF 10x	FedGB 10x	FedRF 15x	FedGB 15x	FedRF 20x	FedGB 20x
Cross-validation AUPRC	0.164	0.170	0.151	0.496	0.184	0.209	0.194	0.192	0.177	0.136
Cross-validation AUROC	0.885	0.899	0.871	0.952	0.892	0.901	0.894	0.905	0.894	0.860
Test set AUPRC	0.158	0.129	0.117	0.084	0.120	0.065	0.124	0.100	0.127	0.093
Test set AUROC	0.899	0.870	0.857	0.846	0.857	0.819	0.895	0.837	0.867	0.797

6.4 Effect of Number of Trees on Federated Random Forest

Figure 6.13 shows how increasing the number of trees built on each site affects the AUPRC performance of the federated random forest on the test set. As expected, the performance is very poor when each site builds only a single tree, but the performance quickly increases when more trees are built. The increase in AUPRC is sharp in the beginning but it starts to level of at around 20 trees per site.

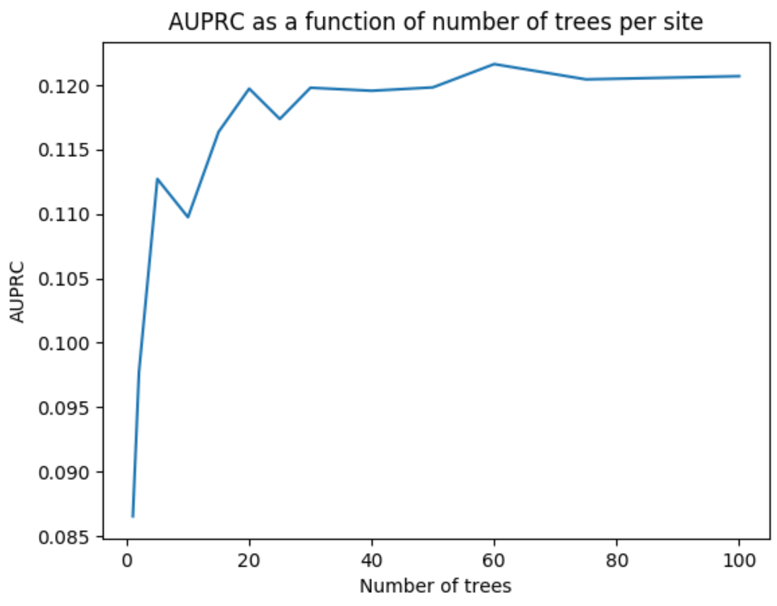

Figure 6.13: Effect of increasing the number of trees built on each site to AUPRC on test set.

For AUROC, a similar result can be seen in Figure 6.14. AUROC in the test set increases sharply in the beginning and starts to level off at around 20 trees per site.

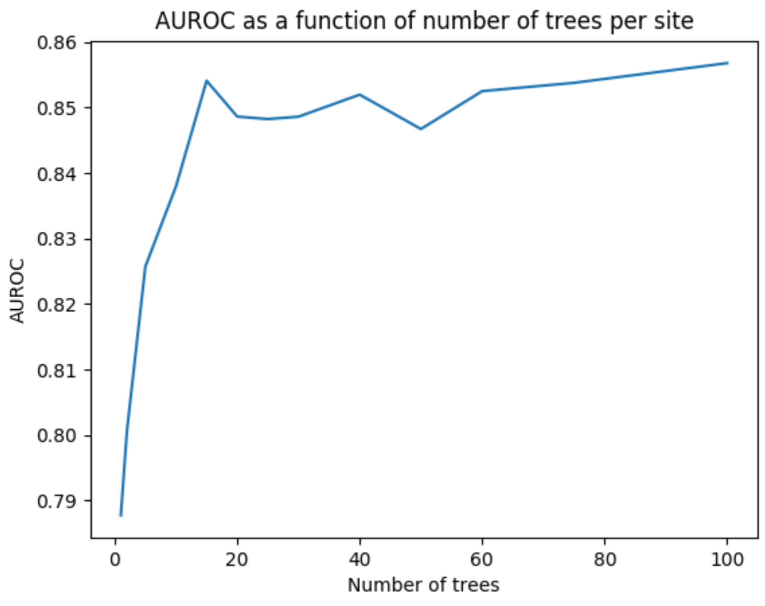

Figure 6.14: Effect of increasing the number of trees built on each site to AUROC on test set.

6.5 Factors affecting Federated Gradient Boosting Performance

Since the federated gradient boosting method used in this thesis is sequential in nature, the effect of going through the hospitals in different order was studied by performing 60 different permutations for the order of hospitals. Table 6.7 shows that the order of hospitals indeed does have an effect. The maximum AUPRC achieved was 57% higher than the lowest and the maximum AUROC achieved was 7% higher than the lowest. The standard deviations were not very high however, 0.007 for the AUPRC and 0.008 for the AUROC. The mean and median for both AUPRC and AUROC were closer to the maximum than the minimum.

Table 6.7: Variability of AUPRC and AUROC of federated gradient boosting when the order of hospitals is permuted

Statistic	AUPRC	AUROC
Minimum	0.069	0.779
Maximum	0.108	0.835
Median	0.100	0.825
Mean	0.098	0.823
Standard Deviation	0.007	0.008

7. Discussion

This thesis was focused on comparing the performance of federated learning and centralized learning on predicting mortality in the ICU. The focus was on tree-based models, namely random forest and gradient boosting, both of which had a federated and a centralized version. In addition to being trained with the full training set, the centralized models were trained with data from the 1, 2 and 3 largest hospitals, and the federated models with the 5, 10, 15, and 20 largest hospitals. In addition, the federated models were trained with the 5, 10, 15, and 20 largest hospitals, but once with the 2 largest hospitals merged into a single dataset and once with the 3 largest hospitals merged into a single data set, in order to examine if the centralized models benefit from additional training with federated learning.

7.1 Performance of the models

The centralized models clearly outperformed the federated models when trained with the full training set, with the centralized gradient boosting achieving a 31% AUPRC compared to the federated random forest and 50% higher AUPRC compared to federated gradient boosting. The centralized random forest achieved 27% and 45% higher AUPRC compared to the federated random forest and federated gradient boosting. The baseline AUPRC was only 0.0089 in the test set and both of the federated models achieved above 10-fold increase in AUPRC compared to the baseline. This suggests that while the performance of the federated models was significantly worse than the centralized models when trained on the full training set, federated learning might still be a viable choice if the data can't be collected into a single location.

To examine if the federated models could outperform the centralized models when given more training data, the centralized models were trained with data from the 1, 2 and 3 largest hospitals and the federated models with data from the 5, 10, 15 and 20 largest hospitals. The results showed that compared to the centralized models with data from a single hospital, all of the federated random forests outperformed the centralized models, with the best one being the one trained on data from the 15 largest hospitals, achieving a 16% and a 45% increase in AUPRC compared to the centralized random forest and centralized gradient boosting. The federated gradient boosting could not outperform the centralized random forest however, achieving a 6% worse AUPRC at best, but could outperform the centralized gradient boosting, achieving a 17% higher AUPRC. It is notable however, that the performance of the federated models increased compared to the federated models trained with the full training set.

When more data was added to the centralized models, they started to outperform the federated models again. From the centralized models trained on the data from the 2 largest hospitals, the gradient boosting was the best, having an AUPRC of 5% higher than the best federated model (RF with 15 hospitals). From the centralized models trained on the data from the 3 largest hospitals on the other hand, the centralized random forest was the best with an AUPRC 22% higher than the best federated model.

Since the AUPRC of the centralized random forest dropped 22% from 0.142 to 0.111 and the AUPRC of the centralized gradient boosting dropped 39% from 0.147 to 0.089 when the training data was reduced from the full training set to just one hospital's data, and all the federated random forest models outperformed the centralized models in this case, the results seem to suggest that if sufficient amount of data can not be collected centrally, federated learning can help improve the performance of the models. When more hospitals were added to the centralized models' training sets, their performance increased significantly however, allowing them to outperform the federated models, suggesting that the increase in the amount of data that federated learning brings must be significant to achieve a boost in performance.

When training the federated models with either 2 or 3 hospitals centralized, to study whether the performance of a centralized model can be improved with further federated training, the results show that the centralized models still outperform the federated models. In fact, the federated gradient boosting with fully federated training largely outperformed the federated gradient boosting models with 2 or 3 hospitals centralized. The federated random forest with 2 hospitals centralized had similar performance to the fully federated version, and the one with 3 hospitals slightly worse than the fully federated one, except for the case of 20 hospitals where the model with 3 centralized hospitals had a minimal improvement compared to the fully federated model. These results imply that the centralized models can not be improved with further federated training and furthermore, if federated training is going to be used, it is not worthwhile to gather portion of the data centrally. One reason why the federated random forest sometimes performed poorly when some of the data was centralized might be that since all the sites built the same number of trees, and thus all sites have equal say in the final prediction, gathering the largest hospitals to a central location diminishes the effect of the largest hospitals on the final prediction. The method would allow scaling the number of trees grown on each individual site by the number of data points on the site, which could possibly improve performance when the size of the datasets differ between sites. Scaling the number of trees was not studied in this thesis and further work is needed to study the effect of scaling.

The performance of the federated random forest method was found to behave as expected when the number of trees grown on each site was varied. When each site grew only a small number of trees the performance was poor but when the number of trees per site was increased the performance quickly improved. The federated random forest was not found to overfit when more trees were added, which is the same behavior that is expected from a centralized random forest [20].

The performance of gradient boosting was found to be dependent on the order that the sites are traversed in. The different between the smallest and largest AUPRC and AUROC was significant but not extreme. The standard deviation for both metrics was small. The distribution was also found to be slightly skewed, with the mean and median for the both metrics being closer to the maximum than the minimum values. These findings show that even though there was a clear difference between the best and the worst performances, the method was still somewhat stable and one can expect the results to be fairly close to the average in most cases. Possible interaction of the random hyperparameter search and the traverseing order cannot be ruled out and further work is needed to investigate the effect of the interaction.

Overall, it was found that the performance of federated random forest on the test set was more stable over the different scenarios and training set sizes than the performance of federated gradient boosting. The standard deviation of AUPRC and AUROC for federated random forest was roughly half of the the standard deviations for federated gradient boosting.

Based on the results of this study, the federated gradient boosting method offers a minor boost in performance compared to centralized gradient boosting trained with less data at best. In most cases the centralized model outperforms the federated model. These results are in line with findings of Zhao et al. [41], where they found that using this sequential method to build trees does not improve the results of the local models. The federated random forest used in this thesis may offer some benefits over the centralized model in some cases, especially if the dataset that could be used for a centralized model is small and insufficient to produce a high-perfroming model. Zhao et al. [41] claimed that their proposed SimFL federated gradient boosting model outperformed the sequential gradient boosting method and achieved a similar performance to the centralized model trained with joint data. No experiments were carried out with SimFL in this thesis and further research is needed to find out how it compares to the federated random forest method.

Liu et al. [55] found that centralized deep learning achieved a 31% higher AUPRC than original federated learning for mortality prediction using eICU dataset. This result is similar to what was found in this thesis for the random forest: the centralzed random forest achieved a 27% (and 31% for the centralized gradient boosting) higher AUPRC than the federated random forest on the full training set. Their federated autonomous deep learning model achieved a higher AUPRC than the centralized model, however. Compared to the results of Huang et al. [56], the federated models of this thesis did not achieve as good performance compared to the centralized models of this thesis. Their centralized model achieved only 9% higher AUPRC than the federated model and 10% higher AUPRC than their best community-based federated model. Beaulieu-Jones et al. [57] had numbers for AUROCs and the results varied from the centralized model having 0.9% higher AUROC than the federated model when trained with 4 hospitals and the federated model having 0.3% higher AUROC than the centralized model when trained with 2 hospitals. On this thesis (with the full data set), the centralized gradient boosting had a 7.7% higher AUROC than the federated gradient boosting and the centralized random forest had a 2.7% higher AUROC than the federated random forest. The gap between the performance of the centralized and federated models in this thesis was much higher than with the models of Beaulieu-Jones et al. [57].

7.2 Limitations

A confounding factor when interpreting the results on whether federated learning using more data can increase the performance compared to centralized model using less data is that the performance of the models in general did not increase in all cases when more data was added. The centralized random forest trained on three hospitals' data outperformed the centralized model trained on the full training set in terms of AUPRC by 11%. Similarly, the federated random forests trained with 5, 10, 15 or 20 hospitals' data outperformed the one trained with the full training set, by 9 to 15%, the best one being the model trained on 15 hospitals' data. The effect was less pronounced with gradient boosting, since the centralized model trained on full training set was the best and the federated model using 20 hospitals was the only one that beat the one trained on full training set. This suggests that some of the smaller hospitals introduced only noise in the data set and that there exists some optimum number of hospitals between using just single hospitals' data and all of the hospitals' data. Based on the results of the federated models, the optimum seemed to lie somewhere near 15 to 20 hospitals, but no further efforts were made to find this optimum. Had the data set contained more hospitals with larger data sets, federated models might have gained a larger increase in AUPRC compared to the centralized models with less data. More research will be needed to more precisely quantify the extent of the possible performance increase gained by having more data with federated learning.

Another factor that most likely affects the performance of the federated random forest is that all the sites (hospitals) trained an equal number of trees. Since the federated model is formed by collecting all the trees from the individual forests and the prediction for the federated model is taken to be the majority vote from all the trees, all of the sites contribute equally to the final prediction since all the sites provided an equal number of trees. This means that a hospital with data from 1000 patients will have an equally large effect on the final prediction as a hospital with 10 patients. As the results showed, using data from all the hospitals for the federated random forest resulted in significantly poorer performance compared to models trained with 5 to 20 largest hospitals. This effect might be magnified because the large and small hospitals had an equal effect on the final prediction.

As mentioned before, scaling the number of trees based on dataset size was nut studied on this thesis and further research is needed to find out the effect of the size of individual sites on the performance of the federated random forest method. Possible experiments would be to scale the number of trees by the number of datapoints or number of patients in each dataset, or to add the number of trees grown on individual sites as a hyperparameter to tune.

Furthermore, the federated random forest here was trained with multiple-source cross-validation, which meant that performance of the federated model, after the individual forests were trained, was evaluated with cross-validation and we do not have any idea how well the individual forests would perform on the data they were trained on. That also means that all the forests share the same hyperparameter values. An alternative way would have been to train the individual forests by regularly cross-validating them on their own data sets and combined the cross validated models to a federated model. This method might be more practical in a real-world scenario since the training can be completely asynchronous but it is computationally expensive to simulate in an experiment with a single machine since all of the sites need to perform cross-validation. There is no guarantee that this method would result in a more accurate model but it might be an interesting question to study in later research.

The experiments showed that the performance of federated gradient boosting depended on the order that the sites where traversed in but the reason behind the effect remains unclear since no experiments where carried out to investigate it. Furher experiments could be done by sorting the hospitals by dataset size to study wether it is better to traverse the hospitals from largest to smallest, smallest to largest or in random order. The order could also be optimzed as a hyperparameter, although it might be computationally intense if the number of hospitals is large.

The AUROCs for all the models in all experiments were generally considerably higher than in the studies presented in Section 3.4, suggesting that in this thesis, the learning problem was easier and the dataset used different, compared to the other studies. Therefore, making conclusions and comparisons about the absolute numbers of the results in this thesis and other related work, as well making comparisons of the models between the studies, is difficult.

Lastly, it is difficult to say how large an effect randomness has on the final results, since for random forest there is inherent randomness in the algorithm and for all models the hyperparameters were tuned with random search. The performance differences generally were not very large so it is difficult to say whether the differences are statistically significant without doing a more thorough statistical analysis.

8. Conclusions

The objective for this thesis was to compare the performance centralized random forest and gradient boosting to federated random forest and gradient boosting on ICU data. For federated gradient boosting, the method of Zhao et al. [41] was adopted with omitting differential privacy. For federated random forest, a novel method was developed by combining random forest built on remote sites.

Three scenarios were considered: comparison of centralized and federated methods on the same data set, comparison of centralized training with less data to federated training with more data and federated training with some data centralized.

The federated methods were found to provide performance well above a random classifier, but when trained on the same data set, centralized methods were found to be superior. With access to more data, federated random forest was found to have superior performance compared to the centralized models trained with a single hospital's data. Federated gradient boosting was found to have similar level of performance compared to the centralized gradient boosting in this scenario. When more hospitals were added to the centralized models' data sets they started outperforming the federated models. Having some of the data centralized was not found to improve the performance of the federated models.

In general, federated random forest was found to perform better than federated gradient boosting. The performance of federated random forest was also found to be more stable with different data sets compared to federated gradient boosting. In addition to the superior performance, federated random forest is simpler to implement, can be trained asynchronously or in parallel, has smaller communication costs and sites can be easily added or removed from the model later.

The results of the thesis suggest that in some scenarios, federated random forest might provide superior performance compared to centralized models but based on the experiments of this thesis, the effect is not likely to be large. For a practical application, consideration is needed wether it is worthwhile to use federated random forest instead of centralized models, even if more data could be acquired with the federated method, leading to potentially slightly better performance. Even though federated learning eliminates the need to collect all the data centrally, gaining access to multiple data sources still requires more work than gaining access to a single source. The extra work required for gaining access to the data might outweigh the benefits gained from a slight performance boost associated with using federated random forest, possibly making the method unfit for practical use. The poor and unstable performance of the federated gradient boosting model suggest that it will not be beneficial compared to centralized learning or federated random forest.

Appendix A. Features

Table A.1: Full table of features

Variable	Window	Last	First	Mean	Median	Min	Max	Slope	Std Dev	Delta	$P_{90\%}$	$P_{10\%}$
Heart Rate	8h	x	x	x	x	x	x	x	x	x	x	x
Respiration Rate	8h	x	x	x	x	x	x	x	x	x	x	x
BPsyst	8h	x	x	x	x	x	x	x	x	x	x	x
BPdiast	8h	x	x	x	x	x	x	x	x	x	x	x
BPmean	8h	x	x	x	x	x	x	x	x	x	x	x
CVP	8h	x	x	x	x	x	x	x	x	x	x	x
SpO_2	8h	x	x	x	x	x	x	x	x	x	x	x
Temperature	8h	x	x	x	x	x	x	x	x	x	x	x
FiO_2	1h	x										
PEEP	1h	x										
GCS verbal	24h	x										
GCS motor	24h	x										
GCS eye	24h	x										
Hematocrit	48h	x										
Hemoglobin	48h	x										
Leukocytes	48h	x										
Erythrocytes	48h	x										
Thrombocytes	48h	x										
Eosinophils	48h	x										
Monocytes	48h	x										
Neutrophils	48h	x										
Basophils	48h	x										
RDW	48h	x										
MCV	48h	x										
MCHC	48h	x										
MCH	48h	x										
MPV	48h	x										
Potassium	48h	x										
Sodium	48h	x										
Creatinine	48h	x										
Glucose	48h	x										
BUN	48h	x										
Calcium	48h	x										
Chloride	48h	x										
$PaCO_2$	48h	x										
PaO_2	48h	x										
Lactate	48h	x										
CRP	48h	x										
Total Bilirubin	48h	x										
Troponin T	48h	x										
Troponin I	48h	x										
Total Protein	48h	x										
Total CO_2	48h	x										
Magnesium	48h	x										
Albumin	48h	x										
Aspartate Aminotransferase	48h	x										
Alanine Aminotransferease	48h	x										
Alkaline Phosphatase	48h	x										

Bibliography

[1] Veerajalandhar Allareddy, Deepti Karhade, Madhuradhar Chegondi, Aditya Badheka, and Veerasathpurush Allareddy. 1384: Machine learning methods in healthcare: An overview. *Critical Care Medicine*, 48:669, 2020.

[2] Carol Bova, Deborah Drexler, and Susan Sullivan-Bolyai. Reframing the influence of the health insurance portability and accountability act on research. *Chest*, (3):782–786, 2012.

[3] Luca Marelli and Giuseppe Testa. Scrutinizing the EU general data protection regulation. *Science*, 360(6388):496–498, 2018.

[4] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-Efficient Learning of Deep Networks from Decentralized Data. In Aarti Singh and Jerry Zhu, editors, *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, volume 54 of *Proceedings of Machine Learning Research*, pages 1273–1282, Fort Lauderdale, FL, USA, 2017. PMLR.

[5] Kevin Murphy. *Machine Learning: A Probabilistic Perspective*, volume 58. MIT Press, 2012.

[6] Mark S. Aldenderfer and Roger K. Blashfield. *Cluster Analysis.* Quantitative Applications in the Social Sciences. SAGE Publications Inc, 1984.

[7] Kishan Mehrotra, Chilukuri Mohan, and HuaMing Huang. *Anomaly Detection Principles and Algorithms.* Springer, 2017.

[8] Henning Best and Christof Wolf. *The SAGE Handbook of Regression Analysis and Causal Inference.* SAGE Publications Ltd, 2014.

[9] Alan Fielding. *Cluster and Classification Techniques for the Biosciences.* Cambridge University Press, 2007.

[10] Douglas Montgomery, Elizabeth Peck, and Geoffrey Vining. *Introduction to Linear Regression Analysis.* John Wiley & Sons Ltd, fifth edition, 2012.

[11] Olivier Chapelle, Bernhard Schölkopf, and Alexander Zien. *Semi-Supervised Learning*. MIT Press, 2006.

[12] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[13] Chi Song and Heping Zhang. Fifty years of classification and regression trees discussion. *International Statistical Review*, 82:359–361, 2014.

[14] Lior Rokach and Oded Maimon. *Data mining with decision trees : theory and applications*, volume 81 of *Series in Machine Perception and Artificial Intelligence*. World Scientific Pub. Co, second edition, 2015.

[15] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An Introduction to Statistical Learning with applications in R*. Springer, 2013.

[16] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *Elements of Statistical Learning Data Mining, Inference, and Prediction*. Springer, 2009.

[17] Christopher Bishop. *Pattern Recognition and Machine Learning*. Springer, 2007.

[18] Jerome H. Friedman. Greedy function approximation: A gradient boosting machine. *Ann. Statist.*, 29(5):1189–1232, 2001.

[19] Shai Shalev-Shwartz and Shai Ben-David. Understanding machine learning. from theory to algorithms. *Understanding Machine Learning: From Theory to Algorithms*, 2013.

[20] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[21] Hervé Abdi, Betty Edelman, and Dominique Valentin. *Neural Nerworks*. SAGE Publications Inc, 1999.

[22] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[23] Krzysztof Geras and Charles Sutton. Multiple-source cross-validation. In Sanjoy Dasgupta and David McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 1292–1300, Atlanta, Georgia, USA, 2013. PMLR.

[24] Allison McCoy, Adam Wright, Michael Kahn, Jason Shapiro, Elmer Bernstam, and Dean Sittig. Matching identifiers in electronic health records: Implications for duplicate records and patient safety. *BMJ quality & safety*, 22, 2013.

[25] Rokia Bey, Romain Goussault, Mehdi Benchoufi, and Raphal Porcher. Stratified cross-validation for unbiased and privacy-preserving federated learning. *ArXiv*, abs/2001.08090, 2020.

[26] Alistair E. W. Johnson, Tom J. Pollard, and Tristan Naumann. Generalizability of predictive models for intensive care unit patients. *ArXiv*, abs/1812.02275, 2018.

[27] Tom Fawcett. An introduction to ROC analysis. *Pattern Recognition Letters*, 27(8):861–874, 2006.

[28] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12(85):2825–2830, 2011.

[29] Javed Aslam, Emine Yilmaz, and Virgil Pavlu. A geometric interpretation of r-precision and its correlation with average precision. pages 573–574, 2005.

[30] J.A. Hanley and Barbara Mcneil. The meaning and use of the area under a receiver operating characteristic (ROC) curve. *Radiology*, 143:29–36, 1982.

[31] Takaya Saito and Marc Rehmsmeier. The precision-recall plot is more informative than the ROC plot when evaluating binary classifiers on imbalanced datasets. *PLoS One*, 10(3), 2015.

[32] Philipp Probst, Marvin N. Wright, and Anne-Laure Boulesteix. Hyperparameters and tuning strategies for random forest. *WIREs Data Mining and Knowledge Discovery*, 9(3), 2019.

[33] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(10):281–305, 2012.

[34] Bob Hickish, David I. Fletcher, and Robert F. Harrison. Investigating Bayesian optimization for rail network optimization. *International Journal of Rail Transportation*, 0(0):1–17, 2019.

[35] Yoshua Bengio. Gradient-based optimization of hyperparameters. *Neural computation*, 12(8):1889–1900, 2000.

[36] Hua Cui and Jie Bai. A new hyperparameters optimization method for convolutional neural networks. *Pattern Recognition Letters*, 125:828–834, 2019.

[37] Ron Bekkerman, Mikhail Bilenko, and John Langford. *Scaling up machine learning: parallel and distributed approaches.* Cambridge University Press, 2012.

[38] Jakub Konecny, H. Brendan McMahan, and Daniel Ramage. Federated optimization: Distributed optimization beyond the datacenter. *arXiv:1511.03575*, 2015.

[39] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. Federated machine learning: Concept and applications. *ACM Transactions on Intelligent Systems and Technology*, 10:1–19, 2019.

[40] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H. Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for federated learning on user-held data. In *NIPS Workshop on Private Multi-Party Machine Learning*, 2016.

[41] Lingchen Zhao, Lihao Ni, Shengshan Hu, Yanjiao Chen, Pan Zhou, Fu Xiao, and Libing Wu. Inprivate digging: Enabling tree-based distributed data mining with differential privacy. *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, 2018.

[42] Mayur Datar, Piotr Indyk, Nicole Immorlica, and Vahab Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. *Proceedings of the Annual Symposium on Computational Geometry*, 2004.

[43] Qinbin Li, Zeyi Wen, and Bingsheng He. Practical federated gradient boosting decision trees. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34:4642–4649, 2020.

[44] Stacey Truex, Ling Liu, Mehmet Gursoy, and Lei Yu. Demystifying membership inference attacks in machine learning as a service. *IEEE Transactions on Services Computing*, pages 1–1, 2019.

[45] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 3–18, 2017.

[46] Michael Veale, Reuben Binns, and Lilian Edwards. Algorithms that remember: model inversion attacks and data protection law. *Philosophical Transactions A: Mathematical, Physical and Engineering Sciences*, 376, 2018.

[47] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. Model inversion attacks that exploit confidence information and basic countermeasures. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*,

CCS '15, pages 1322–1333, New York, NY, USA, 2015. Association for Computing Machinery.

[48] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. *Theory of Cryptography*, 3876:265–284, 2006.

[49] Cynthia Dwork. A firm foundation for private data analysis. *Commun. ACM*, 54:86–95, 2011.

[50] Cynthia Dwork. Differential privacy. *Automata, Languages and Programmin*, 4052:1–12, 2006.

[51] Yang Liu, Yingting Liu, Zhijie Liu, Yuxuan Liang, Chuishi Meng, Junbo Zhang, and Yu Zheng. Federated forest. *IEEE Transactions on Big Data*, pages 1–1, 2020.

[52] Zhuzhu Wang, Yilong Yang, Yang Liu, Ximeng Liu, Brij Gupta, and Jianfeng Ma. Cloud-based federated boosting for mobile crowdsensing. *arXiv:2005.05304*, 2020.

[53] Kewei Cheng, Tao Fan, Yilun Jin, Yang Liu, Tianjian Chen, and Qiang Yang. Secureboost: A lossless federated learning framework. *arXiv:1901.08755*, 2019.

[54] Theodora Brisimi, Ruidi Chen, Theofanie Mela, Alex Olshevsky, Ioannis Paschalidis, and Wei Shi. Federated learning of predictive models from federated electronic health records. *International Journal of Medical Informatics*, 112, 2018.

[55] Dianbo Liu, Timothy Miller, Raheel Sayeed, and Kenneth Mandl. Fadl:federated-autonomous deep learning for distributed electronic health record. *arXiv:1811.11400*, 2018.

[56] Li Huang, Andrew Shea, Huining Qian, Aditya Masurkar, Hao Deng, and Dianbo Liu. Patient clustering improves efficiency of federated machine learning to predict mortality and hospital stay time using distributed electronic medical records. *Journal of Biomedical Informatics*, 99:103291, 2019.

[57] Brett Beaulieu-Jones, William Yuan, Samuel Finlayson, and Zhiwei Wu. Privacy-preserving distributed deep learning for clinical data. *arXiv:1812.01484*, 2018.

[58] Seyedmostafa Sheikhalishahi, Vevake Balaraman, and Venet Osmani. Benchmarking machine learning models on multi-centre eicu critical care dataset. *PLoS One*, 15(7), 2020.

[59] Jack Zimmerman, Andrew Kramer, Douglas McNair, and Fern Malila. Acute physiology and chronic health evaluation (APACHE) IV: Hospital mortality assessment for today's critically ill patients. *Critical Care Medicine*, 34(1297-1310), 2006.

[60] William A Knaus, Elizabeth A Draper, Douglas P Wagner, and Jack E Zimmerman. APACHE II: A severity of disease classification system. *Critical care medicine*, 13(10):818–829, 1985.

[61] Yangyang Ding, Youqing Wang, and Donghua Zhou. Mortality prediction for ICU patients combining just-in-time learning and extreme learning machine. *Neurocomputing*, 2017.

[62] Mohammed Saeed, Mauricio Villarroel, Andrew Reisner, Gari Clifford, Li-wei Lehman, George Moody, Thomas Heldt, Tin Kyaw, Benjamin Moody, and Roger Mark. Multiparameter intelligent monitoring in intensive care II (MIMIC-II): A public-access intensive care unit database. *Critical care medicine*, 39:952–60, 2011.

[63] Wendong Ge, Jin-Won Huh, Yu Rang Park, Jae Ho Lee, Y.H. Kim, and Alexander Turchin. An interpretable ICU mortality prediction model based on logistic regression and recurrent neural networks with LSTM units. *AMIA Annual Symposium proceedings*, 2018:460–469, 2018.

[64] Gilles Louppe. *Understanding Random Forests: From Theory to Practice.* PhD thesis, 2014.

[65] Tom Pollard, Alistair Johnson, Jesse Raffa, Leo Celi, Roger Mark, and Omar Badawi. The eICU Collaborative Research Database, a freely available multi-center database for critical care research. *Scientific Data*, 5:180178, 2018.

[66] Craig Lilly, Ilene Zuckerman, Omar Badawi, and Richard Riker. Benchmark data from more than 240,000 adults that reflect the current practice of critical care in the United States. *Chest*, 140:1232–42, 2011.

[67] Yaseen Arabi, Srinivas Venkatesh, Samir Haddad, Abdullah Shimemeri, and Salim Malik. A prospective study of prolonged stay in the intensive care unit: Predictors and impact on resource utilization. *International journal for quality in health care*, 14:403–10, 2002.