Budapest University of Technology and Economics
Faculty of Electrical Engineering and Informatics
Department of Measurement and Information Systems

Federated boosting in semi-supervised learning

Scientific Students' Association Report

Author
Dániel Sándor

Advisor
Dr. Péter Antal

November 1, 2022

Contents

Kivonat

A boosting algoritmusok lényege, hogy gyenge modellekből építenek egy olyan együttest, amely már erős tanulóként használható. Ennek során az adatot minden iterációban adaptív módon súlyozza újra úgy, hogy figyelembe veszi, hogy az együttes az adat mely részét képes már megfelelően becsülni. Ezáltal az együttes modelljei más-más reprezentációt alakítanak ki és képesek olyan összefüggések megtalálására, amelyekre egy modell nem.

A federált bioaktivitás predikció egy olyan feladat, ahol több kliens egy közös modellt tanít, amely képes lesz gyógyszer hatóanyag molekulákból megbecsülni, hogy azok mely biológiai célpontra mekkora hatással lesznek. Ez az adat szempontjából egy igen összetett feladat, hiszen nem csak az elosztottság okoz nehézséget, hanem az is, hogy a ma rendelkezésre álló bioaktivitás adatokban jellemző a nagyfokú hiányzás. Annak érdekében, hogy ezeket a hátrányokat ellensúlyozzák gyakran többfeladatos módon végzik ezen modellek tanítását. Azonban a multitaszk megközelítés saját nehézségeivel jár: Az egyik kulcs probléma, hogy meg kell találni azon feladatokat az adathalmazban, amelyek segítik egymás tanulását és becslését. Erre egy lehetséges megoldás a boosting súlyozása.

A munkám során egy elterjedt boosting megoldást, az AdaBoost algoritmust veszem alapul egy olyan megoldás elkészítéséhez, amely federált módon működik és képes több feladatot kezelni, valamint alkalmas hiányos adatok kezelésére. A FedBoost jelentősége az, hogy képes kihasználni a boosting adaptív adatsúlyozó módszerét arra, hogy feladatok közötti összefüggéseket adaptívan minden iterációban újraértelmezze és más következtetéseket képes levonni így mint a nem együttes alapú modellek.

A dolgozatomban a boosting kapja a hangsúlyt, mindent területnél kitérek a lehetséges boosting megoldásokra, hogy teljes képet tudjak adni a módszer hasznosságáról az egyes feladatokban. Kiemelten foglalkozom ezen kívül a federált tanulás formáival és a többfeladatos tanuláshoz való kapcsolódással, valamint a hiányos adatból való tanulás nehézségeivel. Az módszert tehát ezen négy megközelítés mentén értékelem ki: federáltság, többfeladatosság, hiányzás és boosting. Vizsgálom ezen megoldások páronkénti alkalmazhatóságát és prediktív teljesítménybeli különbségeket a kombinációk között. Ahhoz, hogy a módszer teljesítményét megfelelően kontextusba tudjam helyezni összevetem a szakirodalomban ismert más boosting módszerekkel. Végül igyekszem teljes képet adni a FedBoost algoritmus potenciális előnyeiről és hátrányairól az ismert federált algoritmusokhoz képest.

Abstract

Boosting algorithms build an ensemble of weak learners, where the ensemble can be considered a strong learner. In this process, the training data gets adaptively reweighted in every iteration. This reweighting is done by measuring which parts of the data can the ensemble already predict, and which parts need more attention. As a result, the models of the ensemble develop different representations and can find connections that a single model cannot.

Federated bioactivity prediction is a task where several clients train a common model that will be able to predict from drug molecules how much effect they will have on given biological targets. When examining the data we can see that the task is complex, not only because it is distributed, but also the fact that the bioactivity data available today is characterized by a high degree of missing values. Multitask learning is often used to overcome these challenges. However multitask learning introduces its own difficulties, such as the problem of finding tasks that can benefit from joint training. One solution for this can be the weighting of boosting.

In my work I use Adaboost, a popular boosting algorithm, as a base to develop a method, which can be used in a federated setting, can accommodate multitask learning, and can deal with missing data. The main importance of FedBoost is that it can utilize the adaptive data weights of boosting to detect and reevaluate connections of tasks in an adaptive manner, and can develop different predictions when compared to a non-ensemble model.

In my report, the focus will be on boosting, when analyzing a field I always consider the possible boosting methods for application, this way I can provide a complete picture of the usage of the approach. In addition, I emphasize federated learning with its possible forms and its connection to multitask learning, with the use of missing data. Thus my methods will be evaluated along these four qualities: federation, multitask, missing values, and boosting. I will further analyze the possible pairwise application of these methods, and their differences in predictive performance. To contextualize the performance of the algorithm I will compare it to other known boosting methods. Finally, I intend to present the full scale of advantages and disadvantages of FedBoost when compared to other well-known federated algorithms.

Chapter 1

Introduction

Boosting is a technique where multiple weak learners are combined to form a strong learner [22]. Weak learners are models which perform only slightly better than random guessing in a PAC [34] framework. The goal of boosting usually is to use these weak learners to form an ensemble, which can perform a given task in a better way. The way we can achieve this is by taking the data and calculating the error on each sample and assigning weights in a manner that facilitates the improvement of predictions on the samples which have a higher error score. This means, that data in a boosting algorithm is getting reweighted in every iteration, thus the learner may develop a representation better suited for the data which has not been predicted successfully so far. Boosting has been effectively applied in a variety of machine learning problems: traditionally it is implemented in singletask use-cases, but multitask extensions exist [41, 42]. The same can be said about federated applications [16], however much fewer implementations are available due to the relative novelty of the field. In my work, I attempt to combine these approaches and create a technique where boosting can be applied in a federated and multitask scenario, with the added complexity of missing data.

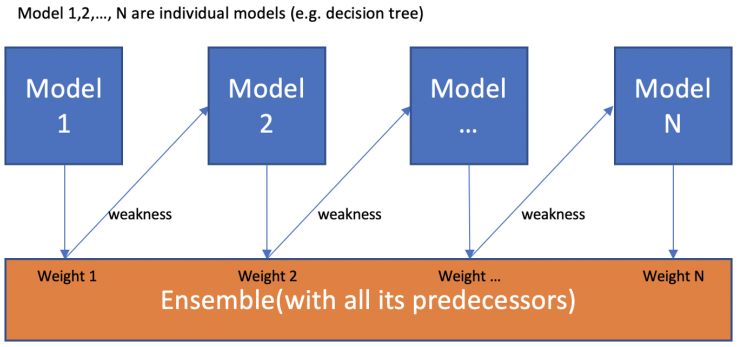


Figure 1.1: Boosting in machine learning. [32]

The main motivation for my work is a federated drug-target interaction problem. In the past years, drug discovery has become more and more reliant on the use of machine learning [35]. For a long time, drug discovery was mostly conducted with in vitro tests on candidates. But these types of tests are expensive and time-consuming to execute thus, they do not scale well for the large amounts of available drug candidates [29]. Machine learning has the advantage of scalability and relatively low resource costs thus, it can complement traditional tests well. Modern drug discovery problems usually involve a

model which can select or narrow down a list of drug candidates which are further tested in laboratories [3].

One of the most prominent fields in drug discovery is drug-target interaction prediction (DTI). A drug is a chemical substance that aims to inflict change in the human body [20]. Biological targets are part of the body through which the drugs can exert their effects. These targets are usually proteins that bind to the molecules of the drug and react in some way which creates change in the organism. Most diseases are characterized by a set of targets that affect them. If we can identify these targets, and predict which molecules might bind to said targets, then we have accelerated the drug discovery part of the research and facilitated the development of a cure.

DTI prediction intends to assess which drugs will bind to which proteins. For this task, there is already a large amount of data available [30, 8, 26]. The historically used compounds of drugs usually have multiple targets to which they are likely to bind hence the only job is to find these targets and the compounds can be repurposed to treat other diseases as well. The data on targets is available through assays. Assays are procedures to analyze the qualities of a given target. In this case, this means the identification of drugs that bind or do not bind to any given target. One target can be identified through multiple assays and these are not always consistent, which makes it hard for machine learning models to make predictions. A usual DTI task is built up in the following way: multiple assays are available for one target, and assays are only conducted for some compounds (and not all available) to save resources. After the collection of assays, they can be arranged into a bioactivity matrix, which contains one assay in a column and one compound in a row. Because the assays only have available data on interactions where they have been measured and the measurements are only conducted where the researchers saw a chance of interaction the resulting data is sparse and missing-not-at-random (MNAR) type data.

This type of data presents a considerable challenge for traditional machine learning, but it has been proven that multitask learning is a useful approach in the field. When learning from MNAR datasets in this case the missing data is the label, which means it is a semi-supervised learning problem [11]. Semi-supervised learning means, that the labels are available for only parts of the data, and this approach introduces new challenges in the learning process.

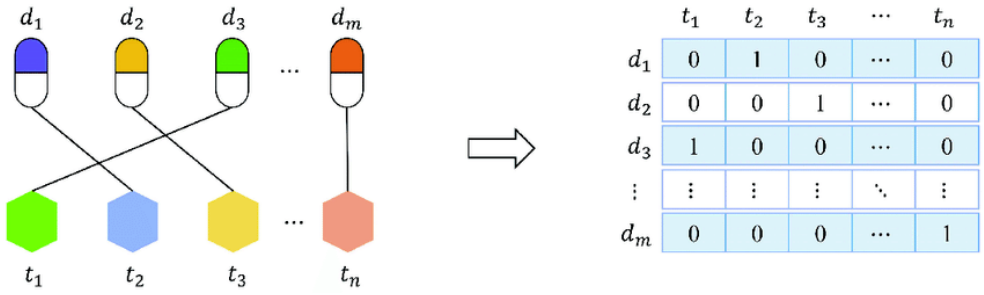


Figure 1.2: Drug target interactions and the bioactivity matrix. [39]

To balance the aforementioned challenges DTI predictors are often trained in a multitask environment. Multitask learning is the branch of machine learning where multiple functions are performed by a given model. This approach was first proposed by Rich Caruana [1]. The multitask paradigm is motivated by the improvement of performances on individual tasks as opposed to a traditional singletask approach. The improvement is traditionally explained by the inductive bias coming from the supporting tasks' signals. This bias helps to create a more generalized representation from which the tasks can benefit.

Multitask learning is most often associated with neural networks. In prediction problems, this usually means, that the neural network gets one input and predicts multiple tasks on the output. This multimodal output is can vary in a wide range depending on the problem, but usually, the tasks are related in a way where the shared representation can benefit each of them [40]. In a neural network, the signals of inductive bias can be imagined as the modification of weights during backpropagation. The shared representation can be observed in the deeper layers of the network, where the features are shaped by all the tasks. This approach is especially popular in the world of computer vision [17] where the deeper layers usually represent general characteristics of images.

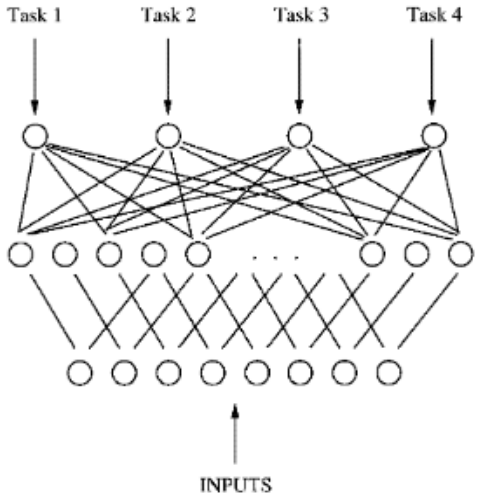


Figure 1.3: Multitask learning in a neural network [1]

Multitask learning presents its own set of challenges, one of these is the problem of tasks subset selection [28]. This means that to succeed in multitask training the nature of tasks has to be considered for the best performance because unfortunately unrelated tasks can deteriorate the prediction leading to worse overall performance. Multitask learning performs best when the tasks are correlated [38]. To select the correct subsets from thousands of tasks we would need to compute the pairwise correlation of all of them, which is usually not feasible. A possible solution for this problem is boosting, specifically the reweighting of data in the boosting iterations. By constant reweighting the models can learn different representations, which can lead to the selection of beneficial subsets in the data, thus improving the training process. The explanation of course comes from the inductive biases mentioned before. As the data is assigned a different weight the "signals" change, and this leads to the forming of different representations.

In my work, my main goal is to combine these approaches and create a federated multitask-boosting scheme, which can handle MNAR data. This can be beneficial because it can leverage the advantages of both methods creating a new way to perform boosting. In my report, I am creating a multitask learning environment to predict DTI tasks. The available data has small molecules described with an Extended Connectivity Fingerprint (ECFP) [2], this serves as the input of the model. While the outputs are the previously described assays in a bioactivity matrix. In every single algorithm, my base classifier will be a Multi-Layer Perceptron (MLP), which will try to predict binary labels for given assay-drug pairs. In some experiments, I will be working with merged assays as these can

represent a protein better and results in better quality data, but the structure of the data will remain the same in these cases as well.

To explain the motivation of federated learning, we have to look at the nature of the data. When working with high-value data, like data on drugs companies can be reluctant to share it. Although in some cases the demand presents itself to learn from a larger dataset and cooperate with other companies or research institutions. In these cases, privacy-preserving federated learning can be a useful tool for performing the training. In federated learning, multiple parties agree to train their models together for better performance, while their data sets remain private and not shared. This can be achieved in a multitude of ways. To give some examples Federated Averaging (FedAvg) uses a shared model architecture and averages the partners' models every few iterations, while Federated Model Distillation (FedMD) uses a shared public data set where information can be shared by knowledge distillation without restricting the architectures of the models.

In my work, I examine different federated architectures in the context of boosting. I implement boosting in a federated DTI problem where each partner can run their own boosting algorithm locally and in a FedAvg-like algorithm they can average their model weights for every few models in the ensemble.

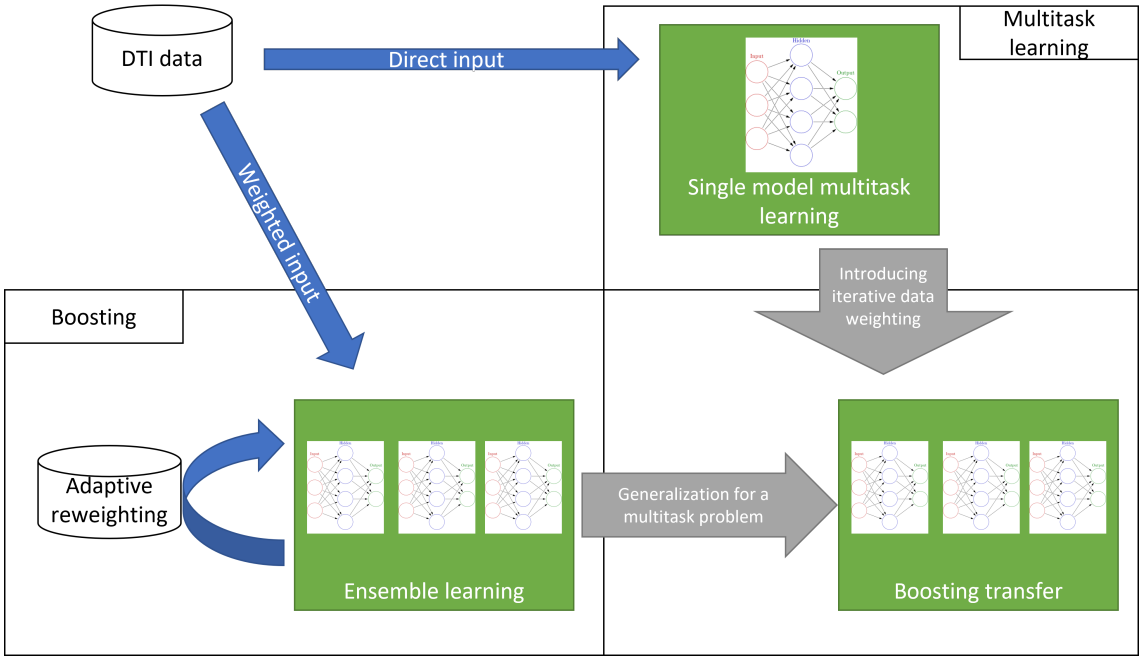


Figure 1.4: Overview of the multitask boosting problem

The structure of my report is the following: In the second chapter, I present the related work from multiple aspects, focusing primarily on the combination of methods. In the third chapter, I discuss boosting: the main focus of my work. I look at the strengths of the approach and I explain one of the most popular boosting algorithms: AdaBoost, I present a slightly modified version of the algorithm for MLP classifiers and multitask data, and I also discuss the different approaches to weighting data adaptively during boosting. In the fourth chapter, I explain the basics of federated learning, I present the two most applicable algorithms for my tasks: FedAvg and FedMD, and I review the history of federated learning in drug research. In the fifth chapter, I explain the importance of multitask learning, with also focusing on the drug research perspective. I present the most important steps in multitask DTI prediction and drug discovery, and in the final part of the chapter, I will overview existing multitask-boosting techniques and their performances. In

the sixth chapter, I describe the data set, and demonstrate the significance of dealing with MNAR-type data and problems in semi-supervised learning. After this, in the seventh, I can present a technique that utilizes the previously mentioned techniques and conforms to the described problems. I demonstrate the usefulness of my algorithm in the world of federated multitask drug research. To give a clear image of the setup I use, I summarize the technical details of my experiments in chapter eight. In chapter nine I define the experiments in a way that shows the improvement in a step-by-step approach, trying out the most prominent combinations of methods, to contextualize the results of my new technique. I analyze the results of these experiments in the second half of chapter nine. Finally, in chapter ten I conclude my work and present the possible use cases and the improvements that can be achieved in the field.

Chapter 2

Related work

This chapter explores the existing combinations of techniques to achieve federated multitask DTI prediction on sparse data. I attempt to give current examples of said combinations and draw a conclusion about the ideal way of combining them.

2.1 Federated multitask learning

Federated multitask learning is a simple and intuitive idea. It is a good combination and it poses little complications for federated models. In [24] Smith et al. create a general framework for solving convex problems in federated multitask frameworks, with the guarantee of convergence.

In [18] Marfoq et al. propose a generalized framework for learning joint models, and recreating personalized distributions from the weighting of these models in an EM-like algorithm. This approach is a key concept for federated boosting as well, where the models can be learned jointly, and the final predictions can be a linear combination of the output of these models.

2.1.1 Federated multitask drug research

In recent years more and more attention has been focused on drug research in a federated setting and utilizing multitask learning to improve performances [37]. Recently the MELLODDY project [10] proved that it could also be conducted on large-scale datasets, and it is feasible in real-world applications.

2.2 Multitask boosting

When talking about multitask boosting, the basic question is how to adapt a singletask basic algorithm to exploit task relationships and learn from the correlating tasks. Wang et al. [36] created the Online Multitask Boosting (OMB) algorithm, which is the generalization of a transfer learning boosting algorithm. Their algorithm captures task relatedness by calculating the differences in errors of the ensemble on the two tasks. In their algorithm, some tasks are used to train models of the ensemble and every task learns to adapt them by assigning its own weight to the output of the classifier.

A different approach is described by Zhang et al. [41] in a technique they call MTBoost. In MTBoost they learn the relationships of tasks in the form of a task covariance matrix. First, they learn an ensemble, for a base hypothesis and generate the output for every task by learning the weights specifically for them. The base hypothesis can be formulated by learning an aggregated "super-task" based on the task covariances.

2.3 Federated boosting

As federated learning itself is a relatively new field, only a handful of federated boosting solutions exist. Most of these are from the field of gradient boosting: One of these is SecureBoost [4], which is a framework for gradient boosting by decision trees, however, it only works in vertical FL scenarios, when the data is distributed in the feature space, not in the sample space, which would describe my problem better. In [16] Li et al. create a similarity-based federation for gradient boosting, where each party makes their data public through hashing and they train models on the hashed features. These approaches are significantly different from AdaBoosting, and in the next chapter, I give a detailed explanation, of why AdaBoost is a good fit for the problem at hand.

2.4 Boosting in drug research

Boosting is a highly versatile method, thus it can be used for drug research too. In [27] Svetnik et al. showed that tree-based boosting is especially useful in Quantitative structure-activity relationship (QSAR) problems, where the goal is to predict bioactivity from structural features, much like in DTI. The usefulness comes from the explainability of tree-based models, which can give insights into the important structural features.

Chapter 3

Boosting

To understand the effectiveness of weighting the data in the learning process I explain the details of boosting in this chapter. The origin of boosting comes from the problem of making weak learners into strong ones [12, 13]. A weak learner in a PAC (probably approximately correct) framework is a classifier, which performs only slightly better than random guessing [43]. The common properties of these algorithms are the aggregation of predictions and the weighting of the data. Aggregating predictions is the core element of boosting, as this has to be done in a way, which truly boosts the predictive performance of the weak models. When talking about predictors it is common to calculate the weighted average of their predictions to form the final prediction. This means that the final output is calculated by a linear regression of the individual outputs.

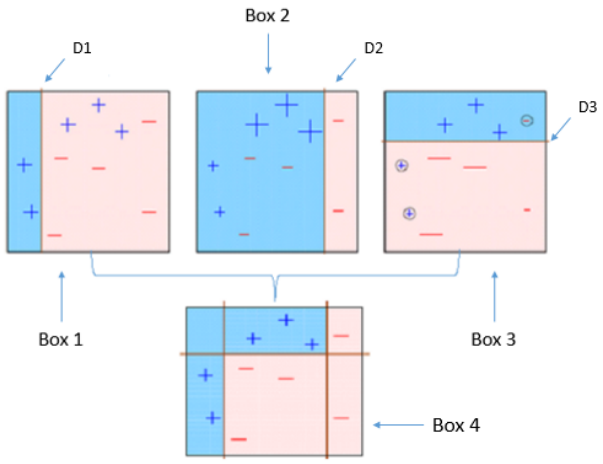


Figure 3.1: Aggregating predictions of stumps in a uniform way [19]

The other key tool of the algorithm family is the weighting of the data. This is the step approach that differentiates it from other ensemble methods. Boosting works in a sequential manner. In early implementations, the weighting was not adaptive [6, 21], but nowadays, only the adaptive methods stayed relevant. This means that data has to be reweighted based on the performance of previous models in the ensemble. One of the first and still widely used boosting methods is AdaBoost.

3.1 AdaBoost

AdaBoost is one of the most widely used boosting algorithms and it has shown great promise in the field of multitask and transfer learning. In this section, I describe the basic algorithm of AdaBoost and explain the modifications I made in the algorithm to use it in a federated multitask DTI prediction scenario.

The basic algorithm follows the following steps: Every sample is initialized with a uniform weight. Next, a weak classifier is trained on this data, usually, this is a decision tree stump, which separates the data linearly. Next, the training data is classified with the trained classifier. The performance of the classifier is measured by an error which we get by the sum of weights over all the incorrectly classified weights.

$$Error_j = \sum_{i=0}^{n} w_{i(j-1)} * I_{y_i \neq \hat{y_i}} \tag{3.1}$$

The say of the classifier in the final prediction or the model's weight in the regression can be calculated in the following way:

$$\alpha_j = \frac{1}{2} log(\frac{1 - Error_j}{Error_j}) \tag{3.2}$$

This means that a model with high error has little weight in the final prediction. In the final step of the algorithm, every sample's weight is refreshed: If a sample was classified correctly its weight is decreased, while an incorrectly classified point's weight is increased so the next model makes it a higher priority to classify it correctly. The weight change is proportional to the classifier's weight.

$$w_{ij} = w_{i(j-1)} * e^{-1^{I_{y_i \neq \hat{y_i}} * \alpha_j}} \tag{3.3}$$

After the new weights are calculated, the whole process is repeated and a new classifier is trained. The algorithm is finished if a given number of classifiers has been trained or a given accuracy is reached on a validation split.

Algorithm 1 AdaBoost

1. Initialize the observation weights $w_{i0} = 1/N$, $i = 1, 2, \ldots, N$
2. For $j = 1$ to M:
   (a) Fit a classifier $G_j(x)$ to the training data using weights $w_i j$.
   (b) Compute error $Error_j = \sum_{i=0}^{n} w_{i(j-1)} * I_{y_i \neq \hat{y_i}}$
   (c) Compute $\alpha_j = \frac{1}{2} log(\frac{1-Error_j}{Error_j})$.
   (d) Set $w_{ij} \leftarrow w_{i(j-1)} * e^{-1^{I_{y_i \neq \hat{y_i}} * \alpha_j}}$, $i = 1, 2, \ldots, N$.
3. Output $G(x) = sign[\sum_{j=1}^{M} \alpha_j G_j(x)]$.

3.1.1 AdaBoosting with neural networks

Textbook AdaBoost uses decision tree stumps as classifiers, however, both in theory and in practice, it is easy to substitute these as an arbitrary classifier. Neural networks are traditionally regarded as strong classifiers. However, in my case, the use of MLPs is motivated by the high dimensionality of the input data. Additionally, I use shallow networks, because previous experiments have shown, that deeper networks cannot pick up on more features in this type of data, despite their higher computational cost. Also, in shallow networks, considerable improvements have been reported [23], thus the use of MLPs is justified as the base classifier for the algorithm.

When using neural networks in AdaBoost it is enough to keep them as weak classifiers: meaning that a few epochs of training are enough. Apart from the dimensionality, the other big advantage of MLPs is the built-in handling of multitask labels, which also works in the AdaBoost framework with relatively short training times. Another addition that I introduced was that for the training, I did not reinitialize the MLPs so they kept the last AdaBoost iteration's weights as initialization for faster convergence (this usually plays the role of accelerating training after the second AdaBoost iteration.

3.1.2 Weighting the data

When working with multitask data another modification must be made for the AdaBoost algorithm and this concerns the weighting of the data. The basic AdaBoost algorithm is created for singletask training, which means that weighting the data can be extended in multiple ways to a multitask environment. The first evident solution would be to assign weights to every cell of the matrix (every bioactivity value), but because of the sparsity of this data, this does not provide a useful way of transferring between the tasks. Measurements support, that a better way to weigh the data is to assign weights to every compound. This is supported by the knowledge, that tasks that have active compounds in common are better candidates for multitask transfer. This type of data weighting allows the models to communicate through the space of compounds. As initial measurements confirmed, this is a better way to weigh the data from the perspective of transfer and predictive performance.

w_1	w_2	w_3	w_4
w_5	w_6	w_7	w_8
w_9	w_{10}	w_{11}	w_{12}
w_{13}	w_{14}	w_{15}	w_{16}

w_1
w_2
w_3
w_4

Table 3.1: Two possible weightings of data

3.2 Alternative approach: Gradient boosting

Another large category in the family of boosting algorithms is gradient boosting methods. In this section, I briefly introduce the method for the case of classification problems. The main difference from AdaBoost, as the name suggests, is that here, the weights are not explicitly assigned, instead, each classifier is trained on the residual error (gradient of the loss function) of the previous classifiers [7].

To initialize the algorithm we have to start with a uniform prediction that minimizes the loss function for every sample. In the case of binary classification, we can view this as the

probability of a random point belonging to class 1. After this, we calculate the residuals for every point and then train a classifier to predict these residuals. These classifiers are usually decision trees, and we can calculate a γ value for every node of the tree, which signifies how much closer the predictions got to the true values. After this, we can recalculate the residuals and start the training over again until we have an ensemble with the desired performance.

Algorithm 2 Gradient Boosting [33]

1. Initialize the model with a constant
   $F_0(x) = \arg\min_x \sum_{i=1}^{n} L(y_i, \gamma)$
2. For $m = 1$ to M:
   (a) Compute residuals $r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)}$ for $i = 1, ..., n$
   (b) Train classifier on x and r_m and create terminal nodes R_{jm} for $j = 1, ..., J_m$
   (c) Compute $\gamma_{jm} = \arg\min_\gamma \sum_{x_i \in R_{jm}} L(y_i, F_{m-1}(x_i) + \gamma)$ for $j = 1, ...J_m$
   (d) Update model: $F_m(x) = F_{m-1}(x) + v \sum_{j=1}^{J_m} \gamma_{jm} 1(x \in R_{jm})$

This approach is different from the previously described AdaBoost algorithm and may yield a difference in results as well. My main focus will be on AdaBoost as it is more flexible for the case of multitask neural networks and the weighting is explicitly assigned to the data instead of calculating residuals. As a reminder, the weights will be important for the selection of tasks in multitask learning. However, as a comparison to the more widely used boosting techniques, I will examine some Gradient Boosting methods in the experiments section.

Chapter 4

Federated learning

Federated learning (FL) refers to multiple clients coordinated with one or more central servers for decentralized machine learning settings [15]. The goal of FL is to minimize data collection, thus mitigating the risks of traditional data center-based machine learning.

Traditional federated learning environments include a multitude of clients with little data (e.g. smartphones learning predictive texts [9]), however, in the case of pharmaceutical applications, this is hardly the case. When multiple companies with large amounts of data work together to create a shared model it is called cross-silo FL. In my report, I am concerned with a cross-silo case with a few partners, each having a varying amount of data.

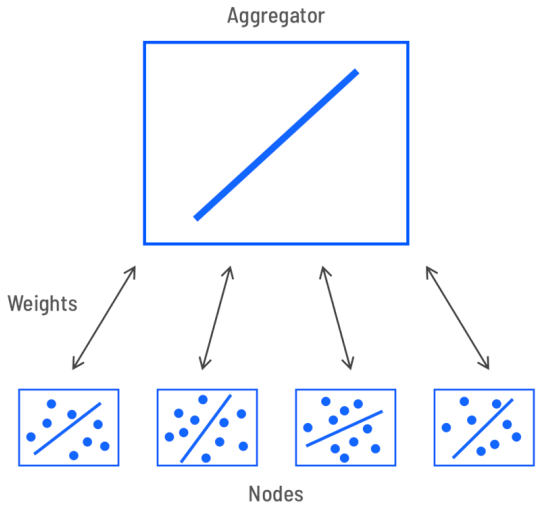


Figure 4.1: Federated averaging

FL has many aspects and still open questions in my work my main contribution is in creating models and methods that improve the predictive performance of federated learning. As the main focus is on the performance, I often omit privacy and communication-related questions, this means, that the methods are usually feasible in the context, in which they are introduced. Most of my methods are derived from already proven techniques, and the privacy and communication aspects used there can be applied in here also, if this is not the case, I will further explain these concerns.

4.1 Federated averaging

Federated averaging (FedAvg) is one of the most basic and widely used forms of secure federated learning. It builds on the assumptions that private data can not be ascertained from aggregated weights of a neural network (or gradients in the learning process). If differential privacy is also considered then no single client can influence a model in a way that might become too specific for the given client. This means that by averaging, the central model on the server learns a shared representation of the clients' data. One of the key conditions of the algorithm is that clients agree on a model architecture and use only that for training, because the models will be averaged, which can only be performed if their weights have the same dimensions and shapes. This is a very strict constraint and determines a lot of the process, still, in most cases, it is desired to use this as it makes the training very simple.

The algorithm has the following steps: First, the server initializes the weight of the central model and distributes it to the clients. Next, each client performs one iteration of gradient descent on their own model with their private data. Next, the server collects the gradients or the weights of the trained models. After this, the server averages the gradients and updates the central weights or if it collected the weights it simply replaces the central model's weights with the average of the collected weights. Finally, it can redistribute the central model to the clients to perform a new round of training.

Algorithm 3 Federated Model Averaging

Computed on the server:
initialization of w_0
for $t = 1, 2, \ldots$ **do**
    S_t = random set of clients
    for $k \in S_t clients$ **do**
        $w_{t+1}^k \leftarrow w_t - \eta g_k$
    end for
    $w_{t+1} \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} w_{t+1}^k$
end for
Computed on clients:
for $epoch \in E$ **do**
    batches $\leftarrow$ partitioning data to B partitions
    for $b \in batches$ **do**
        $w \leftarrow w - \eta \nabla l(w; b)$
    end for
end for
Sending weights to server.

4.2 Federated Model Distillation

Federated model distillation (FedMD) is another method to conduct FL. In this method, the clients do not have to agree on a model beforehand [14]. Instead, they have to construct a shared part for the dataset, on which the distillation is done. The obvious advantage of the method is that clients can personalize their models. This approach fits better with the unique data of clients, as they can construct vastly different models.

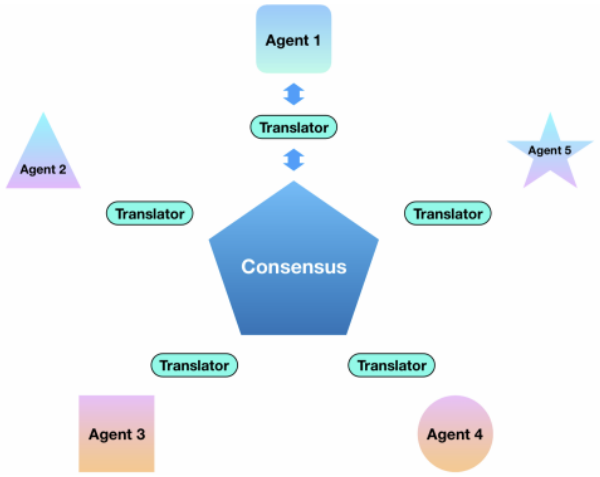


Figure 4.2: Federated model distillation

The algorithm consists of the following loop: At first, every client creates a model for themselves and trains it on their own data and the initialization of the consensus. After this, they predict all the labels of the consensus and communicate it to the server. The server recalculates the consensus as the average of the predictions and sends it back to the clients. Then every client continues their training on the new consensus, periodically retraining on their own data, to refresh the model.

Algorithm 4 FedMD

Input: $D_0, \forall i D_i$ consensus and client data
Output: f_i trained client models
Transfer training: $\forall i f_i$ train all client models until convergence on D_0 and D_i
for $j = 1, ..., P$ **do**
    Communication: Every partner calculates the predictions for D_0: $f_i(x_j^0)$
    Aggregation: Calculate prediction on server $\widetilde{f}(x_j^0) = (1/m) \sum f_i(x_j^0)$
    Distribution: sending $\widetilde{f}(x_j^0)$ to partners
    Processing: train f_i on $\widetilde{f}(x_j^0)$
    Refreshing: train f_i on D_i
end for

4.3 Federated Boosting

When implementing boosting in a federated environment, several challenges present themselves, but the main problem is the information stored in the weights. The weights of the data carry much information about the data itself, meaning that sharing the weights would violate the data owner's privacy.

For a naive federated boosting algorithm, I propose the following: It should be based on the normal boosting method, where every partner trains their own ensemble model. And before the training, they agree to average out every nth model of their ensemble. Preferably the averaging should occur frequently to not let the models diverge. But the partners also learn their own ensemble with their own weighting of models, thus not all models should be shared. Another method to keep the clients from diverging is to initialize

the training of some iterations with the averaged-out model, but the weighing of the data should remain in the hands of each partner.

This method should let partners leverage the transfer effects between tasks, even if the tasks are largely found on other partners' servers. This way, the boosting of the transfer effects is not lost with the federation and the separate weighting of their data.

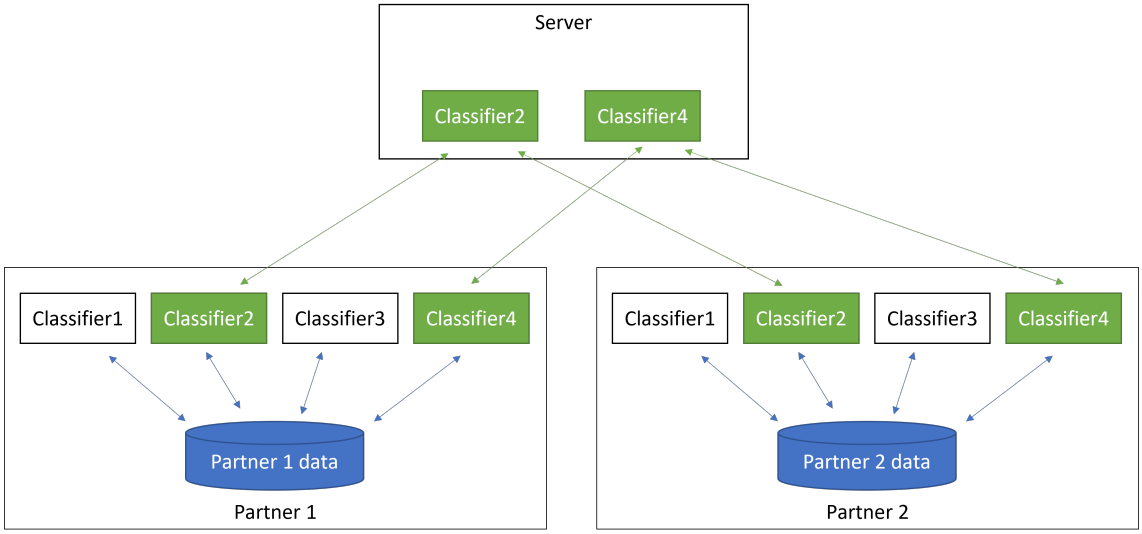


Figure 4.3: Schematic algorithm of federated boosting, with averaging every second classifier

For my initial measurements, I used the method described above with averaging every second model, however, the ensembles started to diverge and after a while, the performance dropped. Thus I changed the method to average every model of the ensemble. From this point of view, it seems to be only a change in hyperparameters, however, in reality, this type of averaging presents a change in philosophy. Now it resembles much more the original FedAvg, with a significant difference in reweighting the data (and the private weights for the regression on the models' outputs). If we approach it from this perspective we can look at it as the conduction of FedAvg with always performing the local training on a new model instead of the previously averaged one. The information stored in the data weights is the only additional information maintained. The other major difference could be the communication of two models in every iteration: the averaged model of the previous round and the newly initialized model for the actual round. This way, the communication overhead is doubled when compared to FedAvg. If this would be too much overhead for the network then the partners can agree initially to always continue training on the averaged model of the previous round. These questions will be further discussed in chapter 7 after the introduction of the algorithm and the description of the sufficient multitask bases.

Chapter 5

Multitask learning and multitask boosting

Multitask learning is a burgeoning field with extensive literature. First, it was proposed by Caruana [1] and it was described as inductive transfer between tasks. Inductive transfer means knowledge is shared between the tasks through the learning process in a manner where each task has the opportunity to shape the representation, which results in a general latent space in neural models.

The approach is most prominently used in deep learning, as it is easy to construct networks with multiple outputs, for the tasks. One of the best-known examples where multitask learning is efficient is Deep Convolutional Neural Networks, where the deeper layers correspond to higher-level features, like edges and corners, which are common among tasks. This is why often the networks are constructed with their deeper layers shared amongst the tasks and the top layers specific for each task [31]. This structure allows for shared deep representations to develop while also preserving specific features for the individual tasks in the top layers. In the case of learning from compound fingerprints, deep networks are not justified, thus I use a shallow architecture with one layer shared between the tasks and every task having its own output in the network.

In DTI prediction, it was rediscovered in recent years and now we understand much more about the task relationships and the nature of transfer [38]. I Xu et al. have found that tasks that correlate can benefit from multitask transfer, and not correlated tasks can have inferior performances even when compared to a singletask method. This was explicitly analyzed in a quantitative structure-activity relationship problem, which is closely related to DTI prediction.

To combine the traditional "signal amplification" and the current "correlating tasks" explanations for multitask learning, we have to look at statistical data amplification. This explains how the network can distinguish data from noise. When one task eavesdrops i.e. takes information from the backpropagation on other tasks, the relevant information which is common in both tasks gets amplified whilst the noise (or information not common for the tasks) diminishes.

Now we can see that the composition of tasks determines the result of the training and, thus, the performance of the model. Training a network on a large number of tasks can help the overall performance, but this increase is usually present on the average of the performances, while the performance of individual tasks can decrease. That is why carefully selecting the tasks for training is crucial. However, calculating correlation for large numbers of tasks is usually not feasible. If the selection can happen adaptively during

training it may simplify the multitask learning process. The iterative weighting of data in boosting gives a possible solution for the selection problem.

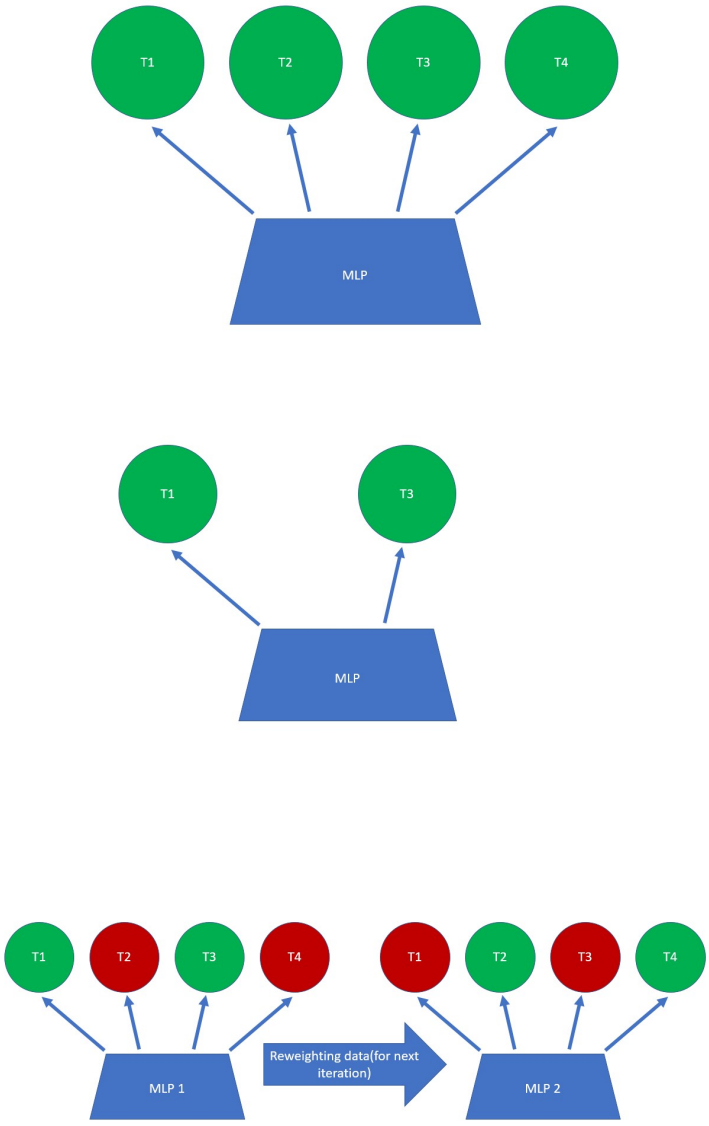


Figure 5.1: Possible ways for multitask learning, from top to bottom: Without task selection (may cause performance decrease). With manual task selection (capacity decrease). With adaptive task selection.

As we can see in figure 5.1, multitask learning can be conducted in many ways, all having their advantages and disadvantages. In practice, it is useful to combine the approaches: Do a selection manually either by clustering or using some kind of heuristic and use the adaptive method to leverage the information that the manual selection missed. In my work, I do manual selection first: I apply multitask boosting in the learning process using only a family of targets, namely tyrosine kinases.

5.1 Multitask boosting

As previously described, there are multiple ways to conduct multitask boosting in my approach, I will use a traditional method to multitask transfer: neural network weights. This has the advantage of not having to formulate a model for the task connections, instead, it is implicitly contained in my classifiers. As we can see, it is similar to the original multitask learning with neural networks, with the main difference being representation.

This type of model contains multiple different representations for the tasks. In every model, the tasks with higher weights for their samples dominate the representations, and they are predicted best by the given models. This means that tasks, which perform weaker, in the beginning, will shape the representations of later models. Here the assumption is that tasks, which perform better from the beginning are already able to leverage information because they usually form the majority. Of course, the key question for the results is the regression on the outputs of the models, to achieve an overall better performance.

Chapter 6

Missing data

Data for drug research is dependent on costly and time-consuming experiments, thus when compared to other fields relatively less data is available. This small amount of data is usually distributed amongst a large number of drug-target pairs, resulting in a sparse representation. This is the case for DTI prediction, where the target bioactivity matrices are known to have a significant level of sparsity.

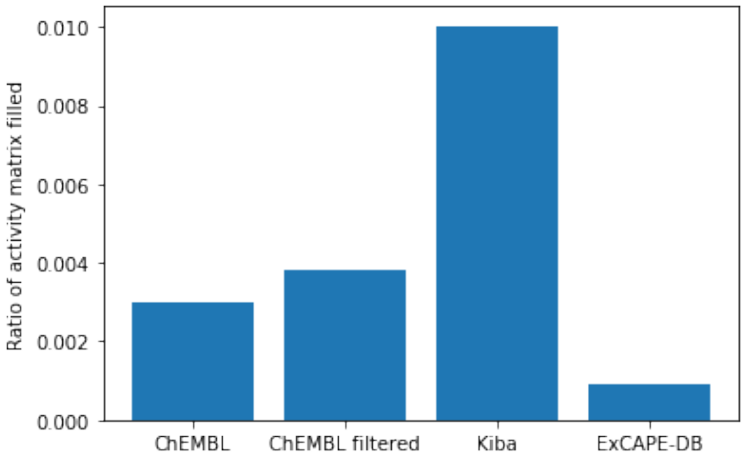


Figure 6.1: Ratio of sparsity in different databases

As we can see from figure 6.1, in reality, less than 1% of the matrices contain useful measurements, and the majority is empty. It is challenging to learn from this data, as in the case of neural networks backpropagation is only possible through the non-zero (i.e. measured activity or measured inactivity) elements of the matrix.

However, if we examine the missing patterns in the matrices we can conclude that there is a system for the missing values. This is called a Missing Not At Random (MNAR) type of data, which means that information about a missing value can be inferred from its absence/presence. This can be explained by the nature of the experiments conducted during the research process: The experiments were conducted at key points of interest. If due to a structural property or some other promising feature a drug is likely to bind to a target that is important for the given research project then this pair has a high probability of being measured already. Also, if a known drug exists for a target then similar drugs are likely to be measured, to see if their effects are similar.

Dealing with MNAR data is a challenging task, as crucial information might not be present during the learning process. Multiple approaches exist when performing learning on partially labeled data. I discuss two distinct approaches, each with its own pros and cons.

The first method is training only on the labels that are present. This means that only exact data enters the training process, and no further assumptions are needed on the nature of the data. However, as expected, this results in a smaller training set, which can cause a decline in predictive performance. At first look, this may seem the logical choice for training as fairly large datasets are available, however, information loss does exist in this scheme, as the location of the data is not explicitly fed to an MLP, thus we cannot draw conclusions based on blocks of existing values.

The other main way to handle the MNAR data is to impute values for the missing parts. In practice, this could mean several things: In a DTI setting, the obvious choice is a ternary representation (e.g. assigning 1 for an active pair -1 for inactive ones and a soft 0 as probably inactive). This representation lives with the assumption that values not measured are most likely inactive. The main of this approach concern is finding the correct weights for the model to learn: Zeros should have a smaller weight in the training process, otherwise, they would distort the predictions, in a boosting case, this should always be constant and multiplied by the algorithmically assigned weights for every element. Further modifications of imputing might include iteratively imputing values that are predicted with higher confidence during training.

Chapter 7

Federated multitask boosting

In this chapter, I describe the combination of the previous approaches and methods into a cohesive algorithm. The basic concepts for using boosting with federated learning and multitask are described in sections 4.3 and 5.1. In this part, I am building on the concepts described in these sections.

7.1 High level overview

To achieve federated multitask boosting, I use the FedAvg and the AdaBoost algorithms as the starting point the point of combination for the algorithms is the ensemble models. As I use neural networks for base classifiers they can be averaged the same way as in the FedAvg algorithm, and for privacy-related concerns, every client keeps their own data and model weights for the AdaBoost part of the ensemble. This allows a natural progression of training for every client while they still make use of information coming from other clients, the same way as they would in the FedAvg. The multitask nature of the training appears in the base classifiers (because they are neural networks fully connected, with multiple outputs), and it is amplified by the selection of data, through weighting, for every training round.

7.2 The Fedboost algorithm

To understand federated boosting, I start from the same setup as the FedAvg algorithm: Every partner has their own dataset, with possible overlaps in both the compound- and target spaces. The clients agree on a neural network architecture before, and (for simplicity) the set of targets, which they aim to use.

7.2.1 Training

After this, the server initializes the model weights and distributes them to the clients. The clients run a local training sequence on the model, in my experiments, this means one epoch of training. Next, the clients send their models back to the server for aggregation (alternatively, like in FedAvg it is enough to send the gradients). The server averages the weights and produces the final model, which will be part of the ensemble. After this, the model is sent back to the clients and they calculate the error of the model on their data using the formula from AdaBoost, and based on this, they assign an individual weight to

the model (same as AdaBoost, but it is unique for every client). If the partners know the weight of the model, they can reweight the data. If the cycle is complete, the server sends a new model for the clients and every client starts the training again.

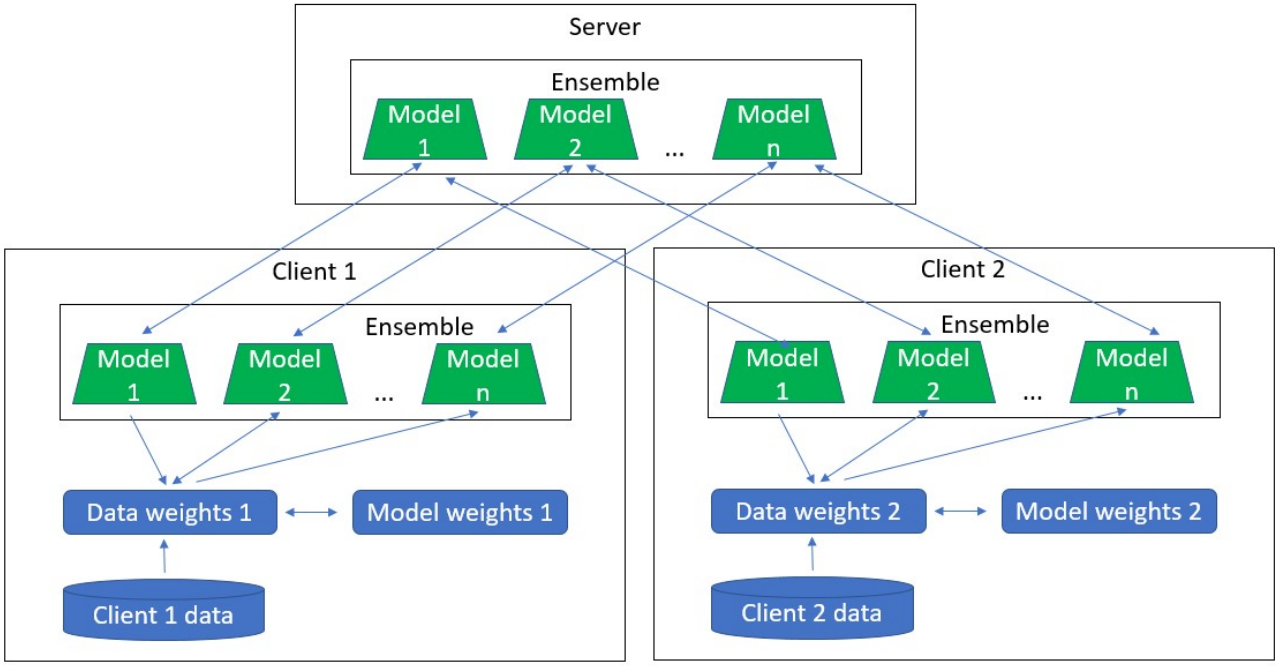


Figure 7.1: Federated multitask boosted training

7.2.2 Prediction

When making predictions every client can rely on the common ensemble with their own weights assigned to each model of the ensemble. They simply feed the data to the models, and calculate the regression with regard to the model weights to produce the final predictions, this process is the same as the one used in AdaBoost.

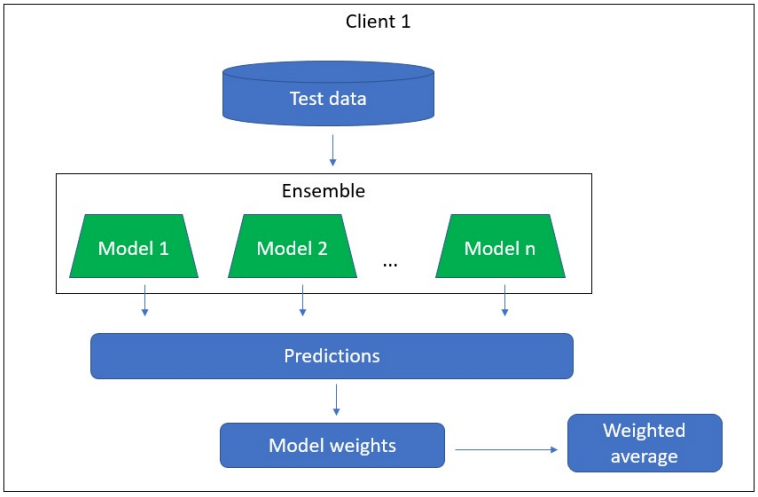


Figure 7.2: Federated multitask boosting prediction

Algorithm 5 FedBoost

Initialization on the clients:
Initialize the observation weights $\omega_{i0} = 1/N$, $i = 1, 2, \ldots, N$
initialization of ensemble $E = \{\}$
Computed on the server:
for $t = 1, 2, \ldots$ **do**
    initialization of w_t
    S_t = random set of clients
    for $k \in S_t$ clients **do**
        $w_t^k \leftarrow w_t - \eta g_k$
    end for
    $w_t \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} w_t^k$
    sending w_t to clients
end for
Computed on clients:
for *epoch* $\in$ *Epochs* **do**
    batches $\leftarrow$ partitioning data to B partitions
    for $b \in$ *batches* **do**
        $w \leftarrow w - \eta \nabla l(w; b; \omega_t)$
    end for
end for
Sending weights to server.
Receiving averaged w_t
$E \leftarrow E \cup \{w_t\}$
Compute error $Error_t = \sum_{i=0}^{n} \omega_{i(t-1)} * I_{y_i \neq \hat{y}_i}$
Compute $\alpha_t = \frac{1}{2} log(\frac{1 - Error_t}{Error_t})$.
Set $\omega_{it} \leftarrow \omega_{i(t-1)} * e^{-1^{I_{y_i \neq \hat{y}_i}} * \alpha_t}$, $i = 1, 2, \ldots, N$.

Chapter 8

Evaluation settings

This chapter deals with the details and the implementation of the model. The model was created in a simulated environment, thus negating the cost of communication and focusing mainly on the predictive performance of the algorithm.

8.1 Data

The dataset I used is based on the KIBA dataset [30], which itself is heavily based on the ChEMBL database. ChEMBL is a public database of small molecules and assays mainly for proteins. It is managed by hand and the data is originated from scientific sources: experiments, articles, and other publications. It is a good source of DTI data. KIBA is a heavily revised and filtered dataset, which means it contains much fewer compounds and targets than the original ChEMBL, but the bioactivity values become more accurate and denser with filtering.

My dataset was extracted from KIBA and the original data contained 467 assays, to present the multitask effect 60 assays were picked, which belonged to proteins in the tyrosine kinase family. The great challenge of the dataset is the sparsity because only about 10% is filled out of the activity matrix, the rest are soft zeros. Although a lot of the dataset is missing, this missing is informative itself as these values are presumably inactive, because it was deemed not worthy to measure them.

For representations of the molecules, the Extended Connectivity Fingerprint [2] was used, which is a popular representation for predicting activities, as it is descriptive of certain features of the molecules. ECFP is a circular topological representation, which is generated by noting the circular atom neighborhoods for every atom and using a hash function on the results. For traditional machine learning, we can think of it as one-hot vectors for parts of the molecules.

8.1.1 Federated partner split

The federated scenarios assume data that is distributed between partners. In reality, both the tasks and the assays are distributed. One partner can own any number of assays and multiple partners can own the same assay, this is also true for compounds. However, for a more measurable scenario, I only federated the data in the compound space and distributed the compounds between the partners. This resulted in seven partners, each having varying amounts of data with some overlap between them.

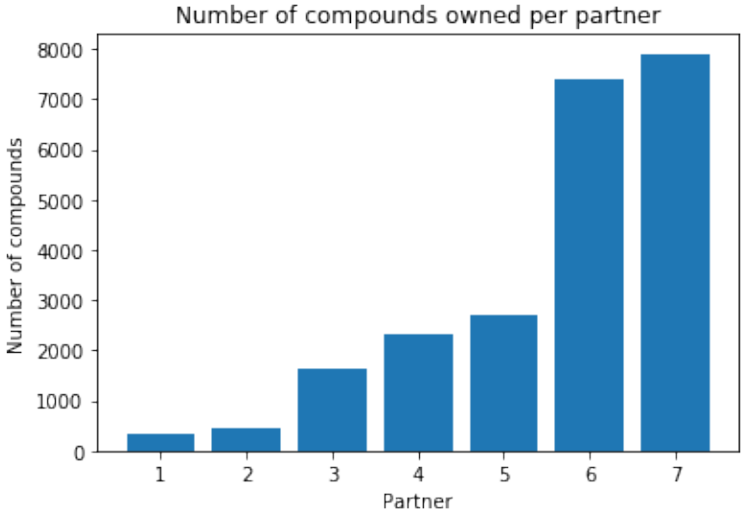


Figure 8.1: Number of compounds owned by federated partners in the distributed dataset

8.2 Framework

The implementation of the model was done in python, using directed frameworks for the specific parts of the algorithm. As the federation is only simulated and done on one machine, the key difficulty in implementation was the sparsity of the data 6.

8.2.1 SparseChem

SparseChem [25] is a neural network framework for training and evaluating models on sparse datasets, used mainly in chemoinformatics for DTI prediction. It is based on PyTorch and allows the creation of neural networks with dense weight matrices that can handle (scipy) sparse data as input and output.

8.2.2 Extending the framework

A key element of the framework is the introduction of sparse datasets, which store the data based on indices and values, which are reconstructed during training when needed. One of my contributions was a reimplementation of the sparse dataset, which can handle the data weights during AdaBoost training, meaning, it stores the weights in a similar way, and reconstructs the during training for each batch of train data.

8.3 Hyperparameters: The architecture of networks

The architecture of the networks was the following: Every network had one hidden layer (the trunk of the model) and the output layer (the head). The input was the 32000-bit long molecule fingerprint, the hidden layer contained 1400 neurons and the output layer was either one neuron for singletask learning or 88 neurons for a multitask case. The activations were ReLU functions in the hidden layers and a sigmoid predicted the output probabilities. The network was optimized using stochastic gradient descent. The loss function in the non-boosting experiments was binary cross-entropy (BCE) and for the

AdaBoost version, I have implemented a weighted version of the BCE, to consider the data weights of the AdaBoost algorithm. My implementation of the loss was based on the PyTorch implementation of BCE, but it was done in python.

Chapter 9

Results

The goal of this chapter is to contextualize the predictive strength and other properties of the FedBoost algorithm when compared to several baseline models also used in the field.

9.1 Realistic measurement scenarios

In my experiments, I use the AdaBoost algorithm with an MLP as discussed in the previous chapter. The experiments can be classified in two ways. The first is the technologies used, in classification, I distinguish between the following categories:

- A single MLP-based solution as the baseline
- An AdaBoost framework using MLPs as classifiers
- Partnerwise training on a single MLP without averaging
- Federated training on a single MLP with FedAvg
- Federated training using FedBoost (averaging every model in the ensemble)

The other distinction can be made in the multitask nature of the training, in this way, the two categories are single- and multitask training. This results in the experiments seen in table 9.1.

As we can see, the first, third, and fourth measurements can serve as a baseline for the performance of machine learning models both in a federated and in a traditional scenario. As these use only the traditional neural network approach for DTI, and in the fourth case, the FedAavg is a well-proven federated learning algorithm, these will be useful comparisons to measure the performance of an AdaBoost model, both on multitask and on federated data.

The experiments are designed in a step-by-step approach, with the hope that every improvement or possible deterioration can be explained by the addition of an aspect in the process. This way, we can see the effect of boosting on the training, and the improvement brought by the boosting of the transfer effect in the multitask scenarios.

	Singletask	Multi task
MLP	One MLP learning one selected task	One MLP learning multiple tasks
AdaBoost	An AdaBoost ensemble of MLPs learning one task	The ensemble learning multiple tasks, with the AdaBoost weights on the data assigned to the compounds
Partnerwise	Splitting one singletask by compounds to partners and every partner conducts their own learning without averaging using one MLP	Splitting the full data realistically to partners and everyone performs multitask learning with one MLP without averaging
FedAvg	Using the same split as above the partners conduct a Federated Averaging scheme on the task	Using the same split as above the partners conduct a Federated Averaging scheme on the data
FedBoost	Using the same split as above, the partners each conduct AdaBoost learning on their part of the task by averaging every model of the ensemble	Using the same split as above, the partners each conduct Adaboost learning on their part of the data with averaging every model of the ensemble, using compound weighting

Table 9.1: Proposed scenarios to measure the effects of boosting and transfer

9.2 Metrics

In this section, I discuss the results of the outlined experiments. At first, I describe the baselines from every scenario, and gradually build on this. The results will be evaluated by focusing mostly on the multitask transfer's effect and resulting in a predictive performance gain. The most important metric of comparison is the AUROC scores, so all experiments will be measured by their improvement in this metric. To get an accurate and fair comparison I have measured the performance on data ordered into five folds. From this, a learning curve can be constructed, where the first measurement will be the AUROC values of a model trained on only one of the folds. The second model will be trained on two folds, and so on until four folds are used for training. The evaluation is always done on the complementary folds of the train data, except for fold 0, which is always used for evaluation. Every plot will have the train folds on the x-axis and the resulting AUROC values on the y-axis. Furthermore, the average AUROC score of the four training sessions will also be visible.

However, improvement in AUROC scores in itself does not mean, that one model is better than another. To see that an improvement is relevant, we have to conduct significance tests. The most common significance test, when comparing AUROC scores, is the DeLong test [5]. This results in a p-value, which can determine the probability of the AUROC scores of two correlated models being significantly different. The DeLong test's null hy-

pothesis is that there is no significant difference between the two models' AUROC scores, thus the lower the p-value the larger the probability of the two models producing significantly different results.

9.3 Baseline MLP measurements

In the first experiment, one MLP based on the SparseChem framework was trained on the data described above, the MLP was trained for 20 epochs every time, the folds were permuted, where it was possible (the 1, 2, and 3 fold scenarios), and the final results were averaged to get plot 9.1.

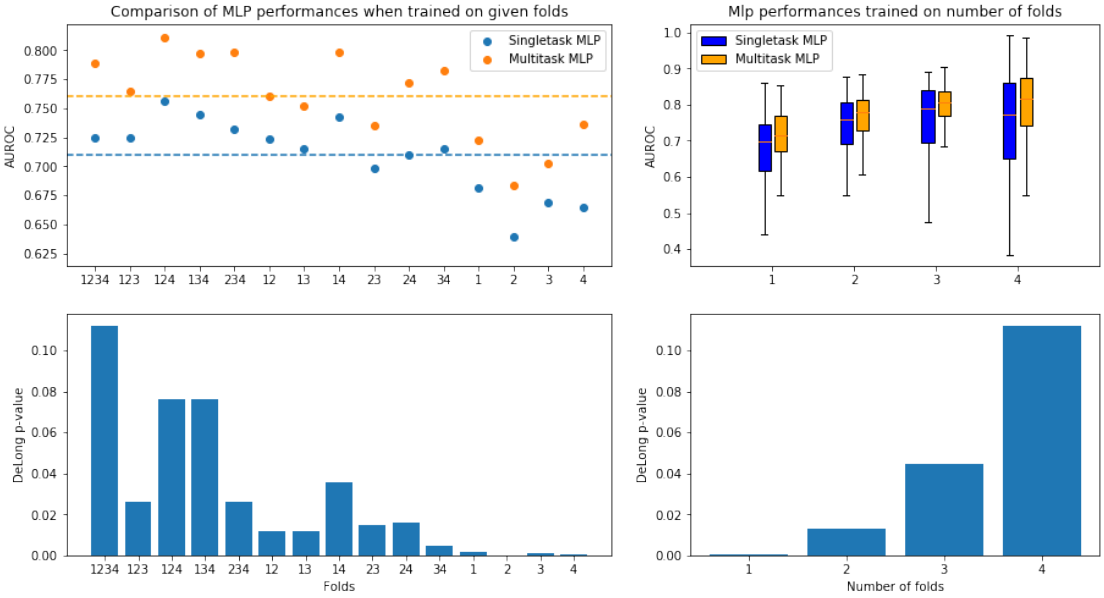


Figure 9.1: AUROC scores averaged over tasks in single- and multitask learning-based MLP shown by folds used for training with the average scores indicated by the dashed lines (left) and boxplot of performances on tasks when averaged over fold permutations (right) with their respective DeLong p-values (bottom)

As we can see, the performance of the model varies, but the single- and multitask scenarios are strongly correlated, and their performances on almost all folds are similar, with a slightly higher variation on the one-fold training (which is expected because in this case, very little data is available for each task. The multitask effect can be seen as the multitask model having a significant edge over the singletask one in nearly every scenario, which also results in smaller variations in the performances, as seen on the boxplot.

9.4 The effect of boosting on the multitask transfer

When using boosting, every neural network can leverage the weighting of the data for its own benefit, meaning that the ensemble is more likely to pick up on connections between tasks. The AdaBoosted models were trained for 3 epochs each and a sum of 10 AdaBoost iterations.

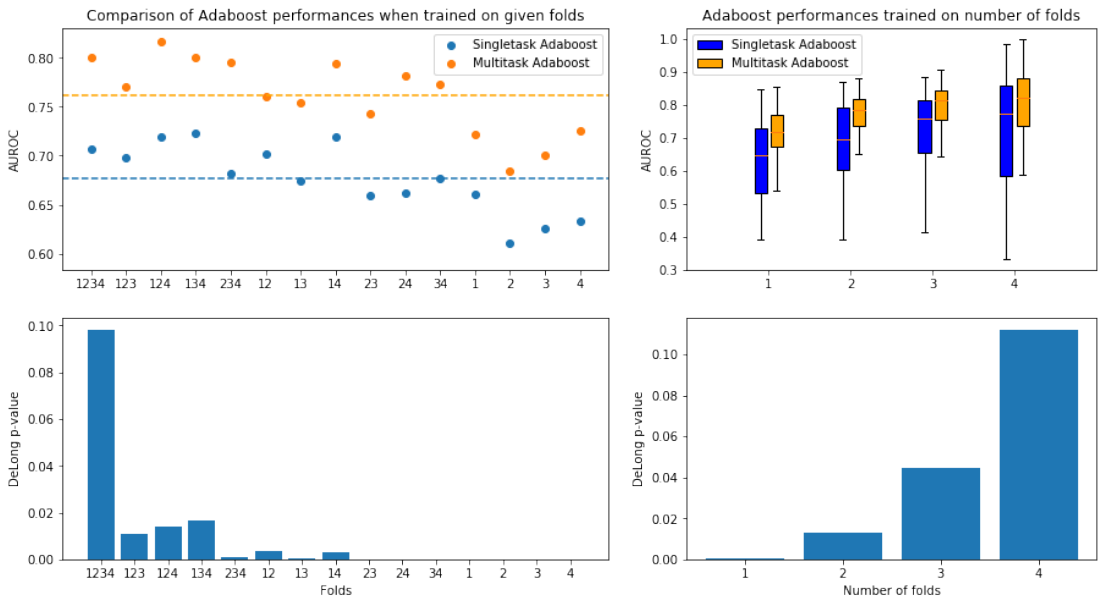


Figure 9.2: AUROC scores averaged over tasks in single- and multitask learning based AdaBoost shown by folds used for training with the average scores indicated by the dashed lines (left) and boxplot of performances on tasks when averaged over fold permutations (right) with their respective DeLong p-values (bottom)

As we can see in plot 9.2 the AdaBoost model cannot perform significantly better, than a basic MLP: the average AUROC scores for multitask training stay around 0.76. However, the singletask scenarios perform significantly worse, this leads to the conclusion, that for boosting to be effective, a large amount of data is required.

As the multitask scenario seems to be similar to the original MLP's, it is worth comparing them. When examined, on 9.3 we can see that the MLP and the AdaBoost models are almost identical, however, when sufficient data is present the AdaBoosted model seems to outperform the baseline, although the differences may not be larger than noise in this case.

9.5 Federated scenarios

In the following scenarios, the usual data was split up between partners, and every partner had a different amount of compounds. The previously described scenarios were modified in part, because of time-consuming training and the performance enhancement seen on the models when not using full federated averaging, instead only averaging the trunks of the models. In the end, the following measurements were conducted: Partner-wise training, this is a scenario, where every partner only trains on their own data using a single MLP, this is analogous to the very first experiment described in this chapter, only on smaller amounts of data. Next, we must see the improvement in the use of federated averaging to have a comparison, if our models perform worse in any way than this algorithm, then there might be flaws that need to be corrected, all in all, this can serve most as a federated baseline. Finally, the federated boosting algorithm described in chapter 7 is used to train using every partner's data. All of the partners' results can be found for all federated

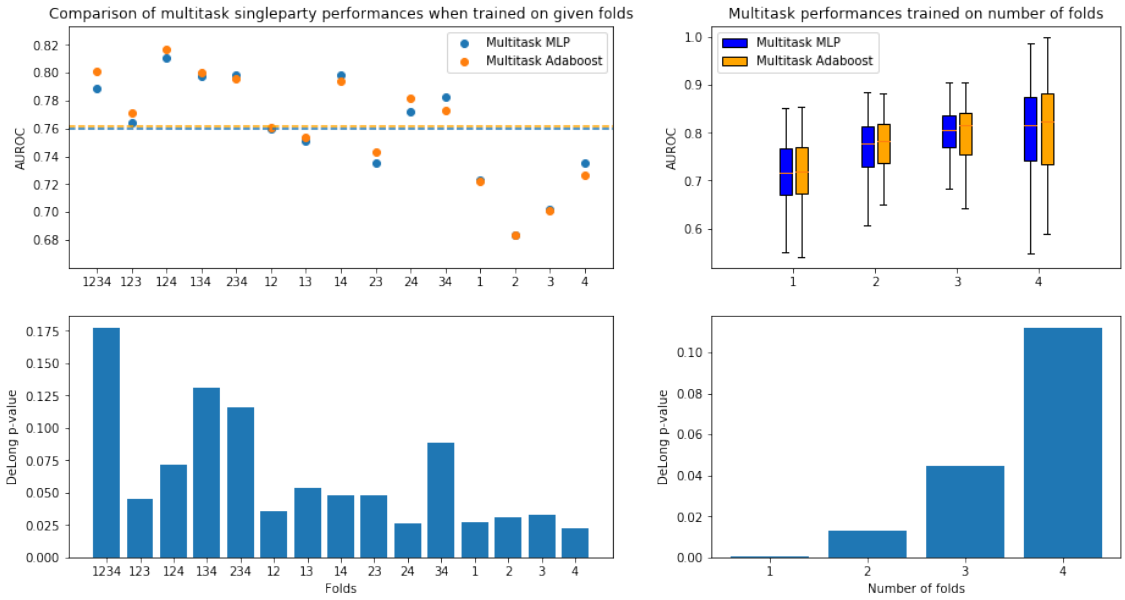


Figure 9.3: AUROC scores averaged over tasks in MLP and AdaBoost based multitask training shown by folds used for training with the average scores indicated by the dashed lines (left) and boxplot of performances on tasks when averaged over fold permutations (right) with their respective DeLong p-values (bottom)

scenarios in appendix A.1. Here only partner 5's data will be presented, this is the partner with the 5th smallest amount of data, this represents a well-balanced case in every way.

9.5.1 Singleparty trainings

In this scenario, every partner trains only on their own data, as expected this brings a significant drop in predictive performance as opposed to the full data singleparty training. In this scenario, no boosting is used, and only the performance of one MLP can be seen. The difference between single- and multitask is more pronounced as less data is available and information is more valuable. The following scenarios should be compared to this as, in reality, federated partners have the same proportion of data when compared with the community's whole aggregated data set.

9.5.2 Federated Averaging

In this scenario, all partners take part in a federated averaging setup. The models are trained for the same number of epochs, but after every epoch, the trunks of the models are averaged out and set to the community's consensus. This results in a more generalized trunk representation, which leads to an increase in predictive performance as expected. The increase, in this case, is small, but for partners with smaller datasets, it is more significant as seen in the appendix.

9.5.3 Federated boosting

This scenario is the most important from the measurements as we can see the performance of a model using boosting in a federated and multitask environment. When looking at

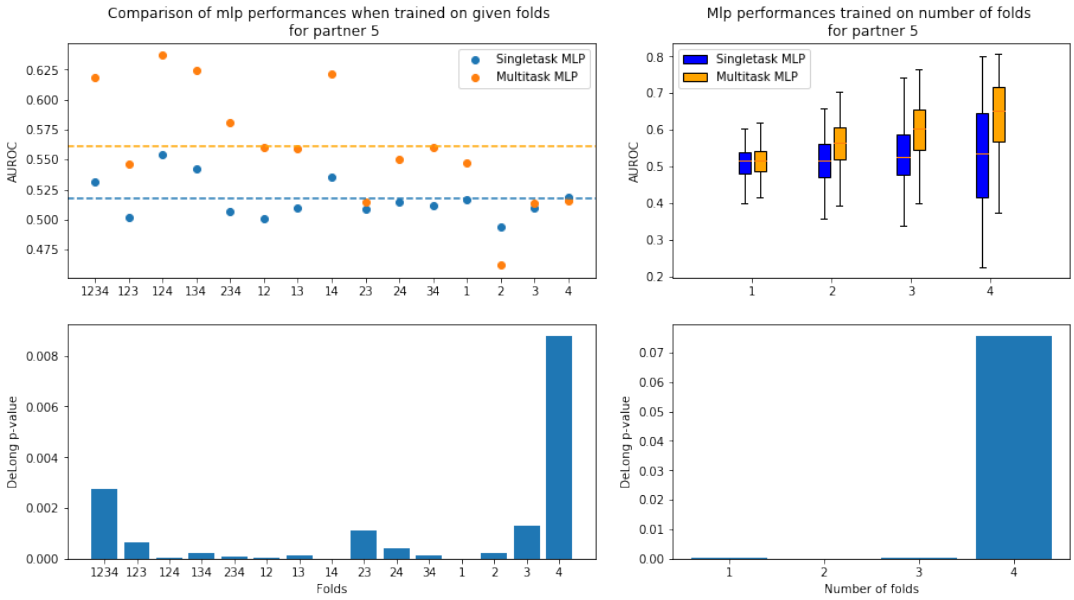


Figure 9.4: AUROC scores averaged over tasks in MLP-based training for partner 5 shown by folds used for training with the average scores indicated by the dashed lines (left) and boxplot of performances on tasks when averaged over fold permutations (right) with their respective DeLong p-values (bottom)

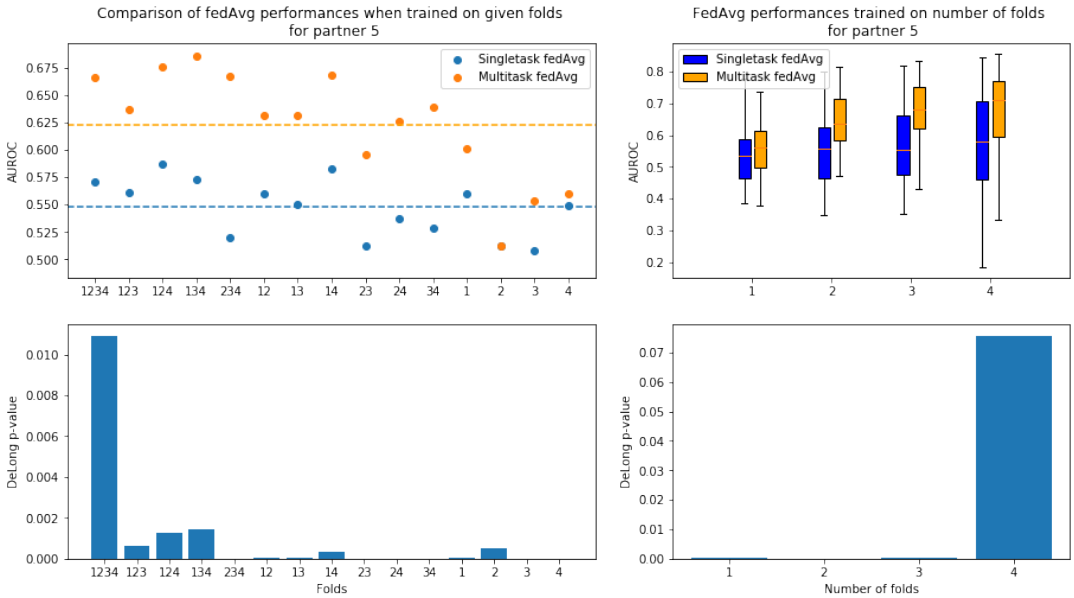


Figure 9.5: AUROC scores averaged over tasks in FedAvg-based training for partner 5 shown by folds used for training with the average scores indicated by the dashed lines (left) and boxplot of performances on tasks when averaged over fold permutations (right) with their respective DeLong p-values (bottom)

the singletask-multitask performances of FedBoost no irregularities can be found, the multitask case outperforms the singletask by a significant amount, especially in cases with extremely small amounts of data.

The next logical step is to compare the method's performance to the performance of FedAvg. As the final goal is to use it in a multitask setting and both methods behave

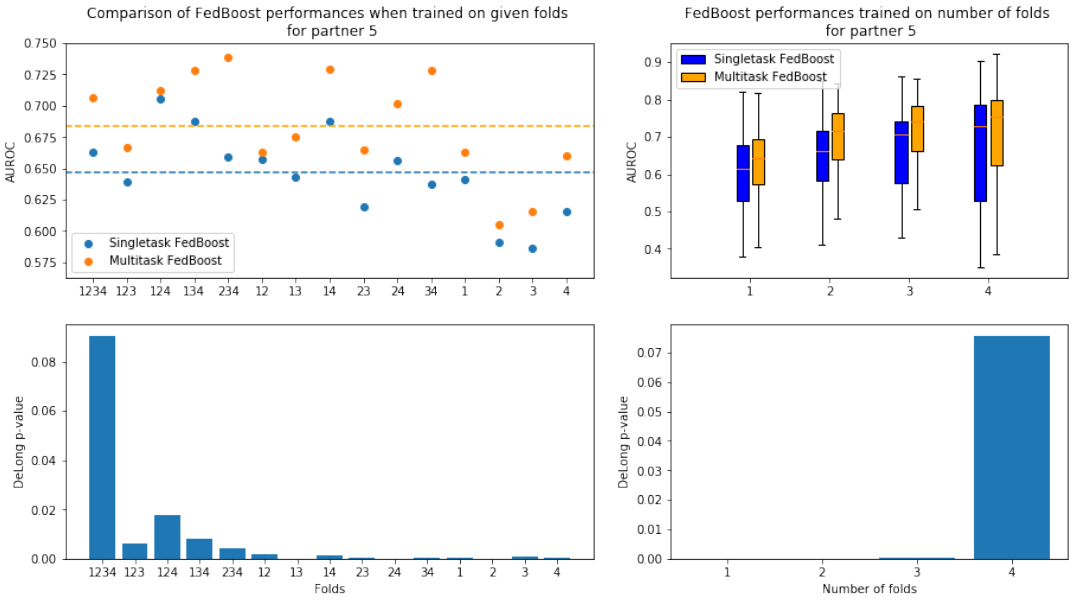


Figure 9.6: AUROC scores averaged over tasks in FedBoost training for partner 5 shown by folds used for training with the average scores indicated by the dashed lines (left) and boxplot of performances on tasks when averaged over fold permutations (right) with their respective DeLong p-values (bottom)

"regularly" (the multitask version is stronger than the singletask), I only compare the performances of multitask learning.

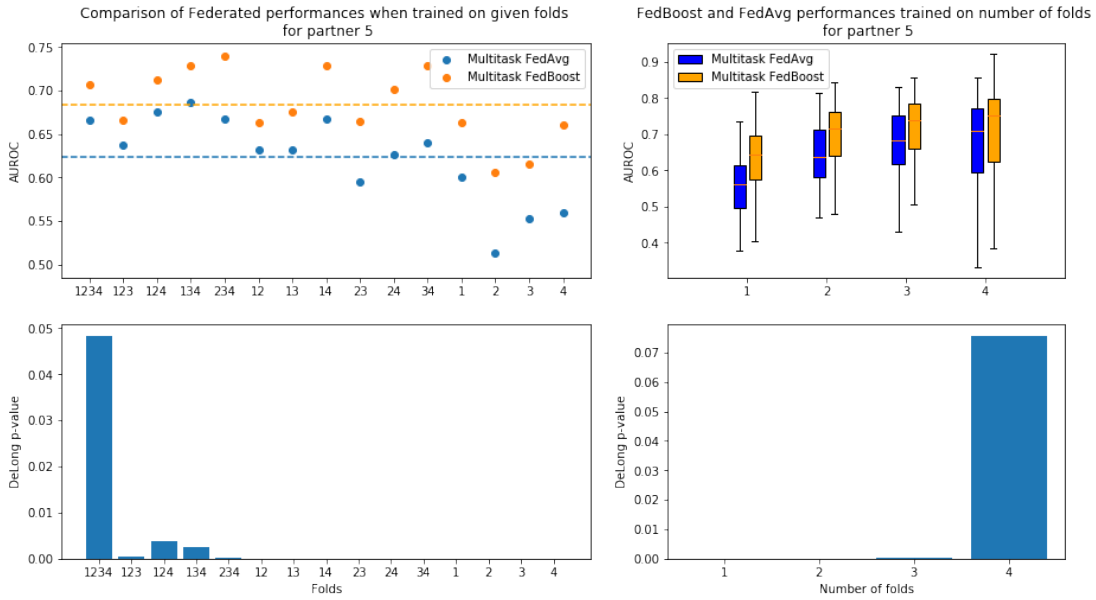


Figure 9.7: AUROC scores averaged over tasks in FedBoost and FedAvg training for partner 5 shown by folds used for training with the average scores indicated by the dashed lines (left) and boxplot of performances on tasks when averaged over fold permutations (right) with their respective DeLong p-values (bottom)

As we can see in figure 9.7 the FedBoost case consistently outperforms the FedAvg in a multitask case, thus proving its effectiveness. To further examine the results, we can

compare them to the partnerwise MLP performances, this way, we can see the ratio of improvements from the federation and the boosting.

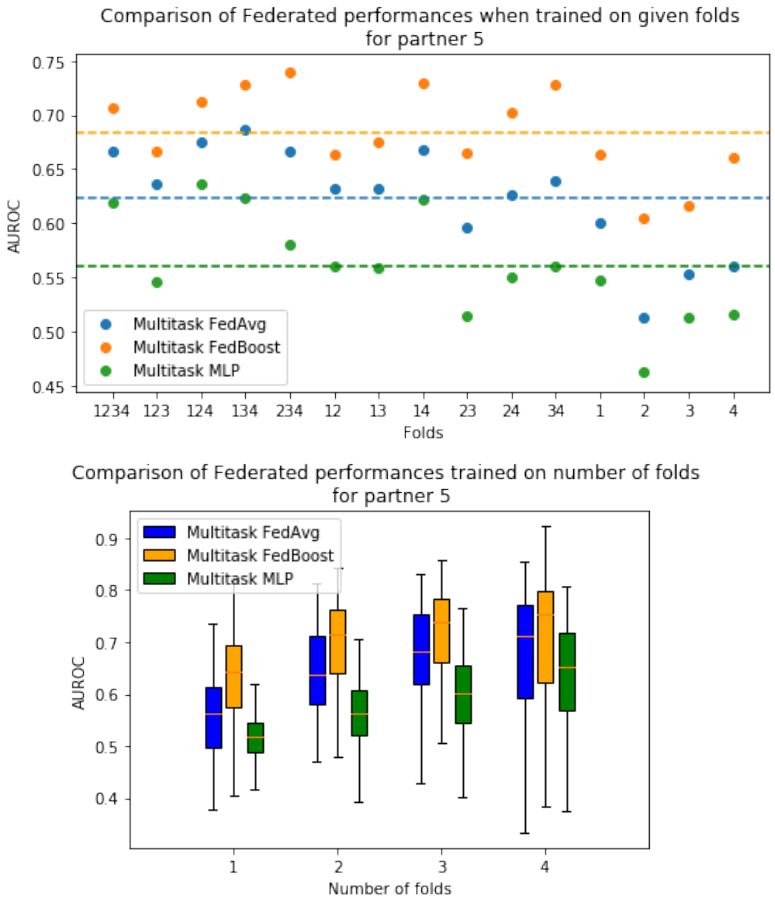


Figure 9.8: AUROC scores averaged over tasks in FedBoost and FedAvg and simple MLP training for partner 5 shown by folds used for training with the average scores of the methods indicated by the dashed lines (top) and boxplot of performances on tasks when averaged over fold permutations (bottom)

As seen in figure 9.8, The MLP cannot perform well on this little data, only getting slightly better when four folds are used for training. The FedAvg improves this by enabling information sharing from the partners and the FedBoost further enhances the predictive performance by utilizing adaptive weighting for the data. This proves that boosting is a suitable technique to improve the performance of federated multitask models.

9.5.4 Comparison on federated schemes

As we can see, the performance of basic federated learning models can be improved by using simple techniques. The FedAvg model already gives a good performance, but in most cases boosting can improve this even with a naive algorithm, which uses averaging itself. Most likely, with an algorithm that can learn explicit task relationships, the multitask transfer effect can be enhanced even more. This means that boosting is a good approach for federated transfer learning and should be further explored.

Chapter 10

Conclusion

In the report, I have presented boosting in a multitask environment and used my findings to construct a federated learning scheme where partners learn by boosting, and knowledge transfer is possible between their respective AdaBoost ensembles.

To achieve this, I have explored the literature on multitask boosting. I have found that it is a possible and efficient form of multitask learning. In traditional boosting schemes, simple models are used, and neural models are usually overlooked as they are too complex for this type of learning. In my work, I have presented a use for the neural network in the boosting algorithm and thus approached multitask boosting by implicitly learning task relationships: in the representations of neural networks.

Federated learning is very often based on a multitask transfer scenario; to simulate this, I have designed experiments to measure the effect of boosting. At first, I showed that in a singleparty scenario boosting does help the multitask transfer. For the multiparty scenarios, the baseline can be federated averaging (one of the most widely used algorithms in federated learning). The federated boosting algorithm is somewhat based on a federated averaging scheme, but it is much more reliant on data weighing. I have shown that the federated boosting algorithm can outperform the basic averaging in multitask learning if the partner in question has enough data in itself. This can be a good way to improve predictive performances in federated learning, and future algorithms can be based on this framework.

My future plans involve a more subtle approach to multitask in federated learning. The boosting ensembles would have single outputs and the partners would learn task relationships to their data. This can produce the final model weighting and get the output of the ensemble. This way, the boosting would be much more strict, and the task relationships should be learned in a new way to allow for federation.

To summarize, boosting is a relevant and simple way to enhance the performance of multitask models; it can and should be adapted to federated learning frameworks, but there are still open questions about the optimal way of utilization for the algorithm.

Acknowledgements

This research was funded by the J. Heim Student Scholarship, OTKA-K139330, the European Union project RRF-2.3.1-21-2022-00004 within the framework of the Artificial Intelligence National Laboratory, New National Excellence Programme of the Ministry of Innovation and Technology, code number ÚNKP-22-2-I-BME-70, funded by the National Research, Development and Innovation Fund.

List of Figures

List of Tables

Bibliography

[1] Rich Caruana. Multitask learning. *Machine learning*, 28(1):41–75, 1997.

[2] ChemAxon. Chemaxon documentation website. `https://docs.chemaxon.com/display/docs/Extended+Connectivity+Fingerprint+ECFP`.

[3] Ruolan Chen, Xiangrong Liu, Shuting Jin, Jiawei Lin, and Juan Liu. Machine learning for drug-target interaction prediction. *Molecules*, 23(9):2208, 2018.

[4] Kewei Cheng, Tao Fan, Yilun Jin, Yang Liu, Tianjian Chen, Dimitrios Papadopoulos, and Qiang Yang. Secureboost: A lossless federated learning framework. *IEEE Intelligent Systems*, 36(6):87–98, 2021.

[5] Elizabeth R DeLong, David M DeLong, and Daniel L Clarke-Pearson. Comparing the areas under two or more correlated receiver operating characteristic curves: a nonparametric approach. *Biometrics*, pages 837–845, 1988.

[6] Yoav Freund. Boosting a weak learning algorithm by majority. *Information and computation*, 121(2):256–285, 1995.

[7] Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

[8] Anna Gaulton, Louisa J Bellis, A Patricia Bento, Jon Chambers, Mark Davies, Anne Hersey, Yvonne Light, Shaun McGlinchey, David Michalovich, Bissan Al-Lazikani, et al. Chembl: a large-scale bioactivity database for drug discovery. *Nucleic acids research*, 40(D1):D1100–D1107, 2012.

[9] Andrew Hard, Kanishka Rao, Rajiv Mathews, Swaroop Ramaswamy, Françoise Beaufays, Sean Augenstein, Hubert Eichner, Chloé Kiddon, and Daniel Ramage. Federated learning for mobile keyboard prediction. *arXiv preprint arXiv:1811.03604*, 2018.

[10] Wouter Heyndrickx, Lewis Mervin, Tobias Morawietz, Noé Sturm, Lukas Friedrich, Adam Zalewski, Anastasia Pentina, Lina Humbeck, Martijn Oldenhof, Ritsuya Niwayama, et al. Melloddy: cross pharma federated learning at unprecedented scale unlocks benefits in qsar without compromising proprietary information. 2022.

[11] Xinting Hu, Yulei Niu, Chunyan Miao, Xian-Sheng Hua, and Hanwang Zhang. On non-random missing labels in semi-supervised learning. *arXiv preprint arXiv:2206.14923*, 2022.

[12] Michael Kearns. Learning boolean formulae or finite automata is as hard as factoring. *Technical Report TR-14-88 Harvard University Aikem Computation Laboratory*, 1988.

[13] Michael Kearns and Leslie Valiant. Cryptographic limitations on learning boolean formulae and finite automata. *Journal of the ACM (JACM)*, 41(1):67–95, 1994.

[14] Daliang Li and Junpu Wang. Fedmd: Heterogenous federated learning via model distillation. *arXiv preprint arXiv:1910.03581*, 2019.

[15] Li Li, Yuxi Fan, Mike Tse, and Kuo-Yi Lin. A review of applications in federated learning. *Computers & Industrial Engineering*, 149:106854, 2020.

[16] Qinbin Li, Zeyi Wen, and Bingsheng He. Practical federated gradient boosting decision trees. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 4642–4649, 2020.

[17] Shikun Liu, Edward Johns, and Andrew J Davison. End-to-end multi-task learning with attention. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 1871–1880, 2019.

[18] Othmane Marfoq, Giovanni Neglia, Aurélien Bellet, Laetitia Kameni, and Richard Vidal. Federated multi-task learning under a mixture of distributions. *Advances in Neural Information Processing Systems*, 34:15434–15447, 2021.

[19] Medium. A quick guide to boosting in ml. `https://medium.com/greyatom/a-quick-guide-to-boosting-in-ml-acf7c1585cb5)`.

[20] Kanica Sachdev and Manoj Kumar Gupta. A comprehensive review of feature based methods for drug target interaction prediction. *Journal of biomedical informatics*, 93: 103159, 2019.

[21] Robert E Schapire. The strength of weak learnability. *Machine learning*, 5(2):197–227, 1990.

[22] Robert E Schapire. A brief introduction to boosting. In *Ijcai*, volume 99, pages 1401–1406. Citeseer, 1999.

[23] Holger Schwenk and Yoshua Bengio. Boosting neural networks. *Neural computation*, 12(8):1869–1887, 2000.

[24] Virginia Smith, Chao-Kai Chiang, Maziar Sanjabi, and Ameet S Talwalkar. Federated multi-task learning. *Advances in neural information processing systems*, 30, 2017.

[25] SparseChem code base. melloddy. `https://github.com/melloddy/SparseChem`.

[26] Jiangming Sun, Nina Jeliazkova, Vladimir Chupakhin, Jose-Felipe Golib-Dzib, Ola Engkvist, Lars Carlsson, Jörg Wegner, Hugo Ceulemans, Ivan Georgiev, Vedrin Jeliazkov, et al. Excape-db: an integrated large scale dataset facilitating big data analysis in chemogenomics. *Journal of cheminformatics*, 9(1):1–9, 2017.

[27] Vladimir Svetnik, Ting Wang, Christopher Tong, Andy Liaw, Robert P Sheridan, and Qinghua Song. Boosting: An ensemble learning tool for compound classification and qsar modeling. *Journal of chemical information and modeling*, 45(3):786–799, 2005.

[28] Dániel Sándor. Federated learning in multitask drug-target interaction prediction, 01 2021.

[29] Nihad AM Tamimi and Peter Ellis. Drug development: from concept to marketing! *Nephron Clinical Practice*, 113(3):c125–c131, 2009.

[30] Jing Tang, Agnieszka Szwajda, Sushil Shakyawar, Tao Xu, Petteri Hintsanen, Krister Wennerberg, and Tero Aittokallio. Making sense of large-scale kinase inhibitor bioactivity data sets: a comparative and integrative analysis. *Journal of Chemical Information and Modeling*, 54(3):735–743, 2014.

[31] Kim-Han Thung and Chong-Yaw Wee. A brief review on multi-task learning. *Multimedia Tools and Applications*, 77(22):29705–29725, 2018.

[32] Towards Data Science. Boosting algorithms explained. `https://towardsdatascience.com/boosting-algorithms-explained-d38f56ef3f30`, .

[33] Towards Data Science. All you need to know about gradient boosting algorithm. `https://towardsdatascience.com/all-you-need-to-know-about-gradient-boosting-algorithm-part-1-regression-2520a34a5`.

[34] Leslie G Valiant. A theory of the learnable. *Communications of the ACM*, 27(11): 1134–1142, 1984.

[35] Jessica Vamathevan, Dominic Clark, Paul Czodrowski, Ian Dunham, Edgardo Ferran, George Lee, Bin Li, Anant Madabhushi, Parantu Shah, Michaela Spitzer, et al. Applications of machine learning in drug discovery and development. *Nature reviews Drug discovery*, 18(6):463–477, 2019.

[36] Boyu Wang and Joelle Pineau. Online boosting algorithms for anytime transfer and multitask learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 29, 2015.

[37] Jie Xu, Benjamin S Glicksberg, Chang Su, Peter Walker, Jiang Bian, and Fei Wang. Federated learning for healthcare informatics. *Journal of Healthcare Informatics Research*, 5(1):1–19, 2021.

[38] Yuting Xu, Junshui Ma, Andy Liaw, Robert P Sheridan, and Vladimir Svetnik. Demystifying multitask deep neural networks for quantitative structure–activity relationships. *Journal of chemical information and modeling*, 57(10):2490–2504, 2017.

[39] Wen Zhang, Yanlin Chen, and Dingfang Li. Drug-target interaction prediction through label propagation with linear neighborhood information. *Molecules*, 22(12): 2056, 2017.

[40] Yu Zhang and Qiang Yang. A survey on multi-task learning. *IEEE Transactions on Knowledge and Data Engineering*, 2021.

[41] Yu Zhang and Dit-Yan Yeung. Multi-task boosting by exploiting task relationships. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 697–710. Springer, 2012.

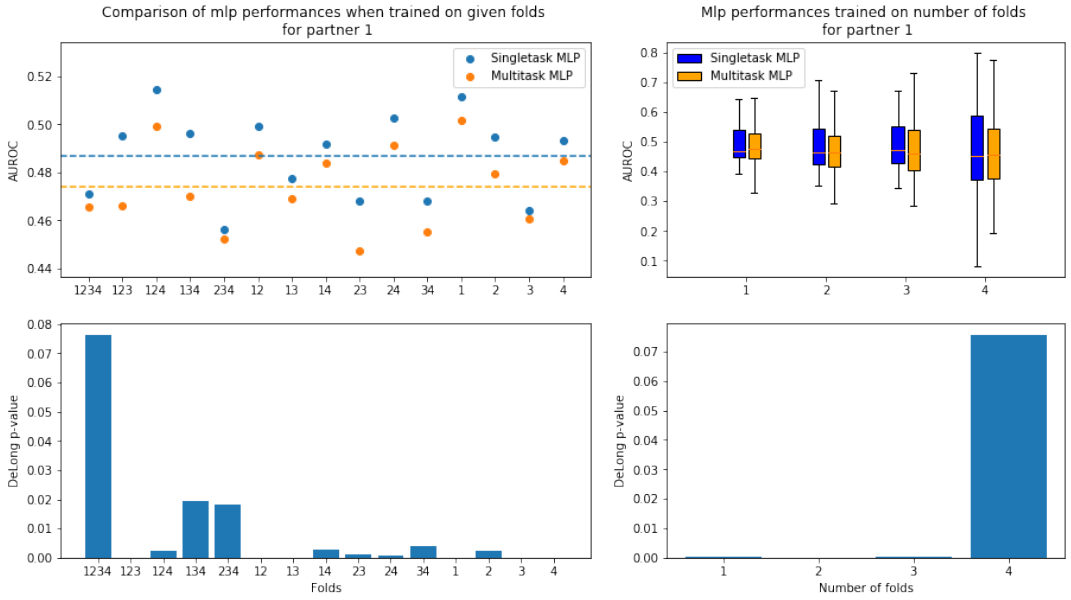
[42] Zhendong Zhang and Cheolkon Jung. Gbdt-mo: gradient-boosted decision trees for multiple outputs. *IEEE transactions on neural networks and learning systems*, 32(7): 3156–3167, 2020.

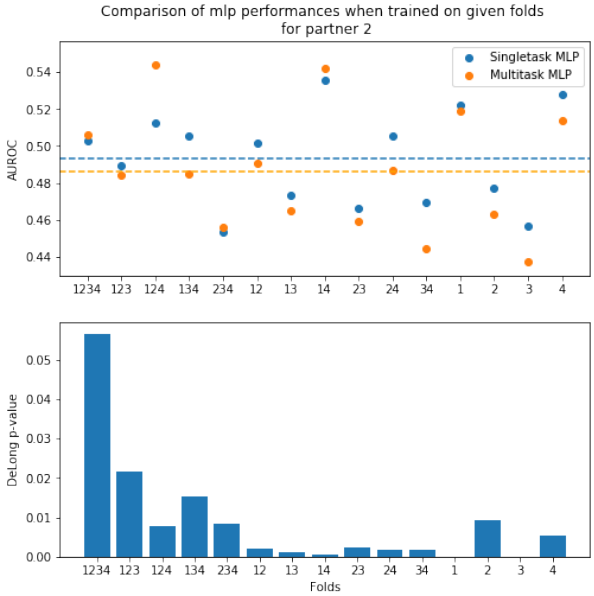
[43] Zhi-Hua Zhou. *Ensemble methods: foundations and algorithms*. CRC press, 2012.

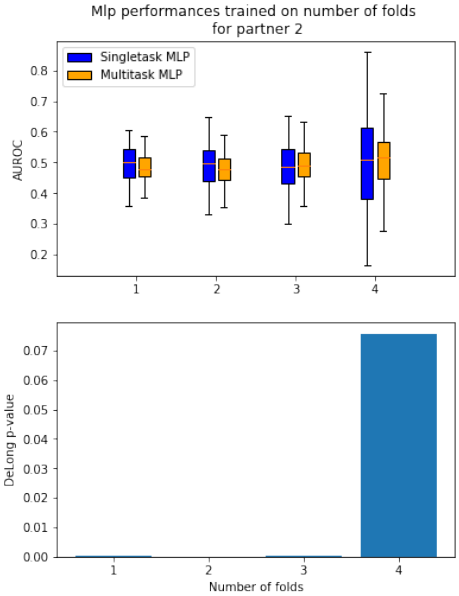
Appendix

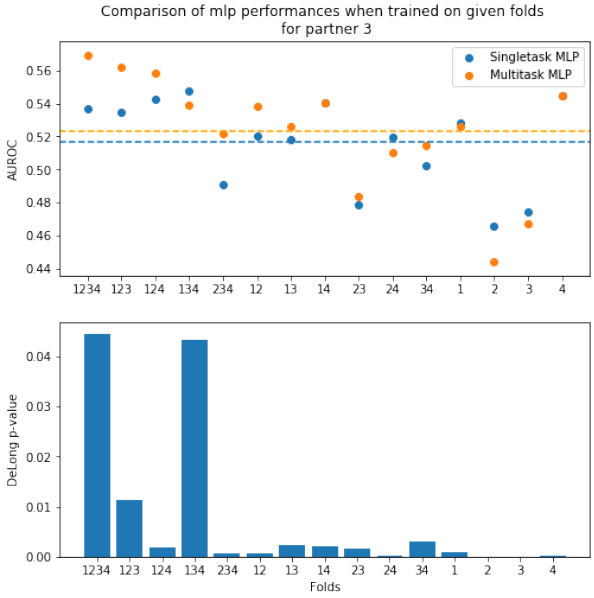
A.1 Partnerwise results of all seven clients

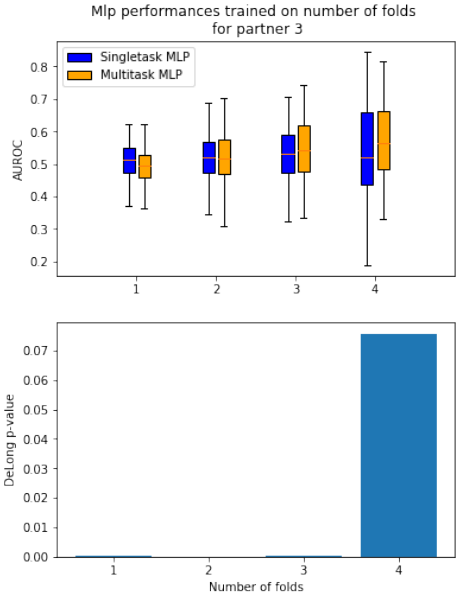
AUROC scores averaged over tasks in MLP-based training for partners shown by folds used for training with the average scores indicated by the dashed lines (left) and boxplot of performances on tasks when averaged over fold permutations (right) with their respective DeLong p-values (bottom):

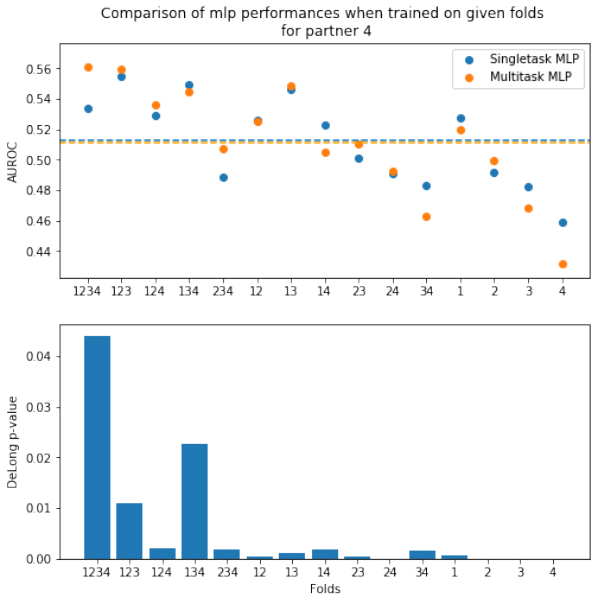


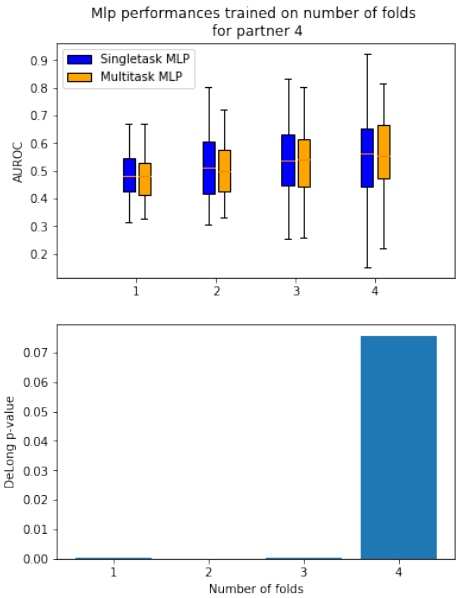

Comparison of mlp performances when trained on given folds
for partner 2
Singletask MLP
Multitask MLP
AUROC
DeLong p-value
Folds
1234 123 124 134 234 12 13 14 23 24 34 1 2 3 4

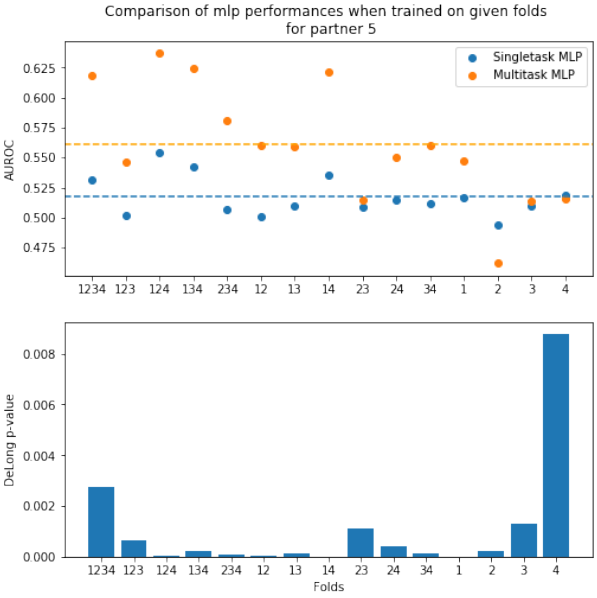

Mlp performances trained on number of folds
for partner 2
Singletask MLP
Multitask MLP
AUROC
DeLong p-value
Number of folds
1 2 3 4

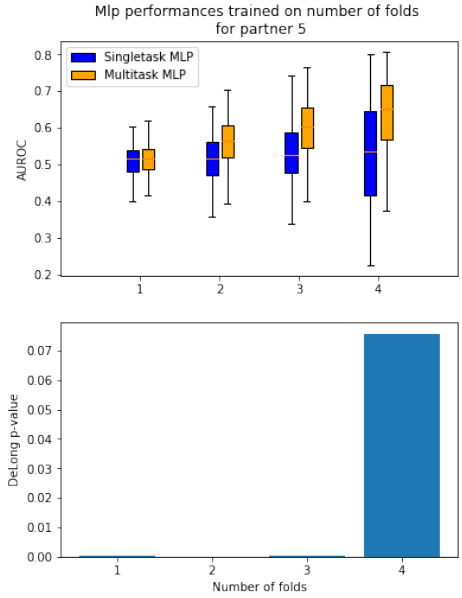

Comparison of mlp performances when trained on given folds
for partner 3
Singletask MLP
Multitask MLP
AUROC
DeLong p-value
Folds
1234 123 124 134 234 12 13 14 23 24 34 1 2 3 4

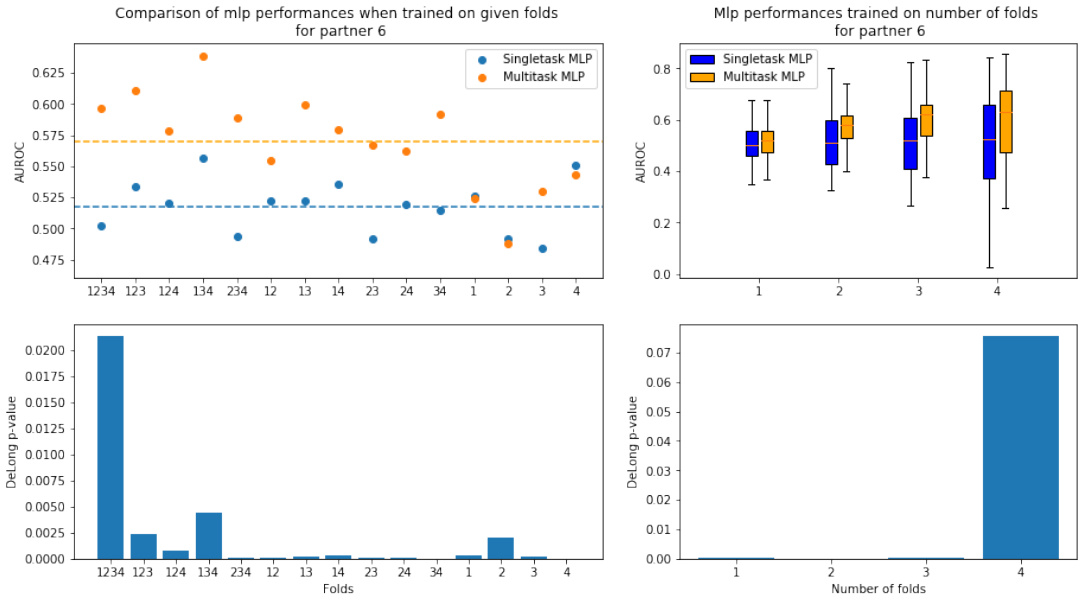

Mlp performances trained on number of folds
for partner 3
Singletask MLP
Multitask MLP
AUROC
DeLong p-value
Number of folds
1 2 3 4

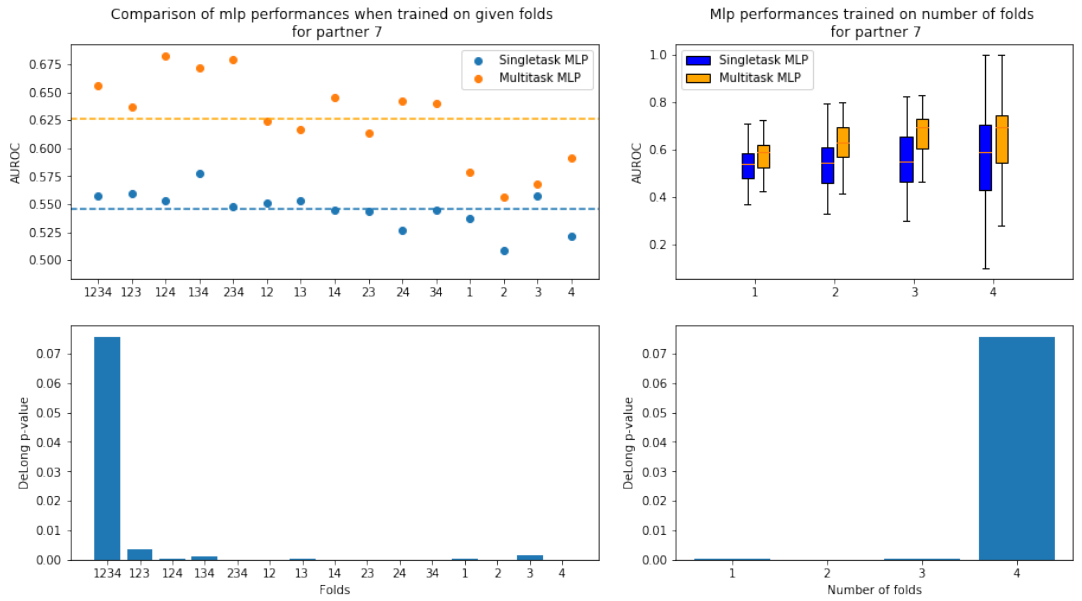

Comparison of mlp performances when trained on given folds
for partner 4
Singletask MLP
Multitask MLP
AUROC
DeLong p-value
Folds
1234 123 124 134 234 12 13 14 23 24 34 1 2 3 4


Mlp performances trained on number of folds
for partner 4
Singletask MLP
Multitask MLP
AUROC
DeLong p-value
Number of folds
1 2 3 4


Comparison of mlp performances when trained on given folds
for partner 5
Singletask MLP
Multitask MLP
AUROC
DeLong p-value
Folds
1234 123 124 134 234 12 13 14 23 24 34 1 2 3 4

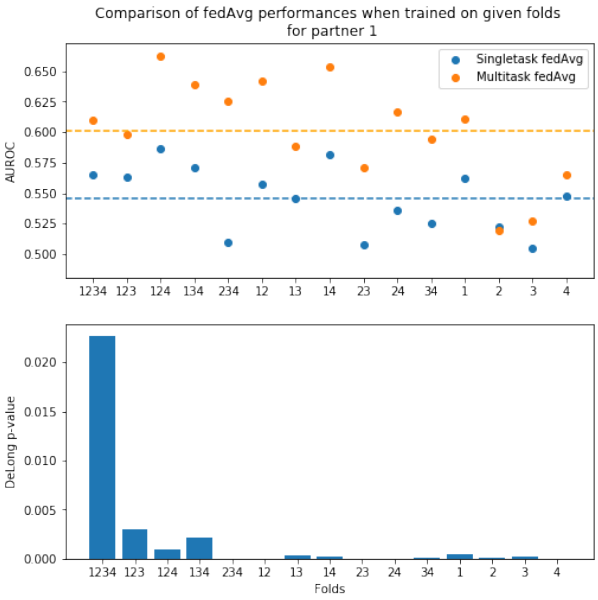

Mlp performances trained on number of folds
for partner 5
Singletask MLP
Multitask MLP
AUROC
DeLong p-value
Number of folds
1 2 3 4

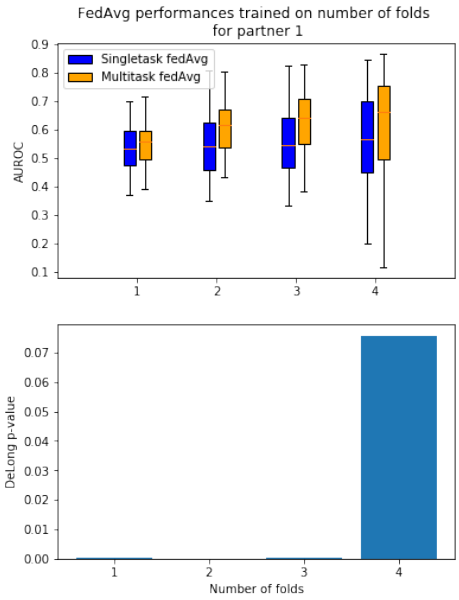


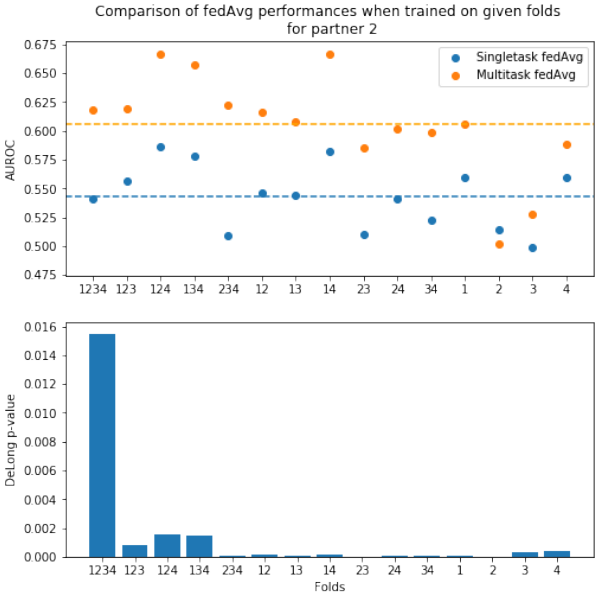


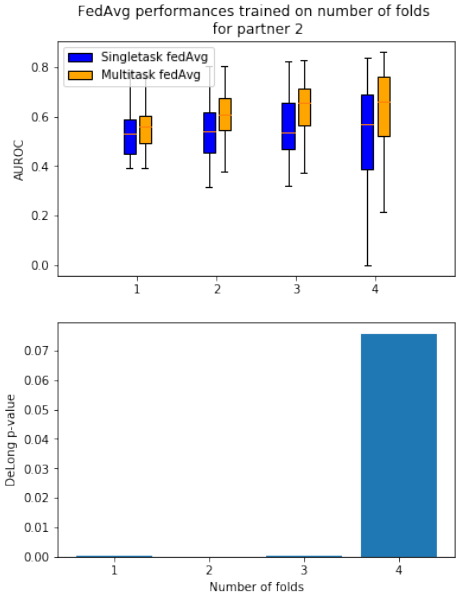
A.2 FedAvg results of all seven clients

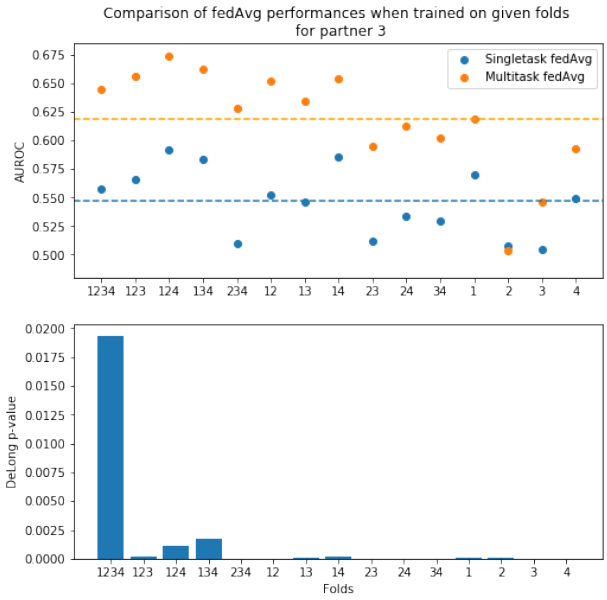
AUROC scores averaged over tasks in FedAvg-based training for partners shown by folds used for training with the average scores indicated by the dashed lines (left) and boxplot of performances on tasks when averaged over fold permutations (right) with their respective DeLong p-values (bottom):

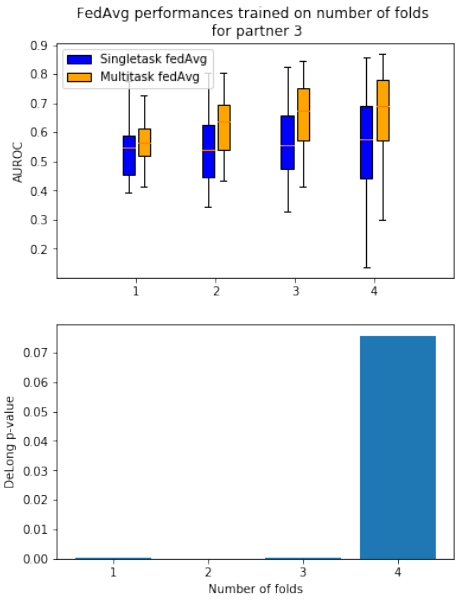

Comparison of fedAvg performances when trained on given folds
for partner 1
Singletask fedAvg
Multitask fedAvg
AUROC
DeLong p-value
Folds
1234 123 124 134 234 12 13 14 23 24 34 1 2 3 4

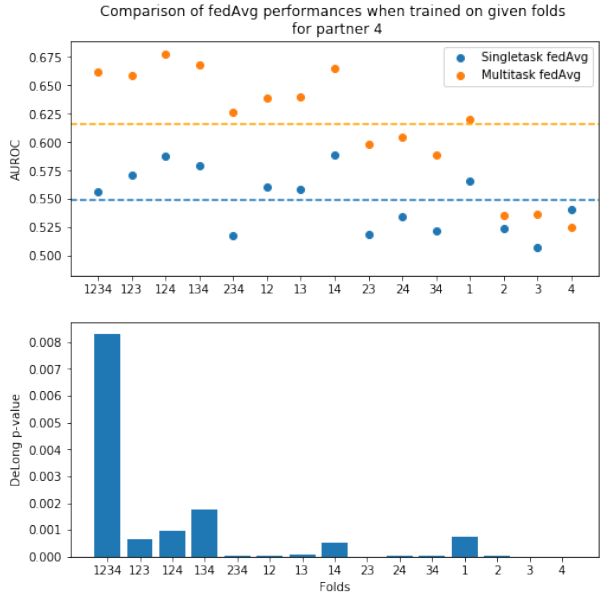

FedAvg performances trained on number of folds
for partner 1
Singletask fedAvg
Multitask fedAvg
AUROC
DeLong p-value
Number of folds

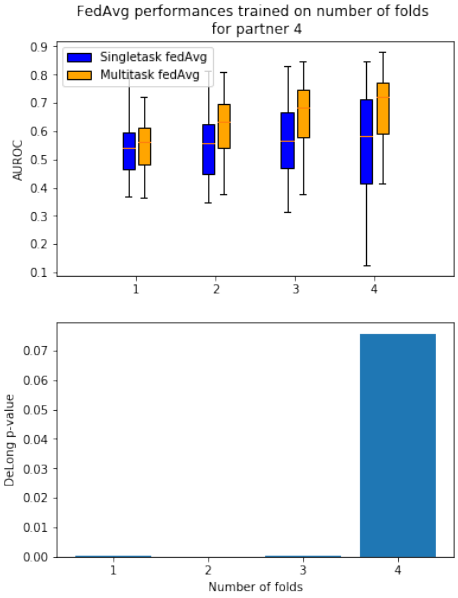

Comparison of fedAvg performances when trained on given folds
for partner 2
Singletask fedAvg
Multitask fedAvg
AUROC
DeLong p-value
Folds
1234 123 124 134 234 12 13 14 23 24 34 1 2 3 4

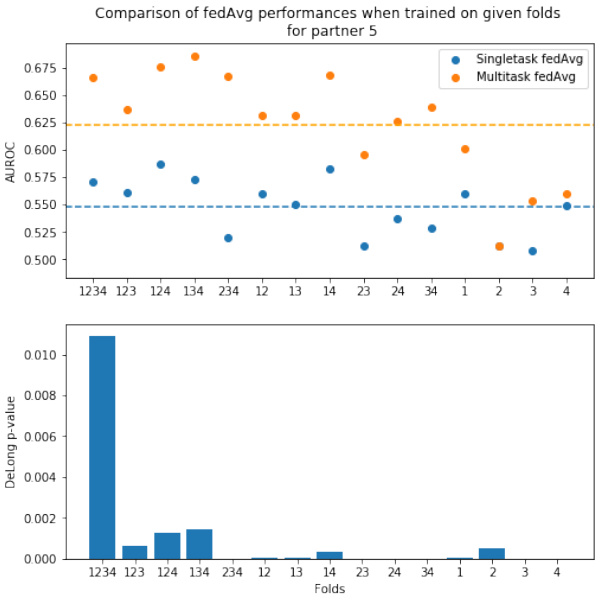

FedAvg performances trained on number of folds
for partner 2
Singletask fedAvg
Multitask fedAvg
AUROC
DeLong p-value
Number of folds

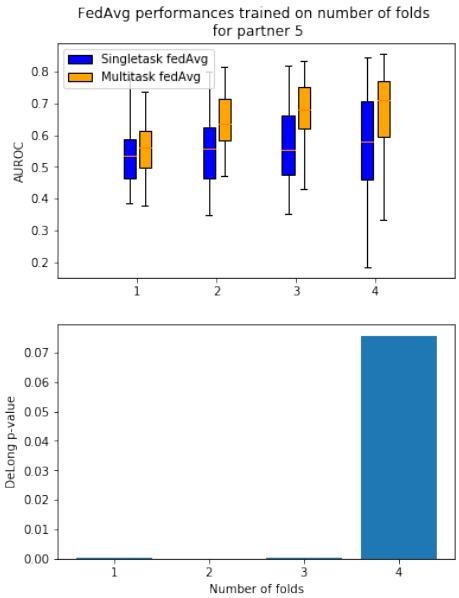

Comparison of fedAvg performances when trained on given folds
for partner 3
Singletask fedAvg
Multitask fedAvg
AUROC
DeLong p-value
Folds
1234 123 124 134 234 12 13 14 23 24 34 1 2 3 4

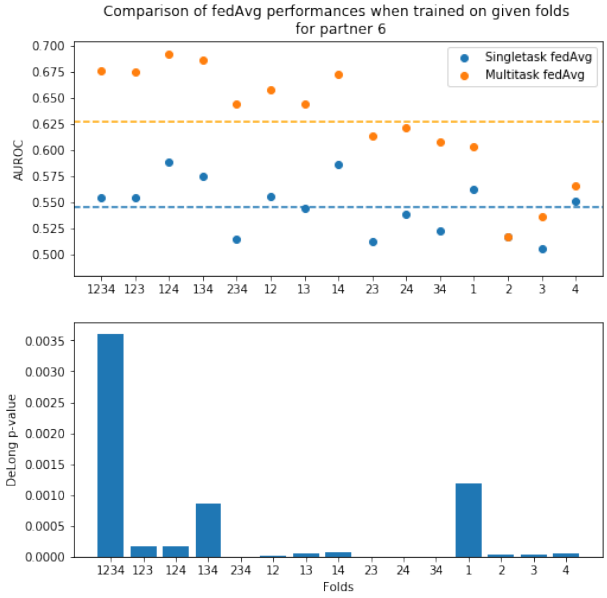

FedAvg performances trained on number of folds
for partner 3
Singletask fedAvg
Multitask fedAvg
AUROC
DeLong p-value
Number of folds
1 2 3 4

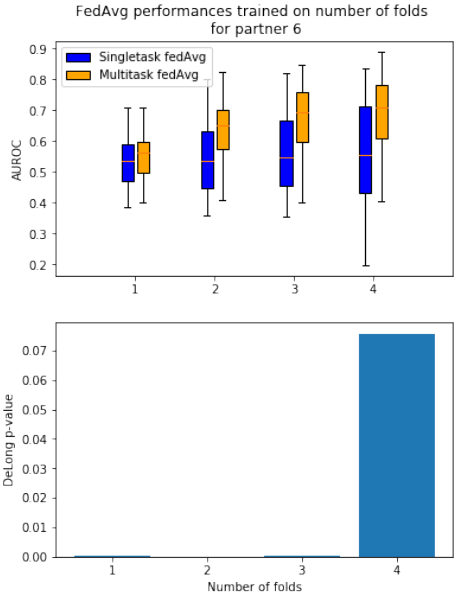

Comparison of fedAvg performances when trained on given folds
for partner 4
Singletask fedAvg
Multitask fedAvg
AUROC
DeLong p-value
Folds
1234 123 124 134 234 12 13 14 23 24 34 1 2 3 4

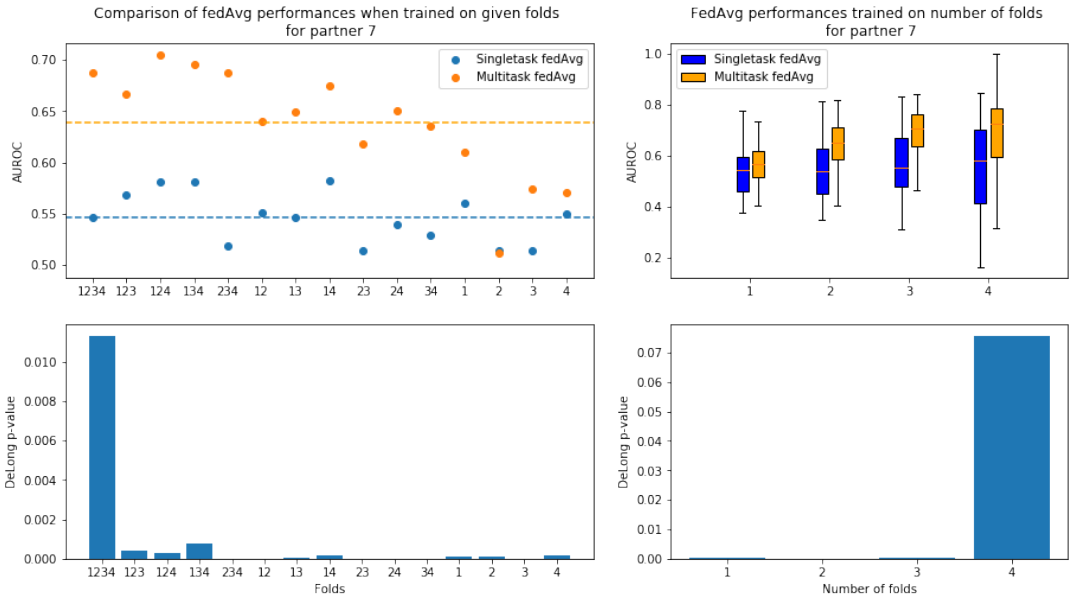

FedAvg performances trained on number of folds
for partner 4
Singletask fedAvg
Multitask fedAvg
AUROC
DeLong p-value
Number of folds
1 2 3 4


Comparison of fedAvg performances when trained on given folds
for partner 5
Singletask fedAvg
Multitask fedAvg
AUROC
DeLong p-value
Folds
1234 123 124 134 234 12 13 14 23 24 34 1 2 3 4


FedAvg performances trained on number of folds
for partner 5
Singletask fedAvg
Multitask fedAvg
AUROC
DeLong p-value
Number of folds

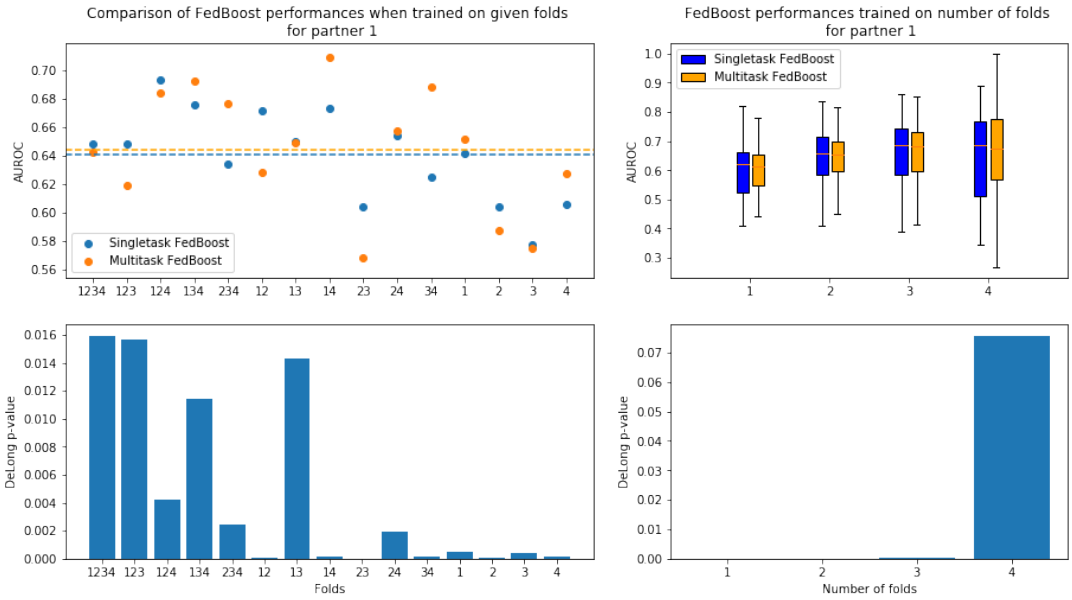

Comparison of fedAvg performances when trained on given folds
for partner 6
Singletask fedAvg
Multitask fedAvg
AUROC
DeLong p-value
Folds
1234 123 124 134 234 12 13 14 23 24 34 1 2 3 4

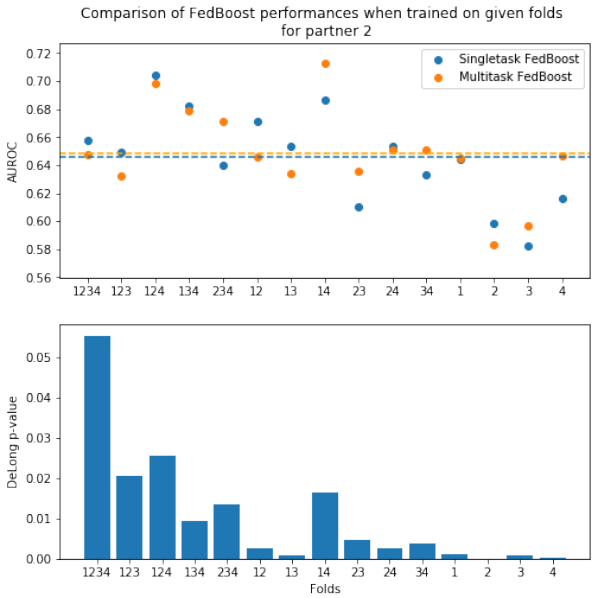

FedAvg performances trained on number of folds
for partner 6
Singletask fedAvg
Multitask fedAvg
AUROC
DeLong p-value
Number of folds

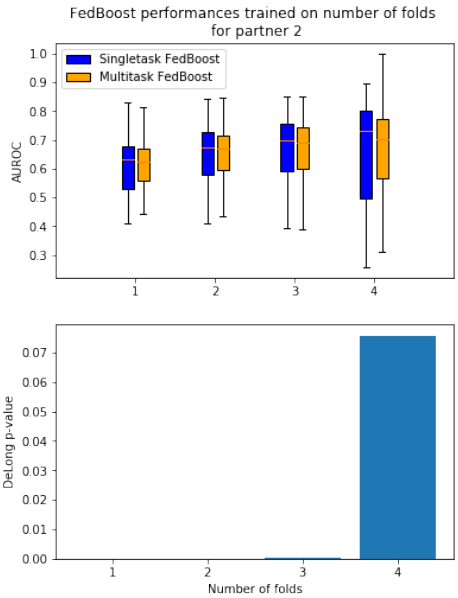


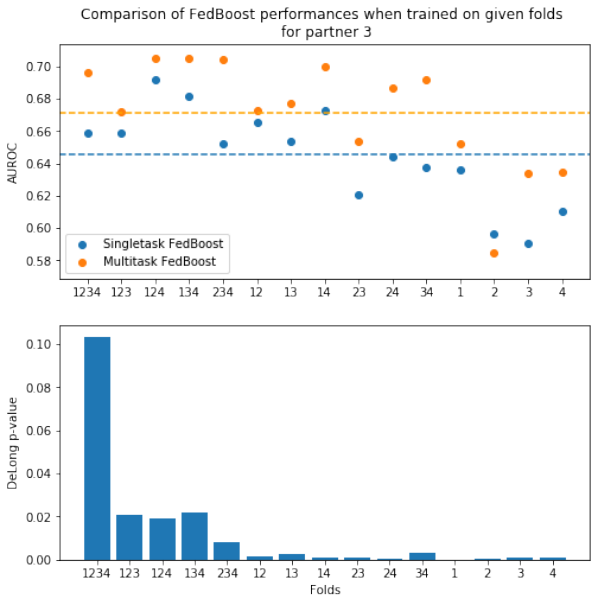
A.3 FedBoost results of all seven clients

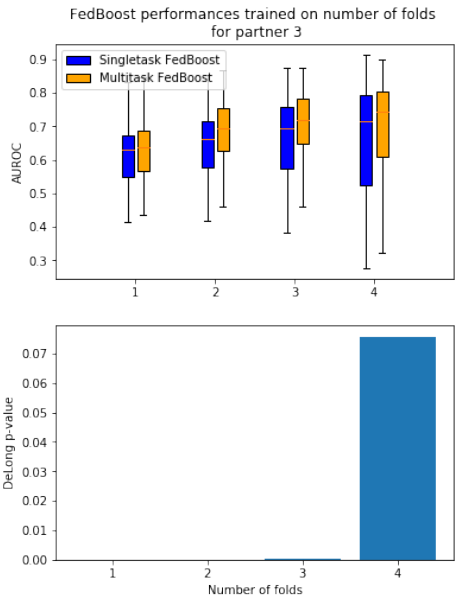
AUROC scores averaged over tasks in FedAvg-based training for partners shown by folds used for training with the average scores indicated by the dashed lines (left) and boxplot of performances on tasks when averaged over fold permutations (right) with their respective DeLong p-values (bottom):

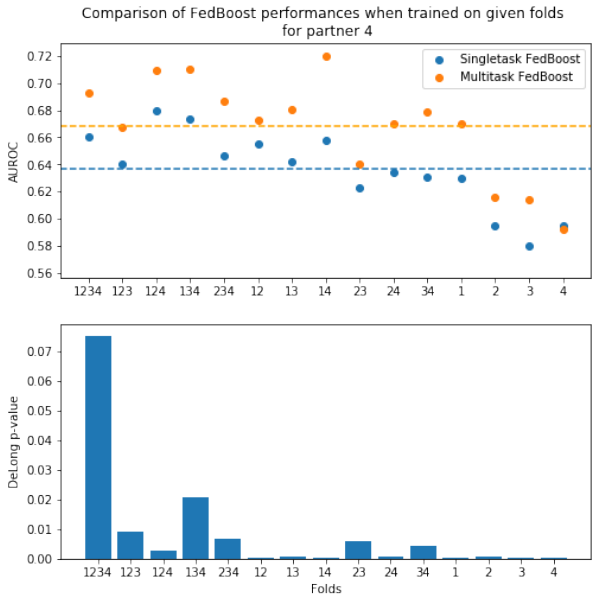


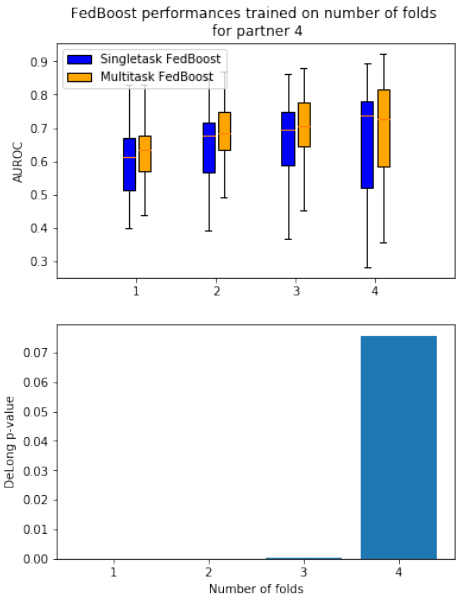

Comparison of FedBoost performances when trained on given folds for partner 2
Singletask FedBoost
Multitask FedBoost
AUROC
DeLong p-value
Folds
1234 123 124 134 234 12 13 14 23 24 34 1 2 3 4

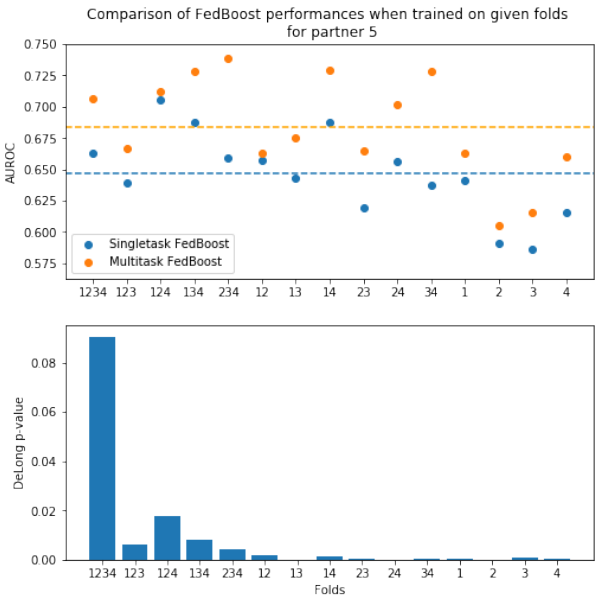

FedBoost performances trained on number of folds for partner 2
Singletask FedBoost
Multitask FedBoost
AUROC
DeLong p-value
Number of folds

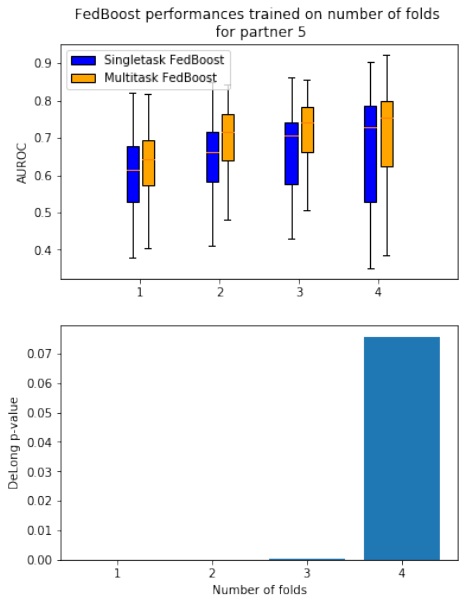

Comparison of FedBoost performances when trained on given folds for partner 3
Singletask FedBoost
Multitask FedBoost
AUROC
DeLong p-value
Folds
1234 123 124 134 234 12 13 14 23 24 34 1 2 3 4

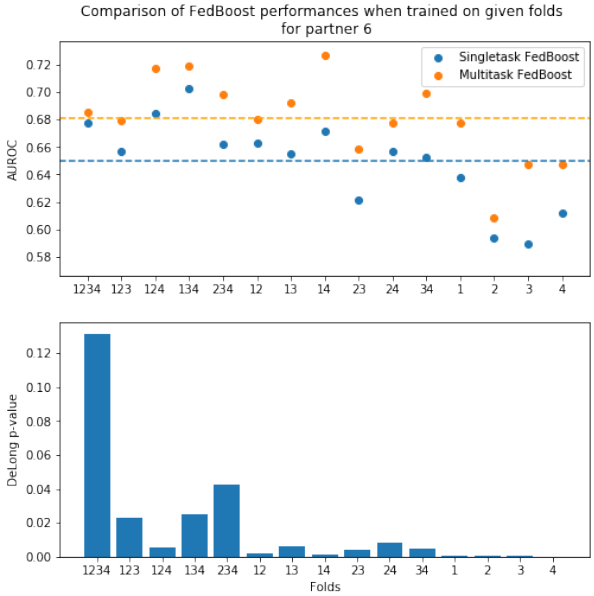

FedBoost performances trained on number of folds for partner 3
Singletask FedBoost
Multitask FedBoost
AUROC
DeLong p-value
Number of folds

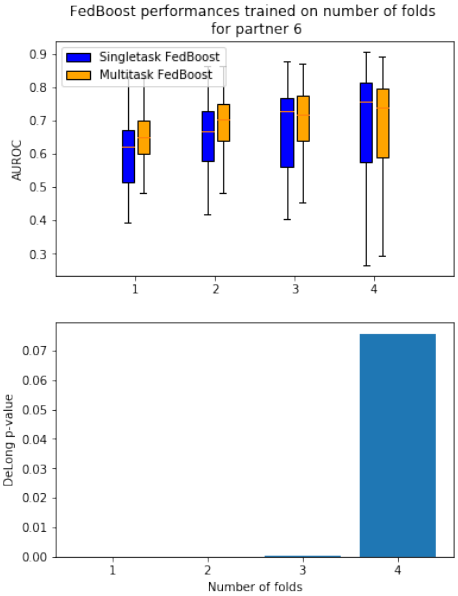

Comparison of FedBoost performances when trained on given folds for partner 4
Singletask FedBoost
Multitask FedBoost
AUROC
DeLong p-value
Folds
1234 123 124 134 234 12 13 14 23 24 34 1 2 3 4

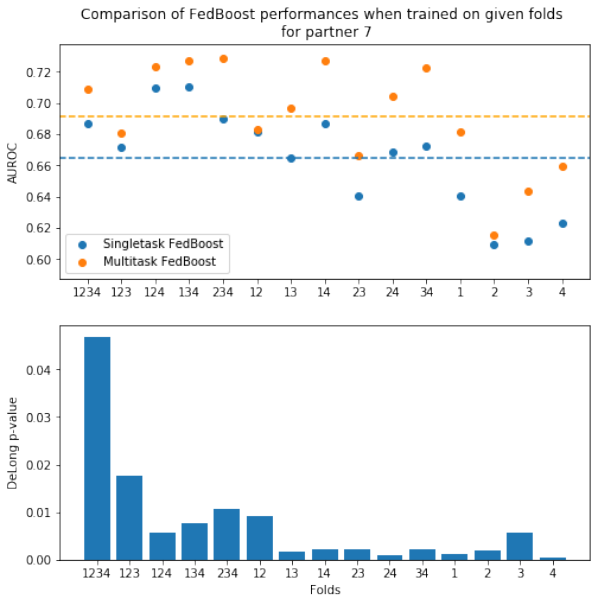

FedBoost performances trained on number of folds for partner 4
Singletask FedBoost
Multitask FedBoost
AUROC
DeLong p-value
Number of folds

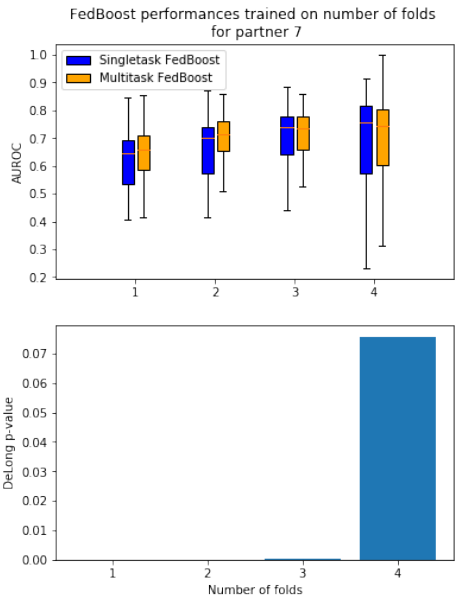

Comparison of FedBoost performances when trained on given folds for partner 5
Singletask FedBoost
Multitask FedBoost
AUROC
DeLong p-value
Folds
1234 123 124 134 234 12 13 14 23 24 34 1 2 3 4


FedBoost performances trained on number of folds for partner 5
Singletask FedBoost
Multitask FedBoost
AUROC
DeLong p-value
Number of folds


Comparison of FedBoost performances when trained on given folds for partner 6
Singletask FedBoost
Multitask FedBoost
AUROC
DeLong p-value
Folds
1234 123 124 134 234 12 13 14 23 24 34 1 2 3 4


FedBoost performances trained on number of folds for partner 6
Singletask FedBoost
Multitask FedBoost
AUROC
DeLong p-value
Number of folds


Comparison of FedBoost performances when trained on given folds for partner 7
Singletask FedBoost
Multitask FedBoost
AUROC
DeLong p-value
Folds
1234 123 124 134 234 12 13 14 23 24 34 1 2 3 4


FedBoost performances trained on number of folds for partner 7
Singletask FedBoost
Multitask FedBoost
AUROC
DeLong p-value
Number of folds