ELIX

# Federated Learning

Report v2.1

October 20, 2021

ELIX

# Report v2.1

October 20, 2021

## Federated Learning

*In this joined project between Elix Inc. and Kyoto University, we implement a small communication module called "Mila" for federated learning, following the implementations of NVIDIA Clara (as close as possible). To validate the results, we analyze the behavior of various datasets and architectures during the distributed learning process.*

# Contents

# Theoretical Background

## 1.1 Federated Learning

Federated learning is a machine learning paradigm focused on distributed training across decentralized machines. As opposed to classic approaches where all the data is in one central location, during federated learning, the data is spread across multiple servers or edge devices. Each device (hereafter called "client") runs training for a number of epochs and sends the weights/checkpoints to a central location (hereafter called "server") which aggregates the information from all clients. The new aggregate model would then be sent to all participating clients, which then repeat the process using the latest model.

This way, each client's data remains private, and is not shared with any of the other clients, nor the server. For companies where data is a sensitive matter (as is the case for big pharmaceutical companies), this is a very attractive approach.

Additional details can be found in the original paper[42].

## 1.2 Datasets

We experiment with a large number of datasets, however, they can be grouped into 3 categories:

- Toxicity datasets
- ADME datasets (absorption, distribution, metabolism, and excretion)
- Protein-Ligand Affinity dataset

An overview of all datasets used can be visualized in Table 1.1.

### *The Tox21 dataset*

The Tox21 dataset contains 7831 entries (SMILES[69] strings) for 12 different toxicological experiments (binary labels) representing activity (active/inactive). It should be noted that the dataset contains missing values, which we ignore in the loss function and during back-propagation. The dataset is very imbalanced, as the number of positive samples range between 186 and 942, depending on the target.

### *The AMES dataset*

We received 3 different versions of the AMES dataset, however, the data structure is similar for all of them. The input format is a SMILES string and the output is a single, binary target.

For the federated learning experiments, we combined the "benchmark" and the "NTP PubChem" subsets, as all of the duplicates, except for one entry, had matching labels. The single entry with a different label was removed ("O=c1c2ccccc2c(=O)c2c1ccc1c2[nH]c2c3c(=O)c4ccccc4c(=O)c3c3[nH]c4c(ccc5c(=O)c6ccccc6c(=O)c54)c3c12"). The S9 Minus dataset was too different from the other 2, and was not merged into the final set.

### *ADME datasets*

The ADME datasets were delivered as a combination of SDF files (structural information) and an Excel file (labels and target values). The excel file also described the molecules in InChI format, but upon further inspection, these did not always match the SDF entries when loaded with RdKit[34]. As a consequence, we matched the molecule IDs between the SDF files ("ID" property) and the excel files ("drumap_id" or "no_salt_drumap_id"), and entries outside the join.

Some of the datasets contain continuous values (regression) while some of them are reduced to binary labels (classification) either directly in the dataset, or manually by us. The final number of entries and the type of task for each dataset can be seen in Table 1.1.

For most datasets, we used the "value", or a numeric format of the "value.class" column directly, but there are a few exceptions. For the intrinsic clearance (CLint) dataset, the "value", "lower_value", and "upper_value" columns were combined into a single target. For the fraction excreted unchanged in urine (fe_human), the continuous "value" column was binarized using a threshold of 0.3, and 0.7 (for federated learning, we used the 0.7 threshold). For the P-gp Net Efflux Ratio (Ner LLC), the "value.class" column was decomposed into 3 classes (multi-class classification).

### *ChEMBL assays dataset*

The Protein-Ligand Affinity dataset is a subset of the ChEMBL dataset and was delivered as an relational database (SQL) wrapped around an command-line interface (CLI). The CLI allows for exports of entries based on protein identifiers, affinity thresholds and metric types, however, we wanted to explore the dataset in more detail and select proteins based on certain criteria, not identifiers. Therefore, we operated directly on the SQL dataset, so updates to the CLI tool should not influence the structure of these datasets.

The input structures are pairs of protein (amino-acid sequences) and ligands (SMILES strings), and the goal is to compute the affinity between them (generally, a binary values and a classification task).

| Name | Category | Samples | Task |
|---|---|---|---|
| Tox21 | Toxicity | 7831 | Classification |
| AMES (Final) | Toxicity | 7441 | Classification |
| AMES (Benchmark)* | Toxicity | 6512 | Classification |
| AMES (NTP PubChem)* | Toxicity | 7403 | Classification |
| AMES (S9 Minus)* | Toxicity | 2876 | Classification |
| Clint Human | ADME | 5216 | Regression |
| Fe Human | ADME | 340 | Classification |
| Fu Brain | ADME | 580 | Regression |
| Fup Human | ADME | 2559 | Regression |
| Fup Rat | ADME | 539 | Regression |
| Ner LLC Human | ADME | 445 | Classification |
| PappCaco2 | ADME | 4294 | Both |
| Papp LLC Human | ADME | 461 | Regression |
| Rb Rat | ADME | 162 | Regression |
| Solubility | ADME | 514 | Classification |
| ChEMBL Assays (v1) | Binding Affinity | 574583 | Classification |
| ChEMBL Assays (v2)* | Binding Affinity | 947586 | Both |

Table 1.1: Datasets used for experiments (subsets marked with [*] were not analyzed through distributed learning)

Initially, we export a subset of this dataset where:

- the "valid" column was equal to 1
- the metric was one of "IC50", "Ki", "EC50", or "Kd"
- and the protein had at least 500 positive samples (where "positive" was classified as a value of 100 nM or less)

For each protein individually, we used the median value when duplicate ligand measures were encountered. This setup resulted in a dataset, referred to as version 1 (v1), containing 574,583 entries spread across 272 unique protein sequences, 390,467 unique ligands (represented as SMILES), and a roughly equal distribution of positive (292,144) and negative (282,439) samples.
After feedback, we created another export with a few slight variations:

- we used the pchembl_value ($-\log_{10}(x)$) instead of the value (affinity) and metric types ("IC50", "Ki", "EC50", or "Kd") when filtering
- the protein filtering was set to a minimum of 100 entries (positive or negative)

Duplicate entries were similarly averaged using the median value. This resulted in a much larger dataset (referred to as version 2, or v2), containing 947,586 entries spread across 1413 unique protein sequences, 583,624 unique ligands. Furthermore, we experimented with 3 different thresholds along with the raw (regression) values. The 3 thresholds suggested were 100 nM (pchembl=7), 1 uM (pchembl=6), and 10 um (pchembl=5), which obviously result in different distributions. The number of positive entries for the 3 splits were 404321, 645186, and 852925 respectively, which in turn meant 543265, 302400, and 94661 negative entries respectively.

The queries used to retrieve the subsets were as follows

```
# SQL query used to extract version 1

SELECT target_accession, target_sequence, standard_type, standard_value, smiles
FROM activities
WHERE standard_type IN ("IC50", "Ki", "EC50", "Kd")
AND valid = 1
AND target_accession IN (
    SELECT target_accession FROM activities
    WHERE valid = 1
```

```
    AND standard_value <= 100
    AND standard_type IN ("IC50", "Ki", "EC50", "Kd")
    GROUP BY target_accession
    HAVING COUNT(*) >= 500
)

# SQL query used to extract version 2 (thresholds were applied programmatically)

SELECT target_accession, target_sequence, standard_type, pchembl_value, smiles
FROM activities
WHERE standard_type IN ("IC50", "Ki", "EC50", "Kd")
AND valid = 1
AND target_accession IN (
    SELECT target_accession FROM activities
    WHERE valid = 1
    AND standard_type IN ("IC50", "Ki", "EC50", "Kd")
    GROUP BY target_accession
    HAVING COUNT(*) >= 100
)
```

One import matter to point out is the distribution of the metrics used in the dataset (column "standard_type"). The number of entries in the dataset for all metric types is listed in Table 1.2.

| Metric | Samples |
|---|---|
| Potency | 1026983 |
| IC50 | 770361 |
| Ki | 379869 |
| EC50 | 109957 |
| AC50 | 58336 |
| Kd | 38456 |
| XC50 | 5 |
| pIC50 | 1 |

Table 1.2: Metric distribution in the ChEMBL dataset

We notice that among the metrics we focused, IC50 and Ki values are abundant, however, there is a "Potency" metric which contains almost 50% of the whole dataset. The "Potency" is generally an inexact metric used when we do not know what the actual measurement is. This can be problematic, because some metrics should be minimized while other metrics should be maximized. Furthermore, looking at the the distribution of samples, we clearly see that entries measured

in "Potency" contain much larger entries, most of which would be classified as negative (although, we stress, this metric is ambiguous). As a consequence, we decided not to include these values in our subsets.

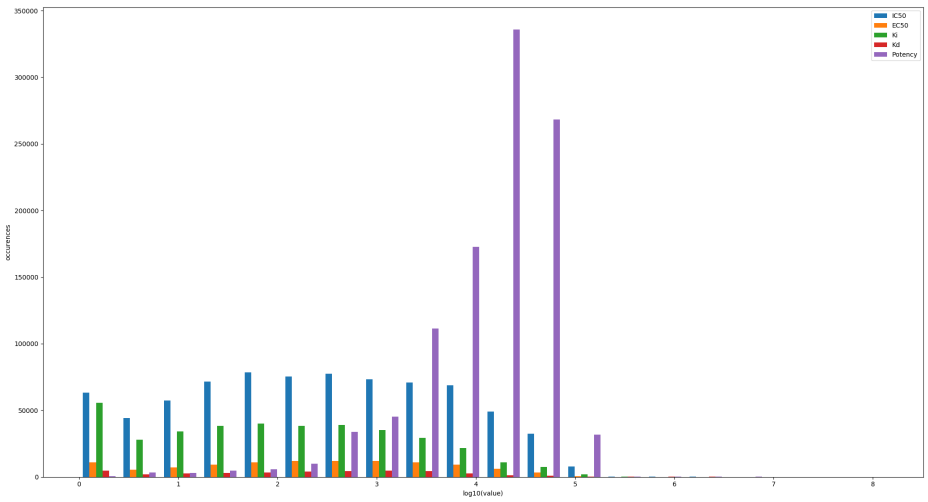


Figure 1.1: Distribution of each metric type

Finally, we should note that these datasets are large and the data formats are big and complex (ie: long amino-acid chains or molecular graph structures). It can take a long time to train a single model on these datasets, and for federated learning, we have to train these models hundreds of times over thousands of epochs. As we only recently started experimenting with version 2, all federated learning experiments were performed on version 1 of the dataset.

### *Additional Datasets*

While no other datasets were used for federated learning experiments, we did test our models and optimization tools on a number of other publicly available ADME-Tox datasets. These datasets were accessed through the Therapeutics Data Commons (TDC) project, which accesses data from the Harward Dataverse.

A list of all datasets we experimented with and their references can be found in Table 1.3. Some of these datasets might be identical to the ones listed under the

"ADME datasets" section. it should also be noted that the Hydration Free Energy dataset (FreeSolv) cannot be used for commercial purposes.

| Name | Category | Task | Samples | References |
|---|---|---|---|---|
| Bioavailability | Absorption | C | 492/148 | [41] |
| Cell Effective Permeability (Caco-2) | Absorption | R | 910 | [66] |
| Human Intestinal Absorption | Absorption | C | 500/78 | [26] |
| Hydration Free Energy | Absorption | R | 642 | [45] |
| Lipophilicity | Absorption | R | 4200 | [73] |
| P-glycoprotein Inhibition | Absorption | C | 651/568 | [9] |
| Solubility | Absorption | R | 9982 | [48] |
| Blood-Brain Barrier | Distribution | C | 1560/479 | [73] |
| Plasma Protein Binding Rate | Distribution | R | 2790 | [70] |
| Volumn of Distribution at steady state | Distribution | R | 1130 | [39] |
| CYP P450 1A2 Inhibition | Metabolism | C | 5829/6750 | [16] |
| CYP P450 2C19 Inhibition | Metabolism | C | 5819/6846 | [16] |
| CYP P450 2C9 Inhibition | Metabolism | C | 4045/8047 | [16] |
| CYP P450 2D6 Inhibition | Metabolism | C | 2514/10616 | [16] |
| CYP P450 3A4 Inhibition | Metabolism | C | 5110/7218 | [16] |
| CYP2C9 Substrate | Metabolism | C | 141/528 | [11] |
| CYP2D6 Substrate | Metabolism | C | 191/476 | [11] |
| CYP3A4 Substrate | Metabolism | C | 355/315 | [11] |
| Clearance | Excretion | R | 1213+1102 | [15] |
| Half Life | Excretion | R | 667 | [49] |
| Acute Toxicity (LD50) | Toxicity | R | 7385 | [76] |
| Carcinogens | Toxicity | C | 60/220 | [33] |
| ClinTox | Toxicity | C | 112/1366 | [20] |
| Drug Induced Liver Injury | Toxicity | C | 236/239 | [75] |
| hERG Blockers | Toxicity | C | 451/204 | [67] |
| Skin Reaction | Toxicity | C | 174/130 | [3] |

Table 1.3: ADME-Tox datasets we experimented with. Tasks are marked as classification (C) or regression (R). For classification tasks we report the positive (left) and the negative samples (right) separately. The "Clearance" dataset contains values for microsomes(1102) and hepatocytes(1213) as well.

## 1.3 Featurization

Molecules are complex structures and they need to be preprocessed and transformed to a machine learning readable format. This process is often referred to as featurization. Different models will support different featurized formats, however, we mainly focus on graph-based models in this project.

We generally use a graph featurization to encode ligands or small-molecules, however, we experimented with circular (Morgan[46]) fingerprints as well. Some of our architectures combine atom-level features with global (molecule) level features. For these features we generally used property descriptor fingerprints computed either with RdKit[34] or Mordred[53].

Amino-acid sequences have been featurized either through one-hot encoding or using an n-gram, bag-of-words approach. We focus on the models in the next section, but to understand the whole pipeline, we briefly introduce the featurizers we employed.

### 1.3.1 Graph Representation

A molecule can be thought of as an undirected graph, where each atom is a node/vertex, and each bond is an edge. Each node in turn can contain several atom related features like: a one-hot encoding of the chemical symbol or the atomic number, a similar encoding of the hybridization, the number of neighbours the atom has (ie: the degree of the node), or the valence. These features would be concatenated into a single array (similar to a fingerprint, for individual atoms) and fed as a matrix into a graph convolutional network (where each row is a separate atom, and each column an atom feature).

Additionally, an adjacency matrix which marks the connections (ie: bonds) between the atoms is also fed to the model. This matrix is symmetric, with both row and columns representing the index of each atom in the feature matrix, and the value of a cell being either "1" if there is a bond between the 2 atoms or "0" otherwise.

In Figure 2.2, we show a possible feature representation of Aspirin. On the left hand-side we have the atom features. Each row represents an atom, and each

column is a feature of that atom (for example, column 3 marks whether the atom is a Carbon, while column 5 marks if it is an a Oxygen atom). On the right hand-side we have the adjacency matrix which marks the bonds between these atoms. For example, we see that the second row atom (which is the core of the carboxyl group) has 3 neighbors (atoms positioned as 1, 3 and 4).

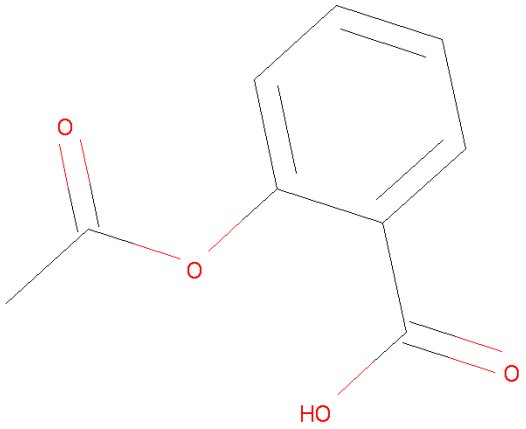


Figure 1.2: 2D structure of Aspirin

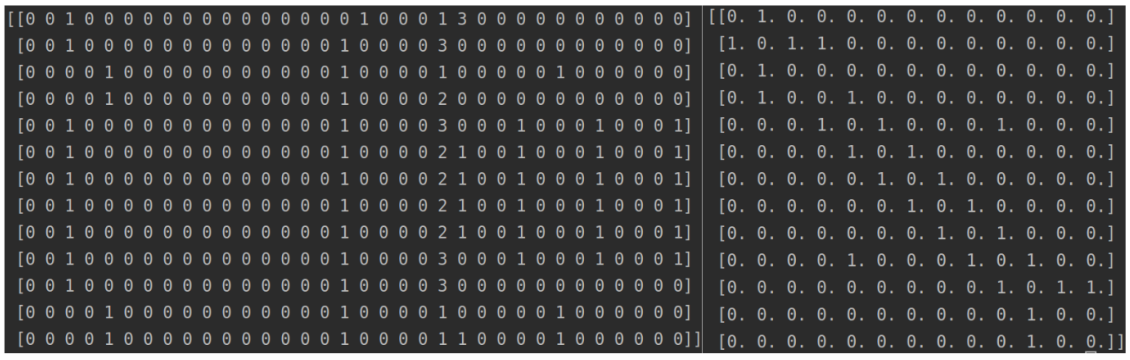


Figure 1.3: A graph representation of Aspirin. The feature matrix (left) encodes properties or descriptors (columns) for each individual atom (row), while the adjacency matrix (right) marks the connections (bonds) between the atoms.

### 1.3.2 Circular Fingerprints (Morgan)

Molecular fingerprints encode the structure of a compound as a series of binary digits (0 or 1), each of which represent the presence or absence of a particular substructure in the molecule. Circular fingerprints do this by considering substructures of atoms and their neighbours which are within a certain radius of the said atoms.

Morgan fingerprints[46] are similar to Extended-Connectivity Fingerprints (ECFPs)[56] in this sense. ECFPs however are a proprietary solution of Pipeline Pilot, and the hashing function is not publicly known. Morgan fingerprints on the other hand are implemented in RdKit[34], which is the software we used in generating them.

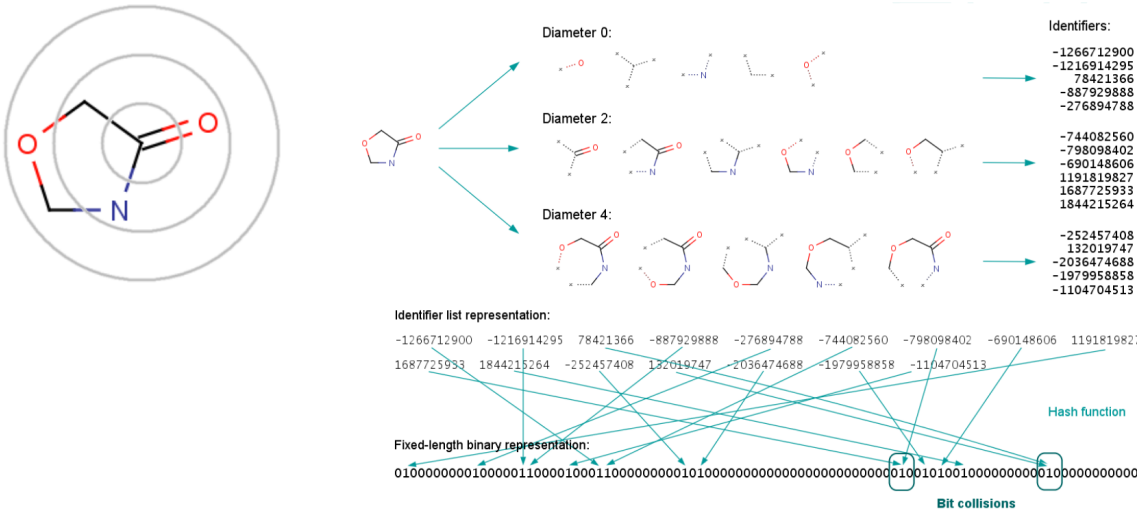


Figure 1.4: Circular Fingerprints. Substructures are created based on a preset radius and mapped to an identifier. A hash function is applied on top of the identifiers which map them to a certain position on the fingerprint vector. As the fingerprint size is fixed, collisions can occur (image source: chemaxon docs)

### 1.3.3 Property Descriptor Fingerprints

Similar to circular fingerprints, a list of properties of the given molecule can be computed an summarized in an array, which then can be easily fed to a model.

### *RdKit Descriptors*

RdKit[34] provides implementations of many such properties. We concatenated 17 such of them to form our rdkit fingerprints:

- Descriptors.MolWt (the molecular weight)
- Descriptors.NumRadicalElectrons
- Descriptors.NumValenceElectrons
- rdMolDescriptors.CalcTPSA (the topological polar surface area (TPSA))
- MolSurf.LabuteASA (Labute's Approximate Surface Area[32])
- GraphDescriptors.BalabanJ (Balaban's J index[4])
- Lipinski.RingCount
- Lipinski.NumAliphaticRings
- Lipinski.NumSaturatedRings
- Lipinski.NumRotatableBonds
- Lipinski.NumHeteroatoms
- Lipinski.HeavyAtomCount
- Lipinski.NumHDonors[38]
- Lipinski.NumHAcceptors[38]
- Lipinski.NumAromaticRings
- Crippen.MolLogP (the partition coefficient)
- QED.qed (the quantitative estimate of drug-likeness[8])

### *Mordred Descriptors*

We also experimented with Mordred Descriptors[53], which provides a list of 1826 molecular descriptors.

### 1.3.4 Sequence Featurizers

Amino-acid sequences are unlike small molecules represented as SMILES, and need different featurizers.

#### *Tokenization*

One approach is to use a one-hot encoding representation of each amino-acid, and concatenate these encodings into a matrix of shape (number of tokens, maximum sequence length). The number of tokens, of course, is equal to 21 (the number of distinct amino-acids we can have), and the maximum sequence length is the length of the longest amino-acid chain from the dataset. Shorter chains than the maximum entry are be 0-padded.

#### *Bag-of-words Encoding*

Bag-of-words approaches are generally applied to text data. Each sentence is split and the occurrence of each word is counted, thus a dictionary of all words present in the dataset is created. Although we are not dealing with words and sentences, our amino-acid sequences can be split by length. As an example, if we have a DNA sequence ATTGATTCG, we could extract all subsequences of length 6: ATTGAT, TTGATT, TGATTC, and GATTCG.

Normally, we would enumerate all possible substrings up to a certain length, count the number of times they occur, and use these counts to form a fingerprint. With DNA sequences, our vocabulary has a length of 4 (A, C, T, G). The unigram would also contain 4 options (A, C, T, G), but the bigram increases in size exponentially (AA, AC, AT, AG, CA, CC, CT, etc.), so there is a limit to the size of the substrings we are able to consider. For our vocabulary of 21 amino-acids, we went up to a length of 3, which resulted in a fingerprint of length 9723.

## 1.4 Models

As mentioned before, our work mainly focuses on graph-based models, however, certain datasets require additional strategies. For binding affinity prediction, we need to consider models which can handle sequence (amino-acid) information.

### 1.4.1 Graph Architectures

There are various graph-based architectures, but almost all of them rely on the base concept which we visualize in Figure 1.5.

As a first step, the input molecules are featurized and converted to a graph representation. These features are then past through a number of graph convolutional layers, which, in a sense, are very similar to classic convolutional layers where fixed size filters are applied in a sliding window fashion across the whole image, enabling the layer to learn features from neighboring pixels. Graph layers are doing the same thing, aggregating information from neighbouring nodes; the main difference is that they work on non-euclidean data formats. The graph layer is applied to each node (here atom), and the more layers are added, the further away information will be brought towards the central node (atom).

After a number of such layers are applied, generally, a graph pooling layer is used, which works very similar to a regular pooling layer, in that it summarizes and shrinks the feature space. These features are then flattened and passed through a shallow neural network.

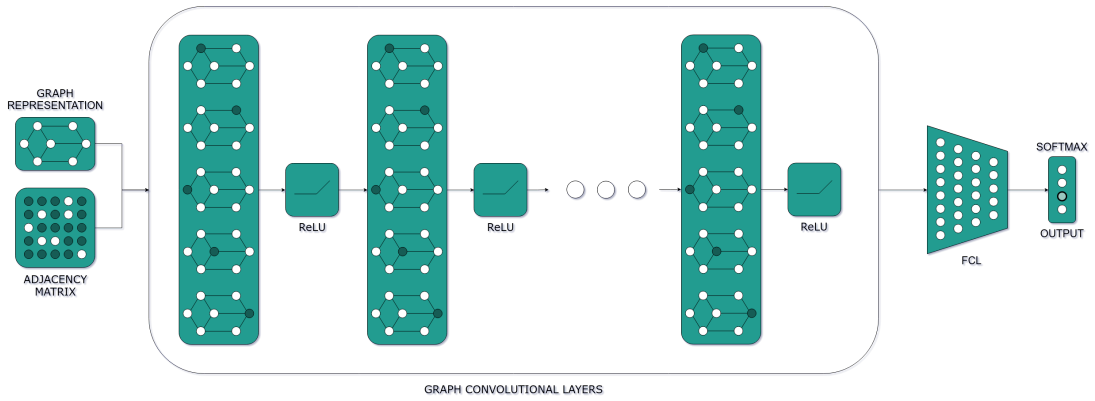


Figure 1.5: Graph convolutional networks aggregate information from neighbouring atoms. As more layers are added, further away information is brought towards the central node. In the end, the node features are downsized using a graph pooling layer, flattened, and passed through a final Multilayer Perceptron (MLP) block.

Different arhitectures perform the aggregation in different ways, which can be as simple as multiplying the input features with the set of training weights and the adjacency matrix. Going through the details of all available graph architectures is beyond the scope of this report, however, we refer the readers to Table 1.4, which lists all architectures we tried and their reference. As a note, our implementation is built upon Pytorch Geometric[19], which provides sparse implementations for many graph convolutional kernels.

| **Name** | **Reference** |
|---|---|
| Classic Graph Network (Kipf & Welling) | [61] |
| Message Passing Network | [21] |
| Chebyshev Spectral Network | [13] |
| GraphSAGE Network | [22] |
| Weisfeiler-Leman Graph Network | [47] |
| Graph Isomorphism Network | [74] |
| ARMA Graph Network | [7] |
| Local Extremum Graph Network | [54] |
| GENeralized Graph Network | [35] |
| Cluster Graph Network | [12] |
| Feature-Steered Graph Netowrk | [64] |
| Graph Attention Network | [63] |
| Topology Adaptive Graph Network | [17] |
| Simple Graph Network | [71] |
| Triplet Message Passing Network (TrimNet) | [36] |

Table 1.4: Graph Convolutional Networks we experimented with.

Our approach goes beyond classic graph convolutional networks which mainly just focus on atom features, and expect the graph convolutional operator to learn the global information from these lower level features. An overview of our architecture, which wraps most of the operators listed in Table 1.4, can be seen in Figure 1.6.

As an initial step, the input SMILES are converted using graph featurization and the generated atom features + adjacency matrix are then fed to a number of graph convolutional layers. For each layer, we can optionally append edge features to

the atom features before passing them to the graph operator (listed in Table 1.4), then an optional residual layer[23], an optional batch normalization layer (we have support for BatchNorm[27], LayerNorm[62], and GraphNorm[10]), a Rectified Linear Unit activation[1] and a dropout[59] layer with configurable probability. The output of one layer acts as the input for the next one.

After the final layer is propagated through, 2 feature sets are generated; one using global max pooling (by taking the channel-wise maximum across the node dimension) and one using global add pooling (were node features are summed across the node dimension).

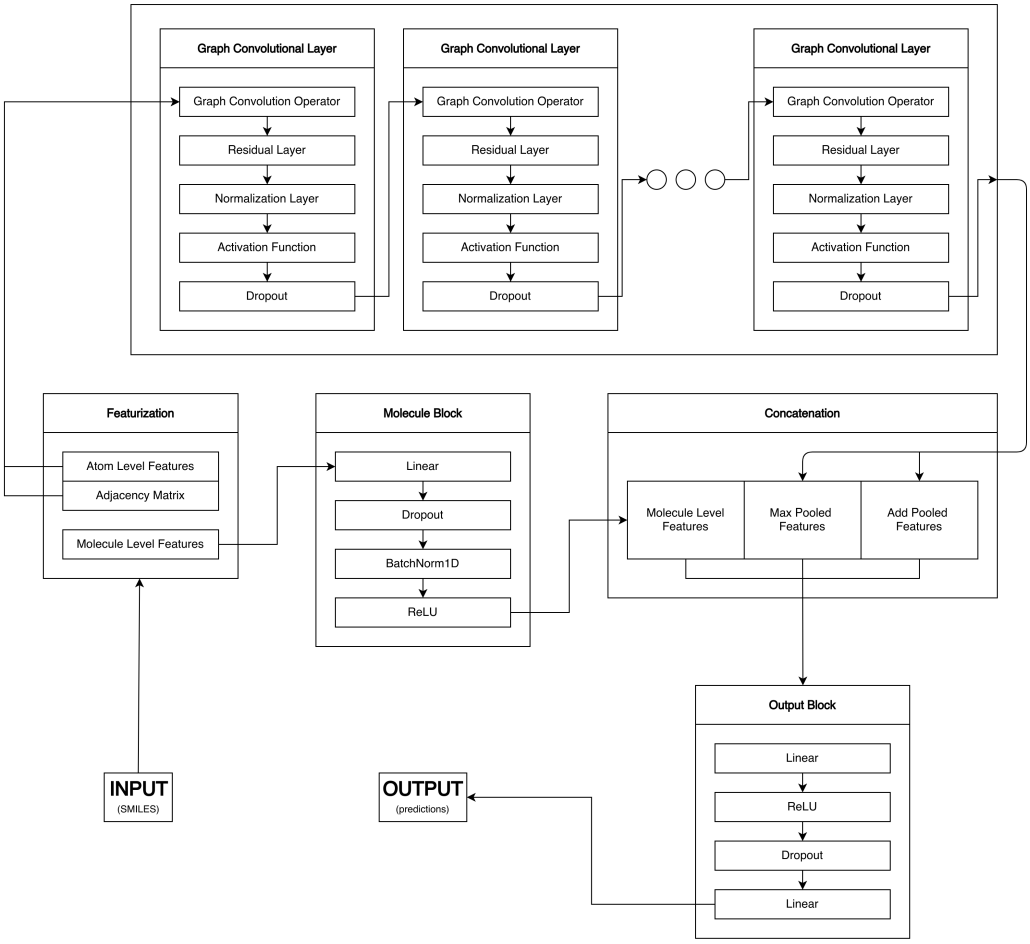


Figure 1.6: Overview of our Graph Network Wrapper

In the meantime, molecular level features (either RdKit or Mordred) are passed through a shallow network (one linear layer, a dropout[59] activation, a batch normalization layer, and a ReLu activation), and the output is concatenated with the above mentioned pooled features. Finally, the features pass through a final output block consisting of 2 linear layers, a ReLu activation and another dropout layer.

### 1.4.2 Multi-Modal Architectures

For ChEMBL assay dataset, while we can still use graph convolutional networks for the ligand encoding, we also have to deal with the amino-acid sequences.

Here, we either used classic 1D convolutions for the tokenized entries, or a shallow neural network for the bag-of-words encoded features. We also experimented with circular fingerprints as a ligand encoding, along with graph representations.

Bag-of-words encodings for the amino-acid and graph representations for the ligand seemed to work best.

## 1.5 Pipelines

Any operation within our framework has to pass through 2 processes:

- data preparation
- execution

Data preparation pipelines start from the raw data format (ie: a file on disk) and end when the data is featurized, normalized, and ready for training in Tensor formats. Execution pipelines contain everything that happens afterwards, including training and inference, but also optimization, learning rate scheduling, and metric computations.

### 1.5.1 Data Preprocessing

There are number of entities with various roles which help bring the data from raw format, to machine learning ready format. To better explain the pipeline, we borrow some terms from the DeepChem[73] library:

- **loaders**: load the raw data with various formats (ie: CSV) from disk into memory (ie: pandas DataFrame);
- **featurizers**: the input fields (ie: SMILES) are prepared by the featurizers (ie: fingerprint featurizer);
- **transformers**: the output fields (ie: toxicity) can be normalized by a transformer (ie: standardization);
- **cache manager**: featurization and transformation can be a lengthy process, so we make use of caching to save our progress and time;
- **splitters**: are used to create subsets of our dataset for various purposes;
- **streamers**: combine all to other elements and dictate how they should coordinate.

### *Loaders*

We provide 3 types of loaders out of the box:

- **CSV Loader**: for comma-separated values
- **Excel Loader**: for spreadsheets
- **SDF Loader**: for SDF files (which also makes the SMILES string available for featurization)

### *Featurizers*

Featurizers work on target columns/fields and it is possible to use more than 1 featurizer. These can be used in parallel, on different columns, or in series, for the same columns.

- **Graph Featurizer**: for graph convolutional networks, with configurable atom features and molecule level features;
- **Circular Fingerprint Featurizer**: for shallow networks and classic machine learning models like Random Forest[25];
- **One-Hot Encoder**;

- **Tokenizer**;
- **Bag of Words Featurizer**;
- **Transpose Featurizer**: if the model expects a matrix to be flipped and we do not want to write custom code;
- **Fixed Featurizer**: will divide the target input by a fixed value.
- **Converter Featurizer**: can convert between different molecular formats (ie: inchi to smiles)

### *Transformers*

Similar to featurizers, multiple transformers can be applied to the same target. Transformers are kept in memory after they are applied and are reverted after inference (only for direct inference, not for metric computations).

- **Log Normalizer**: will convert the targets to their log values;
- **Min-Max Normalizer**;
- **Fixed Normalizer**: will divide the targets by a fixed value;
- **Standardizer**: will perform Z-score normalization on the target value;
- **Cutoff Transformer**: will convert a continuous value to a binary value based on a user-specified threshold (note: this operation is destructive).
- **One-Hot Transformer**: can convert text to numeric values (ie: Low/High to 0/1)

### *Splitters*

The splitters can generate any number of subsets, not just 2 (train-test) or 3 (train-eval-test). We provide 3 ways to perform data splits:

- **Index Split**: will return the entries in the order they appear in the file;
- **Random Split**: will return the entries randomly;

- **Stratified Split**: will return random entries but will try to keep the proportion of a certain output value equal among splits (if the target value in continuous, it can be sorted in a number of configurable bins before sorting is done);
- **Descriptor Split**: performs a stratified split based on an RdKit descriptor;
- **Scaffold Balancer Split**: will split the dataset by scaffold, trying to keep an equal number of samples for each scaffold/split;
- **Scaffold Divider Split**: will split the dataset by scaffold, trying to keep unique scaffolds in each split;
- **Butina Balancer Split**: a similarity based splitter which uses Butina clustering (tries to keep the splits similar);
- **Butina Divider Split**: a similarity based splitter which uses Butina clustering (tries to keep the splits dissimilar);

### *Streamers*

The streamers are of little concern for most users, however we include them for completeness sake.

- **General Streamer**: A general purpose streamer which will use the "loader" to iterate through entries, featurizes and transforms them, then caches the solution before applying the splitter on top. In the end, users can specify which split to use, the batch size to collate, and whether to shuffle the data split before serving.
- **Subset Streamer**: Very useful for federated learning experiments, but for little else. It will basically apply an additional index split and return a subset of the data (for each client in the case of federated learning).
- **Cross Validation Streamer**: The cross-validation streamer overwrites the splitting functionality so it generates a number of user-specified folds instead of a single split. As an example, if a 5-fold split is requested, the specified splitter is applied to split the data into 5 (if possible) equal proportion subsets. When a fold is requested, the given fold will act as the test split and the remaining folds as the train split.

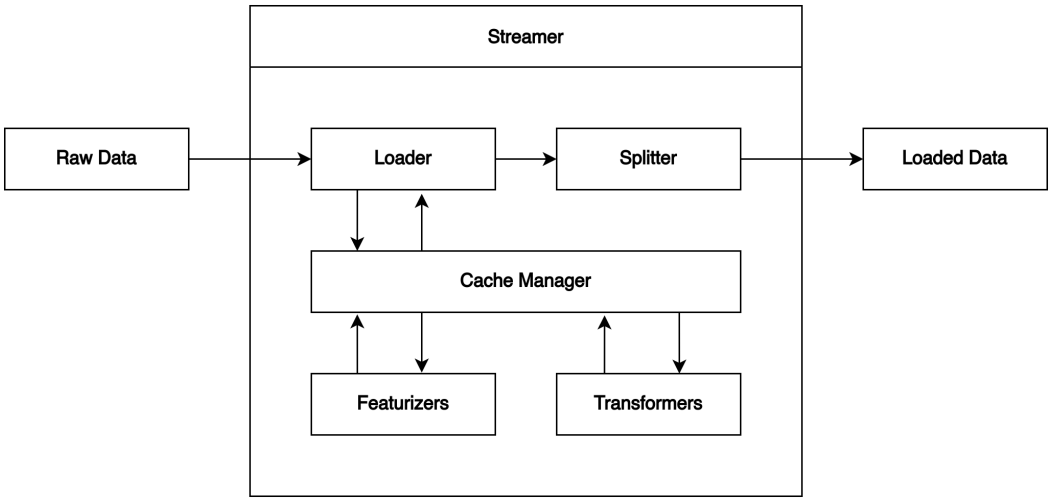


Figure 1.7: Overview of the Data Preparation Pipeline

### 1.5.2 Model Tuning

Similar to the graph convolutional model wrapper, the whole training process was designed to be highly customizable. That is, users should be able to achieve a lot of functionality without having to touch the code. While the details of how this is achieved is left for the technical part of this report, we briefly mention some of the customizable options.

#### *Training Options*

Loss functions (criterions) and optimizers can be directly specified in the configuration files as a module path (ie: "torch.nn.MSELoss"). As far as we are aware, most loss functions and optimizers can be used in this way without having to modify the code.

Additionally, it easy to add custom solutions in separate files, which then can be linked in a similar fashion. As an example, we implement a novel optimizer, the AdaBelief[77] optimizer, which outperformed Adam[30] in many experiments.

We are also making use of learning rate schedules which can be configured in a

similar way.

### *Metrics*

Metrics can be specified to both evaluation and logging purposes during training. These metrics can also be selected as target when using Bayesian Optimization[2]. All supported metrics are listed in Table 1.5

| Name | Shorthand | Task |
|---|---|---|
| Mean Absolute Error | mae | Regression |
| Mean Squared Error | mse | Regression |
| Root Mean Squared Error | rmse | Regression |
| R2 Score | r2 | Regression |
| Pearson Correlation | pearson | Regression |
| Spearman Correlation | spearman | Regression |
| Kullback Leibler Divergence | kl_div | Regression |
| Jensen Shannon Divergence | js_div | Regression |
| Chebyshev Distance | chebyshev | Regression |
| Manhattan/CityBlock Distance | manhattan | Regression |
| Area Under the ROC Curve | roc_auc | Classification |
| Average Precision | pr_auc | Classification |
| Accuracy Score | accuracy | Classification |
| Precision Score | precision | Classification |
| Recall Score | recall | Classification |
| F1 Score | f1 | Classification |
| Cohen's Kappa Coefficient | cohen_kappa | Classification |
| Jaccard Similarity | jaccard | Classification |

Table 1.5: Supported Metrics

### *Bayesian Optimization*

All configurable options can be tuned using Bayesian optimization. This includes model specific options like the model type, layer types, dropout rates or batch normalization layer types, but also training options like the optimizer, criterion, or the learning rate (the list is not exhaustive).

To be more specific, we use a Tree-structured Parzen Estimator (TPE)[5, 6] implemented with Optuna[2]. On each trial, for each parameter, TPE fits one Gaussian Mixture Model (GMM) $l(x)$ to the set of parameter values associated with the best objective values, and another GMM $g(x)$ to the remaining parameter values. It chooses the parameter value $x$ that maximizes the ratio $\frac{l(x)}{g(x)}$.

## 1.6 Communication (Mila)

For the communication part of federated learning, we were planning to use NVIDIA's Clara, but it turned out that Clara only works with Tensorflow models (version 1.14). We, as well as Kyoto University would prefer working with PyTorch or at least Tensorflow 2.x, so we decided to rebuild the communication module, following in the steps of the NVIDIA Clara implementation.

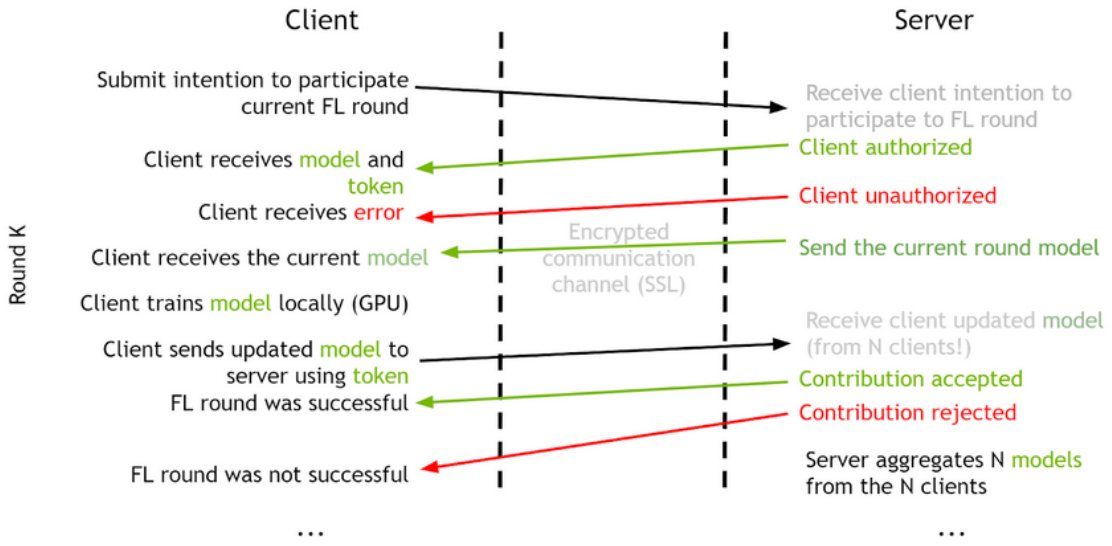


Figure 1.8: Federated Learning workflow followed by NVIDIA Clara and modeled by us (source: NVIDIA Blog)

We named the module "Mila", the greek word for speech. Similar to NVIDIA Clara, we use gRPC and protocol buffers to transmit information between the client and the servers.

Once the server is online, we wait for a number of clients to join before the training starts. The server then sends the latest model to each participant, which then

train the model on their local data for a fixed number of epochs. After training finished, the latest checkpoint is sent back to the server, which will wait for all checkpoints before they are averaged. The new aggregate model is then used for the next round, and the process is repeated.

Each client sends a keepalive signal every minute, and in case one client goes offline or we stop receiving the heartbeat signal, the server halts the process. That is, if one client goes offline, the whole experiment has to be repeated.

### 1.6.1 Security

#### *Secure Communication*

Communication is done through SSL/TLS with both server and client certificate chains. For development purposes, insecure connections are also allowed (no certs required).

When a client authenticates, it receives a token, which will subsequently be used for all operations. Machines not authenticates (ie: without a valid token), or dead clients (with an expired keeplive signal) will not be able to join the training process.

Furthermore, the server has the option to whitelist only a few trusted IP addresses which can join the process; blacklisting suspicious IPs is also possible.

#### *Privacy Attacks*

We generally think of machine learning models as a black box and expect that they would not reveal information about the original dataset they were trained on. In reality, membership inference attacks are possible, where an attacker tries to determine whether an input sample was used as part of the training set. While it is questionable how well research on privacy attacks transfers to large and complex models (let alone graph based models), we cannot ignore the possibility of such attacks.

Over-fitting is often exploited in inference attacks, and regularization tactics have been shown to reduce attack accuracies. Here we are talking about methods such as dropout, regularization, or normalization. The architecture type and complexity

of the model and data can also affect attack accuracy.

An in-depth review of potential privacy attacks is beyond the score of this report, however, we would like to redirect curious readers to a recent survey paper on the topic[55]. It should be noted that research of such attacks on graph-based models is either sparse or nonexistent.

### *Differential Privacy*

Differential privacy can be a useful asset for defence against privacy attacks. It is based on the idea of "learning nothing about an individual while learning useful information about a population"[18, 55]. The definition is based on the notion that if two databases differ only by one record and are used by the same model, the output of that algorithm should be similar.

Formally put, a randomized model $M$ with output $S$ is ($\epsilon$, $\delta$)-differentially private if for any adjacent inputs $D$ and $D'$ that differ in a single example:

$$\forall M, P[M(D) \in S] \leq e^{\epsilon} P[M(D') \in S] + \delta \tag{1.1}$$

where:

- $\epsilon$: is the privacy budget.
- $\delta$: is the probability of information accidentally being leaked. The term was introduced as a relaxation that allows some outputs not to be bounded by $e^{\epsilon}$.

The privacy budget $\epsilon$, measures the distance between the outputs of the model on the 2 datasets. Therefore, $\epsilon$ is a measure of the privacy loss.

It is important to note that differential privacy does not guarantee any privacy in itself, instead, it is a measure of privacy and informs us about how likely the model is to leak knowledge about certain samples from the dataset. With differential privacy, we can set a "privacy budget" and control how much privacy we are willing to give away. Actually making the solution more secure is generally achieved by adding Gaussian or Laplacian noise either to the weights or the gradients of the model, however, this will affect the quality of the trained model (sometimes drastically).

For our solution, we used Opacus[50] to implement differential privacy, however, it comes with limitations. Some layer types are not compatible for differential privacy, with the most prominent example being BatchNorm. As a solution we provide 2 solutions, either removing batch normalization layers, or replacing them with an alternative such as GroupNorm[72].

But more importantly, differential-privacy has a utility-privacy trade off. To make the model more secure, more noise will have to be added, but the more noise we add to the model, the worse it will perform.

### 1.6.2 Differences from NVIDIA Clara

The main advantage of our implementation is that it is framework independent. All models and checkpoints are transmitted as bytes and stored in the same way they are on the server. NVIDIA Clara transports directly (Tensorflow) Tensor objects, and so, only supports Tensorflow models and aggregators.

Another key difference is that our implementation does not require GPUs and can run with all participants (clients and servers) using CPU only, should they wish to do so.

In addition, we added support for IP whitelisting and blacklisting.

On the negative side, our implementation is new and didn't go through as much testing as NVIDIA Clara so far.

# Experiment Results

## 2.1 Introduction

The general pipeline for each dataset can be summarized as follows:

- we first use Bayesian Optimization for both architecture search and hyper-parameter tuning;
- we validate the best architecture using Cross-Validation
- we perform distributed experiments with the best architecture (federated learning)

For federated learning experiments, we analyze how the performance changes when we alter important properties like:

- the number of clients (from 2 to 20);
- the number of epochs trained per round (1, 2, 5, 10, or 20);
- the balance of the datasets between each client (from evenly distributed to highly skewed splits).

All reported metrics all the highest recorded independent values across all checkpoints. The reported mean absolute, mean squared, and root mean squared errors are based on normalized values.

### *Thresholds*

We evaluate experiments on various metrics, depending on the task (classification or regression). For regression tasks we use the $R^2$ Score, the root mean squared error, and the mean absolute error. For classification, we make note of the area under the receiver operating characteristic curve (ROC-AUC, or AUROC), the average precision (PR-AUC, or mAP), the accuracy, the precision, the recall, and Cohen's Kappa Coefficient.

It is worth noting that some classification metrics are threshold based (ie: they are evaluated at a certain threshold), while other ones are threshold independent (ie: evaluated at various thresholds). Besides the ROC-AUC and PR-AUC (average precision) metrics, all classification metrics are threshold based. Threshold based metrics like the accuracy can be dangerous when analyzed individually if the

dataset is imbalanced. For example, if we are working on a classification task on a cancer dataset, over 99% of entries are probably negative. If a model outputs "negative" predictions all the time, it would achieve a 99%+ accuracy without actually providing a discriminative effect.

As an attempt to get a more meaningful metric, the thresholds have been optimized using a precision-recall curve, and have been set to the value where the precision and recall lines intersect. As a consequence, threshold based metrics like the accuracy do not show the highest possible values we can register on the said metric, but the most useful and informative values.

## 2.2 Architecture & Hyper-Parameter Optimization

As a first step, we report the best architectures we found using Bayesian Optimization for each dataset. We considered the following options during the optimization phase:

- the model/layer type (Table 1.4)
- the number of hidden features (between 32 and 256)
- the dropout rate (between 0.0 and 0.7)
- the number of graph layers (between 2 and 12)
- the batch normalization layer (BatchNorm, LayerNorm, or GraphNorm)
- the optimizer (Stochastic Gradient Descent[29], Adam[30], AdamW[40], and AdaBelief[77])
- the learning rate (between 0.0001 and 0.01)
- the weight decay rate (between 0.00001 and 0.001)
- whether to use residual layers in the graph convolution (yes or no)
- whether to use edge features in the graph operation (yes or no)
- the type of molecular descriptors (none, rdKit[34] or Mordred[53])

### 2.2.1 Tox21

Although the Tox21 dataset contains 12 targets, it was taken as a single, multi-task dataset. The model would use the (featurized) SMILES as input and output a prediction for all 12 labels in one go.

The dataset is very imbalanced, so metrics such as accuracy should not be fully trusted (even with threshold tuning). Metrics such as the ROC-AUC and PR-AUC are more trustworthy when it comes to imbalanced datasets.

Furthermore, the Tox21 dataset contains many missing values. To handle these cases, we perform the forward pass of the model as usual, but we set the weight for the missing values to 0 when computing the loss. Similarly, when we compute the metrics, we ignore missing values.

| Task | ROC-AUC | PR-AUC | Accuracy | Precision | Recall | Kappa |
|---|---|---|---|---|---|---|
| NR-AR | 0.813 | 0.524 | 0.971 | 0.929 | 0.471 | 0.567 |
| NR-AR-LBD | 0.908 | 0.610 | 0.980 | 0.730 | 0.711 | 0.678 |
| NR-AhR | 0.900 | 0.659 | 0.909 | 0.655 | 0.840 | 0.579 |
| NR-Aromatase | 0.913 | 0.534 | 0.953 | 0.556 | 0.694 | 0.492 |
| NR-ER | 0.738 | 0.444 | 0.865 | 0.465 | 0.699 | 0.335 |
| NR-ER-LBD | 0.838 | 0.448 | 0.947 | 0.581 | 0.530 | 0.431 |
| NR-PPAR-gamma | 0.897 | 0.394 | 0.973 | 0.556 | 0.621 | 0.433 |
| SR-ARE | 0.844 | 0.553 | 0.844 | 0.491 | 0.842 | 0.446 |
| SR-ATAD5 | 0.918 | 0.464 | 0.968 | 0.571 | 0.635 | 0.494 |
| SR-HSE | 0.855 | 0.435 | 0.949 | 0.625 | 0.662 | 0.481 |
| SR-MMP | 0.924 | 0.729 | 0.889 | 0.616 | 0.900 | 0.614 |
| SR-p53 | 0.884 | 0.368 | 0.937 | 0.365 | 0.696 | 0.350 |
| Minimum | 0.738 | 0.368 | 0.844 | 0.365 | 0.471 | 0.335 |
| Maximum | 0.924 | 0.729 | 0.980 | 0.929 | 0.900 | 0.678 |
| Mean | 0.869 | 0.513 | 0.932 | 0.595 | 0.692 | 0.492 |
| Median | 0.890 | 0.494 | 0.948 | 0.576 | 0.695 | 0.487 |
| Standard Deviation | 0.052 | 0.105 | 0.043 | 0.135 | 0.120 | 0.099 |

Table 2.1: Baseline metrics on the Tox21 dataset

The best performing architecture registered was a Local Extremum Graph Network[54] with 7 graph layers, a hidden layer size of 96, activate residual layers, a dropout rate of 0.1, and BatchNorm as a normalization layer. We used an index split of 80% (training) 20% (test), and rdKit molecule level descriptors. The optimizer used was AdamW[40], with a learning rate of 0.01, and weight decay of 0.00056. As a criterion we used binary cross entropy loss with a mask for missing values. The batch size was set to 128, we trained for 200 epochs, and used a threshold of 0.216 for non-logit based metric computations. We also used a One Cycle Learning Rate Scheduler[58].

### 2.2.2 AMES

The AMES dataset has a single label, however we trained models on 4 versions of the dataset (see the datasets section). Bayesian optimization was performed on the final combo dataset (benchmark + NTP PubChem), and evaluated on all 4 versions.

The best performing architecture registered was a Classic Graph Convolutional Network[61] with 3 graph layers, a hidden layer size of 160, activate residual layers, a dropout rate of 0.1, and BatchNorm as a normalization layer. We used an stratified split of 80% (training) 20% (test). The split was performed with Scikit-Learn[52] and the random seed was set to 42.
The optimizer used was AdaBelief[77], with a learning rate of 0.01, and weight decay of 0.00036. As a criterion we used binary cross entropy loss, and again, a One Cycle Learning Rate Scheduler[58]. The batch size was set to 128, we trained for 200 epochs, and used a threshold of 0.499.

| Task | ROC-AUC | PR-AUC | Accuracy | Precision | Recall | Kappa |
|---|---|---|---|---|---|---|
| Benchmark | 0.871 | 0.876 | 0.804 | 0.833 | 0.906 | 0.603 |
| NTP PubChem | 0.890 | 0.899 | 0.830 | 0.865 | 0.889 | 0.659 |
| S9 Minus | 0.885 | 0.824 | 0.847 | 0.878 | 0.806 | 0.649 |
| Combo/Final | 0.903 | 0.908 | 0.837 | 0.879 | 0.897 | 0.674 |

Table 2.2: Baseline metrics on the AMES datasets

### 2.2.3 ADME

Each ADME related dataset was optimized separately. We present our results in Table 2.3 for classification tasks, and Table 2.4 for regression tasks.

| Task | ROC-AUC | PR-AUC | Accuracy | Precision | Recall | Kappa |
|---|---|---|---|---|---|---|
| FeHuman (th=0.3) | 0.835 | 0.897 | 0.721 | 0.702 | 0.921 | 0.505 |
| FeHuman (th=0.7) | 0.921 | 0.985 | 0.897 | 0.932 | 0.966 | 0.572 |
| NER-LLC (low) | 0.853 | 0.897 | 0.809 | 1.000 | 0.851 | 0.621 |
| NER-LLC (medium) | 0.782 | 0.574 | 0.607 | 0.441 | 1.000 | 0.298 |
| NER-LLC (high) | 0.829 | 0.488 | 0.876 | 1.000 | 1.000 | 0.415 |
| NER-LLC (mean) | 0.821 | 0.653 | 0.764 | 0.813 | 0.950 | 0.444 |
| PappCaco2 | 0.860 | 0.781 | 0.799 | 0.739 | 0.991 | 0.581 |
| Solubility | 0.861 | 0.895 | 0.806 | 0.803 | 1.0 | 0.584 |

Table 2.3: ADME tasks (classification)

| Task | R2 | MAE | RMSE |
|---|---|---|---|
| CLint | 0.489 | 0.806 | 1.536 |
| FuBrain | 0.591 | 1.309 | 2.545 |
| FupHuman | 0.690 | 0.090 | 0.156 |
| FupRat | 0.594 | 0.111 | 0.183 |
| PappCaco2 | 0.451 | 1.090 | 1.411 |
| Papp-LLC | 0.577 | 0.071 | 0.124 |
| RbRat | 0.669 | 0.173 | 0.225 |

Table 2.4: ADME tasks (regression)

We also experimented with multi-task learning on the ADME dataset, however, the overlaps between them is small and the results were not very good. For the PappCaco2 dataset, we did both regression and classification experiments. We report the individual values, however, we experimented with a joined learning approach as well, where we train the classification and regression task in one experiment. For the fraction excreted unchanged in urine (FeHuman) dataset, with experimented with 2 thresholds, 0.3 and 0.7.

A list of the most important options are listed in Table 2.5. All architectures used residual layers, and most used BatchNorm[27] for normalization (exceptions are NER-LLC which used GraphNorm[10], and FupRat which used LayerNorm[62]). As optimizer, all experiments used AdamW[40]. For regression tasks we used random split, for classification we used a stratified split. The batch size was 64 on all experiments, except for RbRat, where we used a batch size of 24. Classification tasks used a binary cross entropy loss, while regression tasks relied on Smooth L1 Loss. Models were trained for 200 epochs, except for fuBrain (210), fupRat (230), and rbRat (230). We used log values for FuBrain, CLint, PappCaco2, and RbRat, while for PappLLC we used a fixed transformer and divided values by 300.

| Dataset | Layer Type | n_layers | hidden | dropout | lr | decay |
|---|---|---|---|---|---|---|
| CLint | GENConv[35] | 2 | 128 | 0.1 | 0.01 | 6e-6 |
| FeHuman | GCNConv[61] | 4 | 64 | 0.2 | 0.003 | 5e-4 |
| FuBrain | TrimConv[36] | 3 | 64 | 0.2 | 0.0045 | 1.6e-6 |
| FupHuman | GINConv[74] | 2 | 128 | 0.15 | 0.003 | 1.5e-6 |
| FupRat | GENConv[35] | 3 | 64 | 0.1 | 0.01 | 2e-6 |
| NER-LLC | GINConv[74] | 2 | 32 | 0.25 | 0.01 | 1e-4 |
| PappCaco2 | GINConv[74] | 4 | 128 | 0.1 | 0.0008 | 2e-4 |
| Papp-LLC | GCNConv[61] | 2 | 32 | 0.2 | 0.01 | 8e-6 |
| RbRat | GCNConv[61] | 3 | 32 | 0.15 | 0.017 | 1e-5 |
| Solubility | GINConv[74] | 2 | 128 | 0.1 | 0.001 | 5e-5 |

Table 2.5: Hyper-parameters of ADME tasks (layer type, number of layers, number of hidden features, dropout rate, learning rate, weight decay)

### 2.2.4 ChEMBL

Parameters were tuned on the first version of the dataset (details in the datasets section), which was then evaluated on the second version subtasks as well.

Protein encoding was done with bag-of-words featurization, while ligand features were encoded as graphs. The protein features were passed through a multi-layer perceptron, with 9723 input features, 160 hidden features, and 16 output features. The ligand features were passed through a GENeralized Graph Network[35] with

5 layers, no dropout, no residual connections, and BatchNorm as normalization layer. The number of hidden features of the graph model was 192. Surprisingly, a Stochastic Gradient Descent[29] worked best, with the learning rate set to 0.0041 and the weight decay to 0.00001. For classification, again we rely on binary cross entropy loss, and for regression on the Smooth L1 loss. We also used a batch size of 128 and trained the models for 50 epochs. For the regression task, we used a log transformation of the target values.

| Task | ROC-AUC | PR-AUC | Accuracy | Precision | Recall | Kappa |
|---|---|---|---|---|---|---|
| v1 (t=100nM) | 0.882 | 0.882 | 0.800 | 0.807 | 0.810 | 0.600 |
| v2 (t=100nM) | 0.891 | 0.855 | 0.809 | 0.779 | 0.818 | 0.61 |
| v2 (t=1um) | 0.9 | 0.948 | 0.836 | 0.868 | 0.934 | 0.616 |
| v2 (t=10um) | 0.916 | 0.989 | 0.918 | 0.941 | 0.999 | 0.476 |

Table 2.6: Baseline Metrics for the ChEMBL Protein-Ligand Binding Affinity datasets (classification)

| Task | R2 | MAE | RMSE |
|---|---|---|---|
| v2 | 0.651 | 1.362 | 1.828 |

Table 2.7: Baseline Metrics for the ChEMBL Protein-Ligand Binding Affinity datasets (regression)

Initially, we used an index split on the first version (v1) which yielded better results than the ones we report in Table 2.6, with a register ROC-AUC of 0.956, PR-AUC of 0.959, and accuracy of 0.878. We later experimented with stratified splits which are the values reported in Table 2.6 and Table 2.7. The stratified split should contain a better representation of the target values, so we think these values are closer to reality.

### 2.2.5 Other Datasets

Finally, we report values registered on the additional ADME-Tox datasets for information purposes.

| Dataset | ROC-AUC | PR-AUC | Acc | R2 | MAE | RMSE |
|---|---|---|---|---|---|---|
| bbb adenot | 0.987 | 0.996 | 0.955 | | | |
| bbb martins | 0.947 | 0.975 | 0.882 | | | |
| bioavailability ma | 0.787 | 0.899 | 0.804 | | | |
| caco2 wang | | | | 0.760 | 0.375 | 0.485 |
| carcinogens lagunin | 0.948 | 0.875 | 0.910 | | | |
| clearance edrug3d | | | | 0.028 | 0.303 | 1.547 |
| clearance hepatocyte az | | | | 0.195 | 0.667 | 0.890 |
| clearance microsome az | | | | 0.383 | 0.558 | 0.784 |
| clintox | 0.951 | 0.765 | 0.959 | | | |
| cyp1a2 veith | 0.943 | 0.938 | 0.874 | | | |
| cyp2c19 veith | 0.903 | 0.880 | 0.831 | | | |
| cyp2c9 substrate | 0.766 | 0.479 | 0.791 | | | |
| cyp2c9 veith | 0.900 | 0.805 | 0.828 | | | |
| cyp2d6 substrate | 0.812 | 0.678 | 0.783 | | | |
| cyp2d6 veith | 0.893 | 0.727 | 0.874 | | | |
| cyp3a4 substrate | 0.723 | 0.713 | 0.634 | | | |
| cyp3a4 veith | 0.915 | 0.878 | 0.839 | | | |
| dili | 0.927 | 0.902 | 0.852 | | | |
| f20 edrug3d | 0.782 | 0.956 | 0.814 | | | |
| f30 edrug3d | 0.806 | 0.948 | 0.728 | | | |
| halflife edrug3d | | | | 0.501 | 0.106 | 0.271 |
| half life obach | | | | 0.405 | 0.103 | 0.171 |
| herg | 0.914 | 0.957 | 0.839 | | | |
| hia hou | 0.998 | 0.999 | 0.982 | | | |
| hydrationfreeenergy freesolv | | | | 0.930 | 0.165 | 0.283 |
| ld50 zhu | | | | 0.669 | 0.418 | 0.578 |
| lipophilicity astrazeneca | | | | 0.763 | 0.365 | 0.492 |
| pgp broccatelli | 0.955 | 0.966 | 0.762 | | | |

Table 2.8: Metrics for other datasets

| Dataset | ROC-AUC | PR-AUC | Acc | R2 | MAE | RMSE |
|---|---|---|---|---|---|---|
| ppbr az | | | | 0.641 | 0.375 | 0.605 |
| ppbr edrug3d | | | | 0.732 | 0.381 | 0.515 |
| ppbr ma | 0.885 | 0.874 | 0.745 | | | |
| skin reaction | 0.896 | 0.947 | 0.802 | | | |
| solubility aqsoldb | | | | 0.837 | 0.267 | 0.396 |
| vd edrug3d | | | | 0.121 | 0.087 | 0.147 |
| vdss lombardo | | | | 0.315 | 0.115 | 0.193 |

Table 2.8: Metrics for other datasets (continued)

| dataset | model | optimizer | layers | dropout | residual |
|---|---|---|---|---|---|
| bbb adenot | SAGEConv | AdamW | 7 | 0.2 | yes |
| bbb martins | GENConv | AdaBelief | 3 | 0.4 | no |
| bioavailability ma | GCNConv | AdaBelief | 2 | 0.3 | no |
| caco2 wang | GraphConv | Adam | 5 | 0.2 | no |
| carcinogens lagunin | GENConv | AdamW | 7 | 0.0 | yes |
| clearance edrug3d | GCNConv | AdamW | 4 | 0.0 | no |
| clearance hepatocyte az | GCNConv | AdaBelief | 3 | 0.2 | yes |
| clearance microsome az | GraphConv | Adam | 7 | 0.2 | yes |
| clintox | GINConv | AdaBelief | 7 | 0.2 | no |
| cyp1a2 veith | SAGEConv | AdaBelief | 7 | 0.1 | no |
| cyp2c19 veith | GCNConv | Adam | 7 | 0.1 | no |
| cyp2c9 substrate | GraphConv | Adam | 7 | 0.4 | yes |
| cyp2c9 veith | GraphConv | AdaBelief | 2 | 0.4 | no |
| cyp2d6 substrate | SAGEConv | AdaBelief | 5 | 0.0 | no |
| cyp2d6 veith | SAGEConv | AdamW | 6 | 0.1 | no |
| cyp3a4 substrate | LEConv | SGD | 4 | 0.0 | no |
| cyp3a4 veith | GraphConv | Adam | 3 | 0.1 | yes |
| dili | GCNConv | SGD | 6 | 0.4 | no |
| f20 edrug3d | GENConv | AdamW | 3 | 0.2 | yes |
| f30 edrug3d | GraphConv | Adam | 6 | 0.4 | no |

Table 2.9: Parameters for other datasets

| dataset | model | optimizer | layers | dropout | residual |
|---|---|---|---|---|---|
| halflife edrug3d | GINConv | AdamW | 6 | 0.2 | yes |
| half life obach | GCNConv | AdamW | 3 | 0.2 | no |
| herg | LEConv | SGD | 2 | 0.1 | yes |
| hia hou | GINConv | AdamW | 4 | 0.0 | yes |
| hydrationfreeenergy freesolv | GENConv | AdaBelief | 4 | 0.0 | yes |
| ld50 zhu | LEConv | AdaBelief | 3 | 0.1 | no |
| lipophilicity astrazeneca | GENConv | AdamW | 5 | 0.1 | no |
| pgp broccatelli | SAGEConv | Adam | 8 | 0.2 | yes |
| ppbr az | GraphConv | AdamW | 4 | 0.1 | no |
| ppbr edrug3d | GCNConv | Adam | 3 | 0.1 | no |
| ppbr ma | GENConv | SGD | 7 | 0.2 | yes |
| skin reaction | GraphConv | SGD | 4 | 0.0 | yes |
| solubility aqsoldb | SAGEConv | Adam | 4 | 0.0 | yes |
| vd edrug3d | GCNConv | AdaBelief | 3 | 0.2 | yes |
| vdss lombardo | GENConv | Adam | 2 | 0.3 | yes |

Table 2.9: Parameters for other datasets (continued)

It is interesting to see the distribution of these hyperparameters.

In terms of model architecture, Classic Graph Networks[61] and Weisfeiler-Leman Graph Networks[47] dominate with 8 counts for each, however GENeralized Graph Networks[35] are not far behind with 7 wins. For optimizers we see an equal distribution amoung AdaBelief, AdamW, and Adam, with 10 entries for each, while SGD is lagging behind with 5 wins. For the number of layers, it seems there is no clear winner, however 3 and 7 layers seem to be the most popular (with 8 wins for each). In terms of dropout, 0.2 (11 wins) and 0.1 (9 wins) is a sweet spot, followed by no dropout at all (8 wins). And finally, we see that some models work better with skip connections while others just do not.

## 2.3 Cross-Validation

Hyper-parameter tuning was done using the same data splits. That is, each set of hyper-parameters was evaluated on the same data samples. It is helpful to evaluate these baselines on multiple splits.

| Dataset | ROC-AUC | PR-AUC | Accuracy |
|---|---|---|---|
| NR-AR | 0.8358±0.0161 | 0.5594±0.0531 | 0.9757±0.0024 |
| NR-AR-LBD | 0.8998±0.0089 | 0.6538±0.0429 | 0.9795±0.0010 |
| NR-AhR | 0.9105±0.0075 | 0.6682±0.0164 | 0.9080±0.0043 |
| NR-Aromatase | 0.8871±0.0156 | 0.4955±0.0405 | 0.9498±0.0073 |
| NR-ER | 0.7627±0.0106 | 0.4937±0.0300 | 0.8769±0.0089 |
| NR-ER-LBD | 0.8604±0.0185 | 0.5402±0.0378 | 0.9592±0.0034 |
| NR-PPAR-gamma | 0.8972±0.0177 | 0.4292±0.0247 | 0.9719±0.0038 |
| SR-ARE | 0.8479±0.0118 | 0.5691±0.0265 | 0.8371±0.0064 |
| SR-ATAD5 | 0.8911±0.0167 | 0.4344±0.0608 | 0.9640±0.0036 |
| SR-HSE | 0.8519±0.0097 | 0.4302±0.0282 | 0.9461±0.0038 |
| SR-MMP | 0.9228±0.0085 | 0.7563±0.0286 | 0.8965±0.0069 |
| SR-p53 | 0.8803±0.0095 | 0.4332±0.0446 | 0.9309±0.0022 |
| Tox21 (mean) | 0.8706±0.0126 | 0.5386±0.0362 | 0.9330±0.0045 |
| AMES (Benchmark) | 0.8814±0.0034 | 0.8929±0.0064 | 0.8153±0.0069 |
| AMES (NTP PubChem) | 0.8950±0.0028 | 0.9051±0.0070 | 0.8302±0.0049 |
| AMES (S9 Minux) | 0.8972±0.0099 | 0.8469±0.0128 | 0.8561±0.0143 |
| AMES (Final) | 0.8927±0.0111 | 0.9000±0.0098 | 0.8277±0.0107 |
| FeHuman (th=0.3) | 0.8255±0.0176 | 0.8714±0.0415 | 0.7849±0.0587 |
| FeHuman (th=0.7) | 0.8650±0.0390 | 0.9710±0.0097 | 0.8673±0.0460 |
| NER-LLC (low) | 0.8283±0.0448 | 0.8177±0.0596 | 0.7884±0.0420 |
| NER-LLC (medium) | 0.7096±0.0484 | 0.6055±0.0427 | 0.5633±0.0444 |
| NER-LLC (high) | 0.8372±0.0306 | 0.5211±0.0856 | 0.8514±0.0198 |
| NER-LLC (mean) | 0.7917±0.0413 | 0.6481±0.0626 | 0.7344±0.0354 |
| PappCaco2 | 0.8645±0.0053 | 0.8075±0.0081 | 0.8049±0.0097 |
| Solubility | 0.8726±0.0221 | 0.9035±0.0138 | 0.8152±0.0216 |
| ChEMBL (v1) | 0.8819±0.0010 | 0.8819±0.0011 | 0.8010±0.0009 |

Table 2.10: Cross-validation Results (classification tasks)

| Dataset | Precision | Recall | Cohen's Kappa |
|---|---|---|---|
| NR-AR | 0.9509±0.0598 | 0.5315±0.0472 | 0.6116±0.0447 |
| NR-AR-LBD | 0.7630±0.0486 | 0.7296±0.0499 | 0.6744±0.0291 |
| NR-AhR | 0.6176±0.0396 | 0.8866±0.0268 | 0.5776±0.0158 |
| NR-Aromatase | 0.5440±0.0756 | 0.7221±0.0509 | 0.4467±0.0257 |
| NR-ER | 0.5376±0.0693 | 0.7728±0.0313 | 0.3902±0.0281 |
| NR-ER-LBD | 0.6481±0.0373 | 0.6419±0.0506 | 0.5101±0.0244 |
| NR-PPAR-gamma | 0.5040±0.0920 | 0.6515±0.0555 | 0.4454±0.0262 |
| SR-ARE | 0.4979±0.0285 | 0.9142±0.0209 | 0.4582±0.0177 |
| SR-ATAD5 | 0.5119±0.0823 | 0.7219±0.0624 | 0.4596±0.0573 |
| SR-HSE | 0.5513±0.0494 | 0.7066±0.0431 | 0.4669±0.0188 |
| SR-MMP | 0.6566±0.0212 | 0.9370±0.0082 | 0.6322±0.0152 |
| SR-p53 | 0.4464±0.0520 | 0.7924±0.0335 | 0.4278±0.0308 |
| Tox21 (mean) | 0.6024±0.0546 | 0.7507±0.0400 | 0.5084±0.0278 |
| AMES (Benchmark) | 0.8636±0.0095 | 0.9155±0.0153 | 0.6286±0.0142 |
| AMES (NTP PubChem) | 0.8941±0.0137 | 0.9106±0.0166 | 0.6593±0.0095 |
| AMES (S9 Minux) | 0.9026±0.0440 | 0.8607±0.0622 | 0.6666±0.0339 |
| AMES (Final) | 0.8787±0.0163 | 0.9031±0.0189 | 0.6545±0.0217 |
| FeHuman (th=0.3) | 0.7911±0.0878 | 1.0000±0.0000 | 0.5197±0.1025 |
| FeHuman (th=0.7) | 0.8895±0.0466 | 0.9965±0.0062 | 0.4459±0.0858 |
| NER-LLC (low) | 0.9778±0.0390 | 0.9498±0.0547 | 0.5666±0.0875 |
| NER-LLC (medium) | 0.5631±0.1945 | 1.0000±0.0000 | 0.2019±0.0756 |
| NER-LLC (high) | 1.0000±0.0000 | 0.9716±0.0313 | 0.3780±0.0730 |
| NER-LLC (mean) | 0.8470±0.0778 | 0.9738±0.0287 | 0.3821±0.0787 |
| PappCaco2 | 0.7670±0.0242 | 0.9709±0.0176 | 0.5888±0.0178 |
| Solubility | 0.8815±0.0429 | 1.0000±0.0000 | 0.6130±0.0486 |
| ChEMBL (v1) | 0.8018±0.0024 | 0.8307±0.0083 | 0.6019±0.0019 |

Table 2.10: Cross-validation Results (classification tasks) (continued)

For this, we use cross-validation, where we divide the dataset into "n" equal splits. For each "fold", the subsample will act as a test set, and the remaining samples as the train set. Each fold is then evaluated and we report the average values for each metric.

For smaller datasets we performed multiple experiments with different random splits. All Tox21, AMES and ChEMBL tasks, along with the fupHuman, fuBrain, CLint, and PappCaco2 tasks were evaluated once on 5-fold split. The Solubility, fupRat, PappLLC, NER-LLC, and feHuman tasks were evaluated on 5 different 5-fold splits. The RbRat task was evaluated on 10 different 5-fold splits.

| Dataset | R2 | MAE | RMSE |
|---|---|---|---|
| CLint | 0.4583±0.0280 | 1.9015±0.1890 | 0.8279±0.0480 |
| FuBrain | 0.6068±0.0770 | 2.066±0.5110 | 1.168±0.1800 |
| FupHuman | 0.6629±0.0330 | 0.1559±0.0110 | 0.0927±0.0060 |
| FupRat | 0.6019±0.0900 | 0.1954±0.0210 | 0.1218±0.0150 |
| PappCaco2 | 0.5053±0.010 | 1.2707±0.0100 | 0.9552±0.0140 |
| Papp-LLC | 0.6044±0.080 | 0.1213±0.0180 | 0.0694±0.0120 |
| RbRat | 0.5205±0.1490 | 0.3842±0.1970 | 0.2642±0.0830 |

Table 2.11: Cross-validation Results (regression tasks)

## 2.4 Distributed Learning

To perform federated learning, we split the training set among the clients. For client-based an epoch-based experiments, the ratios are equal. As an example, for a 2 client experiment, both clients would get an equal 50/50 share of the training set, for a 10 client experiment, they would each receive 10% of the training set. For imbalanced experiments, we specify the ratios explicitly.

Before we present our findings, it is helpful to discuss the aggregation strategies we investigated. Training progress charts for all experiments can be found in the Appendix section.

All client-based and imbalanced split experiments were trained for 10 epochs per round, and all epoch-based experiments were trained on 2 client splits.

### 2.4.1 Aggregators

The aggregation process is an important part of the federated learning process. We experimented with 3 different strategies:

- plain averaging
- weighted averaging
- benchmarked averaging

With plain averaging, checkpoints are averaged as they are, however, this can be a naive approach.

Some clients will often have a lot more data and their contribution matters more. To account for these differences, we multiply the weights for each client by the ratio of samples they posses. We call this aggregation "weighted averaging".

Still, not all datasets are equal. Possession of a benchmark dataset for evaluation on the server side can be very helpful. For these experiments we use the test set to evaluate each checkpoint received by the client in each round. Then, we compute a user specified metric for them (the ROC-AUC for classification and the R2-score for regression in our case), and multiply the weights according to the performance of each one.

We expect this aggregator to perform much better, however, the benchmark should be evaluated carefully. If the dataset is too small, or if it is not representative of the overall distribution, it can negatively affect the federated learning process. Some clients might have very useful, but rare samples which score low on the benchmark dataset (and therefore, penalized). On the other hand, benchmarked aggregation can protect against security attacks such as model poisoning[68] or evasion attacks[37].

### 2.4.2 Tox21 - NR-AR

| experiment | roc_auc | pr_auc | accuracy | precision | recall | kappa |
|---|---|---|---|---|---|---|
| Baseline | 0.813 | 0.524 | 0.971 | 0.929 | 0.471 | 0.567 |
| Clients 2 | 0.795 | 0.490 | 0.967 | 0.778 | 0.735 | 0.523 |
| Clients 3 | 0.789 | 0.485 | 0.964 | 0.683 | 0.941 | 0.496 |
| Clients 4 | 0.807 | 0.507 | 0.959 | 0.592 | 0.971 | 0.475 |
| Clients 5 | 0.815 | 0.506 | 0.962 | 0.630 | 0.985 | 0.490 |
| Clients 6 | 0.811 | 0.488 | 0.958 | 0.569 | 1.000 | 0.466 |

| | | | | | | |
|---|---|---|---|---|---|---|
| Clients 7 | 0.809 | 0.495 | 0.961 | 0.617 | 0.971 | 0.485 |
| Clients 8 | 0.811 | 0.497 | 0.958 | 0.569 | 1.000 | 0.466 |
| Clients 9 | 0.809 | 0.511 | 0.959 | 0.592 | 1.000 | 0.475 |
| Clients 10 | 0.820 | 0.490 | 0.963 | 0.667 | 1.000 | 0.499 |
| Clients 11 | 0.801 | 0.484 | 0.967 | 0.778 | 1.000 | 0.523 |
| Clients 12 | 0.803 | 0.474 | 0.961 | 0.622 | 1.000 | 0.476 |
| Clients 13 | 0.797 | 0.478 | 0.961 | 0.622 | 1.000 | 0.480 |
| Clients 14 | 0.814 | 0.483 | 0.966 | 0.732 | 1.000 | 0.534 |
| Clients 15 | 0.805 | 0.482 | 0.961 | 0.617 | 1.000 | 0.485 |
| Clients 16 | 0.807 | 0.491 | 0.946 | 0.423 | 1.000 | 0.423 |
| Clients 17 | 0.803 | 0.484 | 0.960 | 0.605 | 1.000 | 0.480 |
| Clients 18 | 0.805 | 0.490 | 0.964 | 0.690 | 1.000 | 0.510 |
| Clients 19 | 0.812 | 0.494 | 0.963 | 0.659 | 1.000 | 0.499 |
| Clients 20 | 0.814 | 0.480 | 0.963 | 1.000 | 1.000 | 0.499 |
| Epochs 1 | 0.803 | 0.485 | 0.968 | 0.800 | 0.441 | 0.529 |
| Epochs 2 | 0.805 | 0.481 | 0.966 | 0.737 | 0.441 | 0.520 |
| Epochs 5 | 0.820 | 0.500 | 0.969 | 0.848 | 0.456 | 0.540 |
| Epochs 10 | 0.795 | 0.491 | 0.964 | 0.683 | 0.471 | 0.496 |
| Epochs 20 | 0.799 | 0.497 | 0.962 | 0.659 | 0.956 | 0.486 |
| Balanced 60\|40 | 0.796 | 0.504 | 0.962 | 0.643 | 0.529 | 0.490 |
| Balanced 67\|33 | 0.810 | 0.497 | 0.963 | 0.667 | 0.676 | 0.494 |
| Balanced 75\|25 | 0.837 | 0.499 | 0.964 | 0.683 | 0.471 | 0.496 |
| Balanced 80\|20 | 0.806 | 0.514 | 0.965 | 0.718 | 0.662 | 0.507 |
| Balanced 40\|30\|30 | 0.780 | 0.485 | 0.961 | 0.622 | 0.971 | 0.476 |
| Balanced 60\|20\|20 | 0.798 | 0.484 | 0.964 | 0.674 | 1.000 | 0.505 |
| Balanced 80\|10\|10 | 0.812 | 0.503 | 0.962 | 0.651 | 0.912 | 0.494 |
| Balanced 60\|30\|10 | 0.806 | 0.504 | 0.962 | 0.636 | 0.971 | 0.481 |
| Balanced 30\|30\|15\|15\|10 | 0.814 | 0.487 | 0.965 | 0.707 | 0.971 | 0.515 |
| Balanced 60\|10\|10\|10\|10 | 0.797 | 0.496 | 0.966 | 0.750 | 0.971 | 0.503 |

Table 2.12: Best results for plain averaging on NR-AR (Tox21)

| experiment | roc_auc | pr_auc | accuracy | precision | recall | kappa |
|---|---|---|---|---|---|---|
| Baseline | 0.813 | 0.524 | 0.971 | 0.929 | 0.471 | 0.567 |

| | | | | | | |
|---|---|---|---|---|---|---|
| 60\|40 | 0.810 | 0.502 | 0.961 | 0.622 | 0.471 | 0.493 |
| 67\|33 | 0.819 | 0.492 | 0.965 | 0.707 | 0.471 | 0.518 |
| 75\|25 | 0.805 | 0.492 | 0.964 | 0.690 | 0.500 | 0.510 |
| 80\|20 | 0.810 | 0.496 | 0.967 | 0.778 | 0.485 | 0.523 |
| 40\|30\|30 | 0.827 | 0.502 | 0.964 | 0.674 | 1.000 | 0.505 |
| 60\|20\|20 | 0.814 | 0.481 | 0.962 | 0.651 | 0.971 | 0.498 |
| 80\|10\|10 | 0.801 | 0.497 | 0.966 | 0.732 | 0.485 | 0.534 |
| 60\|30\|10 | 0.826 | 0.504 | 0.964 | 0.711 | 0.456 | 0.492 |
| 30\|30\|15\|15\|10 | 0.822 | 0.489 | 0.964 | 0.674 | 0.971 | 0.505 |
| 60\|10\|10\|10\|10 | 0.815 | 0.510 | 0.966 | 0.737 | 0.779 | 0.512 |

Table 2.13: Best results for weighted averaging on NR-AR (Tox21)

| experiment | roc_auc | pr_auc | accuracy | precision | recall | kappa |
|---|---|---|---|---|---|---|
| Baseline | 0.813 | 0.524 | 0.971 | 0.929 | 0.471 | 0.567 |
| Clients 2 | 0.802 | 0.483 | 0.965 | 0.730 | 1.000 | 0.498 |
| Clients 3 | 0.800 | 0.495 | 0.961 | 0.628 | 1.000 | 0.471 |
| Clients 4 | 0.805 | 0.513 | 0.962 | 0.644 | 1.000 | 0.503 |
| Clients 5 | 0.791 | 0.500 | 0.957 | 0.558 | 1.000 | 0.462 |
| Clients 6 | 0.802 | 0.489 | 0.966 | 0.757 | 1.000 | 0.517 |
| Clients 7 | 0.791 | 0.483 | 0.959 | 0.610 | 1.000 | 0.475 |
| Clients 8 | 0.796 | 0.494 | 0.959 | 0.588 | 1.000 | 0.484 |
| Clients 9 | 0.802 | 0.495 | 0.958 | 0.569 | 1.000 | 0.466 |
| Clients 10 | 0.797 | 0.504 | 0.959 | 0.580 | 1.000 | 0.471 |
| Clients 11 | 0.790 | 0.486 | 0.953 | 0.500 | 1.000 | 0.437 |
| Clients 12 | 0.799 | 0.485 | 0.954 | 0.509 | 1.000 | 0.444 |
| Clients 13 | 0.799 | 0.491 | 0.953 | 0.476 | 1.000 | 0.433 |
| Clients 14 | 0.785 | 0.480 | 0.957 | 0.547 | 1.000 | 0.474 |
| Clients 15 | 0.790 | 0.471 | 0.955 | 0.518 | 1.000 | 0.444 |
| Clients 16 | 0.805 | 0.478 | 0.944 | 0.408 | 1.000 | 0.393 |
| Clients 17 | 0.797 | 0.483 | 0.951 | 0.470 | 1.000 | 0.437 |
| Clients 18 | 0.787 | 0.474 | 0.960 | 0.600 | 1.000 | 0.488 |
| Clients 19 | 0.790 | 0.467 | 0.944 | 0.397 | 1.000 | 0.383 |

| | | | | | | |
|---|---|---|---|---|---|---|
| Clients 20 | 0.778 | 0.466 | 0.961 | 0.622 | 1.000 | 0.485 |
| Epochs 1 | 0.803 | 0.492 | 0.968 | 0.844 | 0.471 | 0.529 |
| Epochs 2 | 0.806 | 0.507 | 0.967 | 0.794 | 0.471 | 0.514 |
| Epochs 5 | 0.788 | 0.487 | 0.968 | 0.818 | 0.662 | 0.523 |
| Epochs 10 | 0.810 | 0.508 | 0.964 | 0.700 | 1.000 | 0.505 |
| Epochs 20 | 0.797 | 0.499 | 0.967 | 0.763 | 1.000 | 0.531 |
| Balanced 60\|40 | 0.805 | 0.503 | 0.965 | 0.718 | 0.941 | 0.515 |
| Balanced 67\|33 | 0.795 | 0.501 | 0.966 | 0.737 | 0.971 | 0.512 |
| Balanced 75\|25 | 0.786 | 0.487 | 0.963 | 0.659 | 0.971 | 0.499 |
| Balanced 80\|20 | 0.789 | 0.499 | 0.966 | 0.730 | 0.956 | 0.520 |
| Balanced 40\|30\|30 | 0.801 | 0.494 | 0.964 | 0.667 | 1.000 | 0.513 |
| Balanced 60\|20\|20 | 0.799 | 0.499 | 0.962 | 0.630 | 1.000 | 0.490 |
| Balanced 80\|10\|10 | 0.805 | 0.489 | 0.965 | 0.718 | 1.000 | 0.507 |
| Balanced 60\|30\|10 | 0.793 | 0.503 | 0.961 | 0.622 | 1.000 | 0.476 |
| Balanced 30\|30\|15\|15\|10 | 0.816 | 0.524 | 0.966 | 0.750 | 1.000 | 0.511 |
| Balanced 60\|10\|10\|10\|10 | 0.803 | 0.483 | 0.962 | 0.659 | 1.000 | 0.477 |

Table 2.14: Best results for benchmarked averaging on NR-AR (Tox21)

### 2.4.3 Tox21 - NR-AR-LBD

| experiment | roc_auc | pr_auc | accuracy | precision | recall | kappa |
|---|---|---|---|---|---|---|
| Baseline | 0.908 | 0.610 | 0.980 | 0.730 | 0.711 | 0.678 |
| Clients 2 | 0.896 | 0.594 | 0.976 | 0.630 | 0.711 | 0.625 |
| Clients 3 | 0.899 | 0.591 | 0.973 | 0.577 | 1.000 | 0.605 |
| Clients 4 | 0.906 | 0.593 | 0.972 | 0.554 | 1.000 | 0.599 |
| Clients 5 | 0.893 | 0.633 | 0.976 | 0.620 | 1.000 | 0.640 |
| Clients 6 | 0.872 | 0.593 | 0.972 | 0.569 | 1.000 | 0.598 |
| Clients 7 | 0.884 | 0.585 | 0.972 | 0.554 | 1.000 | 0.599 |
| Clients 8 | 0.893 | 0.576 | 0.967 | 0.500 | 1.000 | 0.555 |
| Clients 9 | 0.873 | 0.613 | 0.972 | 0.564 | 1.000 | 0.606 |
| Clients 10 | 0.875 | 0.572 | 0.972 | 0.558 | 1.000 | 0.593 |
| Clients 11 | 0.877 | 0.570 | 0.977 | 0.646 | 1.000 | 0.655 |
| Clients 12 | 0.872 | 0.576 | 0.969 | 0.526 | 1.000 | 0.581 |

| | | | | | | |
|---|---|---|---|---|---|---|
| Clients 13 | 0.884 | 0.613 | 0.967 | 0.500 | 1.000 | 0.565 |
| Clients 14 | 0.888 | 0.594 | 0.974 | 0.596 | 1.000 | 0.627 |
| Clients 15 | 0.879 | 0.596 | 0.970 | 0.536 | 1.000 | 0.581 |
| Clients 16 | 0.894 | 0.571 | 0.959 | 0.395 | 1.000 | 0.474 |
| Clients 17 | 0.863 | 0.575 | 0.969 | 0.518 | 1.000 | 0.558 |
| Clients 18 | 0.883 | 0.598 | 0.972 | 0.556 | 1.000 | 0.591 |
| Clients 19 | 0.892 | 0.572 | 0.973 | 0.583 | 1.000 | 0.591 |
| Clients 20 | 0.866 | 0.589 | 0.970 | 1.000 | 1.000 | 0.579 |
| Epochs 1 | 0.906 | 0.649 | 0.979 | 0.682 | 0.733 | 0.663 |
| Epochs 2 | 0.892 | 0.624 | 0.977 | 0.659 | 0.733 | 0.655 |
| Epochs 5 | 0.900 | 0.598 | 0.975 | 0.608 | 0.733 | 0.640 |
| Epochs 10 | 0.906 | 0.652 | 0.975 | 0.622 | 0.733 | 0.626 |
| Epochs 20 | 0.912 | 0.659 | 0.974 | 0.600 | 0.933 | 0.599 |
| Balanced 60\|40 | 0.901 | 0.602 | 0.977 | 0.633 | 0.711 | 0.648 |
| Balanced 67\|33 | 0.897 | 0.606 | 0.977 | 0.640 | 0.956 | 0.662 |
| Balanced 75\|25 | 0.887 | 0.613 | 0.978 | 0.674 | 0.756 | 0.648 |
| Balanced 80\|20 | 0.890 | 0.611 | 0.980 | 0.698 | 0.711 | 0.671 |
| Balanced 40\|30\|30 | 0.889 | 0.591 | 0.976 | 0.625 | 1.000 | 0.633 |
| Balanced 60\|20\|20 | 0.883 | 0.566 | 0.978 | 0.674 | 1.000 | 0.648 |
| Balanced 80\|10\|10 | 0.895 | 0.601 | 0.976 | 0.630 | 0.689 | 0.625 |
| Balanced 60\|30\|10 | 0.897 | 0.596 | 0.975 | 0.617 | 1.000 | 0.618 |
| Balanced 30\|30\|15\|15\|10 | 0.883 | 0.602 | 0.979 | 0.682 | 1.000 | 0.663 |
| Balanced 60\|10\|10\|10\|10 | 0.874 | 0.584 | 0.979 | 0.682 | 1.000 | 0.663 |

Table 2.15: Best results for plain averaging on NR-AR-LBD (Tox21)

| experiment | roc_auc | pr_auc | accuracy | precision | recall | kappa |
|---|---|---|---|---|---|---|
| Baseline | 0.908 | 0.610 | 0.980 | 0.730 | 0.711 | 0.678 |
| 60\|40 | 0.889 | 0.589 | 0.976 | 0.625 | 0.689 | 0.633 |
| 67\|33 | 0.909 | 0.612 | 0.977 | 0.644 | 0.711 | 0.640 |
| 75\|25 | 0.911 | 0.605 | 0.976 | 0.625 | 0.733 | 0.633 |
| 80\|20 | 0.901 | 0.599 | 0.977 | 0.638 | 0.711 | 0.640 |
| 40\|30\|30 | 0.901 | 0.630 | 0.974 | 0.592 | 0.689 | 0.603 |
| 60\|20\|20 | 0.906 | 0.600 | 0.975 | 0.612 | 1.000 | 0.625 |

| | | | | | | |
|---|---|---|---|---|---|---|
| 80\|10\|10 | 0.919 | 0.619 | 0.980 | 0.714 | 0.733 | 0.671 |
| 60\|30\|10 | 0.896 | 0.582 | 0.974 | 0.592 | 0.667 | 0.605 |
| 30\|30\|15\|15\|10 | 0.885 | 0.616 | 0.977 | 0.638 | 1.000 | 0.640 |
| 60\|10\|10\|10\|10 | 0.898 | 0.616 | 0.978 | 0.683 | 0.978 | 0.640 |

Table 2.16: Best results for weighted averaging on NR-AR-LBD (Tox21)

| experiment | roc_auc | pr_auc | accuracy | precision | recall | kappa |
|---|---|---|---|---|---|---|
| Baseline | 0.908 | 0.610 | 0.980 | 0.730 | 0.711 | 0.678 |
| Clients 2 | 0.893 | 0.587 | 0.975 | 0.612 | 1.000 | 0.625 |
| Clients 3 | 0.899 | 0.626 | 0.970 | 0.536 | 1.000 | 0.587 |
| Clients 4 | 0.900 | 0.590 | 0.974 | 0.585 | 1.000 | 0.619 |
| Clients 5 | 0.918 | 0.612 | 0.974 | 0.593 | 1.000 | 0.633 |
| Clients 6 | 0.889 | 0.595 | 0.969 | 0.517 | 1.000 | 0.567 |
| Clients 7 | 0.890 | 0.577 | 0.967 | 0.500 | 1.000 | 0.555 |
| Clients 8 | 0.890 | 0.607 | 0.966 | 0.492 | 1.000 | 0.557 |
| Clients 9 | 0.890 | 0.597 | 0.968 | 0.508 | 1.000 | 0.561 |
| Clients 10 | 0.886 | 0.595 | 0.972 | 0.556 | 1.000 | 0.591 |
| Clients 11 | 0.877 | 0.576 | 0.969 | 0.517 | 1.000 | 0.567 |
| Clients 12 | 0.879 | 0.602 | 0.964 | 0.463 | 1.000 | 0.535 |
| Clients 13 | 0.866 | 0.616 | 0.967 | 0.500 | 1.000 | 0.563 |
| Clients 14 | 0.885 | 0.601 | 0.973 | 0.574 | 1.000 | 0.612 |
| Clients 15 | 0.878 | 0.578 | 0.966 | 0.484 | 1.000 | 0.543 |
| Clients 16 | 0.900 | 0.564 | 0.964 | 0.462 | 1.000 | 0.527 |
| Clients 17 | 0.870 | 0.598 | 0.966 | 0.420 | 1.000 | 0.488 |
| Clients 18 | 0.860 | 0.593 | 0.973 | 0.583 | 1.000 | 0.588 |
| Clients 19 | 0.858 | 0.571 | 0.964 | 0.408 | 1.000 | 0.491 |
| Clients 20 | 0.895 | 0.599 | 0.974 | 0.609 | 1.000 | 0.602 |
| Epochs 1 | 0.908 | 0.668 | 0.977 | 0.651 | 0.756 | 0.624 |
| Epochs 2 | 0.901 | 0.627 | 0.978 | 0.675 | 0.733 | 0.655 |
| Epochs 5 | 0.909 | 0.630 | 0.979 | 0.690 | 1.000 | 0.656 |
| Epochs 10 | 0.903 | 0.606 | 0.976 | 0.625 | 1.000 | 0.633 |
| Epochs 20 | 0.906 | 0.604 | 0.971 | 0.545 | 1.000 | 0.585 |

| | | | | | | |
|---|---|---|---|---|---|---|
| Balanced 60\|40 | 0.885 | 0.624 | 0.976 | 0.625 | 1.000 | 0.640 |
| Balanced 67\|33 | 0.899 | 0.623 | 0.979 | 0.682 | 1.000 | 0.663 |
| Balanced 75\|25 | 0.897 | 0.583 | 0.974 | 0.604 | 1.000 | 0.610 |
| Balanced 80\|20 | 0.881 | 0.629 | 0.975 | 0.608 | 0.978 | 0.633 |
| Balanced 40\|30\|30 | 0.921 | 0.643 | 0.973 | 0.574 | 1.000 | 0.612 |
| Balanced 60\|20\|20 | 0.892 | 0.589 | 0.974 | 0.593 | 1.000 | 0.633 |
| Balanced 80\|10\|10 | 0.878 | 0.601 | 0.977 | 0.633 | 1.000 | 0.648 |
| Balanced 60\|30\|10 | 0.921 | 0.592 | 0.975 | 0.617 | 1.000 | 0.618 |
| Balanced 30\|30\|15\|15\|10 | 0.907 | 0.618 | 0.978 | 0.667 | 1.000 | 0.655 |
| Balanced 60\|10\|10\|10\|10 | 0.886 | 0.581 | 0.980 | 0.707 | 1.000 | 0.664 |

Table 2.17: Best results for benchmarked averaging on NR-AR-LBD (Tox21)

### 2.4.4 Tox21 - NR-AhR

| experiment | roc_auc | pr_auc | accuracy | precision | recall | kappa |
|---|---|---|---|---|---|---|
| Baseline | 0.900 | 0.659 | 0.909 | 0.655 | 0.840 | 0.579 |
| Clients 2 | 0.893 | 0.645 | 0.899 | 0.565 | 0.994 | 0.543 |
| Clients 3 | 0.894 | 0.669 | 0.907 | 0.598 | 1.000 | 0.569 |
| Clients 4 | 0.896 | 0.645 | 0.895 | 0.545 | 1.000 | 0.555 |
| Clients 5 | 0.890 | 0.673 | 0.897 | 0.555 | 1.000 | 0.551 |
| Clients 6 | 0.888 | 0.647 | 0.901 | 0.571 | 1.000 | 0.550 |
| Clients 7 | 0.886 | 0.673 | 0.898 | 0.556 | 1.000 | 0.548 |
| Clients 8 | 0.896 | 0.647 | 0.883 | 0.505 | 1.000 | 0.525 |
| Clients 9 | 0.894 | 0.638 | 0.886 | 0.512 | 1.000 | 0.537 |
| Clients 10 | 0.883 | 0.632 | 0.892 | 0.532 | 1.000 | 0.538 |
| Clients 11 | 0.886 | 0.650 | 0.889 | 0.521 | 1.000 | 0.536 |
| Clients 12 | 0.884 | 0.621 | 0.889 | 0.522 | 1.000 | 0.532 |
| Clients 13 | 0.888 | 0.649 | 0.894 | 0.540 | 1.000 | 0.551 |
| Clients 14 | 0.886 | 0.650 | 0.895 | 0.547 | 1.000 | 0.544 |
| Clients 15 | 0.887 | 0.619 | 0.882 | 0.500 | 1.000 | 0.525 |
| Clients 16 | 0.890 | 0.619 | 0.847 | 0.418 | 1.000 | 0.456 |
| Clients 17 | 0.890 | 0.651 | 0.898 | 0.554 | 1.000 | 0.555 |

| | | | | | | |
|---|---|---|---|---|---|---|
| Clients 18 | 0.889 | 0.633 | 0.883 | 0.504 | 1.000 | 0.531 |
| Clients 19 | 0.886 | 0.613 | 0.897 | 0.556 | 1.000 | 0.564 |
| Clients 20 | 0.886 | 0.628 | 0.895 | 0.547 | 1.000 | 0.537 |
| Epochs 1 | 0.901 | 0.671 | 0.914 | 0.631 | 0.968 | 0.590 |
| Epochs 2 | 0.896 | 0.672 | 0.905 | 0.583 | 1.000 | 0.573 |
| Epochs 5 | 0.897 | 0.660 | 0.899 | 0.564 | 0.840 | 0.546 |
| Epochs 10 | 0.896 | 0.651 | 0.901 | 0.573 | 0.968 | 0.562 |
| Epochs 20 | 0.888 | 0.648 | 0.907 | 0.615 | 1.000 | 0.566 |
| Balanced 60\|40 | 0.894 | 0.681 | 0.903 | 0.573 | 0.801 | 0.577 |
| Balanced 67\|33 | 0.888 | 0.638 | 0.904 | 0.597 | 1.000 | 0.559 |
| Balanced 75\|25 | 0.894 | 0.663 | 0.908 | 0.604 | 0.782 | 0.566 |
| Balanced 80\|20 | 0.896 | 0.680 | 0.906 | 0.596 | 0.763 | 0.562 |
| Balanced 40\|30\|30 | 0.899 | 0.663 | 0.905 | 0.591 | 1.000 | 0.564 |
| Balanced 60\|20\|20 | 0.889 | 0.658 | 0.903 | 0.579 | 1.000 | 0.562 |
| Balanced 80\|10\|10 | 0.890 | 0.664 | 0.902 | 0.578 | 1.000 | 0.548 |
| Balanced 60\|30\|10 | 0.893 | 0.632 | 0.905 | 0.586 | 1.000 | 0.575 |
| Balanced 30\|30\|15\|15\|10 | 0.892 | 0.681 | 0.904 | 0.584 | 1.000 | 0.564 |
| Balanced 60\|10\|10\|10\|10 | 0.882 | 0.646 | 0.889 | 0.523 | 1.000 | 0.521 |

Table 2.18: Best results for plain averaging on NR-AhR (Tox21)

| experiment | roc_auc | pr_auc | accuracy | precision | recall | kappa |
|---|---|---|---|---|---|---|
| Baseline | 0.900 | 0.659 | 0.909 | 0.655 | 0.840 | 0.579 |
| 60\|40 | 0.897 | 0.688 | 0.904 | 0.584 | 0.929 | 0.575 |
| 67\|33 | 0.891 | 0.646 | 0.911 | 0.624 | 0.962 | 0.576 |
| 75\|25 | 0.893 | 0.663 | 0.905 | 0.590 | 0.897 | 0.566 |
| 80\|20 | 0.895 | 0.645 | 0.902 | 0.575 | 0.814 | 0.550 |
| 40\|30\|30 | 0.887 | 0.657 | 0.906 | 0.587 | 1.000 | 0.582 |
| 60\|20\|20 | 0.892 | 0.674 | 0.903 | 0.582 | 1.000 | 0.566 |
| 80\|10\|10 | 0.891 | 0.645 | 0.900 | 0.568 | 0.897 | 0.550 |
| 60\|30\|10 | 0.887 | 0.655 | 0.910 | 0.612 | 0.776 | 0.578 |
| 30\|30\|15\|15\|10 | 0.900 | 0.668 | 0.895 | 0.543 | 1.000 | 0.551 |
| 60\|10\|10\|10\|10 | 0.901 | 0.678 | 0.905 | 0.585 | 0.776 | 0.582 |

Table 2.19: Best results for weighted averaging on NR-AhR (Tox21)

| experiment | roc_auc | pr_auc | accuracy | precision | recall | kappa |
|---|---|---|---|---|---|---|
| Baseline | 0.900 | 0.659 | 0.909 | 0.655 | 0.840 | 0.579 |
| Clients 2 | 0.884 | 0.660 | 0.910 | 0.616 | 1.000 | 0.575 |
| Clients 3 | 0.888 | 0.657 | 0.890 | 0.530 | 1.000 | 0.527 |
| Clients 4 | 0.885 | 0.653 | 0.897 | 0.552 | 1.000 | 0.553 |
| Clients 5 | 0.889 | 0.634 | 0.885 | 0.510 | 1.000 | 0.527 |
| Clients 6 | 0.888 | 0.657 | 0.885 | 0.509 | 1.000 | 0.525 |
| Clients 7 | 0.888 | 0.654 | 0.892 | 0.535 | 1.000 | 0.542 |
| Clients 8 | 0.895 | 0.667 | 0.892 | 0.534 | 1.000 | 0.545 |
| Clients 9 | 0.891 | 0.650 | 0.896 | 0.548 | 1.000 | 0.555 |
| Clients 10 | 0.889 | 0.678 | 0.886 | 0.514 | 1.000 | 0.528 |
| Clients 11 | 0.891 | 0.649 | 0.889 | 0.522 | 1.000 | 0.534 |
| Clients 12 | 0.894 | 0.645 | 0.885 | 0.509 | 1.000 | 0.529 |
| Clients 13 | 0.893 | 0.662 | 0.869 | 0.465 | 1.000 | 0.491 |
| Clients 14 | 0.887 | 0.646 | 0.882 | 0.500 | 1.000 | 0.525 |
| Clients 15 | 0.887 | 0.634 | 0.888 | 0.519 | 1.000 | 0.532 |
| Clients 16 | 0.890 | 0.653 | 0.873 | 0.478 | 1.000 | 0.518 |
| Clients 17 | 0.889 | 0.635 | 0.875 | 0.480 | 1.000 | 0.519 |
| Clients 18 | 0.888 | 0.650 | 0.883 | 0.502 | 1.000 | 0.523 |
| Clients 19 | 0.897 | 0.646 | 0.871 | 0.470 | 1.000 | 0.508 |
| Clients 20 | 0.889 | 0.615 | 0.889 | 0.525 | 1.000 | 0.532 |
| Epochs 1 | 0.903 | 0.682 | 0.905 | 0.587 | 0.853 | 0.562 |
| Epochs 2 | 0.893 | 0.660 | 0.902 | 0.575 | 0.814 | 0.555 |
| Epochs 5 | 0.892 | 0.667 | 0.903 | 0.575 | 1.000 | 0.571 |
| Epochs 10 | 0.891 | 0.675 | 0.906 | 0.593 | 1.000 | 0.577 |
| Epochs 20 | 0.889 | 0.662 | 0.908 | 0.616 | 1.000 | 0.576 |
| Balanced 60\|40 | 0.889 | 0.663 | 0.901 | 0.565 | 1.000 | 0.573 |
| Balanced 67\|33 | 0.890 | 0.664 | 0.899 | 0.562 | 1.000 | 0.566 |
| Balanced 75\|25 | 0.900 | 0.659 | 0.905 | 0.584 | 1.000 | 0.580 |
| Balanced 80\|20 | 0.896 | 0.661 | 0.904 | 0.584 | 1.000 | 0.559 |
| Balanced 40\|30\|30 | 0.889 | 0.646 | 0.903 | 0.574 | 1.000 | 0.573 |
| Balanced 60\|20\|20 | 0.884 | 0.634 | 0.888 | 0.519 | 1.000 | 0.530 |
| Balanced 80\|10\|10 | 0.900 | 0.669 | 0.905 | 0.587 | 1.000 | 0.573 |

| | | | | | | |
|---|---|---|---|---|---|---|
| Balanced 60\|30\|10 | 0.894 | 0.665 | 0.886 | 0.514 | 1.000 | 0.537 |
| Balanced 30\|30\|15\|15\|10 | 0.891 | 0.662 | 0.888 | 0.519 | 1.000 | 0.545 |
| Balanced 60\|10\|10\|10\|10 | 0.896 | 0.659 | 0.895 | 0.548 | 1.000 | 0.545 |

Table 2.20: Best results for benchmarked averaging on NR-AhR (Tox21)

### 2.4.5 Tox21 - NR-Aromatase

| experiment | roc_auc | pr_auc | accuracy | precision | recall | kappa |
|---|---|---|---|---|---|---|
| Baseline | 0.913 | 0.534 | 0.953 | 0.556 | 0.694 | 0.492 |
| Clients 2 | 0.893 | 0.523 | 0.946 | 0.485 | 0.968 | 0.479 |
| Clients 3 | 0.881 | 0.530 | 0.942 | 0.451 | 0.984 | 0.450 |
| Clients 4 | 0.875 | 0.518 | 0.942 | 0.451 | 1.000 | 0.453 |
| Clients 5 | 0.859 | 0.499 | 0.934 | 0.407 | 1.000 | 0.439 |
| Clients 6 | 0.878 | 0.508 | 0.940 | 0.443 | 0.984 | 0.465 |
| Clients 7 | 0.870 | 0.478 | 0.931 | 0.383 | 1.000 | 0.398 |
| Clients 8 | 0.867 | 0.486 | 0.926 | 0.341 | 1.000 | 0.361 |
| Clients 9 | 0.884 | 0.499 | 0.931 | 0.375 | 1.000 | 0.389 |
| Clients 10 | 0.876 | 0.503 | 0.939 | 0.362 | 1.000 | 0.378 |
| Clients 11 | 0.872 | 0.494 | 0.927 | 0.330 | 1.000 | 0.375 |
| Clients 12 | 0.856 | 0.453 | 0.926 | 0.315 | 1.000 | 0.357 |
| Clients 13 | 0.858 | 0.476 | 0.941 | 0.295 | 1.000 | 0.337 |
| Clients 14 | 0.838 | 0.441 | 0.941 | 0.306 | 1.000 | 0.332 |
| Clients 15 | 0.865 | 0.427 | 0.911 | 0.264 | 1.000 | 0.304 |
| Clients 16 | 0.870 | 0.472 | 0.911 | 0.227 | 1.000 | 0.284 |
| Clients 17 | 0.845 | 0.431 | 0.943 | 0.280 | 1.000 | 0.327 |
| Clients 18 | 0.848 | 0.421 | 0.945 | 0.286 | 1.000 | 0.291 |
| Clients 19 | 0.855 | 0.458 | 0.937 | 0.298 | 1.000 | 0.342 |
| Clients 20 | 0.871 | 0.419 | 0.948 | 0.500 | 1.000 | 0.371 |
| Epochs 1 | 0.893 | 0.498 | 0.939 | 0.421 | 0.645 | 0.431 |
| Epochs 2 | 0.903 | 0.532 | 0.941 | 0.444 | 0.613 | 0.446 |
| Epochs 5 | 0.879 | 0.543 | 0.946 | 0.487 | 0.629 | 0.508 |
| Epochs 10 | 0.889 | 0.519 | 0.945 | 0.474 | 0.629 | 0.439 |

| | | | | | | |
|---|---|---|---|---|---|---|
| Epochs 20 | 0.881 | 0.540 | 0.947 | 0.493 | 1.000 | 0.483 |
| Balanced 60\|40 | 0.876 | 0.500 | 0.942 | 0.453 | 1.000 | 0.466 |
| Balanced 67\|33 | 0.906 | 0.552 | 0.942 | 0.453 | 1.000 | 0.466 |
| Balanced 75\|25 | 0.882 | 0.512 | 0.940 | 0.440 | 0.645 | 0.461 |
| Balanced 80\|20 | 0.871 | 0.533 | 0.940 | 0.437 | 0.597 | 0.434 |
| Balanced 40\|30\|30 | 0.883 | 0.497 | 0.942 | 0.455 | 1.000 | 0.446 |
| Balanced 60\|20\|20 | 0.876 | 0.511 | 0.947 | 0.492 | 0.968 | 0.460 |
| Balanced 80\|10\|10 | 0.869 | 0.510 | 0.942 | 0.457 | 0.903 | 0.458 |
| Balanced 60\|30\|10 | 0.874 | 0.531 | 0.941 | 0.449 | 1.000 | 0.469 |
| Balanced 30\|30\|15\|15\|10 | 0.871 | 0.482 | 0.933 | 0.384 | 1.000 | 0.390 |
| Balanced 60\|10\|10\|10\|10 | 0.867 | 0.536 | 0.947 | 0.493 | 1.000 | 0.498 |

Table 2.21: Best results for plain averaging on NR-Aromatase (Tox21)

| experiment | roc_auc | pr_auc | accuracy | precision | recall | kappa |
|---|---|---|---|---|---|---|
| Baseline | 0.913 | 0.534 | 0.953 | 0.556 | 0.694 | 0.492 |
| 60\|40 | 0.887 | 0.489 | 0.946 | 0.485 | 0.839 | 0.471 |
| 67\|33 | 0.869 | 0.476 | 0.943 | 0.456 | 0.871 | 0.407 |
| 75\|25 | 0.874 | 0.484 | 0.946 | 0.484 | 0.581 | 0.455 |
| 80\|20 | 0.881 | 0.520 | 0.937 | 0.419 | 0.613 | 0.427 |
| 40\|30\|30 | 0.884 | 0.496 | 0.933 | 0.395 | 0.968 | 0.413 |
| 60\|20\|20 | 0.881 | 0.509 | 0.936 | 0.405 | 0.935 | 0.407 |
| 80\|10\|10 | 0.872 | 0.473 | 0.943 | 0.458 | 0.597 | 0.426 |
| 60\|30\|10 | 0.873 | 0.528 | 0.948 | 0.508 | 0.581 | 0.477 |
| 30\|30\|15\|15\|10 | 0.864 | 0.447 | 0.942 | 0.453 | 1.000 | 0.435 |
| 60\|10\|10\|10\|10 | 0.892 | 0.505 | 0.943 | 0.462 | 0.871 | 0.443 |

Table 2.22: Best results for weighted averaging on NR-Aromatase (Tox21)

| experiment | roc_auc | pr_auc | accuracy | precision | recall | kappa |
|---|---|---|---|---|---|---|
| Baseline | 0.913 | 0.534 | 0.953 | 0.556 | 0.694 | 0.492 |
| Clients 2 | 0.883 | 0.529 | 0.937 | 0.418 | 1.000 | 0.435 |

| | | | | | | |
|---|---|---|---|---|---|---|
| Clients 3 | 0.885 | 0.461 | 0.937 | 0.416 | 1.000 | 0.431 |
| Clients 4 | 0.890 | 0.536 | 0.942 | 0.451 | 1.000 | 0.450 |
| Clients 5 | 0.879 | 0.462 | 0.926 | 0.337 | 1.000 | 0.348 |
| Clients 6 | 0.874 | 0.417 | 0.904 | 0.284 | 1.000 | 0.340 |
| Clients 7 | 0.873 | 0.459 | 0.921 | 0.343 | 1.000 | 0.383 |
| Clients 8 | 0.878 | 0.452 | 0.931 | 0.375 | 1.000 | 0.386 |
| Clients 9 | 0.887 | 0.456 | 0.922 | 0.337 | 1.000 | 0.363 |
| Clients 10 | 0.869 | 0.415 | 0.899 | 0.258 | 1.000 | 0.305 |
| Clients 11 | 0.876 | 0.416 | 0.926 | 0.305 | 1.000 | 0.329 |
| Clients 12 | 0.887 | 0.441 | 0.904 | 0.289 | 1.000 | 0.349 |
| Clients 13 | 0.873 | 0.449 | 0.928 | 0.300 | 1.000 | 0.334 |
| Clients 14 | 0.861 | 0.434 | 0.937 | 0.295 | 1.000 | 0.334 |
| Clients 15 | 0.871 | 0.402 | 0.909 | 0.261 | 1.000 | 0.299 |
| Clients 16 | 0.861 | 0.409 | 0.904 | 0.272 | 1.000 | 0.323 |
| Clients 17 | 0.840 | 0.410 | 0.946 | 0.333 | 1.000 | 0.284 |
| Clients 18 | 0.860 | 0.456 | 0.944 | 0.268 | 1.000 | 0.297 |
| Clients 19 | 0.859 | 0.440 | 0.925 | 0.248 | 1.000 | 0.299 |
| Clients 20 | 0.854 | 0.398 | 0.937 | 0.241 | 1.000 | 0.280 |
| Epochs 1 | 0.909 | 0.520 | 0.942 | 0.455 | 0.645 | 0.438 |
| Epochs 2 | 0.907 | 0.544 | 0.940 | 0.434 | 0.710 | 0.464 |
| Epochs 5 | 0.902 | 0.535 | 0.947 | 0.492 | 0.645 | 0.476 |
| Epochs 10 | 0.880 | 0.485 | 0.942 | 0.449 | 1.000 | 0.442 |
| Epochs 20 | 0.887 | 0.529 | 0.949 | 0.517 | 1.000 | 0.481 |
| Balanced 60\|40 | 0.894 | 0.477 | 0.939 | 0.429 | 1.000 | 0.439 |
| Balanced 67\|33 | 0.901 | 0.542 | 0.942 | 0.463 | 1.000 | 0.491 |
| Balanced 75\|25 | 0.906 | 0.539 | 0.950 | 0.523 | 0.984 | 0.509 |
| Balanced 80\|20 | 0.894 | 0.537 | 0.941 | 0.433 | 1.000 | 0.431 |
| Balanced 40\|30\|30 | 0.889 | 0.459 | 0.943 | 0.453 | 1.000 | 0.410 |
| Balanced 60\|20\|20 | 0.882 | 0.481 | 0.925 | 0.341 | 1.000 | 0.369 |
| Balanced 80\|10\|10 | 0.877 | 0.511 | 0.934 | 0.392 | 1.000 | 0.392 |
| Balanced 60\|30\|10 | 0.886 | 0.498 | 0.933 | 0.387 | 1.000 | 0.390 |
| Balanced 30\|30\|15\|15\|10 | 0.891 | 0.493 | 0.928 | 0.371 | 1.000 | 0.411 |
| Balanced 60\|10\|10\|10\|10 | 0.893 | 0.517 | 0.932 | 0.393 | 1.000 | 0.418 |

Table 2.23: Best results for benchmarked averaging on NR-Aromatase (Tox21)

### 2.4.6 Tox21 - NR-ER

| experiment | roc_auc | pr_auc | accuracy | precision | recall | kappa |
|---|---|---|---|---|---|---|
| Baseline | 0.738 | 0.444 | 0.865 | 0.465 | 0.699 | 0.335 |
| Clients 2 | 0.728 | 0.437 | 0.818 | 0.340 | 1.000 | 0.283 |
| Clients 3 | 0.719 | 0.443 | 0.816 | 0.340 | 1.000 | 0.294 |
| Clients 4 | 0.713 | 0.430 | 0.807 | 0.322 | 1.000 | 0.273 |
| Clients 5 | 0.716 | 0.440 | 0.841 | 0.367 | 1.000 | 0.293 |
| Clients 6 | 0.724 | 0.445 | 0.827 | 0.356 | 1.000 | 0.307 |
| Clients 7 | 0.719 | 0.439 | 0.825 | 0.359 | 1.000 | 0.302 |
| Clients 8 | 0.726 | 0.410 | 0.827 | 0.305 | 1.000 | 0.259 |
| Clients 9 | 0.734 | 0.431 | 0.838 | 0.346 | 1.000 | 0.275 |
| Clients 10 | 0.734 | 0.416 | 0.860 | 0.379 | 1.000 | 0.261 |
| Clients 11 | 0.727 | 0.416 | 0.820 | 0.338 | 1.000 | 0.270 |
| Clients 12 | 0.721 | 0.408 | 0.807 | 0.321 | 1.000 | 0.265 |
| Clients 13 | 0.721 | 0.416 | 0.861 | 0.414 | 1.000 | 0.273 |
| Clients 14 | 0.726 | 0.405 | 0.828 | 0.319 | 1.000 | 0.261 |
| Clients 15 | 0.730 | 0.426 | 0.803 | 0.319 | 1.000 | 0.273 |
| Clients 16 | 0.723 | 0.403 | 0.765 | 0.260 | 1.000 | 0.226 |
| Clients 17 | 0.710 | 0.411 | 0.847 | 0.330 | 1.000 | 0.254 |
| Clients 18 | 0.723 | 0.410 | 0.834 | 0.353 | 1.000 | 0.290 |
| Clients 19 | 0.721 | 0.411 | 0.837 | 0.322 | 1.000 | 0.248 |
| Clients 20 | 0.729 | 0.396 | 0.860 | 0.419 | 1.000 | 0.270 |
| Epochs 1 | 0.718 | 0.442 | 0.829 | 0.363 | 0.957 | 0.296 |
| Epochs 2 | 0.732 | 0.424 | 0.819 | 0.333 | 0.939 | 0.282 |
| Epochs 5 | 0.725 | 0.441 | 0.826 | 0.360 | 0.939 | 0.298 |
| Epochs 10 | 0.723 | 0.444 | 0.823 | 0.361 | 0.951 | 0.310 |
| Epochs 20 | 0.731 | 0.427 | 0.833 | 0.367 | 0.994 | 0.309 |
| Balanced 60\|40 | 0.716 | 0.430 | 0.827 | 0.360 | 0.693 | 0.295 |
| Balanced 67\|33 | 0.722 | 0.436 | 0.835 | 0.381 | 1.000 | 0.314 |
| Balanced 75\|25 | 0.718 | 0.439 | 0.813 | 0.330 | 1.000 | 0.277 |
| Balanced 80\|20 | 0.734 | 0.407 | 0.792 | 0.301 | 1.000 | 0.249 |
| Balanced 40\|30\|30 | 0.702 | 0.428 | 0.829 | 0.343 | 1.000 | 0.273 |

| | | | | | | |
|---|---|---|---|---|---|---|
| Balanced 60\|20\|20 | 0.709 | 0.446 | 0.834 | 0.380 | 0.988 | 0.329 |
| Balanced 80\|10\|10 | 0.714 | 0.423 | 0.816 | 0.330 | 1.000 | 0.261 |
| Balanced 60\|30\|10 | 0.714 | 0.427 | 0.818 | 0.338 | 1.000 | 0.273 |
| Balanced 30\|30\|15\|15\|10 | 0.714 | 0.430 | 0.834 | 0.353 | 1.000 | 0.262 |
| Balanced 60\|10\|10\|10\|10 | 0.720 | 0.406 | 0.810 | 0.318 | 1.000 | 0.249 |

Table 2.24: Best results for plain averaging on NR-ER (Tox21)

| experiment | roc_auc | pr_auc | accuracy | precision | recall | kappa |
|---|---|---|---|---|---|---|
| Baseline | 0.738 | 0.444 | 0.865 | 0.465 | 0.699 | 0.335 |
| 60\|40 | 0.732 | 0.456 | 0.830 | 0.369 | 0.926 | 0.304 |
| 67\|33 | 0.727 | 0.429 | 0.819 | 0.347 | 0.914 | 0.291 |
| 75\|25 | 0.723 | 0.446 | 0.822 | 0.348 | 0.546 | 0.284 |
| 80\|20 | 0.724 | 0.441 | 0.865 | 0.451 | 0.558 | 0.299 |
| 40\|30\|30 | 0.716 | 0.426 | 0.806 | 0.315 | 1.000 | 0.264 |
| 60\|20\|20 | 0.705 | 0.419 | 0.826 | 0.344 | 1.000 | 0.261 |
| 80\|10\|10 | 0.714 | 0.452 | 0.834 | 0.372 | 0.564 | 0.297 |
| 60\|30\|10 | 0.708 | 0.430 | 0.816 | 0.329 | 0.975 | 0.282 |
| 30\|30\|15\|15\|10 | 0.712 | 0.437 | 0.818 | 0.342 | 1.000 | 0.292 |
| 60\|10\|10\|10\|10 | 0.706 | 0.427 | 0.829 | 0.361 | 0.945 | 0.302 |

Table 2.25: Best results for weighted averaging on NR-ER (Tox21)

| experiment | roc_auc | pr_auc | accuracy | precision | recall | kappa |
|---|---|---|---|---|---|---|
| Baseline | 0.738 | 0.444 | 0.865 | 0.465 | 0.699 | 0.335 |
| Clients 2 | 0.727 | 0.432 | 0.815 | 0.332 | 1.000 | 0.279 |
| Clients 3 | 0.723 | 0.446 | 0.800 | 0.320 | 1.000 | 0.276 |
| Clients 4 | 0.717 | 0.425 | 0.796 | 0.312 | 1.000 | 0.264 |
| Clients 5 | 0.720 | 0.450 | 0.799 | 0.318 | 1.000 | 0.276 |
| Clients 6 | 0.717 | 0.410 | 0.761 | 0.274 | 1.000 | 0.227 |
| Clients 7 | 0.720 | 0.422 | 0.809 | 0.325 | 1.000 | 0.276 |
| Clients 8 | 0.724 | 0.421 | 0.811 | 0.332 | 1.000 | 0.286 |

| | | | | | | |
|---|---|---|---|---|---|---|
| Clients 9 | 0.714 | 0.416 | 0.794 | 0.302 | 1.000 | 0.246 |
| Clients 10 | 0.730 | 0.418 | 0.797 | 0.313 | 1.000 | 0.264 |
| Clients 11 | 0.718 | 0.404 | 0.819 | 0.328 | 1.000 | 0.280 |
| Clients 12 | 0.719 | 0.407 | 0.793 | 0.291 | 1.000 | 0.237 |
| Clients 13 | 0.723 | 0.393 | 0.794 | 0.274 | 1.000 | 0.217 |
| Clients 14 | 0.719 | 0.406 | 0.819 | 0.300 | 1.000 | 0.254 |
| Clients 15 | 0.726 | 0.398 | 0.809 | 0.312 | 1.000 | 0.247 |
| Clients 16 | 0.720 | 0.395 | 0.738 | 0.253 | 1.000 | 0.213 |
| Clients 17 | 0.716 | 0.410 | 0.845 | 0.291 | 1.000 | 0.243 |
| Clients 18 | 0.722 | 0.396 | 0.828 | 0.315 | 1.000 | 0.254 |
| Clients 19 | 0.719 | 0.398 | 0.744 | 0.273 | 1.000 | 0.238 |
| Clients 20 | 0.711 | 0.393 | 0.815 | 0.339 | 1.000 | 0.284 |
| Epochs 1 | 0.724 | 0.438 | 0.817 | 0.343 | 0.632 | 0.286 |
| Epochs 2 | 0.743 | 0.446 | 0.816 | 0.351 | 0.975 | 0.312 |
| Epochs 5 | 0.727 | 0.439 | 0.811 | 0.338 | 0.988 | 0.293 |
| Epochs 10 | 0.722 | 0.423 | 0.829 | 0.349 | 1.000 | 0.269 |
| Epochs 20 | 0.740 | 0.432 | 0.830 | 0.358 | 1.000 | 0.286 |
| Balanced 60\|40 | 0.720 | 0.431 | 0.820 | 0.346 | 1.000 | 0.285 |
| Balanced 67\|33 | 0.732 | 0.431 | 0.819 | 0.332 | 1.000 | 0.272 |
| Balanced 75\|25 | 0.741 | 0.450 | 0.815 | 0.344 | 1.000 | 0.292 |
| Balanced 80\|20 | 0.731 | 0.431 | 0.800 | 0.316 | 1.000 | 0.270 |
| Balanced 40\|30\|30 | 0.733 | 0.447 | 0.803 | 0.324 | 1.000 | 0.285 |
| Balanced 60\|20\|20 | 0.723 | 0.427 | 0.797 | 0.312 | 1.000 | 0.261 |
| Balanced 80\|10\|10 | 0.717 | 0.426 | 0.801 | 0.302 | 1.000 | 0.253 |
| Balanced 60\|30\|10 | 0.732 | 0.431 | 0.792 | 0.314 | 1.000 | 0.276 |
| Balanced 30\|30\|15\|15\|10 | 0.721 | 0.434 | 0.801 | 0.300 | 1.000 | 0.250 |
| Balanced 60\|10\|10\|10\|10 | 0.715 | 0.442 | 0.800 | 0.311 | 1.000 | 0.253 |

Table 2.26: Best results for benchmarked averaging on NR-ER (Tox21)

### 2.4.7 Tox21 - NR-ER-LBD

| experiment | roc_auc | pr_auc | accuracy | precision | recall | kappa |
|---|---|---|---|---|---|---|

| | | | | | | |
|---|---|---|---|---|---|---|
| Baseline | 0.838 | 0.448 | 0.947 | 0.581 | 0.530 | 0.431 |
| Clients 2 | 0.817 | 0.479 | 0.941 | 0.500 | 0.952 | 0.427 |
| Clients 3 | 0.814 | 0.449 | 0.933 | 0.432 | 0.952 | 0.413 |
| Clients 4 | 0.830 | 0.452 | 0.934 | 0.433 | 0.988 | 0.415 |
| Clients 5 | 0.804 | 0.457 | 0.936 | 0.457 | 1.000 | 0.424 |
| Clients 6 | 0.820 | 0.459 | 0.935 | 0.446 | 1.000 | 0.411 |
| Clients 7 | 0.803 | 0.459 | 0.936 | 0.459 | 1.000 | 0.443 |
| Clients 8 | 0.813 | 0.460 | 0.934 | 0.414 | 1.000 | 0.413 |
| Clients 9 | 0.803 | 0.447 | 0.934 | 0.412 | 1.000 | 0.407 |
| Clients 10 | 0.817 | 0.466 | 0.937 | 0.459 | 1.000 | 0.430 |
| Clients 11 | 0.807 | 0.456 | 0.929 | 0.410 | 1.000 | 0.410 |
| Clients 12 | 0.801 | 0.428 | 0.934 | 0.385 | 1.000 | 0.388 |
| Clients 13 | 0.788 | 0.411 | 0.939 | 0.412 | 1.000 | 0.357 |
| Clients 14 | 0.800 | 0.426 | 0.936 | 0.336 | 1.000 | 0.351 |
| Clients 15 | 0.798 | 0.410 | 0.924 | 0.361 | 1.000 | 0.366 |
| Clients 16 | 0.801 | 0.362 | 0.922 | 0.236 | 1.000 | 0.268 |
| Clients 17 | 0.775 | 0.398 | 0.938 | 0.370 | 1.000 | 0.348 |
| Clients 18 | 0.809 | 0.429 | 0.939 | 0.383 | 1.000 | 0.398 |
| Clients 19 | 0.792 | 0.403 | 0.934 | 0.322 | 1.000 | 0.336 |
| Clients 20 | 0.791 | 0.413 | 0.941 | 1.000 | 1.000 | 0.383 |
| Epochs 1 | 0.824 | 0.478 | 0.941 | 0.493 | 0.566 | 0.451 |
| Epochs 2 | 0.827 | 0.481 | 0.931 | 0.430 | 0.795 | 0.434 |
| Epochs 5 | 0.821 | 0.454 | 0.938 | 0.468 | 0.554 | 0.436 |
| Epochs 10 | 0.838 | 0.444 | 0.935 | 0.442 | 0.506 | 0.396 |
| Epochs 20 | 0.830 | 0.432 | 0.939 | 0.481 | 0.988 | 0.430 |
| Balanced 60\|40 | 0.830 | 0.490 | 0.937 | 0.465 | 0.976 | 0.440 |
| Balanced 67\|33 | 0.818 | 0.487 | 0.943 | 0.512 | 0.976 | 0.473 |
| Balanced 75\|25 | 0.817 | 0.490 | 0.938 | 0.473 | 0.976 | 0.464 |
| Balanced 80\|20 | 0.824 | 0.469 | 0.936 | 0.456 | 0.976 | 0.431 |
| Balanced 40\|30\|30 | 0.822 | 0.470 | 0.934 | 0.449 | 1.000 | 0.451 |
| Balanced 60\|20\|20 | 0.819 | 0.474 | 0.942 | 0.506 | 0.976 | 0.463 |
| Balanced 80\|10\|10 | 0.820 | 0.493 | 0.943 | 0.513 | 0.976 | 0.469 |
| Balanced 60\|30\|10 | 0.804 | 0.453 | 0.932 | 0.424 | 0.976 | 0.392 |
| Balanced 30\|30\|15\|15\|10 | 0.817 | 0.458 | 0.936 | 0.455 | 0.988 | 0.442 |

| | | | | | | |
|---|---|---|---|---|---|---|
| Balanced 60\|10\|10\|10\|10 | 0.809 | 0.469 | 0.941 | 0.493 | 1.000 | 0.430 |

Table 2.27: Best results for plain averaging on NR-ER-LBD (Tox21)

| experiment | roc_auc | pr_auc | accuracy | precision | recall | kappa |
|---|---|---|---|---|---|---|
| Baseline | 0.838 | 0.448 | 0.947 | 0.581 | 0.530 | 0.431 |
| 60\|40 | 0.837 | 0.460 | 0.936 | 0.457 | 0.711 | 0.452 |
| 67\|33 | 0.799 | 0.471 | 0.943 | 0.515 | 0.542 | 0.427 |
| 75\|25 | 0.797 | 0.452 | 0.936 | 0.458 | 0.566 | 0.451 |
| 80\|20 | 0.819 | 0.454 | 0.937 | 0.398 | 0.542 | 0.392 |
| 40\|30\|30 | 0.817 | 0.465 | 0.934 | 0.441 | 0.988 | 0.431 |
| 60\|20\|20 | 0.808 | 0.463 | 0.935 | 0.448 | 0.952 | 0.424 |
| 80\|10\|10 | 0.807 | 0.467 | 0.941 | 0.481 | 0.566 | 0.443 |
| 60\|30\|10 | 0.812 | 0.434 | 0.934 | 0.442 | 0.506 | 0.415 |
| 30\|30\|15\|15\|10 | 0.834 | 0.483 | 0.942 | 0.507 | 1.000 | 0.475 |
| 60\|10\|10\|10\|10 | 0.789 | 0.462 | 0.940 | 0.488 | 0.904 | 0.453 |

Table 2.28: Best results for weighted averaging on NR-ER-LBD (Tox21)

| experiment | roc_auc | pr_auc | accuracy | precision | recall | kappa |
|---|---|---|---|---|---|---|
| Baseline | 0.838 | 0.448 | 0.947 | 0.581 | 0.530 | 0.431 |
| Clients 2 | 0.841 | 0.478 | 0.934 | 0.430 | 1.000 | 0.436 |
| Clients 3 | 0.819 | 0.442 | 0.930 | 0.417 | 1.000 | 0.410 |
| Clients 4 | 0.813 | 0.463 | 0.928 | 0.404 | 1.000 | 0.404 |
| Clients 5 | 0.840 | 0.472 | 0.927 | 0.400 | 1.000 | 0.417 |
| Clients 6 | 0.795 | 0.480 | 0.912 | 0.331 | 1.000 | 0.350 |
| Clients 7 | 0.804 | 0.431 | 0.921 | 0.362 | 1.000 | 0.374 |
| Clients 8 | 0.816 | 0.433 | 0.934 | 0.343 | 1.000 | 0.353 |
| Clients 9 | 0.811 | 0.428 | 0.923 | 0.375 | 1.000 | 0.376 |
| Clients 10 | 0.812 | 0.411 | 0.923 | 0.377 | 1.000 | 0.383 |
| Clients 11 | 0.815 | 0.436 | 0.934 | 0.380 | 1.000 | 0.383 |
| Clients 12 | 0.807 | 0.428 | 0.919 | 0.342 | 1.000 | 0.359 |

| | | | | | | |
|---|---|---|---|---|---|---|
| Clients 13 | 0.794 | 0.424 | 0.931 | 0.342 | 1.000 | 0.364 |
| Clients 14 | 0.801 | 0.430 | 0.932 | 0.354 | 1.000 | 0.376 |
| Clients 15 | 0.801 | 0.430 | 0.930 | 0.368 | 1.000 | 0.359 |
| Clients 16 | 0.793 | 0.394 | 0.931 | 0.303 | 1.000 | 0.333 |
| Clients 17 | 0.788 | 0.377 | 0.939 | 0.349 | 1.000 | 0.353 |
| Clients 18 | 0.787 | 0.406 | 0.941 | 0.369 | 1.000 | 0.381 |
| Clients 19 | 0.801 | 0.345 | 0.935 | 0.301 | 1.000 | 0.321 |
| Clients 20 | 0.795 | 0.402 | 0.936 | 0.342 | 1.000 | 0.353 |
| Epochs 1 | 0.822 | 0.461 | 0.935 | 0.414 | 0.771 | 0.413 |
| Epochs 2 | 0.820 | 0.460 | 0.934 | 0.442 | 0.566 | 0.415 |
| Epochs 5 | 0.824 | 0.457 | 0.934 | 0.438 | 0.988 | 0.415 |
| Epochs 10 | 0.825 | 0.455 | 0.941 | 0.493 | 1.000 | 0.443 |
| Epochs 20 | 0.827 | 0.447 | 0.935 | 0.448 | 0.964 | 0.424 |
| Balanced 60\|40 | 0.814 | 0.471 | 0.938 | 0.463 | 0.952 | 0.418 |
| Balanced 67\|33 | 0.813 | 0.469 | 0.936 | 0.453 | 0.988 | 0.427 |
| Balanced 75\|25 | 0.840 | 0.475 | 0.936 | 0.456 | 1.000 | 0.445 |
| Balanced 80\|20 | 0.823 | 0.460 | 0.932 | 0.432 | 0.988 | 0.425 |
| Balanced 40\|30\|30 | 0.818 | 0.481 | 0.931 | 0.420 | 1.000 | 0.415 |
| Balanced 60\|20\|20 | 0.832 | 0.442 | 0.928 | 0.406 | 1.000 | 0.407 |
| Balanced 80\|10\|10 | 0.841 | 0.478 | 0.945 | 0.535 | 1.000 | 0.465 |
| Balanced 60\|30\|10 | 0.826 | 0.454 | 0.923 | 0.380 | 1.000 | 0.401 |
| Balanced 30\|30\|15\|15\|10 | 0.830 | 0.457 | 0.927 | 0.396 | 1.000 | 0.395 |
| Balanced 60\|10\|10\|10\|10 | 0.808 | 0.480 | 0.936 | 0.462 | 1.000 | 0.457 |

Table 2.29: Best results for benchmarked averaging on NR-ER-LBD (Tox21)

### 2.4.8 Tox21 - NR-PPAR-gamma

| experiment | roc_auc | pr_auc | accuracy | precision | recall | kappa |
|---|---|---|---|---|---|---|
| Baseline | 0.897 | 0.394 | 0.973 | 0.556 | 0.621 | 0.433 |
| Clients 2 | 0.863 | 0.414 | 0.967 | 0.429 | 0.973 | 0.439 |
| Clients 3 | 0.884 | 0.425 | 0.961 | 0.362 | 0.973 | 0.422 |
| Clients 4 | 0.832 | 0.366 | 0.964 | 0.311 | 0.973 | 0.365 |
| Clients 5 | 0.845 | 0.375 | 0.960 | 0.310 | 0.973 | 0.357 |

| | | | | | | |
|---|---|---|---|---|---|---|
| Clients 6 | 0.844 | 0.345 | 0.963 | 0.312 | 0.973 | 0.349 |
| Clients 7 | 0.828 | 0.350 | 0.960 | 0.296 | 0.973 | 0.335 |
| Clients 8 | 0.849 | 0.347 | 0.966 | 0.262 | 1.000 | 0.302 |
| Clients 9 | 0.849 | 0.378 | 0.964 | 0.333 | 0.973 | 0.375 |
| Clients 10 | 0.844 | 0.382 | 0.969 | 0.359 | 1.000 | 0.384 |
| Clients 11 | 0.835 | 0.428 | 0.965 | 0.263 | 0.973 | 0.317 |
| Clients 12 | 0.828 | 0.280 | 0.968 | 0.286 | 1.000 | 0.335 |
| Clients 13 | 0.822 | 0.292 | 0.969 | 0.290 | 1.000 | 0.340 |
| Clients 14 | 0.845 | 0.284 | 0.966 | 0.302 | 1.000 | 0.323 |
| Clients 15 | 0.840 | 0.308 | 0.968 | 0.295 | 1.000 | 0.308 |
| Clients 16 | 0.843 | 0.343 | 0.966 | 0.239 | 1.000 | 0.288 |
| Clients 17 | 0.802 | 0.278 | 0.969 | 0.350 | 1.000 | 0.356 |
| Clients 18 | 0.814 | 0.294 | 0.970 | 0.304 | 1.000 | 0.343 |
| Clients 19 | 0.838 | 0.340 | 0.969 | 0.314 | 1.000 | 0.342 |
| Clients 20 | 0.841 | 0.263 | 0.972 | 1.000 | 1.000 | 0.329 |
| Epochs 1 | 0.869 | 0.464 | 0.969 | 0.439 | 0.622 | 0.445 |
| Epochs 2 | 0.868 | 0.384 | 0.970 | 0.450 | 0.622 | 0.435 |
| Epochs 5 | 0.859 | 0.407 | 0.962 | 0.349 | 0.568 | 0.361 |
| Epochs 10 | 0.871 | 0.384 | 0.965 | 0.385 | 0.838 | 0.377 |
| Epochs 20 | 0.848 | 0.313 | 0.959 | 0.293 | 0.919 | 0.317 |
| Balanced 60\|40 | 0.841 | 0.417 | 0.968 | 0.366 | 0.595 | 0.388 |
| Balanced 67\|33 | 0.848 | 0.352 | 0.963 | 0.324 | 0.838 | 0.342 |
| Balanced 75\|25 | 0.827 | 0.368 | 0.952 | 0.286 | 0.730 | 0.342 |
| Balanced 80\|20 | 0.867 | 0.369 | 0.962 | 0.362 | 0.946 | 0.388 |
| Balanced 40\|30\|30 | 0.859 | 0.353 | 0.959 | 0.319 | 0.973 | 0.350 |
| Balanced 60\|20\|20 | 0.837 | 0.400 | 0.966 | 0.409 | 0.784 | 0.427 |
| Balanced 80\|10\|10 | 0.840 | 0.340 | 0.960 | 0.326 | 0.649 | 0.341 |
| Balanced 60\|30\|10 | 0.852 | 0.351 | 0.969 | 0.423 | 0.919 | 0.375 |
| Balanced 30\|30\|15\|15\|10 | 0.819 | 0.387 | 0.968 | 0.311 | 1.000 | 0.365 |
| Balanced 60\|10\|10\|10\|10 | 0.845 | 0.343 | 0.962 | 0.324 | 0.973 | 0.323 |

Table 2.30: Best results for plain averaging on NR-PPAR-gamma (Tox21)

| experiment | roc_auc | pr_auc | accuracy | precision | recall | kappa |
|---|---|---|---|---|---|---|

| | | | | | | |
|---|---|---|---|---|---|---|
| Baseline | 0.897 | 0.394 | 0.973 | 0.556 | 0.621 | 0.433 |
| 60\|40 | 0.845 | 0.405 | 0.969 | 0.444 | 0.919 | 0.431 |
| 67\|33 | 0.854 | 0.405 | 0.972 | 1.000 | 0.595 | 0.394 |
| 75\|25 | 0.855 | 0.348 | 0.972 | 0.500 | 0.595 | 0.396 |
| 80\|20 | 0.863 | 0.389 | 0.969 | 0.346 | 0.541 | 0.384 |
| 40\|30\|30 | 0.877 | 0.388 | 0.957 | 0.310 | 0.865 | 0.361 |
| 60\|20\|20 | 0.840 | 0.399 | 0.963 | 0.340 | 0.568 | 0.403 |
| 80\|10\|10 | 0.843 | 0.366 | 0.967 | 0.394 | 0.568 | 0.354 |
| 60\|30\|10 | 0.866 | 0.372 | 0.968 | 0.347 | 0.622 | 0.375 |
| 30\|30\|15\|15\|10 | 0.824 | 0.400 | 0.966 | 0.346 | 0.973 | 0.365 |
| 60\|10\|10\|10\|10 | 0.850 | 0.359 | 0.966 | 0.375 | 0.892 | 0.346 |

Table 2.31: Best results for weighted averaging on NR-PPAR-gamma (Tox21)

| experiment | roc_auc | pr_auc | accuracy | precision | recall | kappa |
|---|---|---|---|---|---|---|
| Baseline | 0.897 | 0.394 | 0.973 | 0.556 | 0.621 | 0.433 |
| Clients 2 | 0.879 | 0.340 | 0.963 | 0.364 | 1.000 | 0.376 |
| Clients 3 | 0.877 | 0.354 | 0.963 | 0.350 | 1.000 | 0.361 |
| Clients 4 | 0.850 | 0.356 | 0.957 | 0.292 | 1.000 | 0.313 |
| Clients 5 | 0.851 | 0.328 | 0.944 | 0.227 | 1.000 | 0.273 |
| Clients 6 | 0.876 | 0.359 | 0.936 | 0.224 | 1.000 | 0.283 |
| Clients 7 | 0.839 | 0.344 | 0.957 | 0.263 | 1.000 | 0.305 |
| Clients 8 | 0.838 | 0.324 | 0.964 | 0.217 | 1.000 | 0.265 |
| Clients 9 | 0.881 | 0.312 | 0.950 | 0.277 | 1.000 | 0.329 |
| Clients 10 | 0.859 | 0.331 | 0.947 | 0.234 | 1.000 | 0.289 |
| Clients 11 | 0.852 | 0.317 | 0.961 | 0.241 | 1.000 | 0.311 |
| Clients 12 | 0.868 | 0.338 | 0.950 | 0.253 | 1.000 | 0.328 |
| Clients 13 | 0.825 | 0.337 | 0.967 | 0.213 | 1.000 | 0.266 |
| Clients 14 | 0.856 | 0.327 | 0.963 | 0.230 | 1.000 | 0.279 |
| Clients 15 | 0.836 | 0.351 | 0.964 | 0.256 | 1.000 | 0.322 |
| Clients 16 | 0.849 | 0.292 | 0.956 | 0.242 | 1.000 | 0.277 |
| Clients 17 | 0.825 | 0.268 | 0.970 | 0.176 | 1.000 | 0.230 |
| Clients 18 | 0.816 | 0.336 | 0.971 | 0.350 | 1.000 | 0.344 |

| | | | | | | |
|---|---|---|---|---|---|---|
| Clients 19 | 0.841 | 0.315 | 0.966 | 0.209 | 1.000 | 0.267 |
| Clients 20 | 0.827 | 0.259 | 0.967 | 0.217 | 1.000 | 0.272 |
| Epochs 1 | 0.907 | 0.410 | 0.972 | 0.500 | 0.649 | 0.399 |
| Epochs 2 | 0.885 | 0.439 | 0.967 | 0.393 | 0.595 | 0.380 |
| Epochs 5 | 0.885 | 0.391 | 0.967 | 0.372 | 0.649 | 0.381 |
| Epochs 10 | 0.840 | 0.360 | 0.962 | 0.348 | 1.000 | 0.383 |
| Epochs 20 | 0.872 | 0.350 | 0.957 | 0.293 | 0.973 | 0.335 |
| Balanced 60\|40 | 0.866 | 0.370 | 0.962 | 0.362 | 1.000 | 0.387 |
| Balanced 67\|33 | 0.869 | 0.391 | 0.960 | 0.347 | 0.649 | 0.379 |
| Balanced 75\|25 | 0.885 | 0.423 | 0.966 | 0.400 | 0.973 | 0.413 |
| Balanced 80\|20 | 0.868 | 0.407 | 0.955 | 0.317 | 1.000 | 0.370 |
| Balanced 40\|30\|30 | 0.896 | 0.414 | 0.953 | 0.269 | 1.000 | 0.333 |
| Balanced 60\|20\|20 | 0.860 | 0.374 | 0.949 | 0.282 | 1.000 | 0.369 |
| Balanced 80\|10\|10 | 0.864 | 0.334 | 0.954 | 0.283 | 0.973 | 0.314 |
| Balanced 60\|30\|10 | 0.887 | 0.373 | 0.957 | 0.328 | 0.973 | 0.385 |
| Balanced 30\|30\|15\|15\|10 | 0.872 | 0.360 | 0.952 | 0.260 | 1.000 | 0.320 |
| Balanced 60\|10\|10\|10\|10 | 0.886 | 0.423 | 0.960 | 0.361 | 1.000 | 0.429 |

Table 2.32: Best results for benchmarked averaging on NR-PPAR-gamma (Tox21)

### 2.4.9 Tox21 - SR-ARE

| experiment | roc_auc | pr_auc | accuracy | precision | recall | kappa |
|---|---|---|---|---|---|---|
| Baseline | 0.844 | 0.553 | 0.844 | 0.491 | 0.842 | 0.446 |
| Clients 2 | 0.845 | 0.550 | 0.836 | 0.472 | 1.000 | 0.437 |
| Clients 3 | 0.840 | 0.546 | 0.832 | 0.462 | 1.000 | 0.418 |
| Clients 4 | 0.830 | 0.563 | 0.837 | 0.475 | 1.000 | 0.448 |
| Clients 5 | 0.836 | 0.537 | 0.831 | 0.460 | 1.000 | 0.434 |
| Clients 6 | 0.834 | 0.537 | 0.824 | 0.445 | 1.000 | 0.419 |
| Clients 7 | 0.830 | 0.520 | 0.838 | 0.476 | 1.000 | 0.436 |
| Clients 8 | 0.838 | 0.521 | 0.810 | 0.426 | 1.000 | 0.420 |
| Clients 9 | 0.836 | 0.527 | 0.836 | 0.474 | 1.000 | 0.457 |
| Clients 10 | 0.834 | 0.553 | 0.837 | 0.476 | 1.000 | 0.457 |

| | | | | | | |
|---|---|---|---|---|---|---|
| Clients 11 | 0.826 | 0.515 | 0.834 | 0.469 | 1.000 | 0.445 |
| Clients 12 | 0.823 | 0.518 | 0.825 | 0.442 | 1.000 | 0.409 |
| Clients 13 | 0.826 | 0.525 | 0.837 | 0.475 | 1.000 | 0.462 |
| Clients 14 | 0.832 | 0.520 | 0.830 | 0.459 | 1.000 | 0.446 |
| Clients 15 | 0.832 | 0.503 | 0.810 | 0.421 | 1.000 | 0.408 |
| Clients 16 | 0.815 | 0.432 | 0.740 | 0.344 | 1.000 | 0.336 |
| Clients 17 | 0.826 | 0.499 | 0.839 | 0.479 | 1.000 | 0.443 |
| Clients 18 | 0.836 | 0.506 | 0.831 | 0.462 | 1.000 | 0.442 |
| Clients 19 | 0.826 | 0.516 | 0.840 | 0.481 | 1.000 | 0.434 |
| Clients 20 | 0.829 | 0.506 | 0.827 | 0.454 | 1.000 | 0.429 |
| Epochs 1 | 0.847 | 0.561 | 0.827 | 0.450 | 0.831 | 0.425 |
| Epochs 2 | 0.846 | 0.565 | 0.822 | 0.447 | 0.944 | 0.438 |
| Epochs 5 | 0.832 | 0.558 | 0.840 | 0.481 | 0.977 | 0.437 |
| Epochs 10 | 0.841 | 0.557 | 0.839 | 0.477 | 1.000 | 0.444 |
| Epochs 20 | 0.835 | 0.547 | 0.837 | 0.474 | 1.000 | 0.441 |
| Balanced 60\|40 | 0.824 | 0.525 | 0.832 | 0.463 | 0.966 | 0.423 |
| Balanced 67\|33 | 0.846 | 0.571 | 0.825 | 0.450 | 1.000 | 0.441 |
| Balanced 75\|25 | 0.842 | 0.534 | 0.820 | 0.438 | 1.000 | 0.432 |
| Balanced 80\|20 | 0.851 | 0.562 | 0.833 | 0.466 | 1.000 | 0.436 |
| Balanced 40\|30\|30 | 0.830 | 0.549 | 0.832 | 0.460 | 1.000 | 0.422 |
| Balanced 60\|20\|20 | 0.838 | 0.543 | 0.835 | 0.468 | 0.977 | 0.431 |
| Balanced 80\|10\|10 | 0.833 | 0.547 | 0.822 | 0.443 | 1.000 | 0.431 |
| Balanced 60\|30\|10 | 0.834 | 0.551 | 0.831 | 0.461 | 1.000 | 0.431 |
| Balanced 30\|30\|15\|15\|10 | 0.834 | 0.541 | 0.835 | 0.470 | 1.000 | 0.437 |
| Balanced 60\|10\|10\|10\|10 | 0.829 | 0.557 | 0.834 | 0.465 | 1.000 | 0.419 |

Table 2.33: Best results for plain averaging on SR-ARE (Tox21)

| experiment | roc_auc | pr_auc | accuracy | precision | recall | kappa |
|---|---|---|---|---|---|---|
| Baseline | 0.844 | 0.553 | 0.844 | 0.491 | 0.842 | 0.446 |
| 60\|40 | 0.842 | 0.550 | 0.835 | 0.468 | 1.000 | 0.451 |
| 67\|33 | 0.845 | 0.528 | 0.829 | 0.455 | 0.989 | 0.423 |
| 75\|25 | 0.847 | 0.546 | 0.839 | 0.479 | 0.972 | 0.433 |
| 80\|20 | 0.844 | 0.515 | 0.820 | 0.438 | 0.785 | 0.428 |

| | | | | | | |
|---|---|---|---|---|---|---|
| 40\|30\|30 | 0.837 | 0.532 | 0.829 | 0.455 | 1.000 | 0.413 |
| 60\|20\|20 | 0.839 | 0.548 | 0.826 | 0.450 | 1.000 | 0.423 |
| 80\|10\|10 | 0.857 | 0.567 | 0.835 | 0.468 | 0.791 | 0.465 |
| 60\|30\|10 | 0.850 | 0.566 | 0.836 | 0.471 | 0.915 | 0.433 |
| 30\|30\|15\|15\|10 | 0.833 | 0.541 | 0.834 | 0.463 | 1.000 | 0.412 |
| 60\|10\|10\|10\|10 | 0.841 | 0.557 | 0.836 | 0.469 | 1.000 | 0.428 |

Table 2.34: Best results for weighted averaging on SR-ARE (Tox21)

| experiment | roc_auc | pr_auc | accuracy | precision | recall | kappa |
|---|---|---|---|---|---|---|
| Baseline | 0.844 | 0.553 | 0.844 | 0.491 | 0.842 | 0.446 |
| Clients 2 | 0.832 | 0.527 | 0.844 | 0.491 | 1.000 | 0.444 |
| Clients 3 | 0.839 | 0.521 | 0.826 | 0.449 | 1.000 | 0.425 |
| Clients 4 | 0.834 | 0.521 | 0.828 | 0.455 | 1.000 | 0.423 |
| Clients 5 | 0.838 | 0.514 | 0.806 | 0.412 | 1.000 | 0.401 |
| Clients 6 | 0.832 | 0.524 | 0.795 | 0.404 | 1.000 | 0.401 |
| Clients 7 | 0.837 | 0.543 | 0.811 | 0.424 | 1.000 | 0.406 |
| Clients 8 | 0.834 | 0.528 | 0.809 | 0.421 | 1.000 | 0.403 |
| Clients 9 | 0.841 | 0.532 | 0.815 | 0.432 | 1.000 | 0.416 |
| Clients 10 | 0.825 | 0.504 | 0.793 | 0.396 | 1.000 | 0.380 |
| Clients 11 | 0.833 | 0.507 | 0.807 | 0.415 | 1.000 | 0.397 |
| Clients 12 | 0.832 | 0.509 | 0.801 | 0.413 | 1.000 | 0.410 |
| Clients 13 | 0.822 | 0.494 | 0.813 | 0.422 | 1.000 | 0.391 |
| Clients 14 | 0.841 | 0.521 | 0.818 | 0.438 | 1.000 | 0.425 |
| Clients 15 | 0.831 | 0.499 | 0.804 | 0.410 | 1.000 | 0.402 |
| Clients 16 | 0.819 | 0.488 | 0.785 | 0.387 | 1.000 | 0.380 |
| Clients 17 | 0.828 | 0.501 | 0.775 | 0.376 | 1.000 | 0.368 |
| Clients 18 | 0.825 | 0.504 | 0.813 | 0.428 | 1.000 | 0.420 |
| Clients 19 | 0.833 | 0.469 | 0.796 | 0.407 | 1.000 | 0.406 |
| Clients 20 | 0.828 | 0.466 | 0.796 | 0.403 | 1.000 | 0.395 |
| Epochs 1 | 0.844 | 0.551 | 0.823 | 0.444 | 1.000 | 0.420 |
| Epochs 2 | 0.848 | 0.566 | 0.826 | 0.452 | 1.000 | 0.438 |
| Epochs 5 | 0.841 | 0.542 | 0.832 | 0.462 | 1.000 | 0.436 |

| | | | | | | |
|---|---|---|---|---|---|---|
| Epochs 10 | 0.834 | 0.555 | 0.839 | 0.480 | 1.000 | 0.439 |
| Epochs 20 | 0.829 | 0.557 | 0.822 | 0.440 | 1.000 | 0.413 |
| Balanced 60\|40 | 0.833 | 0.533 | 0.822 | 0.445 | 1.000 | 0.428 |
| Balanced 67\|33 | 0.836 | 0.530 | 0.829 | 0.457 | 1.000 | 0.430 |
| Balanced 75\|25 | 0.842 | 0.538 | 0.830 | 0.457 | 1.000 | 0.420 |
| Balanced 80\|20 | 0.846 | 0.537 | 0.826 | 0.452 | 1.000 | 0.443 |
| Balanced 40\|30\|30 | 0.843 | 0.545 | 0.813 | 0.425 | 1.000 | 0.412 |
| Balanced 60\|20\|20 | 0.849 | 0.546 | 0.816 | 0.428 | 1.000 | 0.406 |
| Balanced 80\|10\|10 | 0.851 | 0.541 | 0.815 | 0.428 | 1.000 | 0.423 |
| Balanced 60\|30\|10 | 0.836 | 0.559 | 0.832 | 0.463 | 1.000 | 0.439 |
| Balanced 30\|30\|15\|15\|10 | 0.838 | 0.538 | 0.821 | 0.441 | 1.000 | 0.414 |
| Balanced 60\|10\|10\|10\|10 | 0.839 | 0.515 | 0.813 | 0.424 | 1.000 | 0.390 |

Table 2.35: Best results for benchmarked averaging on SR-ARE (Tox21)

### 2.4.10 Tox21 - SR-ATAD5

| experiment | roc_auc | pr_auc | accuracy | precision | recall | kappa |
|---|---|---|---|---|---|---|
| Baseline | 0.918 | 0.464 | 0.968 | 0.571 | 0.635 | 0.494 |
| Clients 2 | 0.899 | 0.457 | 0.957 | 0.418 | 0.769 | 0.415 |
| Clients 3 | 0.911 | 0.473 | 0.951 | 0.382 | 0.962 | 0.429 |
| Clients 4 | 0.895 | 0.442 | 0.957 | 0.418 | 0.981 | 0.431 |
| Clients 5 | 0.883 | 0.457 | 0.950 | 0.366 | 0.981 | 0.406 |
| Clients 6 | 0.876 | 0.413 | 0.948 | 0.326 | 1.000 | 0.382 |
| Clients 7 | 0.885 | 0.441 | 0.950 | 0.354 | 0.981 | 0.401 |
| Clients 8 | 0.880 | 0.399 | 0.955 | 0.295 | 1.000 | 0.342 |
| Clients 9 | 0.892 | 0.434 | 0.949 | 0.298 | 1.000 | 0.367 |
| Clients 10 | 0.873 | 0.436 | 0.960 | 0.346 | 1.000 | 0.389 |
| Clients 11 | 0.877 | 0.407 | 0.954 | 0.309 | 1.000 | 0.347 |
| Clients 12 | 0.848 | 0.377 | 0.947 | 0.300 | 1.000 | 0.334 |
| Clients 13 | 0.878 | 0.441 | 0.956 | 0.281 | 1.000 | 0.334 |
| Clients 14 | 0.893 | 0.441 | 0.957 | 0.299 | 1.000 | 0.345 |
| Clients 15 | 0.860 | 0.381 | 0.956 | 0.272 | 1.000 | 0.316 |
| Clients 16 | 0.865 | 0.336 | 0.946 | 0.206 | 1.000 | 0.254 |

| | | | | | | |
|---|---|---|---|---|---|---|
| Clients 17 | 0.860 | 0.366 | 0.959 | 0.346 | 1.000 | 0.389 |
| Clients 18 | 0.875 | 0.382 | 0.961 | 0.293 | 1.000 | 0.345 |
| Clients 19 | 0.848 | 0.391 | 0.957 | 0.301 | 1.000 | 0.329 |
| Clients 20 | 0.863 | 0.350 | 0.964 | 1.000 | 1.000 | 0.335 |
| Epochs 1 | 0.917 | 0.509 | 0.958 | 0.431 | 0.769 | 0.437 |
| Epochs 2 | 0.908 | 0.511 | 0.959 | 0.436 | 0.654 | 0.461 |
| Epochs 5 | 0.904 | 0.481 | 0.957 | 0.419 | 0.673 | 0.473 |
| Epochs 10 | 0.913 | 0.436 | 0.957 | 0.429 | 0.692 | 0.448 |
| Epochs 20 | 0.890 | 0.432 | 0.960 | 0.456 | 1.000 | 0.456 |
| Balanced 60\|40 | 0.896 | 0.445 | 0.954 | 0.394 | 0.673 | 0.417 |
| Balanced 67\|33 | 0.893 | 0.456 | 0.960 | 0.444 | 0.712 | 0.453 |
| Balanced 75\|25 | 0.889 | 0.505 | 0.957 | 0.429 | 0.962 | 0.457 |
| Balanced 80\|20 | 0.904 | 0.507 | 0.962 | 0.472 | 0.673 | 0.475 |
| Balanced 40\|30\|30 | 0.919 | 0.480 | 0.952 | 0.390 | 1.000 | 0.441 |
| Balanced 60\|20\|20 | 0.893 | 0.451 | 0.955 | 0.417 | 0.942 | 0.461 |
| Balanced 80\|10\|10 | 0.910 | 0.500 | 0.949 | 0.367 | 0.846 | 0.417 |
| Balanced 60\|30\|10 | 0.888 | 0.482 | 0.953 | 0.389 | 0.981 | 0.435 |
| Balanced 30\|30\|15\|15\|10 | 0.892 | 0.455 | 0.958 | 0.310 | 1.000 | 0.353 |
| Balanced 60\|10\|10\|10\|10 | 0.881 | 0.416 | 0.948 | 0.333 | 0.981 | 0.381 |

Table 2.36: Best results for plain averaging on SR-ATAD5 (Tox21)

| experiment | roc_auc | pr_auc | accuracy | precision | recall | kappa |
|---|---|---|---|---|---|---|
| Baseline | 0.918 | 0.464 | 0.968 | 0.571 | 0.635 | 0.494 |
| 60\|40 | 0.897 | 0.472 | 0.957 | 0.422 | 0.712 | 0.446 |
| 67\|33 | 0.909 | 0.473 | 0.958 | 0.438 | 0.962 | 0.461 |
| 75\|25 | 0.899 | 0.453 | 0.956 | 0.397 | 0.615 | 0.413 |
| 80\|20 | 0.906 | 0.487 | 0.960 | 0.456 | 0.654 | 0.456 |
| 40\|30\|30 | 0.908 | 0.484 | 0.952 | 0.373 | 0.962 | 0.410 |
| 60\|20\|20 | 0.898 | 0.491 | 0.957 | 0.422 | 0.962 | 0.443 |
| 80\|10\|10 | 0.908 | 0.467 | 0.962 | 0.467 | 0.635 | 0.469 |
| 60\|30\|10 | 0.895 | 0.473 | 0.957 | 0.417 | 0.962 | 0.424 |
| 30\|30\|15\|15\|10 | 0.898 | 0.460 | 0.949 | 0.346 | 0.981 | 0.394 |
| 60\|10\|10\|10\|10 | 0.887 | 0.481 | 0.955 | 0.409 | 0.865 | 0.435 |

Table 2.37: Best results for weighted averaging on SR-ATAD5 (Tox21)

| experiment | roc_auc | pr_auc | accuracy | precision | recall | kappa |
|---|---|---|---|---|---|---|
| Baseline | 0.918 | 0.464 | 0.968 | 0.571 | 0.635 | 0.494 |
| Clients 2 | 0.912 | 0.456 | 0.954 | 0.391 | 1.000 | 0.429 |
| Clients 3 | 0.897 | 0.427 | 0.951 | 0.371 | 1.000 | 0.401 |
| Clients 4 | 0.892 | 0.419 | 0.947 | 0.354 | 1.000 | 0.406 |
| Clients 5 | 0.891 | 0.464 | 0.938 | 0.297 | 0.981 | 0.374 |
| Clients 6 | 0.884 | 0.417 | 0.931 | 0.276 | 1.000 | 0.344 |
| Clients 7 | 0.882 | 0.397 | 0.943 | 0.310 | 1.000 | 0.372 |
| Clients 8 | 0.890 | 0.436 | 0.942 | 0.324 | 1.000 | 0.400 |
| Clients 9 | 0.893 | 0.399 | 0.933 | 0.277 | 1.000 | 0.334 |
| Clients 10 | 0.867 | 0.375 | 0.939 | 0.266 | 1.000 | 0.341 |
| Clients 11 | 0.887 | 0.407 | 0.952 | 0.248 | 1.000 | 0.318 |
| Clients 12 | 0.873 | 0.422 | 0.946 | 0.277 | 1.000 | 0.346 |
| Clients 13 | 0.867 | 0.412 | 0.948 | 0.238 | 1.000 | 0.301 |
| Clients 14 | 0.888 | 0.350 | 0.956 | 0.245 | 1.000 | 0.299 |
| Clients 15 | 0.863 | 0.357 | 0.950 | 0.301 | 1.000 | 0.341 |
| Clients 16 | 0.864 | 0.358 | 0.945 | 0.206 | 1.000 | 0.254 |
| Clients 17 | 0.880 | 0.401 | 0.962 | 0.269 | 1.000 | 0.330 |
| Clients 18 | 0.861 | 0.454 | 0.963 | 0.315 | 1.000 | 0.368 |
| Clients 19 | 0.865 | 0.444 | 0.956 | 0.227 | 1.000 | 0.285 |
| Clients 20 | 0.881 | 0.445 | 0.958 | 0.318 | 1.000 | 0.379 |
| Epochs 1 | 0.905 | 0.487 | 0.963 | 0.400 | 0.673 | 0.414 |
| Epochs 2 | 0.896 | 0.468 | 0.957 | 0.424 | 0.635 | 0.452 |
| Epochs 5 | 0.902 | 0.445 | 0.958 | 0.420 | 0.962 | 0.418 |
| Epochs 10 | 0.909 | 0.445 | 0.952 | 0.387 | 0.981 | 0.432 |
| Epochs 20 | 0.896 | 0.422 | 0.956 | 0.407 | 1.000 | 0.433 |
| Balanced 60\|40 | 0.884 | 0.435 | 0.952 | 0.373 | 1.000 | 0.396 |
| Balanced 67\|33 | 0.907 | 0.454 | 0.955 | 0.403 | 0.885 | 0.420 |
| Balanced 75\|25 | 0.910 | 0.481 | 0.956 | 0.418 | 1.000 | 0.448 |
| Balanced 80\|20 | 0.911 | 0.471 | 0.955 | 0.403 | 0.981 | 0.416 |
| Balanced 40\|30\|30 | 0.911 | 0.461 | 0.946 | 0.346 | 1.000 | 0.394 |
| Balanced 60\|20\|20 | 0.903 | 0.444 | 0.940 | 0.307 | 1.000 | 0.376 |
| Balanced 80\|10\|10 | 0.920 | 0.470 | 0.954 | 0.400 | 1.000 | 0.436 |

| | | | | | | |
|---|---|---|---|---|---|---|
| Balanced 60\|30\|10 | 0.919 | 0.442 | 0.948 | 0.347 | 0.981 | 0.388 |
| Balanced 30\|30\|15\|15\|10 | 0.909 | 0.450 | 0.941 | 0.330 | 1.000 | 0.396 |
| Balanced 60\|10\|10\|10\|10 | 0.891 | 0.449 | 0.952 | 0.380 | 1.000 | 0.415 |

Table 2.38: Best results for benchmarked averaging on SR-ATAD5 (Tox21)

### 2.4.11 Tox21 - SR-HSE

| experiment | roc_auc | pr_auc | accuracy | precision | recall | kappa |
|---|---|---|---|---|---|---|
| Baseline | 0.855 | 0.435 | 0.949 | 0.625 | 0.662 | 0.481 |
| Clients 2 | 0.842 | 0.431 | 0.938 | 0.481 | 1.000 | 0.421 |
| Clients 3 | 0.840 | 0.432 | 0.943 | 0.528 | 1.000 | 0.480 |
| Clients 4 | 0.859 | 0.389 | 0.935 | 0.460 | 1.000 | 0.453 |
| Clients 5 | 0.835 | 0.417 | 0.942 | 0.517 | 1.000 | 0.455 |
| Clients 6 | 0.831 | 0.424 | 0.940 | 0.500 | 1.000 | 0.451 |
| Clients 7 | 0.820 | 0.387 | 0.927 | 0.404 | 1.000 | 0.395 |
| Clients 8 | 0.824 | 0.413 | 0.930 | 0.420 | 1.000 | 0.415 |
| Clients 9 | 0.821 | 0.374 | 0.930 | 0.411 | 1.000 | 0.375 |
| Clients 10 | 0.794 | 0.322 | 0.938 | 0.403 | 1.000 | 0.364 |
| Clients 11 | 0.806 | 0.401 | 0.930 | 0.420 | 1.000 | 0.396 |
| Clients 12 | 0.808 | 0.371 | 0.937 | 0.471 | 1.000 | 0.408 |
| Clients 13 | 0.807 | 0.373 | 0.930 | 0.418 | 1.000 | 0.386 |
| Clients 14 | 0.824 | 0.379 | 0.935 | 0.398 | 1.000 | 0.373 |
| Clients 15 | 0.810 | 0.350 | 0.920 | 0.367 | 1.000 | 0.372 |
| Clients 16 | 0.822 | 0.365 | 0.920 | 0.270 | 1.000 | 0.307 |
| Clients 17 | 0.806 | 0.337 | 0.935 | 0.372 | 1.000 | 0.334 |
| Clients 18 | 0.808 | 0.346 | 0.938 | 0.354 | 1.000 | 0.320 |
| Clients 19 | 0.802 | 0.341 | 0.924 | 0.353 | 1.000 | 0.334 |
| Clients 20 | 0.806 | 0.307 | 0.940 | 1.000 | 1.000 | 0.297 |
| Epochs 1 | 0.836 | 0.435 | 0.937 | 0.473 | 0.571 | 0.430 |
| Epochs 2 | 0.840 | 0.433 | 0.935 | 0.455 | 0.714 | 0.433 |
| Epochs 5 | 0.861 | 0.450 | 0.935 | 0.466 | 0.610 | 0.464 |
| Epochs 10 | 0.852 | 0.455 | 0.937 | 0.474 | 1.000 | 0.443 |

| | | | | | | |
|---|---|---|---|---|---|---|
| Epochs 20 | 0.840 | 0.445 | 0.937 | 0.474 | 1.000 | 0.444 |
| Balanced 60\|40 | 0.828 | 0.436 | 0.933 | 0.437 | 0.675 | 0.414 |
| Balanced 67\|33 | 0.857 | 0.444 | 0.942 | 0.519 | 1.000 | 0.466 |
| Balanced 75\|25 | 0.842 | 0.423 | 0.936 | 0.464 | 0.545 | 0.405 |
| Balanced 80\|20 | 0.835 | 0.441 | 0.945 | 0.544 | 0.922 | 0.481 |
| Balanced 40\|30\|30 | 0.847 | 0.390 | 0.931 | 0.432 | 1.000 | 0.433 |
| Balanced 60\|20\|20 | 0.840 | 0.393 | 0.937 | 0.468 | 1.000 | 0.398 |
| Balanced 80\|10\|10 | 0.848 | 0.447 | 0.943 | 0.534 | 1.000 | 0.434 |
| Balanced 60\|30\|10 | 0.824 | 0.406 | 0.939 | 0.492 | 1.000 | 0.413 |
| Balanced 30\|30\|15\|15\|10 | 0.837 | 0.423 | 0.937 | 0.470 | 1.000 | 0.415 |
| Balanced 60\|10\|10\|10\|10 | 0.822 | 0.394 | 0.938 | 0.481 | 1.000 | 0.401 |

Table 2.39: Best results for plain averaging on SR-HSE (Tox21)

| experiment | roc_auc | pr_auc | accuracy | precision | recall | kappa |
|---|---|---|---|---|---|---|
| Baseline | 0.855 | 0.435 | 0.949 | 0.625 | 0.662 | 0.481 |
| 60\|40 | 0.828 | 0.403 | 0.940 | 0.500 | 0.636 | 0.422 |
| 67\|33 | 0.840 | 0.432 | 0.930 | 0.416 | 0.597 | 0.383 |
| 75\|25 | 0.833 | 0.427 | 0.938 | 0.478 | 0.636 | 0.412 |
| 80\|20 | 0.815 | 0.425 | 0.930 | 0.429 | 0.558 | 0.427 |
| 40\|30\|30 | 0.828 | 0.363 | 0.938 | 0.481 | 1.000 | 0.460 |
| 60\|20\|20 | 0.848 | 0.452 | 0.938 | 0.478 | 1.000 | 0.427 |
| 80\|10\|10 | 0.830 | 0.423 | 0.938 | 0.483 | 0.545 | 0.416 |
| 60\|30\|10 | 0.846 | 0.451 | 0.944 | 0.535 | 0.896 | 0.484 |
| 30\|30\|15\|15\|10 | 0.824 | 0.402 | 0.932 | 0.438 | 1.000 | 0.410 |
| 60\|10\|10\|10\|10 | 0.837 | 0.419 | 0.939 | 0.493 | 0.805 | 0.448 |

Table 2.40: Best results for weighted averaging on SR-HSE (Tox21)

| experiment | roc_auc | pr_auc | accuracy | precision | recall | kappa |
|---|---|---|---|---|---|---|
| Baseline | 0.855 | 0.435 | 0.949 | 0.625 | 0.662 | 0.481 |
| Clients 2 | 0.826 | 0.407 | 0.932 | 0.444 | 1.000 | 0.443 |

| | | | | | | |
|---|---|---|---|---|---|---|
| Clients 3 | 0.827 | 0.434 | 0.935 | 0.457 | 1.000 | 0.427 |
| Clients 4 | 0.855 | 0.424 | 0.935 | 0.455 | 1.000 | 0.430 |
| Clients 5 | 0.826 | 0.364 | 0.919 | 0.348 | 1.000 | 0.357 |
| Clients 6 | 0.830 | 0.413 | 0.916 | 0.360 | 1.000 | 0.382 |
| Clients 7 | 0.817 | 0.389 | 0.925 | 0.391 | 1.000 | 0.377 |
| Clients 8 | 0.813 | 0.410 | 0.922 | 0.384 | 1.000 | 0.391 |
| Clients 9 | 0.820 | 0.359 | 0.917 | 0.347 | 1.000 | 0.345 |
| Clients 10 | 0.822 | 0.381 | 0.908 | 0.315 | 1.000 | 0.332 |
| Clients 11 | 0.809 | 0.364 | 0.918 | 0.351 | 1.000 | 0.373 |
| Clients 12 | 0.820 | 0.388 | 0.910 | 0.314 | 1.000 | 0.335 |
| Clients 13 | 0.809 | 0.369 | 0.928 | 0.405 | 1.000 | 0.380 |
| Clients 14 | 0.831 | 0.383 | 0.931 | 0.370 | 1.000 | 0.397 |
| Clients 15 | 0.802 | 0.329 | 0.916 | 0.310 | 1.000 | 0.331 |
| Clients 16 | 0.821 | 0.330 | 0.913 | 0.320 | 1.000 | 0.335 |
| Clients 17 | 0.800 | 0.328 | 0.936 | 0.312 | 1.000 | 0.344 |
| Clients 18 | 0.825 | 0.370 | 0.936 | 0.394 | 1.000 | 0.372 |
| Clients 19 | 0.820 | 0.343 | 0.925 | 0.322 | 1.000 | 0.351 |
| Clients 20 | 0.806 | 0.311 | 0.934 | 0.315 | 1.000 | 0.340 |
| Epochs 1 | 0.836 | 0.419 | 0.935 | 0.453 | 0.610 | 0.414 |
| Epochs 2 | 0.855 | 0.454 | 0.938 | 0.481 | 0.623 | 0.454 |
| Epochs 5 | 0.836 | 0.401 | 0.931 | 0.421 | 1.000 | 0.415 |
| Epochs 10 | 0.838 | 0.396 | 0.931 | 0.429 | 1.000 | 0.410 |
| Epochs 20 | 0.822 | 0.430 | 0.938 | 0.478 | 1.000 | 0.443 |
| Balanced 60\|40 | 0.843 | 0.434 | 0.937 | 0.470 | 1.000 | 0.427 |
| Balanced 67\|33 | 0.841 | 0.424 | 0.942 | 0.514 | 0.987 | 0.472 |
| Balanced 75\|25 | 0.829 | 0.410 | 0.935 | 0.444 | 1.000 | 0.391 |
| Balanced 80\|20 | 0.835 | 0.450 | 0.942 | 0.515 | 1.000 | 0.452 |
| Balanced 40\|30\|30 | 0.831 | 0.399 | 0.938 | 0.481 | 1.000 | 0.401 |
| Balanced 60\|20\|20 | 0.839 | 0.454 | 0.934 | 0.451 | 1.000 | 0.430 |
| Balanced 80\|10\|10 | 0.846 | 0.441 | 0.935 | 0.463 | 1.000 | 0.447 |
| Balanced 60\|30\|10 | 0.871 | 0.464 | 0.939 | 0.492 | 1.000 | 0.456 |
| Balanced 30\|30\|15\|15\|10 | 0.838 | 0.425 | 0.931 | 0.425 | 1.000 | 0.409 |
| Balanced 60\|10\|10\|10\|10 | 0.831 | 0.398 | 0.931 | 0.420 | 1.000 | 0.390 |

Table 2.41: Best results for benchmarked averaging on SR-HSE (Tox21)

### 2.4.12 Tox21 - SR-MMP

| experiment | roc_auc | pr_auc | accuracy | precision | recall | kappa |
|---|---|---|---|---|---|---|
| Baseline | 0.924 | 0.729 | 0.889 | 0.616 | 0.900 | 0.614 |
| Clients 2 | 0.921 | 0.736 | 0.888 | 0.612 | 0.989 | 0.604 |
| Clients 3 | 0.905 | 0.704 | 0.874 | 0.569 | 1.000 | 0.567 |
| Clients 4 | 0.914 | 0.744 | 0.875 | 0.563 | 1.000 | 0.593 |
| Clients 5 | 0.908 | 0.705 | 0.886 | 0.601 | 1.000 | 0.604 |
| Clients 6 | 0.911 | 0.702 | 0.866 | 0.543 | 1.000 | 0.557 |
| Clients 7 | 0.911 | 0.720 | 0.877 | 0.570 | 1.000 | 0.591 |
| Clients 8 | 0.900 | 0.686 | 0.856 | 0.520 | 1.000 | 0.544 |
| Clients 9 | 0.914 | 0.712 | 0.868 | 0.548 | 1.000 | 0.572 |
| Clients 10 | 0.909 | 0.692 | 0.870 | 0.553 | 1.000 | 0.570 |
| Clients 11 | 0.911 | 0.719 | 0.864 | 0.538 | 1.000 | 0.561 |
| Clients 12 | 0.899 | 0.651 | 0.855 | 0.517 | 1.000 | 0.534 |
| Clients 13 | 0.909 | 0.695 | 0.867 | 0.546 | 1.000 | 0.555 |
| Clients 14 | 0.897 | 0.679 | 0.862 | 0.534 | 1.000 | 0.552 |
| Clients 15 | 0.901 | 0.667 | 0.856 | 0.519 | 1.000 | 0.534 |
| Clients 16 | 0.896 | 0.616 | 0.779 | 0.398 | 1.000 | 0.425 |
| Clients 17 | 0.906 | 0.670 | 0.866 | 0.544 | 1.000 | 0.551 |
| Clients 18 | 0.904 | 0.674 | 0.848 | 0.502 | 1.000 | 0.530 |
| Clients 19 | 0.892 | 0.642 | 0.852 | 0.512 | 1.000 | 0.508 |
| Clients 20 | 0.903 | 0.658 | 0.862 | 0.534 | 1.000 | 0.554 |
| Epochs 1 | 0.928 | 0.744 | 0.881 | 0.583 | 1.000 | 0.605 |
| Epochs 2 | 0.919 | 0.749 | 0.879 | 0.581 | 1.000 | 0.599 |
| Epochs 5 | 0.920 | 0.741 | 0.889 | 0.606 | 0.944 | 0.615 |
| Epochs 10 | 0.915 | 0.719 | 0.884 | 0.601 | 0.989 | 0.593 |
| Epochs 20 | 0.912 | 0.693 | 0.886 | 0.601 | 1.000 | 0.604 |
| Balanced 60\|40 | 0.920 | 0.734 | 0.888 | 0.612 | 0.961 | 0.598 |
| Balanced 67\|33 | 0.919 | 0.741 | 0.872 | 0.561 | 0.861 | 0.567 |
| Balanced 75\|25 | 0.917 | 0.730 | 0.879 | 0.583 | 0.978 | 0.580 |
| Balanced 80\|20 | 0.914 | 0.722 | 0.879 | 0.581 | 0.911 | 0.595 |
| Balanced 40\|30\|30 | 0.913 | 0.731 | 0.884 | 0.604 | 1.000 | 0.592 |

| | | | | | | |
|---|---|---|---|---|---|---|
| Balanced 60\|20\|20 | 0.918 | 0.720 | 0.880 | 0.588 | 0.989 | 0.585 |
| Balanced 80\|10\|10 | 0.920 | 0.739 | 0.873 | 0.561 | 1.000 | 0.572 |
| Balanced 60\|30\|10 | 0.922 | 0.740 | 0.889 | 0.610 | 1.000 | 0.609 |
| Balanced 30\|30\|15\|15\|10 | 0.907 | 0.712 | 0.880 | 0.584 | 1.000 | 0.586 |
| Balanced 60\|10\|10\|10\|10 | 0.905 | 0.703 | 0.875 | 0.568 | 1.000 | 0.581 |

Table 2.42: Best results for plain averaging on SR-MMP (Tox21)

| experiment | roc_auc | pr_auc | accuracy | precision | recall | kappa |
|---|---|---|---|---|---|---|
| Baseline | 0.924 | 0.729 | 0.889 | 0.616 | 0.900 | 0.614 |
| 60\|40 | 0.913 | 0.720 | 0.885 | 0.604 | 0.856 | 0.601 |
| 67\|33 | 0.918 | 0.726 | 0.881 | 0.585 | 0.928 | 0.593 |
| 75\|25 | 0.923 | 0.737 | 0.884 | 0.596 | 0.872 | 0.591 |
| 80\|20 | 0.920 | 0.728 | 0.890 | 0.623 | 0.856 | 0.608 |
| 40\|30\|30 | 0.915 | 0.715 | 0.871 | 0.558 | 1.000 | 0.563 |
| 60\|20\|20 | 0.917 | 0.696 | 0.876 | 0.571 | 1.000 | 0.577 |
| 80\|10\|10 | 0.916 | 0.714 | 0.883 | 0.589 | 0.867 | 0.599 |
| 60\|30\|10 | 0.921 | 0.723 | 0.885 | 0.600 | 0.911 | 0.598 |
| 30\|30\|15\|15\|10 | 0.910 | 0.721 | 0.879 | 0.582 | 1.000 | 0.579 |
| 60\|10\|10\|10\|10 | 0.914 | 0.721 | 0.879 | 0.586 | 0.961 | 0.580 |

Table 2.43: Best results for weighted averaging on SR-MMP (Tox21)

| experiment | roc_auc | pr_auc | accuracy | precision | recall | kappa |
|---|---|---|---|---|---|---|
| Baseline | 0.924 | 0.729 | 0.889 | 0.616 | 0.900 | 0.614 |
| Clients 2 | 0.916 | 0.723 | 0.887 | 0.608 | 1.000 | 0.601 |
| Clients 3 | 0.911 | 0.687 | 0.867 | 0.547 | 1.000 | 0.564 |
| Clients 4 | 0.910 | 0.720 | 0.877 | 0.576 | 1.000 | 0.577 |
| Clients 5 | 0.913 | 0.673 | 0.857 | 0.521 | 1.000 | 0.555 |
| Clients 6 | 0.905 | 0.669 | 0.859 | 0.526 | 1.000 | 0.549 |
| Clients 7 | 0.900 | 0.692 | 0.846 | 0.498 | 1.000 | 0.524 |
| Clients 8 | 0.905 | 0.657 | 0.849 | 0.504 | 1.000 | 0.530 |

| | | | | | | |
|---|---|---|---|---|---|---|
| Clients 9 | 0.915 | 0.701 | 0.862 | 0.532 | 1.000 | 0.569 |
| Clients 10 | 0.905 | 0.681 | 0.867 | 0.543 | 1.000 | 0.572 |
| Clients 11 | 0.908 | 0.677 | 0.857 | 0.522 | 1.000 | 0.544 |
| Clients 12 | 0.898 | 0.653 | 0.839 | 0.484 | 1.000 | 0.516 |
| Clients 13 | 0.899 | 0.661 | 0.859 | 0.526 | 1.000 | 0.548 |
| Clients 14 | 0.900 | 0.668 | 0.862 | 0.534 | 1.000 | 0.554 |
| Clients 15 | 0.900 | 0.653 | 0.856 | 0.518 | 1.000 | 0.544 |
| Clients 16 | 0.889 | 0.659 | 0.860 | 0.530 | 1.000 | 0.534 |
| Clients 17 | 0.886 | 0.647 | 0.838 | 0.481 | 1.000 | 0.505 |
| Clients 18 | 0.890 | 0.636 | 0.850 | 0.505 | 1.000 | 0.532 |
| Clients 19 | 0.891 | 0.638 | 0.839 | 0.483 | 1.000 | 0.502 |
| Clients 20 | 0.906 | 0.653 | 0.856 | 0.520 | 1.000 | 0.549 |
| Epochs 1 | 0.918 | 0.746 | 0.870 | 0.555 | 0.906 | 0.569 |
| Epochs 2 | 0.919 | 0.735 | 0.878 | 0.573 | 0.900 | 0.591 |
| Epochs 5 | 0.914 | 0.719 | 0.889 | 0.607 | 0.989 | 0.614 |
| Epochs 10 | 0.913 | 0.712 | 0.888 | 0.608 | 1.000 | 0.605 |
| Epochs 20 | 0.914 | 0.727 | 0.878 | 0.574 | 1.000 | 0.588 |
| Balanced 60\|40 | 0.913 | 0.729 | 0.887 | 0.606 | 0.989 | 0.601 |
| Balanced 67\|33 | 0.911 | 0.708 | 0.886 | 0.605 | 1.000 | 0.597 |
| Balanced 75\|25 | 0.917 | 0.719 | 0.879 | 0.581 | 1.000 | 0.595 |
| Balanced 80\|20 | 0.920 | 0.709 | 0.879 | 0.577 | 1.000 | 0.593 |
| Balanced 40\|30\|30 | 0.912 | 0.710 | 0.881 | 0.584 | 1.000 | 0.594 |
| Balanced 60\|20\|20 | 0.918 | 0.694 | 0.869 | 0.550 | 1.000 | 0.573 |
| Balanced 80\|10\|10 | 0.919 | 0.728 | 0.870 | 0.553 | 1.000 | 0.572 |
| Balanced 60\|30\|10 | 0.907 | 0.666 | 0.864 | 0.538 | 1.000 | 0.563 |
| Balanced 30\|30\|15\|15\|10 | 0.914 | 0.705 | 0.867 | 0.544 | 1.000 | 0.577 |
| Balanced 60\|10\|10\|10\|10 | 0.909 | 0.697 | 0.872 | 0.558 | 1.000 | 0.577 |

Table 2.44: Best results for benchmarked averaging on SR-MMP (Tox21)

### 2.4.13 Tox21 - SR-p53

| experiment | roc_auc | pr_auc | accuracy | precision | recall | kappa |
|---|---|---|---|---|---|---|

| | | | | | | |
|---|---|---|---|---|---|---|
| Baseline | 0.884 | 0.368 | 0.937 | 0.365 | 0.696 | 0.350 |
| Clients 2 | 0.868 | 0.371 | 0.936 | 0.386 | 0.855 | 0.387 |
| Clients 3 | 0.874 | 0.384 | 0.927 | 0.340 | 1.000 | 0.361 |
| Clients 4 | 0.865 | 0.328 | 0.927 | 0.337 | 0.971 | 0.357 |
| Clients 5 | 0.846 | 0.331 | 0.935 | 0.316 | 1.000 | 0.332 |
| Clients 6 | 0.879 | 0.365 | 0.927 | 0.296 | 0.986 | 0.332 |
| Clients 7 | 0.867 | 0.342 | 0.928 | 0.280 | 1.000 | 0.316 |
| Clients 8 | 0.864 | 0.320 | 0.933 | 0.248 | 1.000 | 0.299 |
| Clients 9 | 0.865 | 0.375 | 0.928 | 0.301 | 1.000 | 0.343 |
| Clients 10 | 0.864 | 0.352 | 0.941 | 0.289 | 1.000 | 0.321 |
| Clients 11 | 0.839 | 0.280 | 0.929 | 0.252 | 1.000 | 0.282 |
| Clients 12 | 0.839 | 0.280 | 0.928 | 0.274 | 1.000 | 0.288 |
| Clients 13 | 0.822 | 0.318 | 0.941 | 0.263 | 1.000 | 0.303 |
| Clients 14 | 0.835 | 0.320 | 0.944 | 0.288 | 1.000 | 0.334 |
| Clients 15 | 0.819 | 0.293 | 0.913 | 0.239 | 1.000 | 0.273 |
| Clients 16 | 0.829 | 0.234 | 0.917 | 0.189 | 1.000 | 0.229 |
| Clients 17 | 0.837 | 0.260 | 0.945 | 0.286 | 1.000 | 0.295 |
| Clients 18 | 0.841 | 0.253 | 0.946 | 0.252 | 1.000 | 0.269 |
| Clients 19 | 0.835 | 0.290 | 0.934 | 0.264 | 1.000 | 0.285 |
| Clients 20 | 0.844 | 0.284 | 0.949 | 0.279 | 1.000 | 0.319 |
| Epochs 1 | 0.895 | 0.419 | 0.930 | 0.352 | 0.681 | 0.375 |
| Epochs 2 | 0.867 | 0.359 | 0.931 | 0.337 | 0.609 | 0.379 |
| Epochs 5 | 0.873 | 0.394 | 0.930 | 0.352 | 0.667 | 0.388 |
| Epochs 10 | 0.875 | 0.371 | 0.928 | 0.337 | 0.638 | 0.357 |
| Epochs 20 | 0.855 | 0.332 | 0.925 | 0.330 | 1.000 | 0.347 |
| Balanced 60\|40 | 0.869 | 0.363 | 0.928 | 0.330 | 0.739 | 0.358 |
| Balanced 67\|33 | 0.868 | 0.366 | 0.926 | 0.318 | 0.652 | 0.349 |
| Balanced 75\|25 | 0.874 | 0.435 | 0.930 | 0.362 | 0.855 | 0.381 |
| Balanced 80\|20 | 0.888 | 0.372 | 0.941 | 0.410 | 0.913 | 0.402 |
| Balanced 40\|30\|30 | 0.859 | 0.341 | 0.930 | 0.352 | 0.971 | 0.363 |
| Balanced 60\|20\|20 | 0.853 | 0.348 | 0.939 | 0.397 | 0.884 | 0.362 |
| Balanced 80\|10\|10 | 0.867 | 0.371 | 0.931 | 0.356 | 0.971 | 0.371 |
| Balanced 60\|30\|10 | 0.861 | 0.430 | 0.935 | 0.372 | 1.000 | 0.364 |
| Balanced 30\|30\|15\|15\|10 | 0.861 | 0.380 | 0.939 | 0.365 | 1.000 | 0.367 |

| | | | | | | |
|---|---|---|---|---|---|---|
| Balanced 60\|10\|10\|10\|10 | 0.865 | 0.339 | 0.923 | 0.320 | 0.957 | 0.344 |

Table 2.45: Best results for plain averaging on SR-p53 (Tox21)

| experiment | roc_auc | pr_auc | accuracy | precision | recall | kappa |
|---|---|---|---|---|---|---|
| Baseline | 0.884 | 0.368 | 0.937 | 0.365 | 0.696 | 0.350 |
| 60\|40 | 0.869 | 0.423 | 0.937 | 0.390 | 0.594 | 0.378 |
| 67\|33 | 0.867 | 0.382 | 0.933 | 0.366 | 0.855 | 0.383 |
| 75\|25 | 0.886 | 0.398 | 0.929 | 0.333 | 0.768 | 0.363 |
| 80\|20 | 0.870 | 0.424 | 0.946 | 0.365 | 0.638 | 0.368 |
| 40\|30\|30 | 0.881 | 0.376 | 0.923 | 0.327 | 0.913 | 0.354 |
| 60\|20\|20 | 0.876 | 0.384 | 0.929 | 0.341 | 1.000 | 0.345 |
| 80\|10\|10 | 0.879 | 0.374 | 0.932 | 0.322 | 0.681 | 0.362 |
| 60\|30\|10 | 0.885 | 0.367 | 0.929 | 0.321 | 0.623 | 0.358 |
| 30\|30\|15\|15\|10 | 0.869 | 0.357 | 0.922 | 0.301 | 0.957 | 0.319 |
| 60\|10\|10\|10\|10 | 0.850 | 0.331 | 0.927 | 0.326 | 0.870 | 0.329 |

Table 2.46: Best results for weighted averaging on SR-p53 (Tox21)

| experiment | roc_auc | pr_auc | accuracy | precision | recall | kappa |
|---|---|---|---|---|---|---|
| Baseline | 0.884 | 0.368 | 0.937 | 0.365 | 0.696 | 0.350 |
| Clients 2 | 0.884 | 0.366 | 0.917 | 0.305 | 1.000 | 0.343 |
| Clients 3 | 0.871 | 0.388 | 0.921 | 0.304 | 1.000 | 0.334 |
| Clients 4 | 0.847 | 0.340 | 0.916 | 0.275 | 1.000 | 0.297 |
| Clients 5 | 0.869 | 0.311 | 0.916 | 0.298 | 1.000 | 0.338 |
| Clients 6 | 0.852 | 0.273 | 0.884 | 0.240 | 1.000 | 0.291 |
| Clients 7 | 0.864 | 0.340 | 0.898 | 0.253 | 1.000 | 0.302 |
| Clients 8 | 0.841 | 0.289 | 0.913 | 0.229 | 1.000 | 0.256 |
| Clients 9 | 0.847 | 0.276 | 0.894 | 0.247 | 1.000 | 0.289 |
| Clients 10 | 0.866 | 0.306 | 0.900 | 0.258 | 1.000 | 0.312 |
| Clients 11 | 0.849 | 0.316 | 0.924 | 0.250 | 1.000 | 0.300 |
| Clients 12 | 0.839 | 0.319 | 0.903 | 0.212 | 1.000 | 0.247 |

| | | | | | | |
|---|---|---|---|---|---|---|
| Clients 13 | 0.836 | 0.313 | 0.930 | 0.244 | 1.000 | 0.300 |
| Clients 14 | 0.829 | 0.289 | 0.938 | 0.243 | 1.000 | 0.284 |
| Clients 15 | 0.832 | 0.272 | 0.915 | 0.254 | 1.000 | 0.287 |
| Clients 16 | 0.836 | 0.270 | 0.889 | 0.214 | 1.000 | 0.244 |
| Clients 17 | 0.830 | 0.265 | 0.946 | 0.205 | 1.000 | 0.239 |
| Clients 18 | 0.850 | 0.261 | 0.947 | 0.243 | 1.000 | 0.273 |
| Clients 19 | 0.823 | 0.255 | 0.927 | 0.199 | 1.000 | 0.232 |
| Clients 20 | 0.838 | 0.288 | 0.942 | 0.252 | 1.000 | 0.297 |
| Epochs 1 | 0.872 | 0.398 | 0.925 | 0.313 | 0.681 | 0.350 |
| Epochs 2 | 0.898 | 0.437 | 0.930 | 0.355 | 0.739 | 0.373 |
| Epochs 5 | 0.879 | 0.422 | 0.931 | 0.360 | 0.913 | 0.369 |
| Epochs 10 | 0.870 | 0.410 | 0.931 | 0.349 | 1.000 | 0.378 |
| Epochs 20 | 0.861 | 0.339 | 0.920 | 0.311 | 1.000 | 0.344 |
| Balanced 60\|40 | 0.843 | 0.399 | 0.933 | 0.358 | 1.000 | 0.351 |
| Balanced 67\|33 | 0.876 | 0.396 | 0.931 | 0.356 | 1.000 | 0.371 |
| Balanced 75\|25 | 0.893 | 0.376 | 0.920 | 0.328 | 1.000 | 0.371 |
| Balanced 80\|20 | 0.878 | 0.421 | 0.918 | 0.316 | 1.000 | 0.357 |
| Balanced 40\|30\|30 | 0.876 | 0.425 | 0.918 | 0.303 | 1.000 | 0.332 |
| Balanced 60\|20\|20 | 0.876 | 0.385 | 0.911 | 0.290 | 1.000 | 0.334 |
| Balanced 80\|10\|10 | 0.877 | 0.424 | 0.920 | 0.318 | 1.000 | 0.352 |
| Balanced 60\|30\|10 | 0.871 | 0.402 | 0.931 | 0.363 | 1.000 | 0.377 |
| Balanced 30\|30\|15\|15\|10 | 0.862 | 0.369 | 0.906 | 0.263 | 1.000 | 0.306 |
| Balanced 60\|10\|10\|10\|10 | 0.872 | 0.347 | 0.919 | 0.295 | 1.000 | 0.322 |

Table 2.47: Best results for benchmarked averaging on SR-p53 (Tox21)

### 2.4.14 Tox21 - Mean

| experiment | roc_auc | pr_auc | accuracy | precision | recall | kappa |
|---|---|---|---|---|---|---|
| Baseline | 0.869 | 0.513 | 0.932 | 0.595 | 0.692 | 0.492 |
| Clients 2 | 0.838 | 0.482 | 0.914 | 0.431 | 0.908 | 0.420 |
| Clients 3 | 0.844 | 0.481 | 0.912 | 0.425 | 0.984 | 0.420 |
| Clients 4 | 0.837 | 0.478 | 0.912 | 0.428 | 0.990 | 0.426 |
| Clients 5 | 0.832 | 0.484 | 0.915 | 0.444 | 0.995 | 0.430 |

| | | | | | | |
|---|---|---|---|---|---|---|
| Clients 6 | 0.836 | 0.478 | 0.910 | 0.422 | 0.995 | 0.416 |
| Clients 7 | 0.832 | 0.475 | 0.910 | 0.406 | 0.994 | 0.403 |
| Clients 8 | 0.836 | 0.462 | 0.896 | 0.376 | 1.000 | 0.385 |
| Clients 9 | 0.836 | 0.480 | 0.903 | 0.396 | 0.998 | 0.405 |
| Clients 10 | 0.831 | 0.479 | 0.914 | 0.415 | 1.000 | 0.410 |
| Clients 11 | 0.827 | 0.474 | 0.904 | 0.410 | 0.998 | 0.406 |
| Clients 12 | 0.818 | 0.444 | 0.900 | 0.383 | 1.000 | 0.386 |
| Clients 13 | 0.827 | 0.466 | 0.911 | 0.390 | 1.000 | 0.395 |
| Clients 14 | 0.828 | 0.459 | 0.908 | 0.397 | 1.000 | 0.394 |
| Clients 15 | 0.824 | 0.442 | 0.894 | 0.368 | 1.000 | 0.378 |
| Clients 16 | 0.828 | 0.420 | 0.865 | 0.280 | 1.000 | 0.304 |
| Clients 17 | 0.818 | 0.437 | 0.904 | 0.388 | 1.000 | 0.385 |
| Clients 18 | 0.828 | 0.444 | 0.899 | 0.383 | 1.000 | 0.383 |
| Clients 19 | 0.826 | 0.450 | 0.901 | 0.392 | 1.000 | 0.385 |
| Clients 20 | 0.822 | 0.431 | 0.916 | 0.588 | 1.000 | 0.386 |
| Epochs 1 | 0.846 | 0.502 | 0.913 | 0.447 | 0.623 | 0.426 |
| Epochs 2 | 0.845 | 0.496 | 0.908 | 0.427 | 0.630 | 0.425 |
| Epochs 5 | 0.841 | 0.492 | 0.916 | 0.439 | 0.628 | 0.434 |
| Epochs 10 | 0.845 | 0.488 | 0.914 | 0.441 | 0.666 | 0.425 |
| Epochs 20 | 0.842 | 0.470 | 0.912 | 0.422 | 0.982 | 0.418 |
| Balanced 60\|40 | 0.833 | 0.486 | 0.914 | 0.430 | 0.610 | 0.421 |
| Balanced 67\|33 | 0.843 | 0.491 | 0.914 | 0.441 | 0.848 | 0.430 |
| Balanced 75\|25 | 0.836 | 0.491 | 0.912 | 0.439 | 0.762 | 0.434 |
| Balanced 80\|20 | 0.843 | 0.487 | 0.914 | 0.469 | 0.733 | 0.432 |
| Balanced 40\|30\|30 | 0.835 | 0.481 | 0.912 | 0.431 | 0.993 | 0.422 |
| Balanced 60\|20\|20 | 0.835 | 0.476 | 0.917 | 0.456 | 0.959 | 0.438 |
| Balanced 80\|10\|10 | 0.839 | 0.488 | 0.913 | 0.439 | 0.890 | 0.425 |
| Balanced 60\|30\|10 | 0.834 | 0.484 | 0.916 | 0.441 | 0.987 | 0.429 |
| Balanced 30\|30\|15\|15\|10 | 0.835 | 0.490 | 0.910 | 0.424 | 0.997 | 0.413 |
| Balanced 60\|10\|10\|10\|10 | 0.830 | 0.470 | 0.911 | 0.436 | 0.990 | 0.417 |

Table 2.48: Best results for plain averaging on Tox21 (average)

| experiment | roc_auc | pr_auc | accuracy | precision | recall | kappa |
|---|---|---|---|---|---|---|

| | | | | | | |
|---|---|---|---|---|---|---|
| Baseline | 0.869 | 0.513 | 0.932 | 0.595 | 0.692 | 0.492 |
| 60\|40 | 0.846 | 0.487 | 0.916 | 0.453 | 0.619 | 0.436 |
| 67\|33 | 0.841 | 0.474 | 0.912 | 0.441 | 0.620 | 0.423 |
| 75\|25 | 0.842 | 0.472 | 0.912 | 0.429 | 0.636 | 0.415 |
| 80\|20 | 0.841 | 0.478 | 0.907 | 0.420 | 0.601 | 0.412 |
| 40\|30\|30 | 0.841 | 0.483 | 0.909 | 0.424 | 0.941 | 0.421 |
| 60\|20\|20 | 0.835 | 0.488 | 0.910 | 0.422 | 0.938 | 0.413 |
| 80\|10\|10 | 0.844 | 0.476 | 0.913 | 0.447 | 0.617 | 0.428 |
| 60\|30\|10 | 0.844 | 0.478 | 0.914 | 0.441 | 0.610 | 0.426 |
| 30\|30\|15\|15\|10 | 0.837 | 0.484 | 0.912 | 0.421 | 0.990 | 0.417 |
| 60\|10\|10\|10\|10 | 0.837 | 0.481 | 0.913 | 0.437 | 0.875 | 0.422 |

Table 2.49: Best results for weighted averaging on Tox21 (average)

| experiment | roc_auc | pr_auc | accuracy | precision | recall | kappa |
|---|---|---|---|---|---|---|
| Baseline | 0.869 | 0.513 | 0.932 | 0.595 | 0.692 | 0.492 |
| Clients 2 | 0.839 | 0.474 | 0.913 | 0.427 | 1.000 | 0.422 |
| Clients 3 | 0.843 | 0.470 | 0.907 | 0.407 | 1.000 | 0.409 |
| Clients 4 | 0.838 | 0.469 | 0.905 | 0.395 | 1.000 | 0.398 |
| Clients 5 | 0.841 | 0.457 | 0.899 | 0.378 | 0.998 | 0.388 |
| Clients 6 | 0.832 | 0.453 | 0.887 | 0.345 | 1.000 | 0.365 |
| Clients 7 | 0.832 | 0.464 | 0.895 | 0.368 | 1.000 | 0.379 |
| Clients 8 | 0.834 | 0.464 | 0.896 | 0.364 | 1.000 | 0.380 |
| Clients 9 | 0.839 | 0.451 | 0.894 | 0.363 | 1.000 | 0.376 |
| Clients 10 | 0.836 | 0.449 | 0.894 | 0.371 | 1.000 | 0.376 |
| Clients 11 | 0.833 | 0.450 | 0.896 | 0.356 | 1.000 | 0.373 |
| Clients 12 | 0.833 | 0.456 | 0.883 | 0.338 | 1.000 | 0.364 |
| Clients 13 | 0.824 | 0.458 | 0.889 | 0.348 | 1.000 | 0.367 |
| Clients 14 | 0.831 | 0.451 | 0.897 | 0.373 | 1.000 | 0.385 |
| Clients 15 | 0.823 | 0.442 | 0.893 | 0.349 | 1.000 | 0.362 |
| Clients 16 | 0.825 | 0.426 | 0.879 | 0.324 | 1.000 | 0.340 |
| Clients 17 | 0.817 | 0.434 | 0.905 | 0.318 | 1.000 | 0.336 |
| Clients 18 | 0.816 | 0.453 | 0.896 | 0.368 | 1.000 | 0.372 |

| | | | | | | |
|---|---|---|---|---|---|---|
| Clients 19 | 0.819 | 0.433 | 0.875 | 0.313 | 1.000 | 0.337 |
| Clients 20 | 0.825 | 0.434 | 0.891 | 0.369 | 1.000 | 0.373 |
| Epochs 1 | 0.847 | 0.493 | 0.903 | 0.415 | 0.616 | 0.405 |
| Epochs 2 | 0.848 | 0.496 | 0.909 | 0.432 | 0.622 | 0.420 |
| Epochs 5 | 0.847 | 0.481 | 0.911 | 0.438 | 0.885 | 0.428 |
| Epochs 10 | 0.838 | 0.480 | 0.913 | 0.444 | 0.998 | 0.426 |
| Epochs 20 | 0.844 | 0.478 | 0.911 | 0.422 | 0.995 | 0.413 |
| Balanced 60\|40 | 0.837 | 0.477 | 0.912 | 0.434 | 0.990 | 0.428 |
| Balanced 67\|33 | 0.846 | 0.482 | 0.911 | 0.431 | 0.957 | 0.429 |
| Balanced 75\|25 | 0.847 | 0.491 | 0.912 | 0.430 | 0.994 | 0.427 |
| Balanced 80\|20 | 0.842 | 0.489 | 0.911 | 0.440 | 0.992 | 0.430 |
| Balanced 40\|30\|30 | 0.848 | 0.494 | 0.907 | 0.416 | 1.000 | 0.413 |
| Balanced 60\|20\|20 | 0.844 | 0.477 | 0.899 | 0.395 | 1.000 | 0.396 |
| Balanced 80\|10\|10 | 0.845 | 0.486 | 0.911 | 0.437 | 0.998 | 0.424 |
| Balanced 60\|30\|10 | 0.853 | 0.486 | 0.906 | 0.417 | 0.996 | 0.413 |
| Balanced 30\|30\|15\|15\|10 | 0.848 | 0.476 | 0.899 | 0.407 | 1.000 | 0.400 |
| Balanced 60\|10\|10\|10\|10 | 0.841 | 0.476 | 0.907 | 0.421 | 1.000 | 0.413 |

Table 2.50: Best results for benchmarked averaging on Tox21 (average)

### 2.4.15 ADME - CLint

| experiment | r2 | mae | rmse |
|---|---|---|---|
| Baseline | 0.489 | 0.806 | 1.536 |
| Clients 2 | 0.508 | 2.877 | 3.392 |
| Clients 3 | 0.492 | 3.670 | 4.092 |
| Clients 4 | 0.493 | 2.871 | 3.392 |
| Clients 5 | 0.496 | 3.357 | 3.814 |
| Clients 6 | 0.450 | 3.426 | 3.876 |
| Clients 7 | 0.456 | 3.253 | 3.721 |
| Clients 8 | 0.373 | 3.505 | 3.947 |
| Clients 9 | 0.415 | 3.345 | 3.803 |
| Clients 10 | 0.336 | 3.428 | 3.877 |
| Clients 11 | 0.367 | 3.464 | 3.910 |

| | | | |
|---|---|---|---|
| Clients 12 | 0.255 | 3.487 | 3.932 |
| Clients 13 | 0.344 | 3.431 | 3.880 |
| Clients 14 | 0.327 | 3.501 | 3.944 |
| Clients 15 | 0.292 | 3.463 | 3.909 |
| Clients 16 | 0.227 | 3.534 | 3.973 |
| Clients 17 | 0.311 | 3.465 | 3.911 |
| Clients 18 | 0.191 | 3.451 | 3.899 |
| Clients 19 | 0.269 | 3.474 | 3.920 |
| Clients 20 | 0.121 | 3.471 | 3.916 |
| Epochs 1 | 0.470 | 3.446 | 3.896 |
| Epochs 2 | 0.507 | 3.570 | 3.996 |
| Epochs 5 | 0.510 | 2.998 | 3.498 |
| Epochs 10 | 0.525 | 3.379 | 3.837 |
| Epochs 20 | 0.484 | 2.524 | 3.096 |
| Balanced 60\|40 | 0.517 | 2.358 | 2.957 |
| Balanced 67\|33 | 0.492 | 3.065 | 3.555 |
| Balanced 75\|25 | 0.522 | 3.297 | 3.759 |
| Balanced 80\|20 | 0.497 | 3.095 | 3.578 |
| Balanced 40\|30\|30 | 0.504 | 3.590 | 4.022 |
| Balanced 60\|20\|20 | 0.494 | 3.507 | 3.951 |
| Balanced 80\|10\|10 | 0.507 | 3.115 | 3.600 |
| Balanced 60\|30\|10 | 0.519 | 3.109 | 3.598 |
| Balanced 30\|30\|15\|15\|10 | 0.479 | 3.305 | 3.766 |
| Balanced 60\|10\|10\|10\|10 | 0.468 | 3.560 | 3.997 |

Table 2.51: Best results for plain averaging on CLint (ADME)

| experiment | r2 | mae | rmse |
|---|---|---|---|
| Baseline | 0.489 | 0.806 | 1.536 |
| 60\|40 | 0.510 | 1.673 | 2.408 |
| 67\|33 | 0.505 | 3.200 | 3.669 |
| 75\|25 | 0.504 | 1.921 | 2.602 |
| 80\|20 | 0.513 | 1.627 | 2.367 |
| 40\|30\|30 | 0.499 | 3.356 | 3.814 |

| | | | |
|---|---|---|---|
| 60\|20\|20 | 0.513 | 2.601 | 3.158 |
| 80\|10\|10 | 0.506 | 2.001 | 2.653 |
| 60\|30\|10 | 0.492 | 3.250 | 3.723 |
| 30\|30\|15\|15\|10 | 0.499 | 3.376 | 3.831 |
| 60\|10\|10\|10\|10 | 0.494 | 3.191 | 3.662 |

Table 2.52: Best results for weighted averaging on CLint (ADME)

| experiment | r2 | mae | rmse |
|---|---|---|---|
| Baseline | 0.489 | 0.806 | 1.536 |
| Clients 2 | 0.510 | 2.236 | 2.859 |
| Clients 3 | 0.379 | 1.174 | 2.242 |
| Clients 4 | 0.477 | 3.374 | 3.830 |
| Clients 5 | 0.496 | 3.340 | 3.800 |
| Clients 6 | 0.480 | 3.428 | 3.879 |
| Clients 7 | 0.442 | 3.482 | 3.927 |
| Clients 8 | 0.462 | 3.410 | 3.862 |
| Clients 9 | 0.484 | 3.363 | 3.820 |
| Clients 10 | 0.194 | 1.242 | 2.009 |
| Clients 11 | 0.384 | 3.467 | 3.913 |
| Clients 12 | 0.379 | 3.465 | 3.911 |
| Clients 13 | 0.402 | 3.434 | 3.883 |
| Clients 14 | 0.413 | 3.377 | 3.832 |
| Clients 15 | 0.396 | 3.477 | 3.923 |
| Clients 16 | 0.370 | 3.513 | 3.954 |
| Clients 17 | 0.351 | 3.511 | 3.953 |
| Clients 18 | 0.345 | 3.330 | 3.790 |
| Clients 19 | 0.312 | 3.465 | 3.911 |
| Clients 20 | 0.370 | 1.237 | 2.029 |
| Epochs 1 | 0.515 | 2.266 | 2.883 |
| Epochs 2 | 0.527 | 3.237 | 3.709 |
| Epochs 5 | 0.498 | 2.132 | 2.772 |
| Epochs 10 | 0.486 | 2.015 | 2.674 |
| Epochs 20 | 0.493 | 2.551 | 3.124 |

| | | | |
|---|---|---|---|
| Balanced 60\|40 | 0.510 | 2.294 | 2.906 |
| Balanced 67\|33 | 0.466 | 1.062 | 2.022 |
| Balanced 75\|25 | 0.518 | 2.733 | 3.283 |
| Balanced 80\|20 | 0.483 | 1.723 | 2.387 |
| Balanced 40\|30\|30 | 0.498 | 2.982 | 3.486 |
| Balanced 60\|20\|20 | 0.520 | 2.752 | 3.290 |
| Balanced 80\|10\|10 | 0.492 | 2.182 | 2.818 |
| Balanced 60\|30\|10 | 0.494 | 3.321 | 3.789 |
| Balanced 30\|30\|15\|15\|10 | 0.503 | 3.587 | 4.020 |
| Balanced 60\|10\|10\|10\|10 | 0.476 | 1.067 | 2.269 |

Table 2.53: Best results for benchmarked averaging on CLint (ADME)

### 2.4.16 ADME - FeHuman

| experiment | roc_auc | pr_auc | accuracy | precision | recall | kappa |
|---|---|---|---|---|---|---|
| Baseline | 0.921 | 0.985 | 0.897 | 0.932 | 0.966 | 0.572 |
| Clients 2 | 0.871 | 0.977 | 0.912 | 0.919 | 1.000 | 0.579 |
| Clients 3 | 0.847 | 0.973 | 0.882 | 0.903 | 1.000 | 0.438 |
| Clients 4 | 0.874 | 0.978 | 0.897 | 0.918 | 1.000 | 0.531 |
| Clients 5 | 0.853 | 0.975 | 0.882 | 0.917 | 1.000 | 0.489 |
| Clients 6 | 0.800 | 0.962 | 0.853 | 0.909 | 1.000 | 0.327 |
| Clients 7 | 0.843 | 0.972 | 0.897 | 0.917 | 1.000 | 0.489 |
| Clients 8 | 0.843 | 0.970 | 0.897 | 0.918 | 1.000 | 0.531 |
| Clients 9 | 0.850 | 0.974 | 0.897 | 0.918 | 1.000 | 0.531 |
| Clients 10 | 0.822 | 0.967 | 0.882 | 0.909 | 1.000 | 0.376 |
| Clients 11 | 0.819 | 0.968 | 0.868 | 0.914 | 1.000 | 0.414 |
| Clients 12 | 0.817 | 0.966 | 0.868 | 0.925 | 1.000 | 0.369 |
| Clients 13 | 0.829 | 0.967 | 0.882 | 0.912 | 1.000 | 0.438 |
| Clients 14 | 0.831 | 0.971 | 0.868 | 0.911 | 1.000 | 0.398 |
| Clients 15 | 0.810 | 0.964 | 0.868 | 0.900 | 1.000 | 0.361 |
| Clients 16 | 0.790 | 0.956 | 0.882 | 0.909 | 1.000 | 0.398 |
| Clients 17 | 0.812 | 0.966 | 0.882 | 0.914 | 1.000 | 0.414 |

| | | | | | | |
|---|---|---|---|---|---|---|
| Clients 18 | 0.774 | 0.953 | 0.853 | 0.900 | 1.000 | 0.298 |
| Clients 19 | 0.774 | 0.955 | 0.868 | 0.907 | 1.000 | 0.296 |
| Clients 20 | 0.776 | 0.958 | 0.868 | 0.898 | 1.000 | 0.335 |
| Epochs 1 | 0.847 | 0.973 | 0.882 | 0.915 | 1.000 | 0.450 |
| Epochs 2 | 0.850 | 0.975 | 0.868 | 0.915 | 1.000 | 0.450 |
| Epochs 5 | 0.845 | 0.972 | 0.882 | 0.914 | 1.000 | 0.438 |
| Epochs 10 | 0.881 | 0.978 | 0.882 | 0.917 | 1.000 | 0.489 |
| Epochs 20 | 0.852 | 0.974 | 0.868 | 0.912 | 1.000 | 0.381 |
| Balanced 60\|40 | 0.845 | 0.969 | 0.882 | 0.917 | 1.000 | 0.489 |
| Balanced 67\|33 | 0.847 | 0.972 | 0.868 | 0.915 | 1.000 | 0.450 |
| Balanced 75\|25 | 0.883 | 0.979 | 0.897 | 0.915 | 1.000 | 0.483 |
| Balanced 80\|20 | 0.891 | 0.982 | 0.882 | 0.917 | 1.000 | 0.489 |
| Balanced 40\|30\|30 | 0.814 | 0.966 | 0.868 | 0.915 | 1.000 | 0.450 |
| Balanced 60\|20\|20 | 0.819 | 0.965 | 0.882 | 0.914 | 1.000 | 0.414 |
| Balanced 80\|10\|10 | 0.807 | 0.966 | 0.853 | 0.909 | 1.000 | 0.322 |
| Balanced 60\|30\|10 | 0.852 | 0.975 | 0.897 | 0.911 | 1.000 | 0.483 |
| Balanced 30\|30\|15\|15\|10 | 0.819 | 0.966 | 0.868 | 0.915 | 1.000 | 0.450 |
| Balanced 60\|10\|10\|10\|10 | 0.824 | 0.970 | 0.882 | 0.914 | 1.000 | 0.438 |

Table 2.54: Best results for plain averaging on FeHuman with a 0.7 threshold (ADME)

| experiment | roc_auc | pr_auc | accuracy | precision | recall | kappa |
|---|---|---|---|---|---|---|
| Baseline | 0.921 | 0.985 | 0.897 | 0.932 | 0.966 | 0.572 |
| 60\|40 | 0.872 | 0.977 | 0.897 | 0.918 | 1.000 | 0.531 |
| 67\|33 | 0.841 | 0.971 | 0.897 | 0.918 | 1.000 | 0.531 |
| 75\|25 | 0.834 | 0.966 | 0.882 | 0.917 | 1.000 | 0.489 |
| 80\|20 | 0.891 | 0.981 | 0.882 | 0.917 | 1.000 | 0.489 |
| 40\|30\|30 | 0.866 | 0.975 | 0.868 | 0.915 | 1.000 | 0.450 |
| 60\|20\|20 | 0.878 | 0.977 | 0.912 | 0.919 | 1.000 | 0.579 |
| 80\|10\|10 | 0.874 | 0.979 | 0.882 | 0.914 | 1.000 | 0.438 |
| 60\|30\|10 | 0.879 | 0.978 | 0.912 | 0.919 | 1.000 | 0.579 |
| 30\|30\|15\|15\|10 | 0.862 | 0.975 | 0.912 | 0.919 | 1.000 | 0.579 |
| 60\|10\|10\|10\|10 | 0.855 | 0.974 | 0.882 | 0.903 | 1.000 | 0.438 |

Table 2.55: Best results for weighted averaging on FeHuman with a 0.7 threshold (ADME)

| experiment | roc_auc | pr_auc | accuracy | precision | recall | kappa |
|---|---|---|---|---|---|---|
| Baseline | 0.921 | 0.985 | 0.897 | 0.932 | 0.966 | 0.572 |
| Clients 2 | 0.862 | 0.976 | 0.882 | 0.914 | 1.000 | 0.438 |
| Clients 3 | 0.867 | 0.977 | 0.868 | 0.900 | 1.000 | 0.361 |
| Clients 4 | 0.855 | 0.974 | 0.897 | 0.918 | 1.000 | 0.531 |
| Clients 5 | 0.843 | 0.970 | 0.897 | 0.918 | 1.000 | 0.531 |
| Clients 6 | 0.841 | 0.973 | 0.882 | 0.903 | 1.000 | 0.438 |
| Clients 7 | 0.864 | 0.977 | 0.868 | 0.887 | 1.000 | 0.298 |
| Clients 8 | 0.847 | 0.970 | 0.897 | 0.918 | 1.000 | 0.531 |
| Clients 9 | 0.841 | 0.971 | 0.897 | 0.905 | 1.000 | 0.483 |
| Clients 10 | 0.852 | 0.973 | 0.868 | 0.902 | 1.000 | 0.398 |
| Clients 11 | 0.843 | 0.973 | 0.868 | 0.889 | 1.000 | 0.335 |
| Clients 12 | 0.834 | 0.970 | 0.868 | 0.911 | 1.000 | 0.350 |
| Clients 13 | 0.843 | 0.972 | 0.882 | 0.911 | 1.000 | 0.438 |
| Clients 14 | 0.855 | 0.975 | 0.897 | 0.914 | 1.000 | 0.483 |
| Clients 15 | 0.822 | 0.967 | 0.868 | 0.914 | 1.000 | 0.414 |
| Clients 16 | 0.828 | 0.970 | 0.897 | 0.925 | 1.000 | 0.483 |
| Clients 17 | 0.834 | 0.971 | 0.897 | 0.922 | 1.000 | 0.531 |
| Clients 18 | 0.826 | 0.969 | 0.897 | 0.918 | 1.000 | 0.531 |
| Clients 19 | 0.762 | 0.954 | 0.882 | 0.909 | 1.000 | 0.376 |
| Clients 20 | 0.840 | 0.973 | 0.882 | 0.917 | 1.000 | 0.489 |
| Epochs 1 | 0.845 | 0.972 | 0.882 | 0.917 | 1.000 | 0.489 |
| Epochs 2 | 0.891 | 0.982 | 0.868 | 0.915 | 1.000 | 0.450 |
| Epochs 5 | 0.876 | 0.978 | 0.882 | 0.891 | 1.000 | 0.376 |
| Epochs 10 | 0.836 | 0.970 | 0.868 | 0.915 | 1.000 | 0.450 |
| Epochs 20 | 0.864 | 0.976 | 0.912 | 0.919 | 1.000 | 0.579 |
| Balanced 60\|40 | 0.853 | 0.974 | 0.868 | 0.902 | 1.000 | 0.398 |
| Balanced 67\|33 | 0.852 | 0.974 | 0.868 | 0.915 | 1.000 | 0.450 |
| Balanced 75\|25 | 0.831 | 0.967 | 0.897 | 0.918 | 1.000 | 0.531 |
| Balanced 80\|20 | 0.862 | 0.975 | 0.882 | 0.915 | 1.000 | 0.450 |
| Balanced 40\|30\|30 | 0.805 | 0.959 | 0.882 | 0.914 | 1.000 | 0.438 |
| Balanced 60\|20\|20 | 0.817 | 0.964 | 0.868 | 0.915 | 1.000 | 0.450 |

| | | | | | | |
|---|---|---|---|---|---|---|
| Balanced 80\|10\|10 | 0.866 | 0.977 | 0.853 | 0.907 | 1.000 | 0.327 |
| Balanced 60\|30\|10 | 0.828 | 0.967 | 0.868 | 0.911 | 1.000 | 0.350 |
| Balanced 30\|30\|15\|15\|10 | 0.833 | 0.971 | 0.882 | 0.923 | 1.000 | 0.450 |
| Balanced 60\|10\|10\|10\|10 | 0.828 | 0.963 | 0.882 | 0.917 | 1.000 | 0.489 |

Table 2.56: Best results for benchmarked averaging on FeHuman with a 0.7 threshold (ADME)

### 2.4.17 ADME - FuBrain

| experiment | r2 | mae | rmse |
|---|---|---|---|
| Baseline | 0.591 | 1.309 | 2.545 |
| Clients 2 | 0.504 | 3.274 | 5.045 |
| Clients 3 | 0.483 | 3.859 | 5.525 |
| Clients 4 | 0.315 | 3.891 | 5.552 |
| Clients 5 | 0.322 | 3.926 | 5.582 |
| Clients 6 | 0.294 | 3.900 | 5.563 |
| Clients 7 | 0.462 | 3.837 | 5.509 |
| Clients 8 | 0.164 | 3.982 | 5.627 |
| Clients 9 | 0.335 | 3.966 | 5.619 |
| Clients 10 | 0.394 | 3.894 | 5.558 |
| Clients 11 | 0.226 | 3.948 | 5.602 |
| Clients 12 | 0.311 | 3.934 | 5.591 |
| Clients 13 | 0.381 | 3.950 | 5.603 |
| Clients 14 | 0.219 | 3.974 | 5.621 |
| Clients 15 | 0.275 | 3.932 | 5.589 |
| Clients 16 | 0.185 | 3.979 | 5.625 |
| Clients 17 | 0.145 | 3.961 | 5.613 |
| Clients 18 | 0.179 | 3.960 | 5.612 |
| Clients 19 | 0.173 | 3.966 | 5.616 |
| Clients 20 | 0.203 | 3.970 | 5.620 |
| Epochs 1 | 0.457 | 3.944 | 5.597 |
| Epochs 2 | 0.483 | 3.883 | 5.548 |
| Epochs 5 | 0.495 | 3.883 | 5.550 |

| | | | |
|---|---|---|---|
| Epochs 10 | 0.520 | 3.863 | 5.532 |
| Epochs 20 | 0.577 | 3.521 | 5.243 |
| Balanced 60\|40 | 0.492 | 3.702 | 5.395 |
| Balanced 67\|33 | 0.512 | 3.781 | 5.460 |
| Balanced 75\|25 | 0.514 | 3.719 | 5.412 |
| Balanced 80\|20 | 0.483 | 3.677 | 5.369 |
| Balanced 40\|30\|30 | 0.474 | 3.639 | 5.348 |
| Balanced 60\|20\|20 | 0.453 | 3.776 | 5.458 |
| Balanced 80\|10\|10 | 0.462 | 3.633 | 5.339 |
| Balanced 60\|30\|10 | 0.458 | 3.941 | 5.596 |
| Balanced 30\|30\|15\|15\|10 | 0.404 | 3.988 | 5.634 |
| Balanced 60\|10\|10\|10\|10 | 0.385 | 3.879 | 5.548 |

Table 2.57: Best results for plain averaging on FuBrain (ADME)

| experiment | r2 | mae | rmse |
|---|---|---|---|
| Baseline | 0.591 | 1.309 | 2.545 |
| 60\|40 | 0.551 | 3.304 | 5.071 |
| 67\|33 | 0.506 | 3.455 | 5.184 |
| 75\|25 | 0.560 | 3.481 | 5.215 |
| 80\|20 | 0.561 | 3.380 | 5.126 |
| 40\|30\|30 | 0.433 | 3.827 | 5.501 |
| 60\|20\|20 | 0.422 | 3.851 | 5.527 |
| 80\|10\|10 | 0.565 | 3.257 | 5.022 |
| 60\|30\|10 | 0.460 | 4.154 | 5.768 |
| 30\|30\|15\|15\|10 | 0.347 | 4.002 | 5.645 |
| 60\|10\|10\|10\|10 | 0.488 | 3.415 | 5.163 |

Table 2.58: Best results for weighted averaging on FuBrain (ADME)

| experiment | r2 | mae | rmse |
|---|---|---|---|
| Baseline | 0.591 | 1.309 | 2.545 |

| | | | |
|---|---|---|---|
| Clients 2 | 0.516 | 3.787 | 5.461 |
| Clients 3 | 0.492 | 3.794 | 5.476 |
| Clients 4 | 0.398 | 3.960 | 5.607 |
| Clients 5 | 0.386 | 3.907 | 5.567 |
| Clients 6 | 0.446 | 3.890 | 5.554 |
| Clients 7 | 0.351 | 4.006 | 5.645 |
| Clients 8 | 0.358 | 3.935 | 5.590 |
| Clients 9 | 0.362 | 3.957 | 5.610 |
| Clients 10 | 0.357 | 3.972 | 5.620 |
| Clients 11 | 0.240 | 3.961 | 5.612 |
| Clients 12 | 0.289 | 3.948 | 5.602 |
| Clients 13 | 0.271 | 3.977 | 5.624 |
| Clients 14 | 0.347 | 156.747 | 172.060 |
| Clients 15 | 0.362 | 3.989 | 5.634 |
| Clients 16 | 0.388 | 49.924 | 54.748 |
| Clients 17 | 0.328 | 5.844 | 6.592 |
| Clients 18 | 0.270 | 55.438 | 67.088 |
| Clients 19 | 0.280 | 52.190 | 55.873 |
| Clients 20 | 0.328 | 3.970 | 5.620 |
| Epochs 1 | 0.453 | 3.951 | 5.611 |
| Epochs 2 | 0.499 | 4.049 | 5.677 |
| Epochs 5 | 0.462 | 3.785 | 5.464 |
| Epochs 10 | - | - | - |
| Epochs 20 | -0.488 | 2804.875 | 4961.508 |
| Balanced 60\|40 | 0.536 | 3.456 | 5.199 |
| Balanced 67\|33 | 0.504 | 2.194 | 3.955 |
| Balanced 75\|25 | -0.016 | 73.116 | 90.772 |
| Balanced 80\|20 | 0.522 | 3.599 | 5.298 |
| Balanced 40\|30\|30 | 0.481 | 4.006 | 5.647 |
| Balanced 60\|20\|20 | 0.496 | 3.832 | 5.501 |
| Balanced 80\|10\|10 | -0.370 | 3.990 | 5.634 |
| Balanced 60\|30\|10 | 0.463 | 3.735 | 5.423 |
| Balanced 30\|30\|15\|15\|10 | 0.352 | 4.081 | 5.708 |
| Balanced 60\|10\|10\|10\|10 | 0.390 | 4.047 | 5.680 |

Table 2.59: Best results for benchmarked averaging on FuBrain (ADME)

### 2.4.18 ADME - FupHuman

| experiment | r2 | mae | rmse |
|---|---|---|---|
| Baseline | 0.690 | 0.090 | 0.156 |
| Clients 2 | 0.661 | 0.353 | 0.377 |
| Clients 3 | 0.646 | 0.278 | 0.306 |
| Clients 4 | 0.667 | 0.436 | 0.466 |
| Clients 5 | 0.588 | 0.236 | 0.282 |
| Clients 6 | 0.640 | 0.332 | 0.356 |
| Clients 7 | 0.641 | 0.364 | 0.388 |
| Clients 8 | 0.636 | 0.374 | 0.398 |
| Clients 9 | 0.650 | 0.380 | 0.405 |
| Clients 10 | 0.589 | 0.297 | 0.323 |
| Clients 11 | 0.623 | 0.370 | 0.395 |
| Clients 12 | 0.636 | 0.370 | 0.395 |
| Clients 13 | 0.614 | 0.362 | 0.386 |
| Clients 14 | 0.598 | 0.356 | 0.380 |
| Clients 15 | 0.630 | 0.378 | 0.403 |
| Clients 16 | 0.626 | 0.384 | 0.409 |
| Clients 17 | 0.604 | 0.365 | 0.389 |
| Clients 18 | 0.617 | 0.373 | 0.398 |
| Clients 19 | 0.601 | 0.376 | 0.400 |
| Clients 20 | 0.611 | 0.369 | 0.393 |
| Epochs 1 | 0.661 | 0.240 | 0.278 |
| Epochs 2 | 0.672 | 0.264 | 0.296 |
| Epochs 5 | 0.661 | 0.384 | 0.410 |
| Epochs 10 | 0.665 | 0.336 | 0.360 |
| Epochs 20 | 0.692 | 0.309 | 0.333 |
| Balanced 60\|40 | 0.647 | 0.256 | 0.287 |
| Balanced 67\|33 | 0.670 | 0.410 | 0.437 |
| Balanced 75\|25 | 0.660 | 0.346 | 0.370 |
| Balanced 80\|20 | 0.645 | 0.269 | 0.298 |
| Balanced 40\|30\|30 | 0.681 | 0.335 | 0.358 |

| | | | |
|---|---|---|---|
| Balanced 60\|20\|20 | 0.649 | 0.372 | 0.397 |
| Balanced 80\|10\|10 | 0.623 | 0.138 | 0.212 |
| Balanced 60\|30\|10 | 0.653 | 0.310 | 0.334 |
| Balanced 30\|30\|15\|15\|10 | 0.655 | 0.318 | 0.342 |
| Balanced 60\|10\|10\|10\|10 | 0.646 | 0.328 | 0.352 |

Table 2.60: Best results for plain averaging on FupHuman (ADME)

| experiment | r2 | mae | rmse |
|---|---|---|---|
| Baseline | 0.690 | 0.090 | 0.156 |
| 60\|40 | 0.685 | 0.253 | 0.286 |
| 67\|33 | 0.667 | 0.274 | 0.302 |
| 75\|25 | 0.668 | 0.250 | 0.281 |
| 80\|20 | 0.645 | 0.266 | 0.297 |
| 40\|30\|30 | 0.671 | 0.396 | 0.422 |
| 60\|20\|20 | 0.260 | 0.348 | 0.371 |
| 80\|10\|10 | 0.264 | 0.396 | 0.426 |
| 60\|30\|10 | 0.664 | 0.353 | 0.377 |
| 30\|30\|15\|15\|10 | 0.521 | 0.299 | 0.324 |
| 60\|10\|10\|10\|10 | 0.271 | 0.401 | 0.430 |

Table 2.61: Best results for weighted averaging on FupHuman (ADME)

| experiment | r2 | mae | rmse |
|---|---|---|---|
| Baseline | 0.690 | 0.090 | 0.156 |
| Clients 2 | 0.687 | 0.362 | 0.386 |
| Clients 3 | 0.684 | 0.248 | 0.283 |
| Clients 4 | 0.665 | 0.388 | 0.414 |
| Clients 5 | 0.658 | 0.402 | 0.429 |
| Clients 6 | 0.645 | 0.142 | 0.224 |
| Clients 7 | 0.673 | 0.146 | 0.216 |
| Clients 8 | 0.651 | 0.153 | 0.223 |

| | | | |
|---|---|---|---|
| Clients 9 | -0.571 | 0.233 | 0.381 |
| Clients 10 | 0.645 | 0.152 | 0.228 |
| Clients 11 | 0.611 | 0.197 | 0.326 |
| Clients 12 | 0.495 | 0.210 | 0.350 |
| Clients 13 | -0.093 | 0.210 | 0.350 |
| Clients 14 | 0.018 | 0.485 | 0.611 |
| Clients 15 | 0.654 | 0.154 | 0.231 |
| Clients 16 | 0.630 | 0.155 | 0.236 |
| Clients 17 | 0.624 | 0.384 | 0.410 |
| Clients 18 | 0.616 | 0.338 | 0.362 |
| Clients 19 | 0.600 | 0.317 | 0.341 |
| Clients 20 | 0.617 | 0.337 | 0.360 |
| Epochs 1 | 0.115 | 0.210 | 0.349 |
| Epochs 2 | 0.707 | 0.151 | 0.244 |
| Epochs 5 | 0.667 | 0.234 | 0.270 |
| Epochs 10 | 0.680 | 0.303 | 0.327 |
| Epochs 20 | 0.642 | 0.300 | 0.326 |
| Balanced 60\|40 | 0.663 | 0.281 | 0.307 |
| Balanced 67\|33 | 0.662 | 0.408 | 0.434 |
| Balanced 75\|25 | 0.672 | 0.397 | 0.424 |
| Balanced 80\|20 | 0.651 | 0.284 | 0.310 |
| Balanced 40\|30\|30 | 0.679 | 0.136 | 0.206 |
| Balanced 60\|20\|20 | 0.651 | 0.137 | 0.204 |
| Balanced 80\|10\|10 | -0.262 | 0.207 | 0.345 |
| Balanced 60\|30\|10 | 0.632 | 0.134 | 0.206 |
| Balanced 30\|30\|15\|15\|10 | 0.639 | 0.284 | 0.311 |
| Balanced 60\|10\|10\|10\|10 | 0.664 | 0.375 | 0.400 |

Table 2.62: Best results for benchmarked averaging on FupHuman (ADME)

### 2.4.19 ADME - FupRat

| experiment | r2 | mae | rmse |
|---|---|---|---|

| | | | |
|---|---|---|---|
| Baseline | 0.594 | 0.111 | 0.183 |
| Clients 2 | 0.577 | 0.211 | 0.310 |
| Clients 3 | 0.550 | 0.356 | 0.388 |
| Clients 4 | 0.584 | 0.317 | 0.343 |
| Clients 5 | 0.486 | 0.242 | 0.285 |
| Clients 6 | 0.511 | 0.387 | 0.425 |
| Clients 7 | 0.521 | 0.272 | 0.299 |
| Clients 8 | 0.542 | 0.333 | 0.361 |
| Clients 9 | 0.398 | 0.311 | 0.336 |
| Clients 10 | 0.529 | 0.278 | 0.321 |
| Clients 11 | 0.572 | 0.347 | 0.375 |
| Clients 12 | 0.561 | 0.358 | 0.388 |
| Clients 13 | 0.474 | 0.362 | 0.394 |
| Clients 14 | 0.482 | 0.319 | 0.345 |
| Clients 15 | 0.494 | 0.381 | 0.416 |
| Clients 16 | 0.483 | 0.319 | 0.345 |
| Clients 17 | 0.408 | 0.314 | 0.339 |
| Clients 18 | 0.441 | 0.345 | 0.373 |
| Clients 19 | 0.509 | 0.363 | 0.395 |
| Clients 20 | 0.384 | 0.336 | 0.363 |
| Epochs 1 | 0.608 | 0.267 | 0.315 |
| Epochs 2 | 0.557 | 0.241 | 0.284 |
| Epochs 5 | 0.614 | 0.434 | 0.481 |
| Epochs 10 | 0.520 | 0.228 | 0.268 |
| Epochs 20 | 0.584 | 0.234 | 0.267 |
| Balanced 60\|40 | 0.609 | 0.209 | 0.313 |
| Balanced 67\|33 | 0.521 | 0.199 | 0.281 |
| Balanced 75\|25 | 0.574 | 0.217 | 0.277 |
| Balanced 80\|20 | 0.553 | 0.217 | 0.267 |
| Balanced 40\|30\|30 | 0.536 | 0.199 | 0.275 |
| Balanced 60\|20\|20 | 0.463 | 0.255 | 0.304 |
| Balanced 80\|10\|10 | 0.590 | 0.217 | 0.272 |
| Balanced 60\|30\|10 | 0.591 | 0.367 | 0.402 |
| Balanced 30\|30\|15\|15\|10 | 0.295 | 0.245 | 0.337 |

| | | | |
|---|---|---|---|
| Balanced 60\|10\|10\|10\|10 | 0.500 | 0.401 | 0.442 |

Table 2.63: Best results for plain averaging on FupRat (ADME)

| experiment | r2 | mae | rmse |
|---|---|---|---|
| Baseline | 0.594 | 0.111 | 0.183 |
| 60\|40 | 0.552 | 0.208 | 0.253 |
| 67\|33 | 0.653 | 0.208 | 0.279 |
| 75\|25 | 0.528 | 0.277 | 0.303 |
| 80\|20 | 0.525 | 0.200 | 0.283 |
| 40\|30\|30 | 0.563 | 0.266 | 0.294 |
| 60\|20\|20 | 0.607 | 0.488 | 0.542 |
| 80\|10\|10 | 0.513 | 0.232 | 0.270 |
| 60\|30\|10 | 0.617 | 0.245 | 0.275 |
| 30\|30\|15\|15\|10 | 0.553 | 0.236 | 0.280 |
| 60\|10\|10\|10\|10 | 0.607 | 0.247 | 0.272 |

Table 2.64: Best results for weighted averaging on FupRat (ADME)

| experiment | r2 | mae | rmse |
|---|---|---|---|
| Baseline | 0.594 | 0.111 | 0.183 |
| Clients 2 | 0.632 | 0.221 | 0.264 |
| Clients 3 | 0.619 | 0.231 | 0.366 |
| Clients 4 | 0.621 | 0.257 | 0.285 |
| Clients 5 | - | - | - |
| Clients 6 | 0.170 | 0.208 | 0.276 |
| Clients 7 | 0.640 | 0.352 | 0.418 |
| Clients 8 | 0.433 | 0.450 | 0.497 |
| Clients 9 | 0.163 | 0.460 | 0.511 |
| Clients 10 | 0.561 | 0.641 | 0.723 |
| Clients 11 | 0.267 | 0.299 | 0.323 |
| Clients 12 | -0.012 | 0.509 | 0.568 |

| | | | |
|---|---|---|---|
| Clients 13 | 0.158 | 0.452 | 0.500 |
| Clients 14 | 0.228 | 0.339 | 0.368 |
| Clients 15 | 0.192 | 0.610 | 0.677 |
| Clients 16 | 0.600 | 0.729 | 0.787 |
| Clients 17 | -0.009 | 0.617 | 0.683 |
| Clients 18 | 0.220 | 0.614 | 0.701 |
| Clients 19 | 0.414 | 0.562 | 0.645 |
| Clients 20 | -0.190 | 0.735 | 0.794 |
| Epochs 1 | 0.639 | 0.482 | 0.536 |
| Epochs 2 | 0.652 | 0.448 | 0.533 |
| Epochs 5 | 0.541 | 0.554 | 0.640 |
| Epochs 10 | 0.568 | 0.217 | 0.347 |
| Epochs 20 | 0.652 | 0.204 | 0.274 |
| Balanced 60\|40 | 0.575 | 0.165 | 0.252 |
| Balanced 67\|33 | 0.578 | 0.208 | 0.259 |
| Balanced 75\|25 | 0.558 | 0.432 | 0.477 |
| Balanced 80\|20 | 0.529 | 0.174 | 0.247 |
| Balanced 40\|30\|30 | 0.500 | 0.238 | 0.373 |
| Balanced 60\|20\|20 | -0.162 | 0.481 | 0.601 |
| Balanced 80\|10\|10 | -0.361 | 0.331 | 0.468 |
| Balanced 60\|30\|10 | 0.655 | 0.163 | 0.243 |
| Balanced 30\|30\|15\|15\|10 | 0.611 | 0.234 | 0.367 |
| Balanced 60\|10\|10\|10\|10 | 0.578 | 0.435 | 0.537 |

Table 2.65: Best results for benchmarked averaging on FupRat (ADME)

### 2.4.20 ADME - NER-LLC

| experiment | roc_auc | pr_auc | accuracy | precision | recall | kappa |
|---|---|---|---|---|---|---|
| Baseline | 0.821 | 0.653 | 0.764 | 0.813 | 0.950 | 0.444 |
| Clients 2 | 0.786 | 0.617 | 0.655 | 0.731 | 0.775 | 0.347 |
| Clients 3 | 0.794 | 0.594 | 0.644 | 0.740 | 0.810 | 0.271 |
| Clients 4 | 0.762 | 0.561 | 0.659 | 0.725 | 1.000 | 0.282 |
| Clients 5 | 0.793 | 0.603 | 0.652 | 0.738 | 1.000 | 0.337 |

| | | | | | | |
|---|---|---|---|---|---|---|
| Clients 6 | 0.784 | 0.595 | 0.652 | 0.735 | 1.000 | 0.312 |
| Clients 7 | 0.780 | 0.568 | 0.663 | 0.746 | 1.000 | 0.295 |
| Clients 8 | 0.786 | 0.604 | 0.652 | 0.751 | 1.000 | 0.294 |
| Clients 9 | 0.790 | 0.591 | 0.670 | 0.717 | 0.952 | 0.365 |
| Clients 10 | 0.791 | 0.579 | 0.652 | 0.733 | 1.000 | 0.306 |
| Clients 11 | 0.784 | 0.586 | 0.644 | 0.734 | 1.000 | 0.272 |
| Clients 12 | 0.760 | 0.559 | 0.644 | 0.733 | 0.857 | 0.281 |
| Clients 13 | 0.776 | 0.575 | 0.659 | 0.730 | 1.000 | 0.264 |
| Clients 14 | 0.785 | 0.575 | 0.655 | 0.723 | 1.000 | 0.273 |
| Clients 15 | 0.776 | 0.564 | 0.678 | 0.738 | 1.000 | 0.345 |
| Clients 16 | 0.777 | 0.570 | 0.652 | 0.775 | 1.000 | 0.300 |
| Clients 17 | 0.777 | 0.563 | 0.652 | 0.736 | 1.000 | 0.294 |
| Clients 18 | 0.764 | 0.557 | 0.659 | 0.728 | 1.000 | 0.323 |
| Clients 19 | 0.775 | 0.578 | 0.663 | 0.743 | 1.000 | 0.317 |
| Clients 20 | 0.782 | 0.571 | 0.655 | 0.715 | 1.000 | 0.341 |
| Epochs 1 | 0.808 | 0.635 | 0.663 | 0.762 | 0.820 | 0.347 |
| Epochs 2 | 0.802 | 0.600 | 0.655 | 0.738 | 1.000 | 0.338 |
| Epochs 5 | 0.786 | 0.608 | 0.670 | 0.748 | 0.773 | 0.320 |
| Epochs 10 | 0.801 | 0.610 | 0.648 | 0.677 | 0.789 | 0.316 |
| Epochs 20 | 0.795 | 0.586 | 0.652 | 0.564 | 1.000 | 0.310 |
| Balanced 60\|40 | 0.794 | 0.616 | 0.640 | 0.732 | 0.782 | 0.261 |
| Balanced 67\|33 | 0.793 | 0.596 | 0.640 | 0.719 | 0.917 | 0.312 |
| Balanced 75\|25 | 0.811 | 0.637 | 0.670 | 0.751 | 0.905 | 0.306 |
| Balanced 80\|20 | 0.809 | 0.610 | 0.674 | 0.729 | 0.806 | 0.306 |
| Balanced 40\|30\|30 | 0.782 | 0.578 | 0.678 | 0.738 | 0.751 | 0.273 |
| Balanced 60\|20\|20 | 0.815 | 0.653 | 0.667 | 0.734 | 0.976 | 0.279 |
| Balanced 80\|10\|10 | 0.785 | 0.575 | 0.685 | 0.740 | 0.915 | 0.343 |
| Balanced 60\|30\|10 | 0.795 | 0.641 | 0.644 | 0.732 | 1.000 | 0.238 |
| Balanced 30\|30\|15\|15\|10 | 0.798 | 0.621 | 0.655 | 0.722 | 0.787 | 0.336 |
| Balanced 60\|10\|10\|10\|10 | 0.790 | 0.589 | 0.655 | 0.731 | 0.965 | 0.315 |

Table 2.66: Best results for plain averaging on NER-LLC (average) (ADME)

| experiment | roc_auc | pr_auc | accuracy | precision | recall | kappa |
|---|---|---|---|---|---|---|

| | | | | | | |
|---|---|---|---|---|---|---|
| Baseline | 0.821 | 0.653 | 0.764 | 0.813 | 0.950 | 0.444 |
| 60\|40 | 0.804 | 0.597 | 0.633 | 0.730 | 0.765 | 0.272 |
| 67\|33 | 0.796 | 0.602 | 0.652 | 0.699 | 0.808 | 0.296 |
| 75\|25 | 0.806 | 0.616 | 0.719 | 0.729 | 0.803 | 0.338 |
| 80\|20 | 0.807 | 0.621 | 0.648 | 0.724 | 0.986 | 0.331 |
| 40\|30\|30 | 0.808 | 0.625 | 0.648 | 0.738 | 0.804 | 0.309 |
| 60\|20\|20 | 0.811 | 0.628 | 0.655 | 0.772 | 0.787 | 0.319 |
| 80\|10\|10 | 0.796 | 0.596 | 0.663 | 0.741 | 0.818 | 0.343 |
| 60\|30\|10 | 0.805 | 0.623 | 0.670 | 0.720 | 0.869 | 0.333 |
| 30\|30\|15\|15\|10 | 0.796 | 0.597 | 0.640 | 0.731 | 0.908 | 0.279 |
| 60\|10\|10\|10\|10 | 0.802 | 0.683 | 0.663 | 0.746 | 0.940 | 0.309 |

Table 2.67: Best results for weighted averaging on NER-LLC (average) (ADME)

| experiment | roc_auc | pr_auc | accuracy | precision | recall | kappa |
|---|---|---|---|---|---|---|
| Baseline | 0.821 | 0.653 | 0.764 | 0.813 | 0.950 | 0.444 |
| Clients 2 | 0.801 | 0.599 | 0.644 | 0.738 | 0.825 | 0.283 |
| Clients 3 | 0.796 | 0.602 | 0.648 | 0.734 | 0.787 | 0.286 |
| Clients 4 | 0.790 | 0.624 | 0.652 | 0.736 | 0.857 | 0.296 |
| Clients 5 | 0.803 | 0.601 | 0.652 | 0.739 | 1.000 | 0.304 |
| Clients 6 | 0.793 | 0.593 | 0.678 | 0.722 | 1.000 | 0.345 |
| Clients 7 | 0.795 | 0.600 | 0.670 | 0.734 | 1.000 | 0.325 |
| Clients 8 | 0.790 | 0.601 | 0.682 | 0.739 | 1.000 | 0.373 |
| Clients 9 | 0.791 | 0.588 | 0.670 | 0.739 | 1.000 | 0.362 |
| Clients 10 | 0.799 | 0.584 | 0.655 | 0.735 | 1.000 | 0.333 |
| Clients 11 | 0.803 | 0.604 | 0.655 | 0.729 | 1.000 | 0.330 |
| Clients 12 | 0.783 | 0.570 | 0.655 | 0.743 | 1.000 | 0.306 |
| Clients 13 | 0.794 | 0.608 | 0.674 | 0.730 | 1.000 | 0.342 |
| Clients 14 | 0.776 | 0.575 | 0.648 | 0.726 | 1.000 | 0.309 |
| Clients 15 | 0.759 | 0.533 | 0.659 | 0.740 | 1.000 | 0.288 |
| Clients 16 | 0.755 | 0.553 | 0.652 | 0.716 | 1.000 | 0.264 |
| Clients 17 | 0.787 | 0.572 | 0.685 | 0.741 | 1.000 | 0.361 |
| Clients 18 | 0.789 | 0.609 | 0.670 | 0.735 | 1.000 | 0.321 |

| | | | | | | |
|---|---|---|---|---|---|---|
| Clients 19 | 0.772 | 0.573 | 0.655 | 0.742 | 1.000 | 0.303 |
| Clients 20 | 0.787 | 0.583 | 0.648 | 0.741 | 1.000 | 0.319 |
| Epochs 1 | 0.797 | 0.605 | 0.655 | 0.740 | 0.763 | 0.324 |
| Epochs 2 | 0.785 | 0.604 | 0.667 | 0.740 | 0.833 | 0.312 |
| Epochs 5 | 0.797 | 0.606 | 0.659 | 0.732 | 0.808 | 0.331 |
| Epochs 10 | 0.784 | 0.614 | 0.652 | 0.772 | 0.773 | 0.319 |
| Epochs 20 | 0.785 | 0.604 | 0.655 | 0.610 | 0.929 | 0.303 |
| Balanced 60\|40 | 0.810 | 0.644 | 0.618 | 0.701 | 0.768 | 0.281 |
| Balanced 67\|33 | 0.802 | 0.645 | 0.652 | 0.741 | 0.758 | 0.304 |
| Balanced 75\|25 | 0.799 | 0.626 | 0.648 | 0.696 | 0.785 | 0.285 |
| Balanced 80\|20 | 0.803 | 0.619 | 0.682 | 0.734 | 0.765 | 0.316 |
| Balanced 40\|30\|30 | 0.805 | 0.625 | 0.674 | 0.726 | 0.801 | 0.385 |
| Balanced 60\|20\|20 | 0.800 | 0.608 | 0.655 | 0.756 | 0.782 | 0.305 |
| Balanced 80\|10\|10 | 0.784 | 0.593 | 0.655 | 0.746 | 1.000 | 0.271 |
| Balanced 60\|30\|10 | 0.795 | 0.599 | 0.644 | 0.725 | 0.803 | 0.303 |
| Balanced 30\|30\|15\|15\|10 | 0.803 | 0.594 | 0.648 | 0.734 | 1.000 | 0.279 |
| Balanced 60\|10\|10\|10\|10 | 0.786 | 0.604 | 0.659 | 0.740 | 0.773 | 0.328 |

Table 2.68: Best results for benchmarked averaging on NER-LLC (average) (ADME)

### 2.4.21 ADME - PappCaco2 (Classification)

| experiment | roc_auc | pr_auc | accuracy | precision | recall | kappa |
|---|---|---|---|---|---|---|
| Baseline | 0.860 | 0.781 | 0.799 | 0.739 | 0.991 | 0.581 |
| Clients 2 | 0.867 | 0.793 | 0.794 | 0.717 | 1.000 | 0.575 |
| Clients 3 | 0.855 | 0.775 | 0.757 | 0.650 | 1.000 | 0.513 |
| Clients 4 | 0.848 | 0.768 | 0.728 | 0.606 | 1.000 | 0.471 |
| Clients 5 | 0.853 | 0.777 | 0.769 | 0.674 | 1.000 | 0.532 |
| Clients 6 | 0.838 | 0.741 | 0.746 | 0.633 | 1.000 | 0.498 |
| Clients 7 | 0.840 | 0.751 | 0.705 | 0.583 | 1.000 | 0.433 |
| Clients 8 | 0.832 | 0.748 | 0.695 | 0.573 | 1.000 | 0.414 |
| Clients 9 | 0.834 | 0.740 | 0.716 | 0.596 | 1.000 | 0.447 |
| Clients 10 | 0.823 | 0.734 | 0.652 | 0.532 | 1.000 | 0.355 |

| | | | | | | |
|---|---|---|---|---|---|---|
| Clients 11 | 0.826 | 0.741 | 0.614 | 0.504 | 1.000 | 0.301 |
| Clients 12 | 0.811 | 0.721 | 0.536 | 0.457 | 1.000 | 0.191 |
| Clients 13 | 0.817 | 0.724 | 0.686 | 0.563 | 1.000 | 0.401 |
| Clients 14 | 0.813 | 0.710 | 0.601 | 0.495 | 1.000 | 0.283 |
| Clients 15 | 0.805 | 0.708 | 0.586 | 0.486 | 1.000 | 0.261 |
| Clients 16 | 0.795 | 0.694 | 0.532 | 0.455 | 1.000 | 0.186 |
| Clients 17 | 0.807 | 0.698 | 0.593 | 0.490 | 1.000 | 0.272 |
| Clients 18 | 0.795 | 0.690 | 0.558 | 0.470 | 1.000 | 0.223 |
| Clients 19 | 0.794 | 0.694 | 0.536 | 0.457 | 1.000 | 0.192 |
| Clients 20 | 0.795 | 0.691 | 0.561 | 0.471 | 1.000 | 0.222 |
| Epochs 1 | 0.853 | 0.766 | 0.756 | 0.643 | 1.000 | 0.515 |
| Epochs 2 | 0.851 | 0.769 | 0.752 | 0.640 | 0.991 | 0.507 |
| Epochs 5 | 0.858 | 0.776 | 0.778 | 0.679 | 0.991 | 0.552 |
| Epochs 10 | 0.851 | 0.756 | 0.782 | 0.717 | 1.000 | 0.549 |
| Epochs 20 | 0.854 | 0.777 | 0.789 | 0.758 | 0.997 | 0.551 |
| Balanced 60\|40 | 0.861 | 0.780 | 0.794 | 0.704 | 1.000 | 0.583 |
| Balanced 67\|33 | 0.859 | 0.786 | 0.794 | 0.721 | 0.988 | 0.573 |
| Balanced 75\|25 | 0.863 | 0.787 | 0.788 | 0.696 | 0.997 | 0.569 |
| Balanced 80\|20 | 0.860 | 0.786 | 0.782 | 0.697 | 1.000 | 0.554 |
| Balanced 40\|30\|30 | 0.854 | 0.784 | 0.763 | 0.661 | 1.000 | 0.528 |
| Balanced 60\|20\|20 | 0.850 | 0.764 | 0.765 | 0.660 | 1.000 | 0.532 |
| Balanced 80\|10\|10 | 0.866 | 0.789 | 0.794 | 0.708 | 1.000 | 0.578 |
| Balanced 60\|30\|10 | 0.855 | 0.772 | 0.780 | 0.671 | 1.000 | 0.561 |
| Balanced 30\|30\|15\|15\|10 | 0.847 | 0.769 | 0.728 | 0.606 | 1.000 | 0.472 |
| Balanced 60\|10\|10\|10\|10 | 0.851 | 0.764 | 0.717 | 0.594 | 1.000 | 0.460 |

Table 2.69: Best results for plain averaging on PappCaco2 (ADME) (classification)

| experiment | roc_auc | pr_auc | accuracy | precision | recall | kappa |
|---|---|---|---|---|---|---|
| Baseline | 0.860 | 0.781 | 0.799 | 0.739 | 0.991 | 0.581 |
| 60\|40 | 0.874 | 0.795 | 0.789 | 0.699 | 1.000 | 0.571 |
| 67\|33 | 0.863 | 0.790 | 0.783 | 0.719 | 0.958 | 0.556 |
| 75\|25 | 0.851 | 0.773 | 0.779 | 0.702 | 0.985 | 0.542 |

| | | | | | | |
|---|---|---|---|---|---|---|
| 80\|20 | 0.863 | 0.784 | 0.790 | 0.722 | 1.000 | 0.564 |
| 40\|30\|30 | 0.860 | 0.777 | 0.768 | 0.656 | 1.000 | 0.540 |
| 60\|20\|20 | 0.854 | 0.786 | 0.781 | 0.711 | 1.000 | 0.548 |
| 80\|10\|10 | 0.854 | 0.783 | 0.783 | 0.711 | 1.000 | 0.551 |
| 60\|30\|10 | 0.853 | 0.775 | 0.771 | 0.689 | 1.000 | 0.535 |
| 30\|30\|15\|15\|10 | 0.841 | 0.762 | 0.728 | 0.607 | 1.000 | 0.470 |
| 60\|10\|10\|10\|10 | 0.847 | 0.778 | 0.758 | 0.663 | 1.000 | 0.510 |

Table 2.70: Best results for weighted averaging on PappCaco2 (ADME) (classification)

| experiment | roc_auc | pr_auc | accuracy | precision | recall | kappa |
|---|---|---|---|---|---|---|
| Baseline | 0.860 | 0.781 | 0.799 | 0.739 | 0.991 | 0.581 |
| Clients 2 | 0.862 | 0.785 | 0.788 | 0.702 | 1.000 | 0.568 |
| Clients 3 | 0.855 | 0.781 | 0.786 | 0.730 | 1.000 | 0.550 |
| Clients 4 | 0.847 | 0.775 | 0.763 | 0.649 | 1.000 | 0.529 |
| Clients 5 | 0.834 | 0.750 | 0.750 | 0.652 | 1.000 | 0.498 |
| Clients 6 | 0.833 | 0.754 | 0.715 | 0.596 | 1.000 | 0.444 |
| Clients 7 | 0.781 | 0.664 | 0.705 | 0.591 | 1.000 | 0.421 |
| Clients 8 | 0.836 | 0.763 | 0.701 | 0.579 | 1.000 | 0.424 |
| Clients 9 | 0.833 | 0.750 | 0.671 | 0.547 | 1.000 | 0.384 |
| Clients 10 | 0.826 | 0.738 | 0.659 | 0.538 | 1.000 | 0.367 |
| Clients 11 | 0.815 | 0.717 | 0.619 | 0.508 | 1.000 | 0.306 |
| Clients 12 | 0.826 | 0.743 | 0.681 | 0.558 | 1.000 | 0.395 |
| Clients 13 | 0.814 | 0.713 | 0.629 | 0.514 | 1.000 | 0.324 |
| Clients 14 | 0.820 | 0.735 | 0.686 | 0.563 | 1.000 | 0.403 |
| Clients 15 | 0.809 | 0.711 | 0.587 | 0.487 | 1.000 | 0.266 |
| Clients 16 | 0.813 | 0.722 | 0.631 | 0.516 | 1.000 | 0.323 |
| Clients 17 | 0.786 | 0.677 | 0.523 | 0.450 | 1.000 | 0.173 |
| Clients 18 | 0.794 | 0.692 | 0.552 | 0.466 | 1.000 | 0.212 |
| Clients 19 | 0.798 | 0.694 | 0.573 | 0.478 | 1.000 | 0.243 |
| Clients 20 | 0.796 | 0.692 | 0.569 | 0.476 | 1.000 | 0.237 |
| Epochs 1 | 0.861 | 0.782 | 0.769 | 0.665 | 1.000 | 0.543 |
| Epochs 2 | 0.859 | 0.787 | 0.767 | 0.667 | 0.988 | 0.531 |

| | | | | | | |
|---|---|---|---|---|---|---|
| Epochs 5 | 0.867 | 0.795 | 0.800 | 0.740 | 1.000 | 0.587 |
| Epochs 10 | 0.871 | 0.799 | 0.793 | 0.712 | 1.000 | 0.577 |
| Epochs 20 | 0.850 | 0.783 | 0.794 | 0.743 | 1.000 | 0.569 |
| Balanced 60\|40 | 0.851 | 0.770 | 0.779 | 0.708 | 0.991 | 0.547 |
| Balanced 67\|33 | 0.861 | 0.788 | 0.800 | 0.747 | 1.000 | 0.581 |
| Balanced 75\|25 | 0.853 | 0.781 | 0.780 | 0.707 | 1.000 | 0.544 |
| Balanced 80\|20 | 0.868 | 0.785 | 0.797 | 0.725 | 1.000 | 0.581 |
| Balanced 40\|30\|30 | 0.858 | 0.783 | 0.775 | 0.699 | 0.997 | 0.546 |
| Balanced 60\|20\|20 | 0.857 | 0.777 | 0.760 | 0.654 | 1.000 | 0.523 |
| Balanced 80\|10\|10 | 0.865 | 0.788 | 0.788 | 0.691 | 1.000 | 0.571 |
| Balanced 60\|30\|10 | 0.855 | 0.780 | 0.785 | 0.723 | 1.000 | 0.552 |
| Balanced 30\|30\|15\|15\|10 | 0.845 | 0.764 | 0.771 | 0.680 | 1.000 | 0.532 |
| Balanced 60\|10\|10\|10\|10 | 0.858 | 0.779 | 0.785 | 0.696 | 1.000 | 0.560 |

Table 2.71: Best results for benchmarked averaging on PappCaco2 (ADME) (classification)

### 2.4.22 ADME - PappCaco2 (Regression)

| experiment | r2 | mae | rmse |
|---|---|---|---|
| Baseline | 0.451 | 1.090 | 1.411 |
| Clients 2 | 0.367 | 2.511 | 2.914 |
| Clients 3 | 0.389 | 2.948 | 3.352 |
| Clients 4 | 0.337 | 3.693 | 4.064 |
| Clients 5 | 0.328 | 3.676 | 4.048 |
| Clients 6 | 0.331 | 3.366 | 3.751 |
| Clients 7 | 0.231 | 3.591 | 3.965 |
| Clients 8 | 0.062 | 3.826 | 4.193 |
| Clients 9 | -0.187 | 3.800 | 4.168 |
| Clients 10 | 0.101 | 3.697 | 4.068 |
| Clients 11 | -0.134 | 3.708 | 4.079 |
| Clients 12 | -0.141 | 3.749 | 4.119 |
| Clients 13 | 0.016 | 3.822 | 4.188 |
| Clients 14 | -0.067 | 3.637 | 4.010 |

| | | | |
|---|---|---|---|
| Clients 15 | -0.251 | 3.768 | 4.137 |
| Clients 16 | -0.213 | 3.747 | 4.116 |
| Clients 17 | -0.165 | 3.751 | 4.121 |
| Clients 18 | -0.161 | 3.730 | 4.100 |
| Clients 19 | -0.277 | 3.784 | 4.153 |
| Clients 20 | -0.111 | 3.738 | 4.108 |
| Epochs 1 | 0.321 | 3.558 | 3.933 |
| Epochs 2 | 0.400 | 2.820 | 3.227 |
| Epochs 5 | 0.447 | 2.900 | 3.308 |
| Epochs 10 | 0.446 | 3.138 | 3.529 |
| Epochs 20 | 0.448 | 2.183 | 2.561 |
| Balanced 60\|40 | 0.396 | 2.690 | 3.096 |
| Balanced 67\|33 | 0.435 | 2.441 | 2.838 |
| Balanced 75\|25 | 0.400 | 2.531 | 2.932 |
| Balanced 80\|20 | 0.373 | 3.332 | 3.714 |
| Balanced 40\|30\|30 | 0.341 | 2.998 | 3.400 |
| Balanced 60\|20\|20 | 0.381 | 3.187 | 3.576 |
| Balanced 80\|10\|10 | 0.413 | 2.977 | 3.377 |
| Balanced 60\|30\|10 | 0.417 | 3.556 | 3.931 |
| Balanced 30\|30\|15\|15\|10 | 0.318 | 3.691 | 4.063 |
| Balanced 60\|10\|10\|10\|10 | 0.268 | 3.987 | 4.344 |

Table 2.72: Best results for plain averaging on PappCaco2 (ADME) (regression)

| experiment | r2 | mae | rmse |
|---|---|---|---|
| Baseline | 0.451 | 1.090 | 1.411 |
| 60\|40 | 0.448 | 2.429 | 2.825 |
| 67\|33 | 0.350 | 2.710 | 3.115 |
| 75\|25 | 0.396 | 2.214 | 2.589 |
| 80\|20 | 0.409 | 2.918 | 3.303 |
| 40\|30\|30 | 0.412 | 3.145 | 3.538 |
| 60\|20\|20 | 0.473 | 2.180 | 2.567 |
| 80\|10\|10 | 0.390 | 2.269 | 2.653 |
| 60\|30\|10 | 0.343 | 4.296 | 4.635 |

| | | | |
|---|---|---|---|
| 30\|30\|15\|15\|10 | 0.256 | 3.501 | 3.880 |
| 60\|10\|10\|10\|10 | 0.412 | 2.663 | 3.067 |

Table 2.73: Best results for weighted averaging on PappCaco2 (ADME) (regression)

| experiment | r2 | mae | rmse |
|---|---|---|---|
| Baseline | 0.451 | 1.090 | 1.411 |
| Clients 2 | 0.366 | 1.680 | 2.019 |
| Clients 3 | 0.349 | 1.592 | 1.954 |
| Clients 4 | 0.350 | 3.838 | 4.202 |
| Clients 5 | 0.393 | 3.819 | 4.186 |
| Clients 6 | 0.302 | 3.687 | 4.059 |
| Clients 7 | 0.270 | 3.030 | 3.432 |
| Clients 8 | 0.285 | 3.499 | 3.877 |
| Clients 9 | 0.264 | 3.627 | 4.001 |
| Clients 10 | 0.256 | 3.760 | 4.130 |
| Clients 11 | -0.216 | 3.731 | 4.101 |
| Clients 12 | -0.337 | 3.676 | 4.048 |
| Clients 13 | -0.206 | 3.899 | 4.261 |
| Clients 14 | -0.227 | 3.759 | 4.128 |
| Clients 15 | -0.209 | 3.725 | 4.095 |
| Clients 16 | -0.273 | 3.810 | 4.177 |
| Clients 17 | -0.277 | 3.764 | 4.133 |
| Clients 18 | -0.235 | 3.801 | 4.170 |
| Clients 19 | -0.267 | 3.870 | 6.011 |
| Clients 20 | -0.288 | 3.821 | 4.187 |
| Epochs 1 | 0.401 | 2.302 | 2.700 |
| Epochs 2 | 0.398 | 3.018 | 3.421 |
| Epochs 5 | 0.381 | 2.485 | 2.887 |
| Epochs 10 | 0.391 | 1.461 | 1.830 |
| Epochs 20 | 0.495 | 2.648 | 3.035 |
| Balanced 60\|40 | 0.346 | 1.403 | 1.717 |

| | | | |
|---|---|---|---|
| Balanced 67\|33 | 0.406 | 1.408 | 1.756 |
| Balanced 75\|25 | -666.924 | 865.534 | 1131.583 |
| Balanced 80\|20 | 0.387 | 2.907 | 3.741 |
| Balanced 40\|30\|30 | 0.389 | 3.151 | 3.544 |
| Balanced 60\|20\|20 | 0.247 | 1.975 | 2.415 |
| Balanced 80\|10\|10 | -0.242 | 2.236 | 2.651 |
| Balanced 60\|30\|10 | 0.406 | 2.310 | 2.713 |
| Balanced 30\|30\|15\|15\|10 | 0.370 | 3.801 | 4.168 |
| Balanced 60\|10\|10\|10\|10 | -0.125 | 6.013 | 7.276 |

Table 2.74: Best results for benchmarked averaging on PappCaco2 (ADME) (regression)

### 2.4.23 ADME - Papp-LLC

| experiment | r2 | mae | rmse |
|---|---|---|---|
| Baseline | 0.577 | 0.071 | 0.124 |
| Clients 2 | 0.559 | 0.292 | 0.310 |
| Clients 3 | 0.578 | 0.203 | 0.223 |
| Clients 4 | 0.577 | 0.279 | 0.297 |
| Clients 5 | 0.492 | 0.186 | 0.207 |
| Clients 6 | 0.591 | 0.275 | 0.293 |
| Clients 7 | 0.632 | 0.365 | 0.387 |
| Clients 8 | 0.507 | 0.362 | 0.384 |
| Clients 9 | 0.571 | 0.372 | 0.395 |
| Clients 10 | 0.553 | 0.375 | 0.398 |
| Clients 11 | 0.495 | 0.381 | 0.404 |
| Clients 12 | 0.616 | 0.369 | 0.391 |
| Clients 13 | 0.519 | 0.377 | 0.400 |
| Clients 14 | 0.537 | 0.359 | 0.381 |
| Clients 15 | 0.494 | 0.371 | 0.394 |
| Clients 16 | 0.543 | 0.380 | 0.403 |
| Clients 17 | 0.562 | 0.382 | 0.405 |
| Clients 18 | 0.523 | 0.394 | 0.419 |

| | | | |
|---|---|---|---|
| Clients 19 | 0.550 | 0.386 | 0.411 |
| Clients 20 | 0.476 | 0.390 | 0.415 |
| Epochs 1 | 0.548 | 0.324 | 0.343 |
| Epochs 2 | 0.540 | 0.187 | 0.209 |
| Epochs 5 | 0.588 | 0.338 | 0.359 |
| Epochs 10 | 0.565 | 0.120 | 0.168 |
| Epochs 20 | 0.529 | 0.299 | 0.322 |
| Balanced 60\|40 | 0.618 | 0.344 | 0.368 |
| Balanced 67\|33 | 0.541 | 0.147 | 0.177 |
| Balanced 75\|25 | 0.566 | 0.235 | 0.257 |
| Balanced 80\|20 | 0.563 | 0.286 | 0.305 |
| Balanced 40\|30\|30 | 0.606 | 0.288 | 0.306 |
| Balanced 60\|20\|20 | 0.526 | 0.201 | 0.221 |
| Balanced 80\|10\|10 | 0.584 | 0.239 | 0.258 |
| Balanced 60\|30\|10 | 0.601 | 0.328 | 0.347 |
| Balanced 30\|30\|15\|15\|10 | 0.529 | 0.384 | 0.408 |
| Balanced 60\|10\|10\|10\|10 | 0.577 | 0.323 | 0.342 |

Table 2.75: Best results for plain averaging on Papp-LLC (ADME)

| experiment | r2 | mae | rmse |
|---|---|---|---|
| Baseline | 0.577 | 0.071 | 0.124 |
| 60\|40 | 0.523 | 0.196 | 0.217 |
| 67\|33 | 0.533 | 0.254 | 0.271 |
| 75\|25 | 0.522 | 0.193 | 0.214 |
| 80\|20 | 0.497 | 0.199 | 0.219 |
| 40\|30\|30 | 0.596 | 0.193 | 0.213 |
| 60\|20\|20 | 0.560 | 0.279 | 0.297 |
| 80\|10\|10 | 0.522 | 0.166 | 0.189 |
| 60\|30\|10 | 0.559 | 0.335 | 0.356 |
| 30\|30\|15\|15\|10 | 0.542 | 0.240 | 0.258 |
| 60\|10\|10\|10\|10 | 0.512 | 0.304 | 0.323 |

Table 2.76: Best results for weighted averaging on Papp-LLC (ADME)

| experiment | r2 | mae | rmse |
|---|---|---|---|
| Baseline | 0.577 | 0.071 | 0.124 |
| Clients 2 | -0.502 | 0.135 | 0.233 |
| Clients 3 | 0.572 | 0.266 | 0.284 |
| Clients 4 | -0.502 | 0.135 | 0.233 |
| Clients 5 | 0.612 | 0.237 | 0.256 |
| Clients 6 | -0.227 | 0.865 | 0.886 |
| Clients 7 | 0.583 | 0.319 | 0.394 |
| Clients 8 | 0.561 | 0.159 | 0.192 |
| Clients 9 | 0.358 | 0.577 | 0.683 |
| Clients 10 | 0.544 | 0.117 | 0.165 |
| Clients 11 | -0.064 | 0.135 | 0.233 |
| Clients 12 | -0.502 | 0.135 | 0.233 |
| Clients 13 | -0.016 | 0.217 | 0.282 |
| Clients 14 | 0.438 | 0.220 | 0.307 |
| Clients 15 | 0.420 | 0.404 | 0.430 |
| Clients 16 | 0.474 | 0.403 | 0.429 |
| Clients 17 | 0.556 | 0.381 | 0.404 |
| Clients 18 | 0.459 | 0.379 | 0.402 |
| Clients 19 | 0.575 | 0.379 | 0.402 |
| Clients 20 | 0.507 | 0.502 | 0.534 |
| Epochs 1 | 0.486 | 0.275 | 0.297 |
| Epochs 2 | - | - | - |
| Epochs 5 | 0.516 | 0.104 | 0.181 |
| Epochs 10 | -0.471 | 0.135 | 0.233 |
| Epochs 20 | 0.646 | 0.224 | 0.241 |
| Balanced 60\|40 | 0.459 | 0.133 | 0.230 |
| Balanced 67\|33 | 0.603 | 0.134 | 0.233 |
| Balanced 75\|25 | 0.485 | 0.104 | 0.161 |
| Balanced 80\|20 | 0.547 | 0.204 | 0.222 |
| Balanced 40\|30\|30 | 0.541 | 0.102 | 0.159 |
| Balanced 60\|20\|20 | 0.468 | 0.097 | 0.161 |
| Balanced 80\|10\|10 | 0.462 | 0.104 | 0.159 |

| | | | |
|---|---|---|---|
| Balanced 60\|30\|10 | -0.502 | 0.135 | 0.233 |
| Balanced 30\|30\|15\|15\|10 | 0.295 | 0.655 | 0.724 |
| Balanced 60\|10\|10\|10\|10 | 0.268 | 0.138 | 0.236 |

Table 2.77: Best results for benchmarked averaging on Papp-LLC (ADME)

### 2.4.24 ADME - RbRat

| | | | |
|---|---|---|---|
| Baseline | 0.669 | 0.173 | 0.225 |
| Clients 2 | 0.388 | 0.603 | 0.745 |
| Clients 3 | 0.596 | 0.309 | 0.382 |
| Clients 4 | 0.651 | 0.382 | 0.471 |
| Clients 5 | 0.585 | 0.346 | 0.452 |
| Clients 6 | 0.729 | 0.320 | 0.389 |
| Clients 7 | 0.604 | 0.350 | 0.405 |
| Clients 8 | 0.394 | 0.324 | 0.429 |
| Clients 9 | 0.650 | 0.323 | 0.395 |
| Clients 10 | 0.611 | 0.323 | 0.396 |
| Clients 11 | 0.619 | 0.334 | 0.411 |
| Clients 12 | 0.706 | 0.328 | 0.397 |
| Clients 13 | 0.625 | 0.336 | 0.400 |
| Clients 14 | 0.565 | 0.332 | 0.399 |
| Clients 15 | 0.627 | 0.361 | 0.422 |
| Clients 16 | 0.564 | 0.372 | 0.440 |
| Clients 17 | 0.547 | 0.369 | 0.439 |
| Clients 18 | 0.508 | 0.327 | 0.399 |
| Clients 19 | 0.631 | 0.327 | 0.397 |
| Clients 20 | 0.427 | 0.325 | 0.392 |
| Epochs 1 | 0.582 | 0.443 | 0.845 |
| Epochs 2 | 0.570 | 0.399 | 0.635 |
| Epochs 5 | 0.408 | 0.464 | 0.629 |
| Epochs 10 | 0.499 | 0.351 | 0.454 |
| Epochs 20 | 0.549 | 0.386 | 0.557 |

| | | | |
|---|---|---|---|
| Balanced 60\|40 | 0.404 | 0.329 | 0.584 |
| Balanced 67\|33 | 0.613 | 0.381 | 0.464 |
| Balanced 75\|25 | 0.554 | 0.608 | 0.693 |
| Balanced 80\|20 | 0.649 | 0.326 | 0.516 |
| Balanced 40\|30\|30 | 0.628 | 0.666 | 0.791 |
| Balanced 60\|20\|20 | 0.476 | 0.380 | 0.667 |
| Balanced 80\|10\|10 | 0.507 | 0.328 | 0.450 |
| Balanced 60\|30\|10 | 0.593 | 0.341 | 0.465 |
| Balanced 30\|30\|15\|15\|10 | 0.420 | 0.331 | 0.488 |
| Balanced 60\|10\|10\|10\|10 | 0.634 | 0.331 | 0.404 |

Table 2.78: Best results for plain averaging on RbRat (ADME)

| experiment | r2 | mae | rmse |
|---|---|---|---|
| Baseline | 0.669 | 0.173 | 0.225 |
| 60\|40 | 0.524 | 0.344 | 0.604 |
| 67\|33 | 0.536 | 0.417 | 0.516 |
| 75\|25 | 0.424 | 0.359 | 0.609 |
| 80\|20 | 0.495 | 0.342 | 0.580 |
| 40\|30\|30 | 0.675 | 0.324 | 0.410 |
| 60\|20\|20 | 0.483 | 0.418 | 0.700 |
| 80\|10\|10 | 0.320 | 0.340 | 0.584 |
| 60\|30\|10 | 0.492 | 0.360 | 0.460 |
| 30\|30\|15\|15\|10 | 0.611 | 0.330 | 0.408 |
| 60\|10\|10\|10\|10 | 0.412 | 0.402 | 0.718 |

Table 2.79: Best results for weighted averaging on RbRat (ADME)

| experiment | r2 | mae | rmse |
|---|---|---|---|
| Baseline | 0.669 | 0.173 | 0.225 |
| Clients 2 | 0.616 | 0.575 | 0.834 |
| Clients 3 | 0.522 | 1.371 | 2.282 |

| | | | |
|---|---|---|---|
| Clients 4 | 0.137 | 0.807 | 1.006 |
| Clients 5 | 0.580 | 1.073 | 1.520 |
| Clients 6 | 0.210 | 0.714 | 1.305 |
| Clients 7 | 0.346 | 1.555 | 1.936 |
| Clients 8 | 0.354 | 0.990 | 1.328 |
| Clients 9 | 0.050 | 0.398 | 0.507 |
| Clients 10 | 0.108 | 1.714 | 2.184 |
| Clients 11 | 0.131 | 0.626 | 1.049 |
| Clients 12 | 0.111 | 0.697 | 1.125 |
| Clients 13 | -0.030 | 1.393 | 1.580 |
| Clients 14 | 0.279 | 0.784 | 1.118 |
| Clients 15 | 0.528 | 1.111 | 1.392 |
| Clients 16 | -0.118 | 0.868 | 1.419 |
| Clients 17 | 0.441 | 0.634 | 0.850 |
| Clients 18 | -0.048 | 1.142 | 1.734 |
| Clients 19 | 0.301 | 1.374 | 1.607 |
| Clients 20 | -0.014 | 0.628 | 1.128 |
| Epochs 1 | -0.100 | 0.856 | 0.982 |
| Epochs 2 | 0.587 | 0.744 | 0.866 |
| Epochs 5 | 0.708 | 0.582 | 0.707 |
| Epochs 10 | 0.396 | 0.812 | 1.229 |
| Epochs 20 | 0.464 | 11.679 | 15.911 |
| Balanced 60\|40 | 0.529 | 121.003 | 129.052 |
| Balanced 67\|33 | 0.589 | 1425.939 | 1513.100 |
| Balanced 75\|25 | 0.692 | 0.517 | 0.701 |
| Balanced 80\|20 | 0.532 | 1.420 | 1.865 |
| Balanced 40\|30\|30 | -0.109 | 0.483 | 0.726 |
| Balanced 60\|20\|20 | 0.731 | 2.234 | 4.891 |
| Balanced 80\|10\|10 | -0.588 | 0.833 | 0.942 |
| Balanced 60\|30\|10 | 0.603 | 2.110 | 2.277 |
| Balanced 30\|30\|15\|15\|10 | 0.332 | 0.982 | 1.785 |
| Balanced 60\|10\|10\|10\|10 | -0.112 | 0.842 | 0.993 |

Table 2.80: Best results for benchmarked averaging on RbRat (ADME)

### 2.4.25 ADME - Solubility

| experiment | roc_auc | pr_auc | accuracy | precision | recall | kappa |
|---|---|---|---|---|---|---|
| Baseline | 0.861 | 0.895 | 0.806 | 0.803 | 1.0 | 0.584 |
| Clients 2 | 0.837 | 0.867 | 0.806 | 0.810 | 1.000 | 0.590 |
| Clients 3 | 0.831 | 0.850 | 0.806 | 0.797 | 1.000 | 0.587 |
| Clients 4 | 0.824 | 0.859 | 0.796 | 0.788 | 1.000 | 0.568 |
| Clients 5 | 0.848 | 0.872 | 0.816 | 0.789 | 1.000 | 0.606 |
| Clients 6 | 0.831 | 0.852 | 0.806 | 0.800 | 1.000 | 0.587 |
| Clients 7 | 0.823 | 0.848 | 0.806 | 0.803 | 1.000 | 0.593 |
| Clients 8 | 0.830 | 0.860 | 0.816 | 0.797 | 1.000 | 0.609 |
| Clients 9 | 0.814 | 0.832 | 0.816 | 0.806 | 1.000 | 0.612 |
| Clients 10 | 0.832 | 0.863 | 0.796 | 0.767 | 1.000 | 0.562 |
| Clients 11 | 0.805 | 0.827 | 0.806 | 0.770 | 1.000 | 0.581 |
| Clients 12 | 0.822 | 0.850 | 0.786 | 0.764 | 1.000 | 0.543 |
| Clients 13 | 0.822 | 0.843 | 0.786 | 0.788 | 1.000 | 0.552 |
| Clients 14 | 0.827 | 0.856 | 0.796 | 0.800 | 1.000 | 0.574 |
| Clients 15 | 0.803 | 0.840 | 0.777 | 0.750 | 1.000 | 0.517 |
| Clients 16 | 0.813 | 0.845 | 0.806 | 0.778 | 1.000 | 0.584 |
| Clients 17 | 0.823 | 0.849 | 0.786 | 0.750 | 1.000 | 0.536 |
| Clients 18 | 0.826 | 0.863 | 0.786 | 0.750 | 1.000 | 0.536 |
| Clients 19 | 0.809 | 0.846 | 0.786 | 0.771 | 1.000 | 0.546 |
| Clients 20 | 0.824 | 0.865 | 0.757 | 0.727 | 1.000 | 0.471 |
| Epochs 1 | 0.821 | 0.855 | 0.816 | 0.786 | 1.000 | 0.604 |
| Epochs 2 | 0.835 | 0.860 | 0.816 | 0.794 | 1.000 | 0.604 |
| Epochs 5 | 0.849 | 0.872 | 0.816 | 0.766 | 1.000 | 0.598 |
| Epochs 10 | 0.832 | 0.851 | 0.825 | 0.818 | 1.000 | 0.633 |
| Epochs 20 | 0.835 | 0.872 | 0.796 | 0.825 | 1.000 | 0.568 |
| Balanced 60\|40 | 0.833 | 0.867 | 0.806 | 0.794 | 1.000 | 0.590 |
| Balanced 67\|33 | 0.835 | 0.856 | 0.806 | 0.783 | 1.000 | 0.584 |
| Balanced 75\|25 | 0.834 | 0.874 | 0.816 | 0.781 | 1.000 | 0.604 |
| Balanced 80\|20 | 0.831 | 0.851 | 0.786 | 0.779 | 1.000 | 0.549 |
| Balanced 40\|30\|30 | 0.851 | 0.884 | 0.825 | 0.800 | 1.000 | 0.628 |
| Balanced 60\|20\|20 | 0.820 | 0.839 | 0.786 | 0.771 | 1.000 | 0.546 |
| Balanced 80\|10\|10 | 0.829 | 0.862 | 0.777 | 0.761 | 1.000 | 0.524 |

| | | | | | | |
|---|---|---|---|---|---|---|
| Balanced 60\|30\|10 | 0.830 | 0.869 | 0.777 | 0.768 | 1.000 | 0.527 |
| Balanced 30\|30\|15\|15\|10 | 0.830 | 0.868 | 0.786 | 0.771 | 1.000 | 0.546 |
| Balanced 60\|10\|10\|10\|10 | 0.837 | 0.870 | 0.777 | 0.753 | 1.000 | 0.520 |

Table 2.81: Best results for plain averaging on Solubility (ADME)

| experiment | roc_auc | pr_auc | accuracy | precision | recall | kappa |
|---|---|---|---|---|---|---|
| Baseline | 0.861 | 0.895 | 0.806 | 0.803 | 1.0 | 0.584 |
| 60\|40 | 0.843 | 0.862 | 0.816 | 0.825 | 1.000 | 0.617 |
| 67\|33 | 0.834 | 0.871 | 0.806 | 0.794 | 1.000 | 0.590 |
| 75\|25 | 0.829 | 0.862 | 0.796 | 0.800 | 1.000 | 0.574 |
| 80\|20 | 0.823 | 0.851 | 0.806 | 0.770 | 1.000 | 0.581 |
| 40\|30\|30 | 0.837 | 0.867 | 0.806 | 0.778 | 1.000 | 0.584 |
| 60\|20\|20 | 0.837 | 0.866 | 0.796 | 0.791 | 1.000 | 0.571 |
| 80\|10\|10 | 0.833 | 0.856 | 0.796 | 0.810 | 1.000 | 0.577 |
| 60\|30\|10 | 0.824 | 0.845 | 0.796 | 0.781 | 1.000 | 0.565 |
| 30\|30\|15\|15\|10 | 0.832 | 0.855 | 0.786 | 0.781 | 1.000 | 0.549 |
| 60\|10\|10\|10\|10 | 0.837 | 0.876 | 0.816 | 0.797 | 1.000 | 0.609 |

Table 2.82: Best results for weighted averaging on Solubility (ADME)

| experiment | roc_auc | pr_auc | accuracy | precision | recall | kappa |
|---|---|---|---|---|---|---|
| Baseline | 0.861 | 0.895 | 0.806 | 0.803 | 1.0 | 0.584 |
| Clients 2 | 0.839 | 0.870 | 0.806 | 0.794 | 1.000 | 0.590 |
| Clients 3 | 0.840 | 0.864 | 0.806 | 0.814 | 1.000 | 0.593 |
| Clients 4 | 0.810 | 0.823 | 0.796 | 0.787 | 1.000 | 0.565 |
| Clients 5 | 0.834 | 0.866 | 0.806 | 0.800 | 1.000 | 0.587 |
| Clients 6 | 0.826 | 0.849 | 0.816 | 0.797 | 1.000 | 0.609 |
| Clients 7 | 0.828 | 0.852 | 0.806 | 0.783 | 1.000 | 0.581 |
| Clients 8 | 0.816 | 0.846 | 0.786 | 0.768 | 1.000 | 0.543 |
| Clients 9 | 0.822 | 0.839 | 0.796 | 0.783 | 1.000 | 0.568 |
| Clients 10 | 0.837 | 0.873 | 0.786 | 0.750 | 1.000 | 0.536 |

| | | | | | | |
|---|---|---|---|---|---|---|
| Clients 11 | 0.831 | 0.861 | 0.806 | 0.786 | 1.000 | 0.587 |
| Clients 12 | 0.825 | 0.853 | 0.796 | 0.767 | 1.000 | 0.562 |
| Clients 13 | 0.822 | 0.851 | 0.786 | 0.785 | 1.000 | 0.546 |
| Clients 14 | 0.828 | 0.853 | 0.825 | 0.809 | 1.000 | 0.631 |
| Clients 15 | 0.812 | 0.843 | 0.806 | 0.794 | 1.000 | 0.590 |
| Clients 16 | 0.828 | 0.862 | 0.786 | 0.764 | 1.000 | 0.543 |
| Clients 17 | 0.820 | 0.857 | 0.786 | 0.750 | 1.000 | 0.536 |
| Clients 18 | 0.825 | 0.858 | 0.806 | 0.770 | 1.000 | 0.581 |
| Clients 19 | 0.810 | 0.849 | 0.777 | 0.753 | 1.000 | 0.520 |
| Clients 20 | 0.832 | 0.876 | 0.757 | 0.722 | 1.000 | 0.468 |
| Epochs 1 | 0.829 | 0.867 | 0.816 | 0.783 | 1.000 | 0.604 |
| Epochs 2 | 0.855 | 0.887 | 0.806 | 0.786 | 1.000 | 0.587 |
| Epochs 5 | 0.831 | 0.857 | 0.796 | 0.783 | 1.000 | 0.568 |
| Epochs 10 | 0.857 | 0.881 | 0.796 | 0.800 | 1.000 | 0.574 |
| Epochs 20 | 0.849 | 0.877 | 0.786 | 0.781 | 1.000 | 0.549 |
| Balanced 60\|40 | 0.836 | 0.859 | 0.806 | 0.788 | 1.000 | 0.579 |
| Balanced 67\|33 | 0.832 | 0.854 | 0.806 | 0.806 | 1.000 | 0.576 |
| Balanced 75\|25 | 0.831 | 0.852 | 0.796 | 0.788 | 1.000 | 0.556 |
| Balanced 80\|20 | 0.830 | 0.861 | 0.825 | 0.784 | 1.000 | 0.623 |
| Balanced 40\|30\|30 | 0.861 | 0.896 | 0.825 | 0.800 | 1.000 | 0.628 |
| Balanced 60\|20\|20 | 0.846 | 0.856 | 0.825 | 0.818 | 1.000 | 0.633 |
| Balanced 80\|10\|10 | 0.819 | 0.866 | 0.757 | 0.778 | 1.000 | 0.496 |
| Balanced 60\|30\|10 | 0.854 | 0.876 | 0.796 | 0.767 | 1.000 | 0.562 |
| Balanced 30\|30\|15\|15\|10 | 0.826 | 0.850 | 0.786 | 0.779 | 1.000 | 0.549 |
| Balanced 60\|10\|10\|10\|10 | 0.821 | 0.866 | 0.767 | 0.746 | 1.000 | 0.498 |

Table 2.83: Best results for benchmarked averaging on Solubility (ADME)

### 2.4.26 AMES

| experiment | roc_auc | pr_auc | accuracy | precision | recall | kappa |
|---|---|---|---|---|---|---|
| Baseline | 0.903 | 0.908 | 0.837 | 0.879 | 0.897 | 0.674 |
| Clients 2 | 0.900 | 0.908 | 0.830 | 0.866 | 0.842 | 0.660 |
| Clients 3 | 0.894 | 0.902 | 0.827 | 0.870 | 0.977 | 0.653 |

| | | | | | | |
|---|---|---|---|---|---|---|
| Clients 4 | 0.898 | 0.907 | 0.840 | 1.000 | 0.864 | 0.680 |
| Clients 5 | 0.895 | 0.902 | 0.834 | 0.870 | 0.860 | 0.667 |
| Clients 6 | 0.890 | 0.896 | 0.828 | 0.861 | 0.863 | 0.656 |
| Clients 7 | 0.896 | 0.905 | 0.827 | 1.000 | 0.861 | 0.654 |
| Clients 8 | 0.892 | 0.902 | 0.827 | 0.836 | 1.000 | 0.652 |
| Clients 9 | 0.892 | 0.901 | 0.824 | 0.848 | 0.999 | 0.648 |
| Clients 10 | 0.892 | 0.904 | 0.822 | 1.000 | 0.858 | 0.643 |
| Clients 11 | 0.887 | 0.894 | 0.823 | 0.851 | 1.000 | 0.647 |
| Clients 12 | 0.889 | 0.898 | 0.815 | 1.000 | 0.878 | 0.627 |
| Clients 13 | 0.890 | 0.903 | 0.818 | 0.835 | 1.000 | 0.635 |
| Clients 14 | 0.886 | 0.896 | 0.814 | 0.814 | 0.992 | 0.626 |
| Clients 15 | 0.881 | 0.893 | 0.813 | 0.812 | 1.000 | 0.623 |
| Clients 16 | 0.884 | 0.895 | 0.816 | 0.828 | 1.000 | 0.631 |
| Clients 17 | 0.882 | 0.890 | 0.821 | 0.821 | 1.000 | 0.641 |
| Clients 18 | 0.885 | 0.891 | 0.820 | 0.832 | 1.000 | 0.638 |
| Clients 19 | 0.880 | 0.885 | 0.812 | 0.810 | 1.000 | 0.621 |
| Clients 20 | 0.886 | 0.893 | 0.819 | 0.822 | 0.989 | 0.637 |
| Epochs 1 | 0.902 | 0.913 | 0.838 | 0.885 | 0.879 | 0.675 |
| Epochs 2 | 0.901 | 0.911 | 0.839 | 0.877 | 0.863 | 0.677 |
| Epochs 5 | 0.906 | 0.914 | 0.838 | 1.000 | 0.849 | 0.676 |
| Epochs 10 | 0.898 | 0.904 | 0.831 | 0.862 | 0.859 | 0.661 |
| Epochs 20 | 0.898 | 0.902 | 0.838 | 0.862 | 0.912 | 0.676 |
| Balanced 60\|40 | 0.899 | 0.904 | 0.832 | 0.868 | 0.936 | 0.662 |
| Balanced 67\|33 | 0.896 | 0.906 | 0.834 | 0.869 | 0.875 | 0.669 |
| Balanced 75\|25 | 0.892 | 0.901 | 0.830 | 0.873 | 0.839 | 0.659 |
| Balanced 80\|20 | 0.897 | 0.902 | 0.834 | 0.870 | 0.867 | 0.667 |
| Balanced 40\|30\|30 | 0.901 | 0.909 | 0.832 | 0.876 | 0.933 | 0.664 |
| Balanced 60\|20\|20 | 0.894 | 0.904 | 0.830 | 1.000 | 0.869 | 0.660 |
| Balanced 80\|10\|10 | 0.896 | 0.903 | 0.832 | 0.856 | 0.922 | 0.664 |
| Balanced 60\|30\|10 | 0.895 | 0.903 | 0.827 | 0.876 | 0.875 | 0.656 |
| Balanced 30\|30\|15\|15\|10 | 0.894 | 0.901 | 0.834 | 0.860 | 0.865 | 0.668 |
| Balanced 60\|10\|10\|10\|10 | 0.899 | 0.910 | 0.825 | 0.847 | 0.878 | 0.648 |

Table 2.84: Best results for plain averaging on AMES (final)

| experiment | roc_auc | pr_auc | accuracy | precision | recall | kappa |
|---|---|---|---|---|---|---|
| Baseline | 0.903 | 0.908 | 0.837 | 0.879 | 0.897 | 0.674 |
| 60\|40 | 0.896 | 0.908 | 0.828 | 0.871 | 0.836 | 0.655 |
| 67\|33 | 0.895 | 0.903 | 0.831 | 0.871 | 0.855 | 0.662 |
| 75\|25 | 0.894 | 0.903 | 0.827 | 0.871 | 0.854 | 0.653 |
| 80\|20 | 0.897 | 0.907 | 0.829 | 0.872 | 0.849 | 0.658 |
| 40\|30\|30 | 0.896 | 0.905 | 0.836 | 0.873 | 0.851 | 0.670 |
| 60\|20\|20 | 0.896 | 0.904 | 0.832 | 0.877 | 0.870 | 0.665 |
| 80\|10\|10 | 0.902 | 0.907 | 0.831 | 0.871 | 0.920 | 0.662 |
| 60\|30\|10 | 0.899 | 0.905 | 0.832 | 0.872 | 0.848 | 0.664 |
| 30\|30\|15\|15\|10 | 0.901 | 0.905 | 0.830 | 0.860 | 0.867 | 0.659 |
| 60\|10\|10\|10\|10 | 0.898 | 0.905 | 0.828 | 0.876 | 0.841 | 0.656 |

Table 2.85: Best results for weighted averaging on AMES (final)

| experiment | roc_auc | pr_auc | accuracy | precision | recall | kappa |
|---|---|---|---|---|---|---|
| Baseline | 0.903 | 0.908 | 0.837 | 0.879 | 0.897 | 0.674 |
| Clients 2 | 0.872 | 0.883 | 0.804 | 1.000 | 0.866 | 0.605 |
| Clients 3 | 0.886 | 0.898 | 0.814 | 1.000 | 0.884 | 0.624 |
| Clients 4 | 0.877 | 0.886 | 0.807 | 1.000 | 0.906 | 0.612 |
| Clients 5 | 0.886 | 0.896 | 0.815 | 0.843 | 0.893 | 0.629 |
| Clients 6 | 0.869 | 0.879 | 0.809 | 1.000 | 0.862 | 0.615 |
| Clients 7 | 0.874 | 0.886 | 0.804 | 0.824 | 0.987 | 0.605 |
| Clients 8 | 0.881 | 0.890 | 0.812 | 0.827 | 0.889 | 0.621 |
| Clients 9 | 0.872 | 0.886 | 0.804 | 0.814 | 1.000 | 0.603 |
| Clients 10 | 0.877 | 0.888 | 0.806 | 1.000 | 0.897 | 0.609 |
| Clients 11 | 0.875 | 0.888 | 0.806 | 0.815 | 0.894 | 0.609 |
| Clients 12 | 0.868 | 0.882 | 0.800 | 0.817 | 1.000 | 0.597 |
| Clients 13 | 0.876 | 0.885 | 0.805 | 1.000 | 0.904 | 0.607 |
| Clients 14 | 0.876 | 0.890 | 0.798 | 1.000 | 0.900 | 0.593 |
| Clients 15 | 0.875 | 0.887 | 0.797 | 0.809 | 1.000 | 0.590 |
| Clients 16 | 0.874 | 0.880 | 0.797 | 1.000 | 0.893 | 0.587 |
| Clients 17 | 0.874 | 0.885 | 0.810 | 1.000 | 0.892 | 0.618 |

| | | | | | | |
|---|---|---|---|---|---|---|
| Clients 18 | 0.882 | 0.891 | 0.800 | 0.805 | 1.000 | 0.596 |
| Clients 19 | 0.873 | 0.882 | 0.805 | 1.000 | 0.903 | 0.606 |
| Clients 20 | 0.869 | 0.880 | 0.800 | 0.799 | 1.000 | 0.595 |
| Epochs 1 | 0.883 | 0.897 | 0.816 | 0.839 | 0.867 | 0.629 |
| Epochs 2 | 0.883 | 0.892 | 0.814 | 0.846 | 0.887 | 0.627 |
| Epochs 5 | 0.879 | 0.892 | 0.810 | 0.837 | 0.859 | 0.616 |
| Epochs 10 | 0.879 | 0.887 | 0.804 | 0.827 | 0.879 | 0.606 |
| Epochs 20 | 0.877 | 0.884 | 0.809 | 0.857 | 0.887 | 0.616 |
| Balanced 60\|40 | 0.874 | 0.885 | 0.808 | 0.843 | 0.916 | 0.614 |
| Balanced 67\|33 | 0.876 | 0.887 | 0.806 | 0.841 | 0.854 | 0.611 |
| Balanced 75\|25 | 0.877 | 0.887 | 0.807 | 0.836 | 0.854 | 0.612 |
| Balanced 80\|20 | 0.873 | 0.881 | 0.815 | 0.831 | 0.856 | 0.628 |
| Balanced 40\|30\|30 | 0.879 | 0.884 | 0.808 | 0.839 | 0.866 | 0.615 |
| Balanced 60\|20\|20 | 0.879 | 0.891 | 0.808 | 0.830 | 0.863 | 0.614 |
| Balanced 80\|10\|10 | 0.872 | 0.884 | 0.808 | 0.829 | 0.884 | 0.614 |
| Balanced 60\|30\|10 | 0.877 | 0.888 | 0.807 | 0.852 | 0.989 | 0.611 |
| Balanced 30\|30\|15\|15\|10 | 0.881 | 0.892 | 0.812 | 0.832 | 0.886 | 0.622 |
| Balanced 60\|10\|10\|10\|10 | 0.874 | 0.884 | 0.804 | 0.829 | 0.866 | 0.606 |

Table 2.86: Best results for benchmarked averaging on AMES (final)

### 2.4.27 ChEMBL

| experiment | roc_auc | pr_auc | accuracy | precision | recall | kappa |
|---|---|---|---|---|---|---|
| Baseline | 0.882 | 0.882 | 0.800 | 0.807 | 0.810 | 0.600 |
| Clients 2 | 0.877 | 0.875 | 0.799 | 0.810 | 0.817 | 0.597 |
| Clients 3 | 0.878 | 0.878 | 0.799 | 1.000 | 0.835 | 0.598 |
| Clients 4 | 0.878 | 0.877 | 0.799 | 1.000 | 0.818 | 0.598 |
| Clients 5 | 0.879 | 0.877 | 0.800 | 1.000 | 0.830 | 0.600 |
| Clients 6 | 0.879 | 0.880 | 0.799 | 1.000 | 0.863 | 0.598 |
| Clients 7 | 0.879 | 0.878 | 0.800 | 1.000 | 0.821 | 0.600 |
| Clients 8 | 0.878 | 0.877 | 0.798 | 1.000 | 0.934 | 0.595 |
| Clients 9 | 0.876 | 0.876 | 0.796 | 1.000 | 0.816 | 0.591 |
| Clients 10 | 0.873 | 0.874 | 0.794 | 1.000 | 0.810 | 0.588 |

| | | | | | | |
|---|---|---|---|---|---|---|
| Clients 11 | 0.871 | 0.872 | 0.792 | 1.000 | 0.848 | 0.583 |
| Clients 12 | 0.870 | 0.870 | 0.790 | 1.000 | 0.881 | 0.581 |
| Clients 13 | 0.867 | 0.868 | 0.787 | 1.000 | 0.865 | 0.573 |
| Clients 14 | 0.865 | 0.866 | 0.786 | 1.000 | 0.869 | 0.571 |
| Clients 15 | 0.867 | 0.868 | 0.788 | 1.000 | 0.886 | 0.576 |
| Clients 16 | 0.865 | 0.865 | 0.786 | 1.000 | 0.801 | 0.572 |
| Clients 17 | 0.860 | 0.860 | 0.780 | 1.000 | 0.853 | 0.560 |
| Clients 18 | 0.861 | 0.862 | 0.781 | 1.000 | 0.843 | 0.562 |
| Clients 19 | 0.857 | 0.857 | 0.777 | 1.000 | 0.856 | 0.554 |
| Clients 20 | 0.855 | 0.856 | 0.774 | 1.000 | 0.792 | 0.549 |
| Epochs 1 | 0.867 | 0.864 | 0.789 | 0.800 | 0.941 | 0.577 |
| Epochs 2 | 0.869 | 0.869 | 0.790 | 1.000 | 0.814 | 0.579 |
| Epochs 5 | 0.882 | 0.882 | 0.800 | 1.000 | 0.828 | 0.600 |
| Epochs 10 | 0.877 | 0.876 | 0.799 | 1.000 | 0.818 | 0.598 |
| Epochs 20 | 0.870 | 0.870 | 0.791 | 1.000 | 0.803 | 0.582 |
| Balanced 60\|40 | 0.878 | 0.878 | 0.798 | 1.000 | 0.824 | 0.596 |
| Balanced 67\|33 | 0.879 | 0.879 | 0.799 | 1.000 | 0.823 | 0.598 |
| Balanced 75\|25 | 0.878 | 0.877 | 0.799 | 1.000 | 0.825 | 0.597 |
| Balanced 80\|20 | 0.879 | 0.877 | 0.801 | 1.000 | 0.826 | 0.602 |
| Balanced 40\|30\|30 | 0.880 | 0.880 | 0.801 | 1.000 | 0.819 | 0.601 |
| Balanced 60\|20\|20 | 0.881 | 0.879 | 0.801 | 1.000 | 0.836 | 0.602 |
| Balanced 80\|10\|10 | 0.881 | 0.881 | 0.803 | 1.000 | 0.839 | 0.605 |
| Balanced 60\|30\|10 | 0.880 | 0.880 | 0.801 | 0.808 | 0.831 | 0.602 |
| Balanced 30\|30\|15\|15\|10 | 0.880 | 0.879 | 0.801 | 1.000 | 0.828 | 0.601 |
| Balanced 60\|10\|10\|10\|10 | 0.879 | 0.879 | 0.799 | 0.880 | 0.818 | 0.598 |

Table 2.87: Best results for plain averaging on ChEMBL (v1)

### 2.4.28 Discussion

The first thing we notice is that some of the smaller datasets are more chaotic when we look at metrics and are less trustworthy in general. This makes sense. If the dataset has only 200 samples, it will result in a training set with only 10 samples for each client when we split it among 20 parties. Smaller ADME datasets should therefore be analyzed with caution.

We ran over 1000 experiments totaling over 20000 epochs, so it is difficult to find common behaviors. For almost any trend we observe we also find exceptions for it. But one prevalent behavior is that the ROC-AUC metric and especially the PR-AUC metric need longer time to converge as the number of clients increases. This effect is clearly visible on most Tox21 tasks, on AMES, on ChEMBL, as well as some ADME datasets like PappCaco2 and Solubility.

Regression tasks are affected a lot more by the federated learning process. Even for larger datasets like CLint the losses in R2 can surpass 50%, and in case of PappCaco2, it can even fail to converge. Other datasets like FupHuman, FupRat, or even the very small RbRat seem to be less affected, but the differences are still much bigger when compared to classification tasks. These effects seemed to be smaller when using benchmarked aggregation (ie: on CLint), but on small or difficult datasets, this method fails in several rounds because the intermediate checkpoints diverge in early phases (ie: on FuBrain, PappCaco2, PappLLC, or RbRat).

Classification tasks work very well on the other hand on almost all tasks, with small losses registered even with large number of clients. This is especially so for the ROC-AUC metric, which is more stable in general compared to PR-AUC and Cohen's Kappa.

Occasionally, we notice a clear sign that the performance decreases as the number of clients increases, but other times, the scores a scrambled, and there is no clear indication if more or less clients work better. This might seem like a strange occurrence because, we would expect the performance to get worse as the local dataset becomes smaller, even if we later aggregate the result, however, it would seem that the federation process can actually be beneficial and acts as a sort of regularization. What we are doing is very similar to what is often referred to in the literature as Stochastic Weight Averaging[28] which has been shown to lead to better generalization. When the effect is not clear in client-based experiments, the differences are likely just random occurrences, similar to what we would get if we trained the model locally with various random splits.

Looking at the number of epochs trained per round, metrics are generally not significantly affected for the most part, but training for too many epochs (20 per round) does sometimes adversely affect the outcome. Otherwise, plain

aggregation generally seemed to prefer higher epoch rounds, while benchmarked aggregation worked scored better on fewer rounds.

Another matter worth mentioning is that recall (and sometimes precision) values in the tables presented do not seem to offer any insight by themselves. This is because early in the training process many models often just predict the same value for any input, which often leads to a perfect recall score. As we only show the best values registered in these tables, readers are referred to the Appendix to see how recall and precision values change throughout the training progress.

More importantly, the aggregation method we use does not seem to make that much of a difference, especially on classification tasks. Balancing weights by the number of samples often increased performance, but the differences where much smaller than we expected. Also, there were several exceptions where plain, naive averaging worked better.

Weighting the checkpoints by their performance on a metric was surely expected to outperform naive aggregation, but that was not always the case. We should of course remember the difficulties of evaluating checkpoints on difficult and small datasets, which makes this approach even less attractive (as we can see from the large number of gaps in the line plots from the Appendix). Benchmarked aggregation slightly outperforms plain or, occasionally, even weighted aggregation on classification tasks as well, but the difference is seldom significant. However, client-based and imbalanced split experiments were executed with 10 epochs trained per round. Often times we noticed that benchmarked averaging performed better when we train for fewer epochs per round, so decreasing the number of epochs for these experiments might be worth checking out.

## 2.5 Performance Analysis

The time it takes to featurize a dataset and to train one epoch will depend on the architecture and the size of the dataset. We report in Table 2.88 our recorded time for featurizing the training set and the time it takes to train 1 epoch on the traning set.

| Dataset | Featurization | Training |
|---|---|---|
| Tox21 | 49.682 | 14.194 |
| AMES | 48.530 | 7.945 |
| CLint | 43.739 | 12.333 |
| FeHuman | 5.129 | 3.281 |
| FuBrain | 6.848 | 4.576 |
| FupHuman | 21.483 | 4.483 |
| FupRat | 6.675 | 3.334 |
| NER-LLC | 5.866 | 3.200 |
| PappCaco2 (cls) | 38.291 | 6.691 |
| PappCaco2 (reg) | 38.400 | 7.204 |
| PappLLC | 6.311 | 2.989 |
| RbRat | 4.138 | 2.901 |
| Solubility | 6.893 | 3.145 |
| ChEMBL (v1) | 8147.601 | 5171.256 |
| ChEMBL (v2) | 13436.784 | 8528.283 |

Table 2.88: Registered time (in seconds) for featurizing the training set and for training one epoch on CPU.

# Technical Documentation

## 3.1 Installing Dependencies

In order to perform experiments, some preliminary steps must be taken. Namely, we must install all dependencies which we support with conda. If conda is not installed locally, one may follow the online installation guides.

As a first step, we create a new environment using the provided snapshot. Then, we install additional dependencies for PyTorch Geometric (which are downloaded as wheel files).

```
conda env create -f environment.yml
conda activate kmol
bash install.sh
```

This step has to be performed only one time. Once the environment is created and the depencies installed, one may use the new environment at any time by activating it.

```
conda activate kmol
```

## 3.2 Project Structure

Our code is split into several folders:

- **data**: contains datasets, checkpoints, certificates, logs, and everything else data related. This folder is not fixed. Data can be stored anywhere else on the disk.
- **src/kmol**: contains all the custom code implemented for data preprocessing, training, inference, and experimenting with models in general.
- **src/mila**: contains the communication module (federated learning)
- **environment.yml, install.sh**: are installation environments and scripts

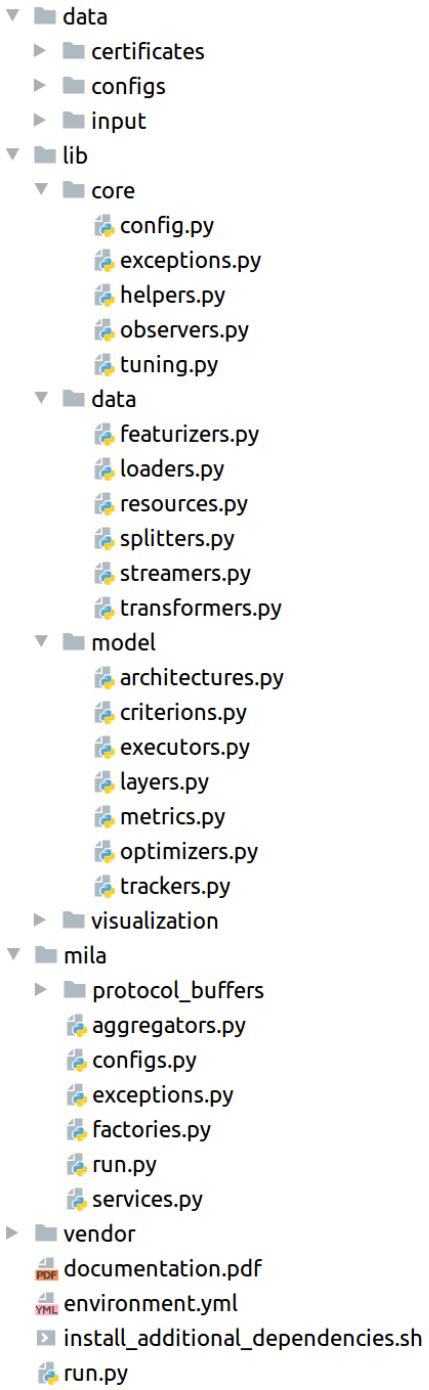


Figure 3.9: Project Structure

- **documentation.pdf**: is this file

The main codebase (**src/kmol**) is furthermore split into 4 folders:

- **core**: contains often used functionality and helper functions
- **data**: contains data handles and pre-processing tools
- **model**: contains architectures, metrics, and modeling tools
- **visualization**: contains functionality from visualizing results

In the **core** folder we have:

- **config.py**: for configuration parsing
- **exceptions.py**: for exception handling
- **helpers.py**: for functionality like caching, time tracking, reflection, and dependency injection
- **observers.py**: for event management
- **tuning.py**: for bayesian optimization

In **data** we have:

- **featurizers.py**: for preparing input fields
- **loaders.py**: for loading raw data formats from disk

- **resources.py**: for common structures
- **splitters.py**: for data splitting
- **streamers.py**: where we combine data loading and preprocessing tools
- **transformers.py**: for preparing output fields

In **model** we have:

- **architectures.py**: for model architectures
- **criterions.py**: for custom loss functions
- **executors.py**: for pipelines (ie: training, inference, evaluation)
- **layers.py**: for custom layers
- **metrics.py**: for handling and computing metrics
- **optimizers.py**: for custom optimizers
- **trackers.py**: for tracking running averages

In **src**/**mila** we have:

- **protocol_buffers**: for service contracts and gRPC messages
- **aggregators.py**: for aggregator implementations
- **configs.py**: for server and client configuration parsing
- **exceptions.py**: for exception handling
- **factories.py**: for abstractions
- **run.py**: is the entry point for federated learning (both server and clients)
- **services.py**: implements the communication between server and clients

## 3.3 Framework

Our vision and efforts for the provided solution are centered are 3 key points:

- good quality and clean code
- extensible with minimum effort
- highly configurable and easy to use

We rely on several concepts and design patterns improve these characteristics. We briefly describe some of the major components.

### 3.3.1 Abstractions

Most of our components, like loaders, featurizers, networks, or execution pipelines are designed by contract. For each group of components, we first design an abstract class (ie: AbstractLoader) with common functionality, but more importantly, common methods. The abstract classes are then extended to implement detailed solutions (ie: CsvLoader).

Then, if high level components (like the streamers) rely on low level ones (like the loaders), it is enough for us to specify the abstraction, and the high level module will be able to handle all subtypes of it (it acts as a contract). In object-oriented design, this is referred to as dependency inversion.

### 3.3.2 Dependency Injection

Dependency injection makes a class independent of its dependencies (ie: input arguments). We achieve this by decoupling the use of a component from its instantiation.

Among other things, this means we can change the code of a lower level objects without having to modify higher level objects which depend on it. This can be a very powerful concept, because it allows us to easily add new functionality without having to write a lot of additional code.

For a more visual way to see the benefits of this concept, see the next section.

### 3.3.3 Dynamic Configuration

To run experiments, we require a configuration file in JSON format (details in a later section). While some of the structure is rigid, for the most part, configurable options are directly linked to the argument lists of the components constructor. This is achieved with dependency injection.

For example, let us assume we have a "Calculator" object we want to make use of:

```
# file: custom/helpers.py

class Calculator:
    def __init__(self, x: int, y: int):
        self._x = x
        self._y = y

    def sum(self) -> int:
        return self._x + self._y
```

To make use of this object, no changes have to be made to the configuration parsing logic whatsoever. We just point to the class we want to use and specify its arguments:

```
{
    "helper": {
        "type": custom.helpers.Calculator,
        "x": 2,
        "y": 3
    }
}
```

Input arguments do not have to be numeric though, we can use any objects whatsoever. If additional classes are required, they will be instantiated recursively. As an example, this would also work:

```
# file: custom/computers.py

from custom.helpers import Calculator

class Computer:

    def __init__(self, calculator: Calculator, constants: list):
        self._calculator = calculator
        self._constants = constants
```

```
    def sum(self, x: int, y: int) -> int:
        return self._calculator.sum(x, y)
```

```
{
    "computer": {
        "type": "custom.computers.Computer",
        "calculator": {
            "type": custom.helpers.Calculator,
            "x": 2,
            "y": 3
        },
        "constants": [1, 5]
    }
}
```

We make use of this functionality a lot and it makes it very easy to use already implemented features like loss functions from the core Pytorch library[51] or graph convolutional operators from Pytorch Geometric[19]. More importantly, we can make use of them without having to write large chunks of complicated logic, or any code at all for most cases.

On the negative side, the compatibility can make the configuration less straight forward and require a level of knowledge about the underlying arguments. We make this easier by providing plenty of examples on how to configure experiments (see the `data/configs/` folder).

### 3.3.4 Events & Observer

Not everything can be solved with dependency injection. Some areas of any framework are just inherently complicated, complicated code is never good because it is messy, hard to maintain and more error prone.

One good example is the training pipeline. Even for minimal functionality, we need to load checkpoints, initiate data loaders, perform forward and backward passes, log progress, and compute metrics. However, as a central area where many components connect, usually things do not stop there and extra functionality continue to pile up in the area.

To somewhat mitigate this problem, we introduced an event manager, also called

an observer pattern. In an observer pattern, we keep track of a list of dependents which are automatically notified when important events happen. In case of the training pipeline for example, we can define such a dependent to receive a payload containing the model and optimizer right before backpropagation happens. Of course, we can then add any changes we want using that payload.

There are several benefits to this approach.

- we will not have an ever growing training script for one
- due to dynamic configuration, anyone can add their own event handlers without having to touch the core codebase at all
- this is very useful for backwards compatibility as well, because changes will not be overwritten upon update

However, adding too many observers which modify the original behavior can hide or cause dependency issues, and make the functionality harder to debug.

The list of dispatchers (events that can be listened to) are listen in the configuration section below. An example of how to write a custom handler is described in the customization section.

## 3.4 Models

Models live under the `lib/` folder.

Each experiment requires a configuration file to run, which we describe in the next section. Furthermore, for the models to be compatible with Mila, it has to extend 2 abstract classes from the Mila library.

- the **AbstractConfiguration** class - responsible for configuration handling
- and the **AbstractExecutor** class - responsible for starting the train, validation, and inference processes.

### 3.4.1 Configuration

Configuration files are stored in JSON format and parsed by a class extending `AbstractConfiguration` . When using Mila with a BenchmarkedAggregator, the

extended configuration must also include a "checkpoint_path" argument.

All experiments and operations make use of the same configuration file. Certain fields are dynamic and used for dependency injection. In these cases a "type" subfield will specify the desired class type, while all other subfields will be considered arguments for that class. Class arguments themselves can contain a "type" subfield, in which case additional objects will be instantiated recursively.

The configurable options are as follows.

### *model (no default)*

The model option is used to specify which model architecture to use and the options of that model. It is a dynamic field and used to instantiate torch modules which extend `kmol.model.architectures.AbstractNetwork` .

As a dynamic field, the "type" argument is used to specify which class to use. At the moment, we support 6 architectures as input for the "type" option:

- **graph_convolutional**
- **message_passing**
- **triplet_message_passing**
- **linear**
- **convolutional**
- **protein_ligand**

```
"model": {
    "type": "graph_convolutional",
    "in_features": 45,
    "out_features": 1,
    "hidden_features": 160,
    "dropout": 0.1,
    "layer_type": "torch_geometric.nn.GCNConv",
    "layers_count": 3,
    "molecule_features": 17,
    "is_residual": 1,
    "norm_layer": "lib.model.layers.BatchNorm"
},
```

Figure 3.10: An example of how to configure a Graph Convolutional Network

The `graph_convolutional` is by far the most configurable architecture. Customizable arguments for this model are as follows:

- **in_features**: the number of input features. This value will depend on the options configured for graph featurizer, but should be "45" by default.
- **hidden_features**: number of hidden features used in the graph convolutional operation.
- **out_features**: the number of output features. This should coincide with the number of concurrent tasks we are trying to predict (ie: 12 for Tox21, and 1 for AMES).
- **molecule_features**: how many molecule level features are we expecting? This should be set to "17" for RdKit descriptors and "1613" for Mordred descriptors.
- **dropout**: the dropout rate
- **layer_type**: The graph convolutional operator (defaults to "torch_geometric.nn.GCNConv"). We leverage implementations from Pytorch Geometric and support a large number of layers listen in torch_geometric.nn. Layers which have been tested to work are as follows:

  - torch_geometric.nn.GCNConv
  - torch_geometric.nn.ChebConv
  - torch_geometric.nn.SAGEConv
  - torch_geometric.nn.GraphConv
  - torch_geometric.nn.ARMAConv
  - torch_geometric.nn.LEConv
  - torch_geometric.nn.GENConv
  - torch_geometric.nn.ClusterGCNConv
  - torch_geometric.nn.FeaStConv
  - torch_geometric.nn.GATConv
  - torch_geometric.nn.TAGConv
  - torch_geometric.nn.SGConv
  - kmol.model.layers.GINConvolution
  - kmol.model.layers.TrimConvolution

- **layers_count**: The number of graph layers to use (defaults to "2")
- **is_residual**: Whether to apply a residual connection to the graph operation (defaults to "True")
- **norm_layer**: Which normalization layer to apply after the graph operation (defaults to "None"). The available options are:
  - None
  - kmol.model.layers.BatchNorm
  - kmol.model.layers.GraphNorm
  - torch_geometric.nn.LayerNorm
- **activation**: What activation function to use? (defaults to "torch.nn.ReLU"[1]). For a list of options, please check the PyTorch documentation on non-linear activations.
- **edge_features**: The number of edge features (defaults to 0). At the moment, the only working options are "0" or "12". If set to 12, edge features will be concatenated with atom features.

This is my no means an exhaustive list, as the options are dynamic. Should someone wish to implement custom models, the input arguments will be automatically passed from the configuration file.

Each graph convolutional operator can have additional options configured based on the input argument lists. We point curious readers to the PyTorch Geometric docs for a list of all available options. As an example, for GENConv, one might easily specify the initial inverse temperature or the initial power for mean aggregation by passing and additional "t" or "p" argument in the configuration file.

As a note, when the "layer_type" is set to "kmol.model.layers.TrimConvolution", 2 additional options should be specified. Namely, "propagate_edge_features" should be set to "True" and "in_edge_features" should be set to "12".
The `message_passing` [21][57] architecture accepts the following options:

- **in_features**: the number of input features. This value will depend on the options configured for graph featurizer, but should be "45" by default.
- **hidden_features**: number of hidden features used in the graph convolutional kernel.
- **out_features**: the number of output features.
- **edge_features**: The number of edge features. At the moment, the only working option is 12.
- **edge_hidden**: the number of hidden features for the edge block.
- **steps**: number of processing steps to run
- **dropout**: the dropout rate (defaults to 0)
- **aggregation**: the aggregation scheme to use. (Options are: "add", "mean", or "max") (defaults to "add")
- **set2set_layers**: number of recurrent layers to use in the the global pooling operator[65] (defaults to "3")
- **set2set_steps**: processing steps for the global pooling operator[65] (defaults to "6")

The `triplet_message_passing` [36] architecture accepts the following options:

- **in_features**: the number of input features. This value will depend on the options configured for graph featurizer, but should be "45" by default.
- **hidden_features**: number of hidden features used in the graph convolutional kernel.
- **out_features**: the number of output features.
- **edge_features**: the number of edge features. At the moment, the only working option is 12.
- **layers_count**: the number of Triplet Message Passing layers to use
- **dropout**: the dropout rate (defaults to 0)
- **set2set_layers**: number of recurrent layers to use in the the global pooling operator[65] (defaults to "1")
- **set2set_steps**: processing steps for the global pooling operator[65] (defaults to "6")

The `linear` architecture is a simple Shallow Neural Network with 2 linear layers.

- **in_features**: the number of input features.
- **hidden_features**: the number of hidden features
- **out_features**: the number of output features
- **activation**: the activation type (defaults to "torch.nn.ReLU"[1])

The `convolutional` architecture is a simple Convolutional Network with a single 1D convolutional layer, and a max pooling layer, followed by a linear block as describe above:

- **in_features**: the number of input features.
- **hidden_features**: the number of hidden features
- **out_features**: the number of output features

The `protein_ligand` is a composite architecture. That is, it takes 2 other modules as input, one for the ligand and one for the protein features. Dependency injection again makes our work easy.

```
"model": {
    "type": "protein_ligand",
    "protein_module": {
        "type": "linear",
        "in_features": 9723,
        "hidden_features": 160,
        "out_features": 16
    },
    "ligand_module": {
        "type": "graph_convolutional",
        "in_features": 45,
        "out_features": 16,
        "hidden_features": 192,
        "dropout": 0.0,
        "layer_type": "torch_geometric.nn.GENConv",
        "layers_count": 5,
        "molecule_features": 17,
        "is_residual": 0,
        "norm_layer": "lib.model.layers.BatchNorm"
    },
    "hidden_features": 32,
    "out_features": 3
},
```

Figure 3.11: An example of how to configure a Protein-Ligand Architecture

- **protein_module**: a list set of options for the protein module. Supported options include a *convolutional* architecture for tokenized inputs, or a *linear* architecture for bag-of-words featurized inputs.
- **ligand_module**: a list set of options for the ligand module. Supported options include a *graph_convolutional*, *message_passing*, or *triplet_message_passing* architecture for graph featurized ligands, or a *linear* architecture for circular fingerprints.
- **hidden_features**: the number of hidden features

- **out_features**: the number of output features

After the individual blocks are passed through, the outputs are concatenated and passed through a final "linear" block.

### *loader (no default)*

The loader option is used to configure how raw data files are loaded. It is a dynamic field used to instantiate components which extend `kmol.data.loaders.AbstractLoader` .
We support 3 loaders at this time, which are specified in the "type":

- **csv**: For comma-separated values
- **excel**: For excel spreadsheets (only one spreadsheet can be loaded at a time)
- **sdf**: For SDF files (where the whole dataset is stored in a single SDF file)

The additional arguments are identical for the most part at this time. One exception is the `excel` loader, which accepts a numeric "sheet_index" to specify which sheet to load (numbering starts from 0 for the first sheet in the file). Otherwise, all loaders are expected to receive 3 fields:

- **input_path**: the relative or absolute path to the input file
- **input_column_names**: a list of columns or properties which should be mapped as input features
- **target_column_names**: a list of columns or properties which should be mapped as output features

```
"loader": {
    "type": "csv",
    "input_path": "data/input/tox21/raw/tox21.csv",
    "input_column_names": ["smiles"],
    "target_column_names": [
        "NR-AR", "NR-AR-LBD", "NR-AhR", "NR-Aromatase",
        "NR-ER", "NR-ER-LBD", "NR-PPAR-gamma", "SR-ARE",
        "SR-ATAD5", "SR-HSE", "SR-MMP", "SR-p53"
    ]
},
```

Figure 3.12: An example of how to configure a CSV loader for the AMES dataset

### *featurizers (no default)*

Featurizers prepare the input features for the training or inference. The expected input is a list, and therefore we can specify more than one featurizer per experiment. Featurizers are dynamic fields used to instantiate components extending `kmol.data.featurizers.AbstractFeaturizer` .

As any dynamic field, the featurizer to be used is specified with the "type" option. The supported featurizers are as follows:

- **graph**: for graph featurization. This featurizer expects a SMILES[69] string for input, and the generated features can be used by graph architectures (*graph_convolutional*, *message_passing*, or *triplet_message_passing*)
- **circular_fingerprint**: for fingerprint featurization. This featurizer expects a SMILES[69] string for input, and the generated features can be used by the *linear* architecture.
- **token**: similar to the one-hot encoder, but will tokenize a whole sentence (like an amino-sequence). The expected input is a sequence (string), and the output can be used by the *convolutional* architecture.
- **bag_of_words**: performs an n-gram, bag-of-words featurization. The expected input is a sequence (string), and the generated features can be used by the *linear* architecture.

- **one_hot_encoder**: can one-hot encode categorical features. The expected input is a string. No architectures make direct use of this featurizer at this point, however, it could be used in a featurization pipeline to prepare inputs for other featurizers.
- **transpose**: is an intermediary featurizer. It expects a Torch Tensor as input and will output a transposed version of it. This of course can be done in a model directly, however, featurized outputs are cached, and can save valuable computational time.
- **fixed**: is an intermediary featurizer. It expects a single float as input and will return that value divided by a user specified value.
- **converter**: convert molecules to another format (ie: inchi to smiles)

```
"featurizers": [
    {
        "type": "graph",
        "inputs": ["smiles"],
        "outputs": ["ligand"],
        "descriptor_calculator": {"type": "rdkit"}
    },{
        "type": "bag_of_words",
        "inputs": ["target_sequence"],
        "outputs": ["protein"],
        "should_cache": true,
        "vocabulary": [
            "A", "C", "D", "E", "F", "G", "H", "I", "K", "L", "M",
            "N", "P", "Q", "R", "S", "T", "V", "W", "Y", "X"
        ],
        "max_length": 3
    }
],
```

Figure 3.13: An example of how to configure featurizers for protein-ligand affinity prediction

Featurizers have custom arguments which can be configured, but all of them will have at least 3 parameters:

- **inputs**: a list of input targets. These values should match the ones specified in the "input_column_names" option of the loader.

- **outputs**: a list of output targets. These values should match the arguments expected by the network architectures. The list of arguments expected by each model is listed in Table 3.1.
- **should_cache**: whether we should perform sample level caching. Note that this is different from global caching, which is performed after featurization. When this option is set to true, each input will be cached in-memory and will not be processed again. This can be very helpful when we are dealing with many duplicates in a certain column which are expensive to compute, like amino-acid sequences. (defaults to "False")

| Architecture | Dependency Name | Compatible Featurizers |
|---|---|---|
| graph_convolutional | graph | graph |
| message_passing | graph | graph |
| triplet_message_passing | graph | graph |
| linear | features | circular_fingerprint, bag_of_words |
| convolutional | features | token |
| protein_ligand | ligand | graph, circular_fingerprint |
| protein_ligand | protein | token, bag_of_words |

Table 3.1: List of featurizers and what architectures they empower. The featurizer "outputs" option should include the "Dependency Name". The "protein_ligand" architecture has 2 dependencies, both of which have to be satisfied.

All featurizers will have these configurable options, and the "inputs" and "outputs" arguments are mandatory. However, most featurizers will have additional configurable options.

The `graph` featurizer supports:

- **descriptor_calculator**: specify how and what molecule level features to compute. The supported options include "rdkit"[34] and "mordred"[53].
- **allowed_atom_types**: specify which atom types should be considered explicitly for (one-hot) encoding. Other atoms will be grouped as a general "other" feature. The default option includes: B, C, N, O, F, Na, Si, P, S, Cl, K, Br, and I.

In additional to symbol based encoding, the graph featurizer also encodes a one-hot atom degree encoding (from 0 to 10), one-hot implicit valence encoding (form 0 to 6), the formal charge, the number of radical electrons, one-hot encoded hybridization, an aromaticity flag, and the total number of Hydrogen atoms. By default, these amounts to 45 input features. If additional atom types are specified, the input features settings should reflect those changes.

For bond level features, we compute a bond type one-hot encoding (single, double, triple, or aromatic), a conjucation flag, a ring membership flag, a stereo configuration encoding. All these amount to 12 edge features.
The `circular_fingerprint` featurizer supports:

- **fingerprint_size**: the number of bits (defaults to 2048)
- **radius**: which defaults to 2

The `one_hot_encoder` featurizer requires:

- **classes**: a list of all possible tokens (ie: "male", "female")

The `token` featurizer supports:

- **vocabulary**: a list of all possible tokens (ie: "A", "C", "T", "G")
- **max_length**: the length of the largest sequence (smaller sequences will be 0-padded)
- **separator**: a separator for the tokens. If an empty string "" is specified, the string will be split character-by-character (defaults to "")

The `bad_of_words` featurizer supports:

- **vocabulary**: a list of all possible tokens (ie: "A", "C", "T", "G")
- **max_length**: the length of the largest sequence (smaller sequences will be 0-padded)

The `fixed` featurizer requires:

- **value**: the value to divide the input by

### *transformers (no default)*

Are very similar featurizers, expect they are applied to output features, and are generally used to normalize values. Transformers are dynamic fields used to instantiate components extending `kmol.data.transformers.AbstractTransformer` .

Each transformer implements an "apply" and a "reverse" operation, which converts the values back after inference (for prediction only, does not apply to metrics). Similar to featurizers, multiple transformers can be used in the same experiment
The `converter` featurizer requires:

- **source_format**: the input format
- **target_format**: the output format

Both of these options should be https://openbabel.org/wiki/BabelFile$_{F}ormats$.

```
"transformers": [
    {"type": "log_normalize", "targets": [0, 2]},
    {"type": "standardize", "target": 1, "mean": 2.1863357142857143, "std": 1.203003713387104}
],
```

Figure 3.14: An example of how to 2 transformers

As a dynamic field, the transformer type is again specified with the "type" argument:

- **log_normalize**: Converts the target to its log value. This transformer expects a `targets` argument, which is a list of (0-based) indices mapping to the "target_column_names" argument of the loader.
- **min_max_normalize**: performs min-max normalization. This transformer has a `target` argument similar to the log transformer, however, in this case, we expect a single value instead of a list. The "minimum" and "maximum" arguments are also mandatory, which should reflect the smallest and largest values recorded for the column/property.

- **fixed_normalize**: Divides the target values by a fixed number. This transformer expects a `targets` argument, which is a list in this case. The "value" argument is used to specify the value used for the division.
- **standardize**: performs z-score normaliztion. This transformer has a `target` argument which is a single 0-based index (not a list). The "mean" and "std" options are also mandatory which denote the average value and the standard deviation for the target.
- **cutoff**: converts a continuous value to a discrete one, using a user-specified cutoff. Note that this operation is destructive and cannot be reversed. We expect a `target` argument with a single 0-based index, and a "cutoff" argument denoting the threshold for binarization.

### *splitter (no default)*

The last of the major abstraction based dynamic settings, the "splitter" specifies how to split the dataset. These options are used to instantiate components extending `kmol.data.splitters.AbstractSplitter` .

```
"splitter": {
    "type": "stratified",
    "seed": 42,
    "target_name": "label",
    "splits": {"train": 0.8, "test": 0.2}
},
```

Figure 3.15: An example configuration for a stratified splitter

As a dynamic option, splitter classes are specified using the "type" argument. We have the following splitter types:

- **index**: for index-based splitting
- **random**: for random splits
- **stratified**: for stratified splits based on a certain output column/property

- **descriptor**: for splits based on a molecular descriptor (ie: molecular weight)
- **scaffold_balancer**: for scaffold based splits where the goal is to keep an equal number of scaffolds between splits
- **scaffold_divider**: for splits where the goal is to have unique scaffolds in each split
- **butina_balancer**: for butina based splits where the goal is to keep an equal number of scaffolds between splits
- **butina_divider**: for splits where the goal is to have unique clusters in each split

All splits require a `splits` argument in a dictionary (key-value) format. The keys denote the name of the split and can be any users-specified value (we generally used "train" and "test"). The values represent the proportions for each split and should add up to 1. Users can specify any number of splits, not only 2.

Besides the "index" spliter, all splitters also require a "seed" argument which is used to set the random state for reproducible splits.

The "stratified" split requires a "target_name" argument, which should be an entry of the loader's "target_column_names" or "input_column_names" argument. If the stratification target is an input column, the "is_target_input" option should also be set to "true" in addition.
If the "target_name" contains continuous values, they can be grouped into a number of bins using quantile-based discretization (ie: equal-sized buckets based on rank). Users can specify the number of bins using the "bins_count" argument.
Butina spliters have a "butina_cutoff" option which specifies the range within which clusters are considered to be neighbors, as well as a "fingerprint_size" and a "radius" option to control the fingerprint bits used for clustering.
The "descriptor" splitter requires a "descriptor" which specifies which descriptor to use. this value should be a member of RdKit's https://www.rdkit.org/docs/source/rdkit.Chem.Descriptors.html.
In addition, it also supports a "bins_count" option similar to the stratified split.

### *criterion (no default)*

The criterion option is used to specify the loss function. It is a dynamic option, however, it is not rely on abstractions.

Users can specify any native Torch or custom written loss functions:

```
"criterion": {
    "type": "torch.nn.BCEWithLogitsLoss"
},
```

We also provide a custom built masked loss function, which wrapps around a regular loss function, but ignores missing labels. Using the masked loss is easy with dependency injection:

```
"criterion": {
    "type": "kmol.model.criterions.MaskedLoss",
    "loss": {"type": "torch.nn.BCEWithLogitsLoss"}
},
```

As a dynamic setting, of course, users can specify any additional input arguments for their loss function. For a list of all criterions, we point readers to the pytorch documentation.

### *optimizer (no default)*

Is a dynamic option used to specify and configure the optimizer. As any other fynamic option, the class is specified with the "type" argument, and everything else is injected into the constructor:

```
"optimizer": {
    "type": "torch.optim.Adam",
    "lr": 0.01,
    "weight_decay": 0.00056
},
```

For a list of native PyTorch optimizers, please see the pytorch documentaiton. Of course, custom optimizers can be used as well. As an example, we provide support for the recent AdaBelief[77] optimizer:

```
"optimizer": {
    "type": "kmol.model.optimizers.AdaBelief",
    "weight_decay": 0,
    "betas": [0.9, 0.999]
},
```

### *scheduler (no default)*

Schedulers are used to adjust the learning rate during training. This is a dynamic option very similar to the criterion or the optimizer.

For a list of native pytorch learning rate schedulers, please visit the pytorch documentation.

```
"scheduler": {
    "type": "torch.optim.lr_scheduler.OneCycleLR",
    "max_lr": 0.01,
    "epochs": 200,
    "pct_start": 0.3,
    "div_factor": 25,
    "final_div_factor": 1000
},
```

### *is_stepwise_scheduler (default: True)*

A static field. If true, we perform scheduler updates after every forward pass. Otherwise, we perform the update after every epoch.

### *is_finetuning (default: False)*

A static field. This should remain "False" if one wishes to continue training after a reboot, and be set to "True" when we wish to train on a new dataset using a previous checkpoint. When this option is "True", only model weights will be loaded, otherwise we also load the optimizer, scheduler, and epoch tracking information.

### *output_path (no default)*

Points to the location where checkpoints will be saved to.

### *checkpoint_path (default: None)*

A static field marking the path to a checkpoint. Some operations like single checkpoint evaluation or inference require a fixed checkpoint. One can also specify a checkpoint path if they wish to continue training on a previous task, or for fine-tuning on another dataset.

### *threshold (default: 0.5)*

Specifies what threshold to use when converting logits to predictions. This is going to affect certain metrics (like accuracy) and predicted values (when running inference).

Of course, the threshold is used only for classification tasks.

### *cross_validation_folds (default: 5)*

How many folds to split the data into when performing cross-validation?

### *train_split (default: "train")*

Specify which split to use for training. This should be one of the keys specified in the "splits" option of the "splitter".

### *test_split (default: "test")*

Specify which split to use for evaluation and inference. This should be one of the keys specified in the "splits" option of the "splitter".

### *train_metrics (default: [])*

Specify which metrics to report during training. This option is expected to be a list of strings, thus multiple metrics can be specified. But please note that metric computations can slow down the training process.

For a list of supported metrics, please use the shorthand code from Table 1.5.

### *test_metrics (default: [])*

Specify which metrics to report during evaluation. This option is expected to be a list of strings, thus multiple metrics can be specified.

For a list of supported metrics, please use the shorthand code from Table 1.5.

### *epochs (default: 200)*

How many epochs to train the model?

### *batch_size (default: 32)*

What batch size to use (in all operations).

### *use_cuda (default=true)*

If true, and if an NVIDIA graphics card is available, the model will use GPU acceleration.

### *cache_location (default: "/tmp/federated/")*

A folder where cached objects will be stored in. It is recommended to monitor the size of this directory.

### *clear_cache (default: False)*

When set to "True", cached data for the specific experiment will be flushed.

### *log_frequency (default: 20)*

After how many iterations should an update be printed when training the model.

### *log_level (default: "info")*

How many logs should be printed. The available options are: "debug", "info", "warn", "error", and "critical".

### *log_format (default: "")*

Can be used to include additional details for logged messages, like timestamps.

### *target_metric (default: "roc_auc")*

Specify which metric to use as feedback for bayesian optimization. This option is also used when to find best performing checkpoints.

### *optuna_trials (default: 1000)*

How many trials should be perform when running Bayesian Optimization?

### *subset (default: None)*

The subset is a static, but composite setting. It expects a dictionary containing an "id" and a "distribution" field.

The option is used to run experiments on a subset of the whole dataset, and is very helpful for distributed learning experiments. Otherwise, it provides little use, as the specific subsets can be optained using the regular splitter functionality.

The "distribution" setting specifies a list of fractions the dataset should be split into and the expected value is a list of floating point values which sum up to 1. The "id" setting specifies which subset should be used from the list of "distribution"s and is a 0-based index.

As an example, the below setting will split the dataset into 2 equal proportions, and will use the first half of the split for training.

```
"subset": {
    "id": 0,
    "distribution": [0.5, 0.5]
}
```

### *visualizer (default: None)*

Another static, but composite option. This setting is used by the visualization operation.

The setting should contains several subfields:

- **mapping_file_path**: This file will contain a mapping between created images and input molecules (in CSV format)
- **targets**: is a list of target columns which will be explained. The values are 0-based indices mapping to the "target_column_names" option of the loader
- **output_path**: a folder where visualized drawings will be saved to

```
"visualizer": {
    "mapping_file_path": "data/visualizations/mapping.csv",
    "targets": [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11],
    "sketcher": {"type": "rdkit", "output_path": "data/visualizations/img/"}
}
```

### *differential_privacy (default: {"enabled": False})*

A static, but composite option used to control differential privacy.

The "enabled" option is a boolean value acting as a switch for differential privacy. The "options" argument is a dictionary of configurable options:

- **delta**: The target delta used when reporting the computed (epsilon, delta)-privacy budget spent so far.
- **alphas**: instructs the privacy engine what Renyi Differential Privacy[43, 44] orders to use for tracking privacy expenditure. (the default value is set to [1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 2.0, 2.1, 2.2, 2.3, 2.4, 2.5, 2.6, 2.7, 2.8, 2.9, 3.0, 3.1, 3.2, 3.3, 3.4, 3.5, 3.6, 3.7, 3.8, 3.9, 4.0, 4.1, 4.2, 4.3, 4.4, 4.5, 4.6, 4.7, 4.8, 4.9, 5.0, 5.1, 5.2, 5.3, 5.4, 5.5, 5.6, 5.7, 5.8, 5.9, 6.0, 6.1, 6.2, 6.3, 6.4, 6.5, 6.6, 6.7, 6.8, 6.9, 7.0, 7.1, 7.2, 7.3, 7.4, 7.5, 7.6, 7.7, 7.8, 7.9, 8.0, 8.1, 8.2, 8.3, 8.4, 8.5, 8.6, 8.7, 8.8, 8.9, 9.0, 9.1, 9.2, 9.3, 9.4, 9.5, 9.6, 9.7, 9.8, 9.9, 10.0, 10.1, 10.2, 10.3, 10.4, 10.5, 10.6, 10.7, 10.8, 10.9,

12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63]).

- **noise_multiplier**: The ratio of the standard deviation of the Gaussian noise to the L2-sensitivity of the function to which the noise is added.
- **max_grad_norm**: The maximum norm of the per-sample gradients. Any gradient with norm higher than this will be clipped to this value.
- **secure_rng**: If on, it will use torchcsprng for secure random number generation. This comes with a significant performance cost, therefore it is recommended to turn off when experimenting.

Differential privacy was implemented using Opacus[50].

```
"differential_privacy": {
    "enabled": true,
    "options": {
        "delta": 1e-5,
        "alphas": [10, 100],
        "noise_multiplier": 1.5,
        "max_grad_norm": 1.0,
        "secure_rng": false
    }
}
```

**Caution!** Not all layer types support differential privacy. Of especial note is that BatchNorm layers are not supported. As a workaround, we dynamically replace BatchNorm layers with GroupNorm[72] layers, which can affect the training results.

### *observers (default: {})*

Observers are handlers which listen for and handle incoming events. We set up various event dispatchers throughout the framework like before a checkpoint is loaded, before training starts, or after training finished. A full list of event dispatchers and their payload is listed in Table 3.2.

Event listeners watch the dispatchers and act on the payload. Features like differential privacy are implemented entirely with observers. As for user specified

handlers, the `add_sigmoid` handler is one of the only useful ones, which adds an additional sigmoid activation to the output of a model. It should be attached to "before_criterion", and "before_predict" dispatchers.

```
"observers": {
    "before_criterion": ["kmol.core.observers.AddSigmoidHandler"],
    "after_predict": ["kmol.core.observers.AddSigmoidHandler"]
},
```

### 3.4.2 Bayesian Optimization

To perform Bayesian Optimization, an experiment configuration file (in JSON format) must be first parameterized. An example of how this looks like is presented below.

```
"model": {
    "type": "graph_convolutional",
    "in_features": 45,
    "out_features": 1,
    "hidden_features": "{{{hidden=32-256-16}}}",
    "dropout": "{{{dropout=0.0-0.7-0.1}}}",
    "layer_type": "{{{layer_type=torch_geometric.nn.GCNConv|torch_geometric.nn.GraphConv|torch_geometric.nn.GENConv}}}",
    "layers_count": "{{{layers_count=2-8-1}}}",
    "molecule_features": 17,
    "is_residual": "{{{is_residual=0-1-1}}}",
    "norm_layer": "lib.model.layers.BatchNorm"
},
```

Figure 3.16: An example configuration with parameterized options.

Parameterized options are wrapped in triple curly brackets and must be enclosed within quotes "{{{}}}". These options are also referenced in the code as "placeholders". We use a parsing mechanism before we pass the options to Optuna[2] for suggestion generation:

- Each placeholder should have a name and some options, separated by an equal sign (ie: **"{{{name=options}}}"**)
- Options separated by a pipe "|" will be categorical (ie: **"{{{aggregate=mean|sum|max}}}"**)
- Numeric values should have 3 options separated by a dash "-" (ie: **"{{{dropout=min-max-step**}}}")

  - The first value is the minimum value
  - The second value is the maximum value
  - The the third value is the incremental step between the minimum and the maximum
  - The [minimum, maximum] is a closed interval

- if numeric values contain a dot ".", a float value will be suggested (ie: **"{{{dropout=0.0-0.7-0.1}}}"**)
- if numeric values do not contain a dot ".", an integer will be suggested (ie: **"{{{layers=2-5-1}}}**")

We present an example for the model arguments, but any option from the configuration file can be parameterized.

### 3.4.3 Commands

#### *Training*

To train a model, the following command can be used:

```
kmol train {config_path}
```

```
(federated) elix@bash:/var/www/federated$ python run.py train data/configs/model/tox21.json
epoch: 1 - iteration: 5 - examples: 640 - loss: 0.7674 - time elapsed: 0:00:01 - progress: 0.1024
epoch: 1 - iteration: 10 - examples: 1280 - loss: 0.6921 - time elapsed: 0:00:02 - progress: 0.2048
epoch: 1 - iteration: 15 - examples: 1920 - loss: 0.6165 - time elapsed: 0:00:03 - progress: 0.3072
epoch: 1 - iteration: 20 - examples: 2560 - loss: 0.5576 - time elapsed: 0:00:04 - progress: 0.4097
epoch: 1 - iteration: 25 - examples: 3200 - loss: 0.5036 - time elapsed: 0:00:05 - progress: 0.5121
epoch: 1 - iteration: 30 - examples: 3840 - loss: 0.4668 - time elapsed: 0:00:05 - progress: 0.6145
epoch: 1 - iteration: 35 - examples: 4480 - loss: 0.4268 - time elapsed: 0:00:06 - progress: 0.7169
epoch: 1 - iteration: 40 - examples: 5120 - loss: 0.3928 - time elapsed: 0:00:07 - progress: 0.8193
epoch: 1 - iteration: 45 - examples: 5760 - loss: 0.3680 - time elapsed: 0:00:08 - progress: 0.9217
Saving checkpoint: data/logs/tox21/checkpoint.1
epoch: 2 - iteration: 5 - examples: 640 - loss: 0.2817 - time elapsed: 0:00:10 - progress: 0.1024
epoch: 2 - iteration: 10 - examples: 1280 - loss: 0.2800 - time elapsed: 0:00:10 - progress: 0.2048
epoch: 2 - iteration: 15 - examples: 1920 - loss: 0.2754 - time elapsed: 0:00:11 - progress: 0.3072
epoch: 2 - iteration: 20 - examples: 2560 - loss: 0.2719 - time elapsed: 0:00:12 - progress: 0.4097
epoch: 2 - iteration: 25 - examples: 3200 - loss: 0.2706 - time elapsed: 0:00:13 - progress: 0.5121
epoch: 2 - iteration: 30 - examples: 3840 - loss: 0.2696 - time elapsed: 0:00:14 - progress: 0.6145
epoch: 2 - iteration: 35 - examples: 4480 - loss: 0.2690 - time elapsed: 0:00:15 - progress: 0.7169
epoch: 2 - iteration: 40 - examples: 5120 - loss: 0.2689 - time elapsed: 0:00:16 - progress: 0.8193
epoch: 2 - iteration: 45 - examples: 5760 - loss: 0.2688 - time elapsed: 0:00:17 - progress: 0.9217
Saving checkpoint: data/logs/tox21/checkpoint.2
epoch: 3 - iteration: 5 - examples: 640 - loss: 0.2091 - time elapsed: 0:00:18 - progress: 0.1024
```

Figure 3.17: Sample output of the training command.

## *Evaluating a Single Checkpoint*

To evaluate the performance of the "checkpoint_path" specified in the configuration file, use the following command:

```
kmol eval {config_path}
```

```
(federated) elix@bash:/var/www/federated$ python run.py eval data/configs/model/other/linear.json
Restoring from Checkpoint: data/logs/tox21/checkpoint.200
--------------------------------------------------------------------
metric,NR-AR,NR-AR-LBD,NR-AhR,NR-Aromatase,NR-ER,NR-ER-LBD,NR-PPAR-gamma,SR-ARE,SR-ATAD5,SR-HSE,SR-MMP,SR-p53
--------------------------------------------------------------------
roc_auc,0.66490238336714,0.8046915435345187,0.7940588403313344,0.7377669039145909,0.6385818932192737,0.7019674202427688,0.689506407816267,0.708024319596203,0.7372072372072371,0.6780777074165503,0.8373150928706484,0.6867292717976138
pr_auc,0.37873271387440444,0.3972700503580749,0.48305480191682 53,0.3738996573753368,0.33409047049328355,0.3675339220814409,0.12729722169658608,0.3839939463075163,0.3023469394338735,0.2253023865348695,0.5937794425794202,0.2410230721019888 7
accuracy,0.95,0.9651162790697675,0.8783068783068783,0.9494097807757167,0.8199052132701422,0.9407616361071932,0.9642585551330799,0.8115318416523236,0.9554937413073713,0.9230769230769231,0.8625954198473282,0.9255474452554745
precision,0.45098039215686275,0.4651162790697674 4,0.48226950354609927,0.5227272727272727,0.3163841807909605,0.4915254237288136,0.2222222222222222,0.39285714285714285,0.3695652173913043,0.31666666666666665,0.5459183673469388,0.2857142857142857
recall,0.3382352941176471,0.4444444444444444,0.4358974358974359,0.3709677419354839,0.34355828220858897,0.3493975903614458,0.10810810810810811,0.4350282485875706,0.3269230769230769,0.24675324675324675,0.5944444444444444,0.3188405797101449
cohen_kappa,0.3610464976859069,0.43653705062362436,0.3895695236130096 7,0.40828205554631636,0.22560082417140193,0.3782065336549002,0.12942121054188438,0.3009652732377034,0.3239900102835317,0.23740865099743236,0.487589337454124,0.2621730361046285
--------------------------------------------------------------------
metric,amin,amax,mean,median,std
--------------------------------------------------------------------
roc_auc,0.6385818932192737,0.8373150928706484,0.7232357517761788,0.7049958699194859,0.05833369285421122
pr_auc,0.12729722169658608,0.5937794425794202,0.3506937187294683 6,0.37071678972838884,0.11610569513775663
accuracy,0.8115318416523236,0.9651162790697675,0.9121669761501833,0.9331545406813339,0.052892943077222 29
precision,0.2222222222222222,0.5459183673469388,0.4051622461848614,0.4219187675070028,0.0989941839147882
recall,0.10810810810810811,0.5944444444444444,0.3593832077909698,0.3464779362850174,0.11310740596102865
cohen_kappa,0.12942121054188438,0.487589337454124,0.3283991669928719 7,0.3425182539847193,0.09759695294028381
(federated) elix@bash:/var/www/federated$
```

Figure 3.18: Sample output of the evaluation command.

## *Evaluating Multiple Checkpoints*

To evaluate all checkpoints found in the "output_path" folder specified in the configuration file, use the following command:

```
kmol analyze {config_path}
```

```
(federated) elix@bash:/var/www/federated$ python run.py analyze data/configs/model/other/linear.json
Restoring from Checkpoint: data/logs/tox21/checkpoint.191
Restoring from Checkpoint: data/logs/tox21/checkpoint.192
Restoring from Checkpoint: data/logs/tox21/checkpoint.193
Restoring from Checkpoint: data/logs/tox21/checkpoint.194
Restoring from Checkpoint: data/logs/tox21/checkpoint.195
Restoring from Checkpoint: data/logs/tox21/checkpoint.196
Restoring from Checkpoint: data/logs/tox21/checkpoint.197
Restoring from Checkpoint: data/logs/tox21/checkpoint.198
Restoring from Checkpoint: data/logs/tox21/checkpoint.199
Restoring from Checkpoint: data/logs/tox21/checkpoint.200
----------------------------------------------------------------
metric,NR-AR,NR-AR-LBD,NR-AhR,NR-Aromatase,NR-ER,NR-ER-LBD,NR-PPAR-gamma,SR-ARE,SR-ATAD5,SR-HSE,SR-MMP,SR-p53,[0]
----------------------------------------------------------------
roc_auc,0.6647544793779581,0.8046414558811253,0.7940423615230813,0.7377812535874182,0.6385373966149208,0.7020035196967647,0.689506407816267,0.7080874128882388,0.7371933621
933622,0.678238703445315,0.8372650428205982,0.6866624335795207
pr_auc,0.37865553650968353,0.39726248056104047,0.4830194672774002,0.37386773542844487,0.3341358378687654,0.3675764142426596,0.1407950405740639,0.3840026516288543,0.3024808
3306023066,0.22527485187235421,0.5936620071741494,0.24100685624874538
accuracy,0.95,0.9651162790697675,0.8783068783068783,0.9494097807757167,0.8199052132701422,0.9407616361071932,0.9642585551330799,0.8115318416523236,0.9554937413073713,0.923
0769230769231,0.8625954198473282,0.9262773722627737
precision,0.45098039215686275,0.46511627906976744,0.48226950354609927,0.5227272727272727,0.3163841807909605,0.4915254237288136,0.2222222222222222,0.39285714285714285,0.369
```

Figure 3.19: Sample output of the analyze command.

## *Inference*

To run inference using the "checkpoint_path" specified in the configuration file, the following command can be used:

```
kmol predict {config_path}
```

```
Restoring from Checkpoint: data/logs/tox21/checkpoint.200redict data/configs/model/other/linear.json
NR-AR,NR-AR-LBD,NR-AhR,NR-Aromatase,NR-ER,NR-ER-LBD,NR-PPAR-gamma,SR-ARE,SR-ATAD5,SR-HSE,SR-MMP,SR-p53
0,0,0,0,0,0,0,0,0,1,0
0,0,0,0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0,0,0,0
0,0,1,0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0,0,0,0
0,0,0,0,0,1,0,1,0,0,0
0,0,0,0,0,0,1,0,0,0,0
0,0,0,0,0,0,0,0,0,0,0
0,0,0,0,0,0,1,1,0,1,0
0,0,0,0,0,0,0,0,0,0,0
0,0,0,0,1,0,0,0,0,0,0
0,0,0,0,1,0,0,1,0,0,1
0,0,0,0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0,0,0,0
0,0,0,0,0,0,1,0,0,0,0
0,0,0,0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0,0,0,0
0,0,0,0,0,0,1,0,0,1,0
0,0,0,0,0,0,0,0,0,0,0
```

Figure 3.20: Sample output of the inference command.

### *Cross-Validation*

We support 3 different ways to perform cross-validation:

- **mean cross-validation**: for each fold we train, then evaluate the fold. We store the best performing metrics and perform fold-wise aggregation of the metrics.
- **full cross-validation**: for each fold we train and retrieve the logits for the fold. These are stored and metrics are computed on the whole dataset after each fold finished training.
- **step cross-validation**: we train each fold one epoch at a time. After an epoch is trained on for each fold, we perform full cross-validation on for the last checkpoints only.

```
kmol mean_cv {config_path}
kmol full_cv {config_path}
kmol step_cv {config_path}
```

```
Restoring from Checkpoint: data/logs/server//.4/checkpoint.223
Restoring from Checkpoint: data/logs/server//.4/checkpoint.224
Restoring from Checkpoint: data/logs/server//.4/checkpoint.225
Restoring from Checkpoint: data/logs/server//.4/checkpoint.226
Restoring from Checkpoint: data/logs/server//.4/checkpoint.227
Restoring from Checkpoint: data/logs/server//.4/checkpoint.228
Restoring from Checkpoint: data/logs/server//.4/checkpoint.229
Restoring from Checkpoint: data/logs/server//.4/checkpoint.230
-------------------------------------------------------------
metric,value
-------------------------------------------------------------
r2,0.5852±0.1624
mae,-0.2396±0.0483
rmse,-0.3553±0.1466
-------------------------------------------------------------
metric,amin,amax,mean,median
-------------------------------------------------------------
r2,0.5852±0.1624,0.5852±0.1624,0.5852±0.1624,0.5852±0.1624
mae,-0.2396±0.0483,-0.2396±0.0483,-0.2396±0.0483,-0.2396±0.0483
rmse,-0.3553±0.1466,-0.3553±0.1466,-0.3553±0.1466,-0.3553±0.1466
```

Figure 3.21: Sample output of mean cross-validation command.

### *Bayesian Optimization*

Bayesian optimization requires a parameterized configuration file as input. Otherwise, we use the "optimize" command.

```
kmol optimize {config_path}
```

```
(federated) elix@bash:/var/www/federated$ python run.py optimize data/configs/bo/caco2_wang.json
[I 2021-03-14 20:37:38,376] A new study created in memory with name: no-name-000b2284-7b3b-40a6-af0a-3dd2fc8e5e25
epoch: 1 - iteration: 5 - examples: 160 - loss: 0.7346 - time elapsed: 0:00:00 - progress: 0.2198
epoch: 1 - iteration: 10 - examples: 320 - loss: 0.7203 - time elapsed: 0:00:00 - progress: 0.4396
epoch: 1 - iteration: 15 - examples: 480 - loss: 0.7208 - time elapsed: 0:00:01 - progress: 0.6593
epoch: 1 - iteration: 20 - examples: 640 - loss: 0.6688 - time elapsed: 0:00:01 - progress: 0.8791
Saving checkpoint: data/bo/caco2_wang//0/checkpoint.1
epoch: 2 - iteration: 5 - examples: 160 - loss: 0.4818 - time elapsed: 0:00:01 - progress: 0.2198
epoch: 2 - iteration: 10 - examples: 320 - loss: 0.4654 - time elapsed: 0:00:01 - progress: 0.4396
epoch: 2 - iteration: 15 - examples: 480 - loss: 0.4644 - time elapsed: 0:00:01 - progress: 0.6593
epoch: 2 - iteration: 20 - examples: 640 - loss: 0.4516 - time elapsed: 0:00:01 - progress: 0.8791
Saving checkpoint: data/bo/caco2_wang//0/checkpoint.2
epoch: 3 - iteration: 5 - examples: 160 - loss: 0.2267 - time elapsed: 0:00:02 - progress: 0.2198
epoch: 3 - iteration: 10 - examples: 320 - loss: 0.2527 - time elapsed: 0:00:02 - progress: 0.4396
epoch: 3 - iteration: 15 - examples: 480 - loss: 0.2705 - time elapsed: 0:00:02 - progress: 0.6593
```

Figure 3.22: Sample output of the bayesian optimization command.

## *Finding the Best Checkpoint*

This functionality relies on the "analyze" command. It will evaluate all checkpoints and return the best one according to the "targe_metric" specified.

```
kmol find_best_checkpoint {config_path}
```

```
-----------------------------------------------------------------
Best checkpoint: data/logs/tox21/checkpoint.191
Restoring from Checkpoint: data/logs/tox21/checkpoint.191
-----------------------------------------------------------------
metric,NR-AR,NR-AR-LBD,NR-AhR,NR-Aromatase,NR-ER,NR-ER-LBD,NR-PPAR-gamma,SR-ARE,SR-ATAD5,SR-HSE,SR-MMP,SR-p53
-----------------------------------------------------------------
roc_auc,0.6647544793779581,0.8046414558811253,0.7940423615230813,0.7377812535874182,0.6385373966149208,0.7020035196967647,0.689506407816267,0.7080874128882388,0.7371933621
933622,0.678238703445315,0.8372650428205982,0.6866624335795207
pr_auc,0.37865553650968353,0.39726248056104047,0.4830194672774002,0.37386773542844487,0.3341358378687654,0.3675764142426596,0.1407950405740639,0.3840026516288543,0.3024808
3306023066,0.22527485187235421,0.5936620071741494,0.24100685624874538
accuracy,0.95,0.9651162790697675,0.8783068783068783,0.9494097807757167,0.8199052132701422,0.9407616361071932,0.9642585551330799,0.8115318416523236,0.9554937413073713,0.923
0769230769231,0.8625954198473282,0.9262773722627737
precision,0.45098039215686275,0.46511627906976744,0.48226950354609927,0.5227272727272727,0.3163841807909605,0.4915254237288136,0.2222222222222222,0.39285714285714285,0.369
5652173913043,0.31666666666666665,0.5459183673469388,0.2894736842105263
recall,0.33823529411764705,0.4444444444444444,0.43589743589743589,0.3709677419354839,0.34355828220858897,0.3493975903614458,0.10810810810810811,0.4350282485875706,0.32692307
69230769,0.24675324675324675,0.5944444444444444,0.31884057971014493
cohen_kappa,0.36104649768590690,0.43653705062362436,0.38956952361300967,0.40828205554631636,0.22560082417140193,0.37820653365490002,0.12942121054188438,0.30096527323770340,0.
3239900102835317,0.23740865099743236,0.48758933745412 4,0.2646230375952636
-----------------------------------------------------------------
metric,amin,amax,mean,median,std
-----------------------------------------------------------------
```

Figure 3.23: Sample output of the best checkpoint finder command.

## *Finding the Best Threshold*

When traning a classifier, different thresholds will yield different predictions. If the dataset is imbalanced, some metrics like "accuracy" become less trustworthy.

One way to get more reliable results is to plot a precision-recall curve and pick

the threshold where the 2 lines intersect. This threshold can be calculated for a certain "checkpoint_path" using the following command:

```
kmol find_threshold {config_path}
```

```
(federated) elix@bash:/var/www/federated$ python run.py find_threshold data/configs/model/other/linear.json
Restoring from Checkpoint: data/logs/tox21/checkpoint.200
Best Thresholds: [0.15974624454975128, 0.5004438161849976, 0.15607455372810364, 0.09827805310487747, 0.14083275198936462, 0.18876782059669495, 0.3295873701572418, 0.33261004090309143, 0.26095089316368103, 0.24408897757530212, 0.3303869664669037, 0.9938067197799683]
Average: 0.3112978506833315
(federated) elix@bash:/var/www/federated$
```

Figure 3.24: Sample output of the best threshold finder command.

## *Finding the Best Learning Rate*

The following command will use an exponential learning rate scheduler to analyze various learning rates and plots a chart to help decide on a good one.

```
kmol find_learning_rate {config_path}
```

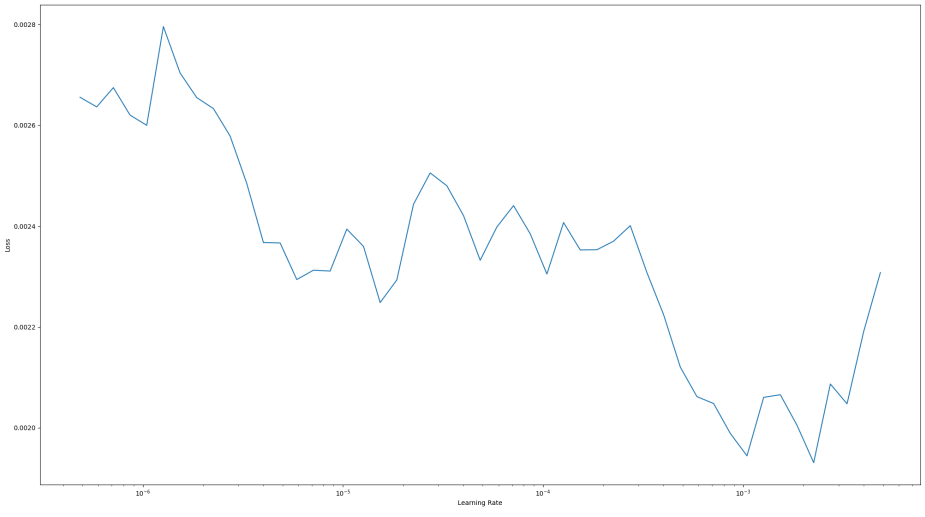


Figure 3.25: Sample output of the best learning rate finder command.

### *Visualizing Graphs*

The following command uses model interpretability and understanding tools like Integrated Gradients[60] to plot graphs for molecules in the test set. Analysis will be performed on the "checkpoint_path" specified.

Implementations rely on the Captum library.

```
kmol visualize {config_path}
```

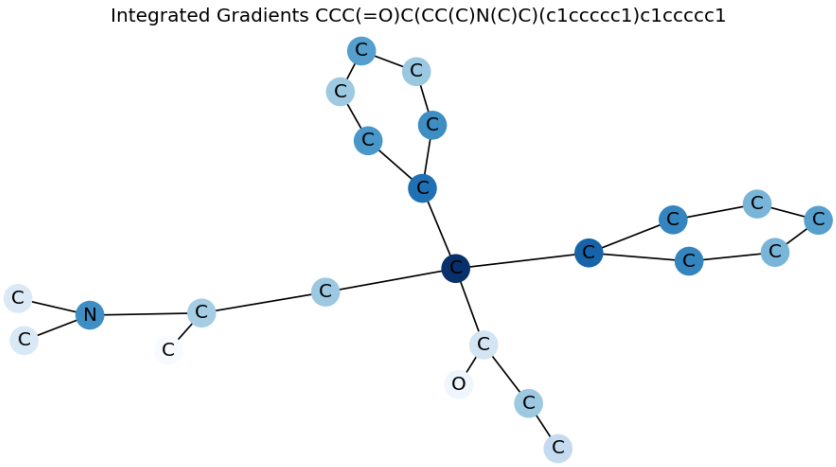


Figure 3.26: Sample output of the visualize command.

### *Preloading Datasets*

Dataset featurization can take a long time. To preload and cached a dataset, one may use the "preload" command and perform training at a later time.

```
kmol preload {config_path}
```

### *View Split Indices*

The sample indices for each split can be visualized with the "splits" command.

```
kmol splits {config_path}
```

### 3.4.4 Customization

Extending the core functionality is can be easy and clean with the provided features. Dependency injection and observers allow anyone to add custom functionality without them having to touch any of the original code.

Indeed, we would recommend developers try to avoid modifying the original files as much as possible, unless absolutely necessary. Keeping custom code decoupled has several advantages. For one, it makes upgrading to a never version much easier, as the original files do not change. But more importantly it helps keep the manageability and maintainability of the library.

In what follows we provide a few examples on how one might add custom functionality, should they desire to do so.

#### *Making Good use of Dependency Injection*

Dependency injection helps with maintainability a lot, because our functionality can be anywhere. For example, maybe our dataset contains InChI[24] strings instead of SMILES[69]. One could dive into the code and start modifying all the featurizers, but there is a much easier and cleaner way. We write our own featurizer for it

We will be adding all our code to a folder called "custom/". Since we are writing a featurizer, we should override the associated abstract class to make sure we follow the expected contract. We see that the abstract class requires us to implement a single method called "$_process$".

```
# in custom/featurizers.py
from kmol.data.featurizers import AbstractFeaturizer

class InchiFeaturizer(AbstractFeaturizer):

    def _process(self, data: Any) -> Any:
        raise NotImplementedError
```

We will be connecting our featurizer directly to the loader, so we expect the input to be a string. However, we could easily connect multiple featurizers and use the output of one as input to the next one. To keep things simple, we will convert our InChI strings to SMILES, as most featurizers are already using SMILES.

```
# in custom/featurizers.py

from kmol.data.featurizers import AbstractFeaturizer
from rdkit import Chem

class InchiFeaturizer(AbstractFeaturizer):

    def _process(self, data: str) -> str:
        return Chem.MolToSmiles(Chem.MolFromInchi(data))
```

We don't even have to worry about exception handling, as problematic entries are filtered at the Streamer level. With this featurizer in place, we can now easily convert all InChI inputs and connect the output to other featurizers.

```
"featurizers": [
    {
        "type": "custom.featurizers.InchiFeaturizer",
        "inputs": ["inchi"],
        "outputs": ["our_custom_generated_smiles"]
    },
    {
        "type": "graph",
        "inputs": ["our_custom_generated_smiles"],
        "outputs": ["graph"],
        "descriptor_calculator": {"type": "rdkit"}
    }
],
```

### *Use Inheritance*

In the previous section we created a general solution which can be plug in to every other featurizer, however, we don't always have to think big. The same thing can be achieved by overriding one of the implemented featurizers:

```
# in custom/featurizers.py

from kmol.data.featurizers import GraphFeaturizer

class InchiGraphFeaturizer(GraphFeaturizer):
```

```
    def _process(self, data: str):
        data = Chem.MolToSmiles(Chem.MolFromInchi(data))
        return super()._process(data)
```

When we pass the "GraphFeaturizer" as argument to the "InchiGraphFeaturizer", we are basically saying "start this class as a copy of GraphFeaturizer". We then alter only the "_process" method, while everything else stays the same. And even in the "_process", all we do is to convert the InChI strings to SMILES, then we pass the input to the parent object to do its job as usual.

All that is left is to use our custom featurizer.

```
"featurizers": [
    {
        "type": "custom.featurizers.InchiGraphFeaturizer",
        "inputs": ["inchi"],
        "outputs": ["graph"],
        "descriptor_calculator": {"type": "rdkit"}
    }
],
```

### *Spying on Pipelines with Observers*

Observers are handlers which listen for and handle incoming events. We set up various event dispatchers throughout the framework which are summarized in Table 3.2 below.

| Event Name | Payload Content |
|---|---|
| before_checkpoint_load | executor, loaded checkpoint |
| after_checkpoint_load | executor |
| after_network_create | executor, config |
| before_train_start | executor, data iterator |
| before_criterion | ground truth, logits |
| after_train_end | executor, data iterator |
| before_checkpoint_save | content to be saved |
| before_train_progress_log | logging information |
| after_predict | ground truth, logits, input features |

Table 3.2: List of event dispatchers and the payload they transmit.

Event listeners watch dispatchers and act on their payload. Some of the core functionality is implemented with observers, however, someone can write custom event handlers and listen in on any of the dispatchers listed above.

Maybe we would like to modify a checkpoint before it is loaded, or we might wish to analyze the logits before they are passed to the loss function. For this example, we will show how one might go about logging the training process to an additional medium like Tensorboard, without having to touch any of the core files at all.

First, we extend from the abstract event handler to make sure we are implementing all expected methods. Again, it looks like we only have one named "run".

```
# in custom/observers.py

from kmol.core.observers import EventHandler

class TensorboardLogHandler(EventHandler):

    def run(self, payload: Namespace):
        raise NotImplementedError
```

Next, we try to identify a dispatcher which contains the information we need. It would seem like "before_train_progress_log" is our best fit. We also see that it transmits a "message" argument in its payload, which is the string printed in the console. Not the best format, but we can work with it.

```
# in custom/observers.py

from kmol.core.observers import EventHandler
import re

class TensorboardLogHandler(EventHandler):

    def _extract_information(self, message: str, desired_metric: str) -> float:
        regular_expression = "{}:\ ([0-9\.]*)".format(desired_metric)
        metric = re.findall(regular_expression, message)

        return float(metric[0])

    def run(self, payload: Namespace):
        message = payload.message

        roc_auc = self._extract_information(message, "roc_auc")
```

```
        loss = self._extract_information(message, "loss")

        self._print_to_tensorboard(roc_auc, loss, payload.epoch, payload.iteration)

    def _print_to_tensorboard(
            self, roc_auc: float, loss: float, epoch: float, iteration: float
    ) -> None:
        raise NotImplementedError
```

With the desired data extracted, logging the values to any medium is a trivial task. All we have to do now is to plug in the event handler in the configuration file:

```
"observers": {
    "before_train_progress_log": ["custom.observers.TensorboardLogHandler"],
}
```

This of course will print updates after every few iterations, based on the configured log frequency. If we wanted to evaluate the model after each epoch, the "before_checkpoint_save" event might have been a better choice.

## 3.5 Federated Learning (Mila)

Server and client processes are started similar to how models work. Each servicer and consumer requires a configuration file, which is fed to the entry point. In what follows, we describe the options for servers and clients separately.

### 3.5.1 Server Configurations

Server configuration files are expected to be in JSON format. The available options include:

#### *task_configuration_file (no default)*

Path to the model configuration file. This will be sent to each client along with the checkpoint.

#### *config_type: (no default)*

Module path to the configuration class (ie: "kmol.config.Config")

### *executor_type: (no default)*

Module path to the executor class (ie: "run.Executor")

### *aggregator_type (no default)*

Which aggregator to use? Options include:

- **mila.aggregators.PlainTorchAggregator**
- **mila.aggregators.WeightedTorchAggregator**
- **mila.aggregators.BenchmarkedTorchAggregator**

### *aggregator_options (default={})*

Input parameters for the aggregator (the expected value is a dictionary).

The "PlainTorchAggregator" does not expect any additional options.

The "WeightedTorchAggregator" expects a "weights" options which is a mapping between the client's "name" parameter and the expected weight. For example, if we have 2 clients named "tester1" and "tester2", this option could be set as follows:

```
"aggregator_options": {
    "weights": {
        "tester1": 0.67,
        "tester2": 0.33
    }
}
```

The "BenchmarkedTorchAggregator" needs additional options to determine how checkpoints are evaluated.

- **config_type**: specifies the configuration parser to use (which should extend "AbstractConfiguration")
- **config_path**: is the path to the actual JSON configuration
- **executor_type**: specifies which executor to use (which should extend "AbstractExecutor")

- **target_metric**: specifies which metric to use for checkpoint evaluation
- **minimized_metric**: should be set to "true" if the "target_metric" should be minimized instead of maximized (ie: "mae", "mse", or "rmse")

```
"aggregator_options": {
    "config_type": "kmol.core.config.Config",
    "config_path": "data/configs/model/tox21.json",
    "executor_type": "run.Executor",
    "target_metric": "roc_auc"
}
```

### *target (default="localhost:8024")*

The gRPC service location URL (that is, the server address).

### *rounds_count (default=10)*

How many rounds to perform

### *save_path (default="data/logs/server/")*

Where to checkpoints received from clients and the aggregate models?

### *start_point (default=null)*

Optionally, specify a checkpoint for the first round. If nothing is specified, clients will start training from scratch.

### *workers (default=2)*

Maximum number of processes handling client requests

### *minimum_clients (default=2)*

Minimum number of clients required to start federated learning. The server won't start the first round until this number is reached.

### *maximum_clients (default=100)*

Maximum number of clients allowed to join the federated learning process.

### *client_wait_time (default=10)*

Once the "minimum_clients" number is reach, the server will wait this many seconds for additional clients before the process starts. After this time expires, no new members will be allowed to join.

### *heartbeat_timeout (default=300)*

How long to wait for a keepalive signal from clients before declaring them "dead"?

### *use_secure_connection (default=false)*

When true, the communication will be performed through HTTPS protocol. The 3 SSL files specified below must be valid for this to work.

### *ssl_private_key (default="data/certificates/server.key")*

gRPC secure communication private key

### *ssl_cert (default="data/certificates/server.crt")*

gRPC secure communication SSL certificate

### *ssl_root_cert (default="data/certificates/rootCA.pem")*

gRPC secure communication trusted root certificate

### *options*

Additional gRPC options. "grpc.max_send_message_length" represents the maximum length of a sent message, and "grpc.max_receive_message_length" the maximum length of a received message. Since we are using peer-to-peer

connections, we also disable host name checks of the certificates with "ssl_target_name_override".

The default value for this option is:

```
[
    ("grpc.max_send_message_length", 1000000000),
    ("grpc.max_receive_message_length", 1000000000),
    ("grpc.ssl_target_name_override", "localhost")
]
```

### *blacklist (default=[])*

A list of IP addresses which will be declined upon authentication.

### *whitelist (default=[])*

A list of IP addresses which will be allowed to join the federated learning process. If "use_whitelist" is True, these will be the only IP addresses allowed to join.

### *use_whitelist (default=false)*

Enables whitelist filtering.

## 3.5.2 Client Configurations

Client configuration files are also expected to be in JSON format. The available options include:

### *name (no default)*

A unique identifier for this client (could be a company name for example)

### *save_path (default="data/logs/client/")*

Where to save checkpoints received from the client?

### *heartbeat_frequency (default=60)*

How often to send keepalive signals to the client?

### *retry_timeout (default=1)*

If a request failes because the server is under heavy load, we retry the connection after this many seconds.

### *model_overwrites*

Used to override model configuration options. Generally, clients might want to change the path where local (model) checkpoints are stored. The default value for this option is:

```
{
    "output_path": "data/logs/local/",
    "epochs": 5
}
```

### *config_type: (no default)*

Module path to the configuration class (ie: "kmol.config.Config")

### *executor_type: (no default)*

Module path to the executor class (ie: "run.Executor")

### *target (default="localhost:8024")*

The gRPC service location URL (that is, the server address).

### *use_secure_connection (default=false)*

When true, the communication will be performed through HTTPS protocol. The 3 SSL files specified below must be valid for this to work.

### *ssl_private_key (default="data/certificates/client.key")*

gRPC secure communication private key

### *ssl_cert (default="data/certificates/client.crt")*

gRPC secure communication SSL certificate

### *ssl_root_cert (default="data/certificates/rootCA.pem")*

gRPC secure communication trusted root certificate

### *options*

Additional gRPC options. "grpc.max_send_message_length" represents the maximum length of a sent message, and "grpc.max_receive_message_length" the maximum length of a received message. Since we are using peer-to-peer connections, we also disable host name checks of the certificates with "ssl_target_name_override".

The default value for this option is:

```
[
    ("grpc.max_send_message_length", 1000000000),
    ("grpc.max_receive_message_length", 1000000000),
    ("grpc.ssl_target_name_override", "localhost")
]
```

## 3.5.3 Creating SSL Certificates

We first need to create the root private key and a self-signed root certificate

```
openssl genrsa -des3 -out rootCA.key 2048

openssl req -x509 -new -nodes -key rootCA.key
-sha256 -days 1024 -out rootCA.pem
```

Next, we create a server private key, a server certificate signing request (CSR), and we sign the CSR using the root certificate:

```
openssl genrsa -out server.key 2048

openssl req -new -key server.key -out server.csr

openssl x509 -req -in server.csr -CA rootCA.pem
-CAkey rootCA.key -CAcreateserial -out server.crt
-days 500 -sha256
```

The client creates the client private key, a client certificate signing request (CSR). A common name should be specified when creating the CSR.

```
openssl genrsa -out client.key 2048
openssl req -new -key client.key -out client3.csr
```

The client sends the CSR to the server which wigns it using the root certificate. The server sends "client.crt" back to the client.

```
openssl x509 -req -in client.csr -CA rootCA.pem
-CAkey rootCA.key -CAcreateserial -out client3.crt
-days 500 -sha256
```

### 3.5.4 Commands

Starting the server can be done using following command:

```
mila server {config_path}
```

Starting a client process is very similar:

```
mila client {config_path}
```

```
(federated) elix@bash:/var/www/federated$ python -m mila.run server data/configs/mila/2/server.json
E1113 18:12:03.679359766   21507 socket_utils_common_posix.cc:223] check for SO_REUSEPORT: {"created":"@1605258723.679350852","description":"SO_REUSEPORT unavailable o
n compiling system","file":"src/core/lib/iomgr/socket_utils_common_posix.cc","file_line":192}
[CAUTION] Connection is insecure!
Starting server at: [localhost:8024]
[tester1|127.0.0.1] Successfully authenticated (clients=1)
[tester2|127.0.0.1] Successfully authenticated (clients=2)
[tester1|127.0.0.1] Sending Model (round=1)
[tester2|127.0.0.1] Sending Model (round=1)
[tester1|127.0.0.1] Checkpoint Received
[tester2|127.0.0.1] Checkpoint Received
Start aggregation (round=1)
Aggregate model saved: [data/logs/server//1.aggregate]
Starting round [2]
[tester1|127.0.0.1] Sending Model (round=2)
[tester2|127.0.0.1] Sending Model (round=2)
[tester1|127.0.0.1] Checkpoint Received
[tester2|127.0.0.1] Checkpoint Received
Start aggregation (round=2)
Aggregate model saved: [data/logs/server//2.aggregate]
Starting round [3]
[tester2|127.0.0.1] Sending Model (round=3)
[tester1|127.0.0.1] Sending Model (round=3)
[tester2|127.0.0.1] Checkpoint Received
[tester1|127.0.0.1] Checkpoint Received
Start aggregation (round=3)
Aggregate model saved: [data/logs/server//3.aggregate]
Starting round [4]
```

Figure 3.27: Sample output of from the server.

```
(federated) elix@bash:/var/www/federated$ python -m mila.run client data/configs/mila/2/client1.json
[CAUTION] Connection is insecure!
[CAUTION] Connection is insecure!
[CAUTION] Connection is insecure!
epoch: 1 - iteration: 5 - examples: 640 - loss: 1.2937450647354125 - time elapsed: 0:00:00 - progress: 0.1525
epoch: 1 - iteration: 10 - examples: 1280 - loss: 1.1932451248168945 - time elapsed: 0:00:00 - progress: 0.305
epoch: 1 - iteration: 15 - examples: 1920 - loss: 1.1230454444885254 - time elapsed: 0:00:00 - progress: 0.4575
epoch: 1 - iteration: 20 - examples: 2560 - loss: 1.0506272077560426 - time elapsed: 0:00:00 - progress: 0.61
epoch: 1 - iteration: 25 - examples: 3200 - loss: 1.0024004578590393 - time elapsed: 0:00:00 - progress: 0.7624
epoch: 1 - iteration: 30 - examples: 3840 - loss: 0.9705534219741822 - time elapsed: 0:00:01 - progress: 0.9149
Saving checkpoint: data/logs/local/tester1/checkpoint.1
epoch: 2 - iteration: 5 - examples: 640 - loss: 0.9004879713058471 - time elapsed: 0:00:01 - progress: 0.1525
epoch: 2 - iteration: 10 - examples: 1280 - loss: 0.8540361762046814 - time elapsed: 0:00:01 - progress: 0.305
epoch: 2 - iteration: 15 - examples: 1920 - loss: 0.8188570618629456 - time elapsed: 0:00:01 - progress: 0.4575
epoch: 2 - iteration: 20 - examples: 2560 - loss: 0.8083093166351318 - time elapsed: 0:00:01 - progress: 0.61
epoch: 2 - iteration: 25 - examples: 3200 - loss: 0.7534210562705994 - time elapsed: 0:00:01 - progress: 0.7624
epoch: 2 - iteration: 30 - examples: 3840 - loss: 0.7510416865348816 - time elapsed: 0:00:01 - progress: 0.9149
Saving checkpoint: data/logs/local/tester1/checkpoint.2
epoch: 3 - iteration: 5 - examples: 640 - loss: 0.6974666357040405 - time elapsed: 0:00:01 - progress: 0.1525
epoch: 3 - iteration: 10 - examples: 1280 - loss: 0.6492686152458191 - time elapsed: 0:00:01 - progress: 0.305
epoch: 3 - iteration: 15 - examples: 1920 - loss: 0.6089567184448242 - time elapsed: 0:00:01 - progress: 0.4575
epoch: 3 - iteration: 20 - examples: 2560 - loss: 0.5941434621810913 - time elapsed: 0:00:01 - progress: 0.61
epoch: 3 - iteration: 25 - examples: 3200 - loss: 0.542203176021576 - time elapsed: 0:00:02 - progress: 0.7624
epoch: 3 - iteration: 30 - examples: 3840 - loss: 0.5095760762691498 - time elapsed: 0:00:02 - progress: 0.9149
Saving checkpoint: data/logs/local/tester1/checkpoint.3
epoch: 4 - iteration: 5 - examples: 640 - loss: 0.4610184669494629 - time elapsed: 0:00:02 - progress: 0.1525
epoch: 4 - iteration: 10 - examples: 1280 - loss: 0.4187736213207245 - time elapsed: 0:00:02 - progress: 0.305
epoch: 4 - iteration: 15 - examples: 1920 - loss: 0.4015202224254608 - time elapsed: 0:00:02 - progress: 0.4575
```

Figure 3.28: Sample output of from the client.

## 3.6 Environment Information

List of 3rd party libraries and frameworks used:

- **Pytorch**[51]: base framework
- **Pytorch Geometric**[19]: graph architectures

- **Scikit-Learn**[52]: metrics
- **Optuna**[2]: Bayesian optimization
- **Opacus**[50]: differential privacy
- **RdKit**[34]: molecule handling and descriptors
- **Mordred**[53]: molecular descriptors
- **DGL-LifeSci**[14]: featurizers
- **Captum**[31]: integrated gradients

Versioning:

- **OS**: Ubuntu 20.04.1 LTS
- **Python Version**: 3.8.3
- **PyTorch Version**: 1.6.0
- **CUDA Toolkit Version**: 10.2.89
- **cuDNN Version**: 7.6.5

# Bibliography


[1] Abien Fred Agarap. Deep learning using rectified linear units (relu), 2019.

[2] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework, 2019.

[3] Vinicius M. Alves, Eugene Muratov, Denis Fourches, Judy Strickland, Nicole Kleinstreuer, Carolina H. Andrade, and Alexander Tropsha. Predicting chemically-induced skin reactions. part i: Qsar models of skin sensitization and their application to identify potentially hazardous compounds. *Toxicology and Applied Pharmacology*, 284(2):262–272, 2015.

[4] Alexandru T. Balaban. Highly discriminating distance-based topological index. *Chemical Physics Letters*, 89(5):399–404, 1982.

[5] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *Proceedings of the 24th International Conference on Neural Information Processing Systems*, NIPS'11, page 2546âĂŞ2554, Red Hook, NY, USA, 2011. Curran Associates Inc.

[6] James Bergstra, Daniel Yamins, and David Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In Sanjoy Dasgupta and David McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 115–123, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR.

[7] Filippo Maria Bianchi, Daniele Grattarola, Lorenzo Livi, and Cesare Alippi. Graph neural networks with convolutional arma filters, 2021.

[8] G. Richard Bickerton, Gaia V. Paolini, Jeremy Besnard, Sorel Muresan, and Andrew L. Hopkins. Quantifying the chemical beauty of drugs. *Nature News*, Jan 2012.

[9] Fabio Broccatelli, Emanuele Carosati, Annalisa Neri, Maria Frosini, Laura Goracci, Tudor I. Oprea, and Gabriele Cruciani. A novel approach for predicting p-glycoprotein (abcb1) inhibition using molecular interaction fields. *Journal of Medicinal Chemistry*, 54(6):1740–1751, 2011. PMID: 21341745.

[10] Tianle Cai, Shengjie Luo, Keyulu Xu, Di He, Tie-Yan Liu, and Liwei Wang. Graphnorm: A principled approach to accelerating graph neural network training, 2021.

[11] Miriam Carbon-Mangels and Michael C. Hutter. Selecting relevant descriptors for classification by bayesian estimates: A comparison with decision trees and support vector machines approaches for disparate data sets. *Molecular Informatics*, 30(10):885–895, 2011.

[12] Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery  Data Mining*, Jul 2019.

[13] MichaÃńl Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering, 2017.

[14] Dgl-lifesci library. Available from lifesci.dgl.ai.

[15] Li Di, Christopher Keefer, Dennis O. Scott, Timothy J. Strelevitz, George Chang, Yi-An Bi, Yurong Lai, Jonathon Duckworth, Katherine Fenner, Matthew D. Troutman, and R. Scott Obach. Mechanistic insights from comparing intrinsic clearance values between human liver microsomes and hepatocytes to guide drug design. *European Journal of Medicinal Chemistry*, 57:441–448, 2012.

[16] Veith H;Southall N;Huang R;James T;Fayne D;Artemenko N;Shen M;Inglese J;Austin CP;Lloyd DG;Auld DS;. Comprehensive characterization of cytochrome p450 isozyme selectivity across chemical libraries.

[17] Jian Du, Shanghang Zhang, Guanhang Wu, Jose M. F. Moura, and Soummya Kar. Topology adaptive graph convolutional networks, 2018.

[18] Cynthia Dwork and Aaron Roth. The algorithmic foundations of differential privacy. *Foundations and TrendsÂő in Theoretical Computer Science*, 9(3âĂŞ4):211–407, 2014.

[19] Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric, 2019.

[20] Kaitlyn M. Gayvert, Neel S. Madhukar, and Olivier Elemento. A data-driven approach to predicting successes and failures of clinical trials. *Cell Chemical Biology*, 23(10):1294–1301, Oct 2016.

[21] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry, 2017.

[22] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs, 2018.

[23] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[24] Stephen R. Heller, Alan McNaught, Igor Pletnev, Stephen Stein, and Dmitrii Tchekhovskoi. Inchi, the iupac international chemical identifier. *Journal of Cheminformatics*, 7(1):23, May 2015.

[25] Tin Kam Ho. Random decision forests. In *Proceedings of 3rd International Conference on Document Analysis and Recognition*, volume 1, pages 278–282 vol.1, 1995.

[26] Tingjun Hou, Junmei Wang, Wei Zhang, and Xiaojie Xu. Adme evaluation in drug discovery. 7. prediction of oral absorption by correlation and classification. *Journal of Chemical Information and Modeling*, 47(1):208–218, 2007. PMID: 17238266.

[27] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015.

[28] Pavel Izmailov, Dmitrii Podoprikhin, Timur Garipov, Dmitry Vetrov, and Andrew Gordon Wilson. Averaging weights leads to wider optima and better generalization, 2019.

[29] J. Kiefer and J. Wolfowitz. Stochastic estimation of the maximum of a regression function. *The Annals of Mathematical Statistics*, 23(3):462–466, 2021/02/26/ 1952. Full publication date: Sep., 1952.

[30] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[31] Narine Kokhlikyan, Vivek Miglani, Miguel Martin, Edward Wang, Bilal Alsallakh, Jonathan Reynolds, Alexander Melnikov, Natalia Kliushkina, Carlos Araya, Siqi Yan, and Orion Reblitz-Richardson. Captum: A unified and generic model interpretability library for pytorch, 2020.

[32] Paul Labute. A widely applicable set of descriptors. *Journal of Molecular Graphics and Modelling*, 18(4):464–477, 2000.

[33] Alexey Lagunin, Dmitrii Filimonov, Alexey Zakharov, Wei Xie, Ying Huang, Fucheng Zhu, Tianxiang Shen, Jianhua Yao, and Vladimir Poroikov. Computer-aided prediction of rodent carcinogenicity by pass and cisoc-psct. *QSAR & Combinatorial Science*, 28(8):806–810, 2009.

[34] Greg Landrum. Rdkit: Open-source cheminformatics.

[35] Guohao Li, Chenxin Xiong, Ali Thabet, and Bernard Ghanem. Deepergcn: All you need to train deeper gcns, 2020.

[36] Pengyong Li, Yuquan Li, Chang-Yu Hsieh, Shengyu Zhang, Xianggen Liu, Huanxiang Liu, Sen Song, and Xiaojun Yao. TrimNet: learning molecular representation from triplet messages for biomedicine. *Briefings in Bioinformatics*, 11 2020. bbaa266.

[37] Yiming Li, Baoyuan Wu, Yong Jiang, Zhifeng Li, and Shu-Tao Xia. Backdoor learning: A survey, 2021.

[38] Christopher A Lipinski, Franco Lombardo, Beryl W Dominy, and Paul J Feeney. Experimental and computational approaches to estimate solubility and permeability in drug discovery and development settings1pii of original

article: S0169-409x(96)00423-1. the article was originally published in advanced drug delivery reviews 23 (1997) 3âĂŞ25.1. *Advanced Drug Delivery Reviews*, 46(1):3–26, 2001. Special issue dedicated to Dr. Eric Tomlinson, Advanced Drug Delivery Reviews, A Selection of the Most Highly Cited Articles, 1991-1998.

[39] Franco Lombardo and Yankang Jing. In silico prediction of volume of distribution in humans. extensive data set and the exploration of linear and nonlinear methods coupled with molecular interaction fields descriptors. *Journal of Chemical Information and Modeling*, 56(10):2042–2052, 2016. PMID: 27602694.

[40] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization, 2019.

[41] Chang-Ying Ma, Sheng-Yong Yang, Hui Zhang, Ming-Li Xiang, Qi Huang, and Yu-Quan Wei. Prediction models of human plasma protein binding rate and oral bioavailability derived by using gaâĂŞcgâĂŞsvm method. *Journal of Pharmaceutical and Biomedical Analysis*, 47(4):677–682, 2008.

[42] H. Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise AgÃijera y Arcas. Communication-efficient learning of deep networks from decentralized data, 2017.

[43] Ilya Mironov. RÃľnyi differential privacy. *2017 IEEE 30th Computer Security Foundations Symposium (CSF)*, Aug 2017.

[44] Ilya Mironov, Kunal Talwar, and Li Zhang. Rényi differential privacy of the sampled gaussian mechanism, 2019.

[45] Guthrie J. Peter Mobley David L. Freesolv: a database of experimental and calculated hydration free energies, with input files.

[46] H. L. Morgan. The generation of a unique machine description for chemical structures-a technique developed at chemical abstracts service. *Journal of Chemical Documentation*, 5(2):107–113, 1965.

[47] Christopher Morris, Martin Ritzert, Matthias Fey, William L. Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks, 2020.

[48] Abhishek Khetan Suleyman Er Murat Cihan Sorkun. Aqsoldb, a curated reference set of aqueous solubility and 2d descriptors for a diverse set of compounds.

[49] R. Scott Obach, Franco Lombardo, and Nigel J. Waters. Trend analysis of a database of intravenous pharmacokinetic parameters in humans for 670 drug compounds. *Drug Metabolism and Disposition*, 36(7):1385–1405, 2008.

[50] Opacus PyTorch library. Available from opacus.ai.

[51] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

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

[53] V. Consonni R. Todeschini, Y. Fu R. Wang, GM. Crippen AK. Ghose, R. Goswami V. Sharma, PC. Jurs DT. Stanton, CW. Yap, Q-S. Xu D-S. Cao, Y-Z. Liang D-S. Cao, N. Xiao D-S. Cao, GR. Hutchison NM. OâĂŹBoyle, and et al. Mordred: a molecular descriptor calculator. *Journal of Cheminformatics*, Jan 1970.

[54] Ekagra Ranjan, Soumya Sanyal, and Partha Pratim Talukdar. Asap: Adaptive structure aware pooling for learning hierarchical graph representations, 2020.

[55] Maria Rigaki and Sebastian Garcia. A survey of privacy attacks in machine learning, 2020.

[56] David Rogers and Mathew Hahn. Extended-connectivity fingerprints. *Journal of Chemical Information and Modeling*, 50(5):742–754, 2010. PMID: 20426451.

[57] Martin Simonovsky and Nikos Komodakis. Dynamic edge-conditioned filters in convolutional neural networks on graphs, 2017.

[58] Leslie N. Smith and Nicholay Topin. Super-convergence: Very fast training of neural networks using large learning rates, 2018.

[59] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

[60] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks, 2017.

[61] Max Welling Thomas N. Kipf. Semi-supervised classification with graph convolutional networks. *Arxiv*, 2017.

[62] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance normalization: The missing ingredient for fast stylization, 2017.

[63] Petar VeliÄ■koviÄĞ, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro LiÃš, and Yoshua Bengio. Graph attention networks, 2018.

[64] Nitika Verma, Edmond Boyer, and Jakob Verbeek. Feastnet: Feature-steered graph convolutions for 3d shape analysis, 2018.

[65] Oriol Vinyals, Samy Bengio, and Manjunath Kudlur. Order matters: Sequence to sequence for sets, 2016.

[66] Ning-Ning Wang, Jie Dong, Yin-Hua Deng, Min-Feng Zhu, Ming Wen, Zhi-Jiang Yao, Ai-Ping Lu, Jian-Bing Wang, and Dong-Sheng Cao. Adme properties evaluation in drug discovery: Prediction of caco-2 cell permeability using a combination of nsga-ii and boosting. *Journal*

*of Chemical Information and Modeling*, 56(4):763–773, 2016. PMID: 27018227.

[67] Shuangquan Wang, Huiyong Sun, Hui Liu, Dan Li, Youyong Li, and Tingjun Hou. Admet evaluation in drug discovery. 16. predicting herg blockers by combining multiple pharmacophores and machine learning approaches. *Molecular Pharmaceutics*, 13(8):2855–2866, 2016. PMID: 27379394.

[68] Kang Wei, Jun Li, Ming Ding, Chuan Ma, Yo-Seb Jeon, and H. Vincent Poor. Covert model poisoning against federated learning: Algorithm design and optimization, 2021.

[69] David Weininger. Smiles, a chemical language and information system. 1. introduction to methodology and encoding rules. *Journal of Chemical Information and Computer Sciences*, 28(1):31–36, 1988.

[70] Mark Wenlock and Nicholas Tomkinson. Experimental in vitro dmpk and physicochemical data on a set of publicly disclosed compounds.

[71] Felix Wu, Tianyi Zhang, Amauri Holanda de Souza Jr. au2, Christopher Fifty, Tao Yu, and Kilian Q. Weinberger. Simplifying graph convolutional networks, 2019.

[72] Yuxin Wu and Kaiming He. Group normalization, 2018.

[73] Zhenqin Wu, Bharath Ramsundar, Evan N. Feinberg, Joseph Gomes, Caleb Geniesse, Aneesh S. Pappu, Karl Leswing, and Vijay Pande. Moleculenet: A benchmark for molecular machine learning, 2018.

[74] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks?, 2019.

[75] Youjun Xu, Ziwei Dai, Fangjin Chen, Shuaishi Gao, Jianfeng Pei, and Luhua Lai. Deep learning for drug-induced liver injury. *Journal of Chemical Information and Modeling*, 55(10):2085–2093, 2015. PMID: 26437739.

[76] Hao Zhu, Todd M. Martin, Lin Ye, Alexander Sedykh, Douglas M. Young, and Alexander Tropsha. Quantitative structureâĹŠactivity relationship modeling of rat acute toxicity by oral exposure. *Chemical Research in Toxicology*, 22(12):1913–1921, 2009. PMID: 19845371.

[77] Juntang Zhuang, Tommy Tang, Yifan Ding, Sekhar Tatikonda, Nicha Dvornek, Xenophon Papademetris, and James S. Duncan. Adabelief optimizer: Adapting stepsizes by the belief in observed gradients, 2020.