# ASCAPE

Artificial intelligence Supporting CAncer Patients across Europe

| | |
|---|---|
| Project Title | Artificial intelligence Supporting CAncer Patients across Europe |
| Project Acronym | ASCAPE |
| Grant Agreement No | 875351 |
| Instrument | Research and Innovation action |
| Call / Topic | H2020-SC1-DTH-2019 / Big data and Artificial Intelligence for monitoring health status and quality of life after the cancer treatment |
| Start Date of Project | 01/01/2020 |
| Duration of Project | 36 months |

# D2.4 ML/DL Training and Evaluation Report-V1

| | |
|---|---|
| Work Package | WP2 – ASCAPE data management, DL/ML algorithms and design |
| Lead Author (Org) | Serge Autexier (DFKI) |
| Contributing Author(s) (Org) | Brankica Bratić (UNSPMF), Lucian Itu (SIE), Mirjana Ivanovic (UNSPMF), Vladimir Kurbalija (UNSPMF), Johannes Rust (DFKI), Miloš Savić (UNSPMF), Ioana Antonia Taca (SIE), Anamaria Vizitiu (SIE) |
| Due Date | 28.02.2021 |
| Actual Date of Submission | 26.02.2021 |
| Version | 1.0 |

**Dissemination Level**

| | |
|---|---|
| X | PU: Public (*on-line platform) |
| | PP: Restricted to other programme participants (including the Commission) |
| | RE: Restricted to a group specified by the consortium (including the Commission) |
| | CO: Confidential, only for members of the consortium (including the Commission) |

The work described in this document has been conducted within the project ASCAPE. This project has received funding from the European Union's Horizon 2020 (H2020) research and innovation programme under the Grant Agreement no 875351. This document does not represent the opinion of the European Union, and the European Union is not responsible for any use that might be made of such content.

## Versioning and contribution history

| Version | Date | Authors | Notes |
|---|---|---|---|
| 0.1 | 04.11.2020 | Serge Autexier (DFKI)<br>Johannes Rust (DFKI) | First Draft TOC |
| 0.2 | 22.01.2021 | Brankica Bratić (UNSPMF)<br>Lucian Itu (SIE)<br>Mirjana Ivanovic (UNSPMF)<br>Vladimir Kurbalija (UNSPMF)<br>Johannes Rust (DFKI)<br>Miloš Savić (UNSPMF)<br>Ioana Antonia Taca (SIE)<br>Anamaria Vizitiu (SIE) | Input to the different section from partners. |
| 0.3 | 25.01.2021 | Serge Autexier (DFKI)<br>Johannes Rust (DFKI) | Added Summary, Introduction and conclusions; first internal editing of the complete the deliverable. |
| 0.4 | 27.01.2021 | Serge Autexier (DFKI)<br>Mirjana Ivanovic (UNSPMF)<br>Johannes Rust (DFKI) | Further corrections, addition of explainability information |
| 0.5 | 29.01.2021 | Serge Autexier (DFKI)<br>Miloš Savić (UNSPMF)<br>Johannes Rust (DFKI)<br>Anamaria Vizitiu (SIE) | Finalizing for internal review |
| 0.6 | 19.02.2021 | Serge Autexier (DFKI)<br>Johannes Rust (DFKI)<br>Miloš Savić (UNSPMF)<br>Anamaria Vizitiu (SIE) | Addressing reviewers' recommendations |
| 0.9 | 22.02.2021 | Serge Autexier (DFKI) | Final check by task leader |
| 1.0 | 26.02.2021 | Christina Stratigaki (UBITECH) | Quality Assurance and final review |

# Table of Contents



# List of Tables

# List of Figures

# List of Acronyms

| | |
|---|---|
| ADAB | Adaptive Boosting (AdaBoost) |
| AI | Artificial Intelligence |
| CON | (Semi-)Concurrent |
| CSV | Comma-separated Values |
| DP | Differential Privacy |
| DT | Decision Tree |
| FHIR | Fast Healthcare Interoperability Resources |
| HE | Homomorphic Encryption |
| HL7 | Health Level 7 |
| INC | Incremental |
| IPSS | International Prostate Symptom Score |
| KNN | K-Nearest-Neighbour |
| LASSO | LASSO regression |
| LIFT | Deep Learning Important FeaTures |
| LIME | Local Interpretable Model-Agnostic Explanations |
| LISAT | Life-Satisfaction Questionnaire |
| MAE | Mean Absolute Error |
| ML | Machine Learning |
| MLP | Multilayer Perceptron |
| MORE | Matrix Operation for Randomization or Encryption |
| MSE | Mean Squared Error |
| NB | Naïve Bayes |
| NN | Neural Network |
| ORB | Örebro |
| PC | Pearson Correlation Coefficient |
| QoL | Quality of Life |
| ReLU | Rectified linear unit |
| RF | Random Forest |
| SGD | Stochastic Gradient Descent |
| SHAP | SHapley Additive exPlanations |
| SVM | Support Vector Machine |
| TFNN | TensorFlow Neural Network |

# Executive Summary

The goal of the ASCAPE project is to provide data driven assistance to preserve and improve the Quality of Life (QoL) of cancer patients. The deliverable reports on the first development of model training algorithms and model-based assistances and their evaluation. The work is based on anonymised retrospective datasets available at the pilot sites of ASCAPE and addresses all steps in the process of incorporating data with possibly missing values, differential privacy methods to prevent inference of personal information from model, the predictive model training styles for federated model training and on homomorphically encrypted data, and up to methods to obtain explanations of predictions and intervention suggestions based on trained models. All developments have been conducted on a fixed set of ten training datasets and evaluated according to exactly the same techniques and metrics. On these datasets regarding methods for initial data preparation steps were evaluated and lead to two results: first, missing value inference by simple methods is as effective as missing value inference by more complex regression methods and thus sufficient. Second, privacy-enhancement by differential privacy should be using the Laplace mechanism with an $\varepsilon$ value around 1. In locally trained models, training of models for classification and for regression was considered. Five algorithms for training classification models and nine algorithms to train regression models were evaluated using the same datasets and the result was that the naïve Bayes classifier yields the best classification models and Lasso regression the best regression models. In a simulation of federated learning, based on a preliminary evaluation, swallow neural networks trained on large batch sizes were selected for classification tasks and deeper neural networks trained on small batch sizes were selected to train models in both incremental and semi-concurrent federated learning mode for two to four edge nodes. The results show no significant differences in the learning modes and were close or even better in classification tasks compared to the locally trained models, but in regression tasks had higher prediction errors with a tendency of increasing errors with the number of edge nodes. Models trained on homomorphically encrypted data are weaker due to the fact that only Stochastic Gradient Descent optimizers are currently available. For explainability of model predictions, the SHAP framework was identified as the best basis for feature attribution and trained surrogate models showed overall good to excellent approximation of the target model. Finally, a simulation method based on the identification of key features in the predictive models was developed, to suggest best treatments for a specific patient, exploiting the knowledge encoded in the trained predictive models. Future next steps have been identified and the overall evaluation whether the foreseen machine learning methods can result in useful, data-driven assistance to help improving the quality-of-life of patients is conclusive.

# 1 Introduction

The goal of the ASCAPE project is to provide data driven assistance to preserve and improve the Quality of Life (QoL) of cancer patients. To this end, datasets from cancer patients shall be used to train predictive models for QoL indicators. These models will then be used for predicting quality of life evolution for specific patients and propose interventions that have a beneficial effect on it. The training of the predictive models will be based on datasets collected continuously at the four pilot site partners in Greece, Spain, Sweden and UK, and extended with datasets from further participating partners included during the open call. Privacy of patient data is among ASCAPE's highest concern and a number of measures are applied in the ASCAPE system, such as that patient data is always pseudonymized and never leaves the premises of the participating site except if completely encrypted. In order to accommodate the distributed training of predictive models, specific federated model training schemes are proposed, that support the federated learning and especially the flexible addition and removal of participating partner sites. The goal of the present deliverable is to report on the first development of model training algorithms and model-based assistances and their evaluation.

The deliverable is structured along the process from incorporating and curating data, developing and evaluating different predictive model training styles using that data, and using such models to obtain explanations of predictions and suggestions for interventions. As prospective data collection was about to start only in ASCAPE pilots, retrospective datasets available at pilot sites were used as the basis for the presented methods. Section 2 presents the selected retrospective datasets, the selected target variables and how a fixed set of 10 training datasets have been derived from them. They are used as a common ground for all further developments to ensure the comparability of their evaluations. Section 3 presents different methods for missing value imputation and compares their performances as well as variants of differential privacy methods and their comparison to prevent inference of information about specific patients from trained models. Section 4 is the main part of the deliverable, which presents the evaluation of different model training methods on local datasets, in federated style and on homomorphically encrypted data. All training algorithms have been applied on the 10 training datasets and evaluated according to the same metrics. Section 5 then evaluates methods to infer explanations for predictions obtained from such models. Section 6 reports on experiments on how to obtain an assessment of interventions with respect to their effect on quality of life based on the predictive models. The main results and achievements of these first experiments towards developing machine learning based assistances are summarized in section 7.

# 2 Training Datasets and Targets

## 2.1 Training Datasets

Since prospective datasets were not available, experiments for the proof of concept have been based on the retrospective datasets. Three datasets have been provided by the pilot site partner from Örebro in Sweden which contained datasets with anonymized medical data of patients diagnosed with breast cancer and of prostate cancer. The datasets vary in the number of samples/patients, the number and composition of medical data fields and the target variables. While this makes the direct comparison and alignment of the datasets difficult, they offer a great variety of data types, formats and other challenges that need to be addressed for machine learning. AI prototypes created based on these datasets therefore are a good reference for the work coming regarding the prospective datasets.

The retrospective datasets are not transformed to the HL7 FHIR format but are arranged as a table saved as CSV-format. Each row represents a different patient and each column represents one variable (a field in a patient's medical data).

A variable can be a date or duration, categorical data, ordinal data, information about a patient like the age of diagnosis, scores for standardized medical questionnaires and measurements done during treatment. At a later stage, prospective datasets will be transformed to FHIR. Not all fields contain valid data, mostly because a certain treatment has not taken place. This yields the possibility to add additional variables during pre-processing which mark if a treatment has taken place or not. To identify these fields human input is needed.

For experimentation, two of three retrospective datasets were deemed suitable: Breast cancer *BcBase* dataset (18988 rows x 47 columns) and Prostate cancer *Orebro* dataset (2466 rows x 124 columns).

The Örebro dataset (ORB) contains 2466 health records of prostate cancer patients with each containing 124 variable fields. After 3 weeks and again in at months 6, 12, 18, 24, 30, 36, 42, 48, 54, 60, 72, 84, 96, 108 and 120 after the diagnosis, a follow-up examination s scheduled for the patient, in which he reports about bowel side effects, erectile function and lower urinary tract symptoms. Additionally, for the follow-up dates at months 36, 60 and 120, the International Prostate Symptom Score (IPSS) and the LISAT-11 Quality of Life Score were collected from the patients. The LISAT score will be used as a target field and always has a value between 11 and 66. The health data collected at the different follow-up examinations forms sequential data, although in non-regular time steps.

The second dataset BcBase contains patient data of 18988 breast cancer patients. It contains 47 variables per sample. The high number of samples makes it especially resistant to overfitting, thus deep neural networks can be trained with it easily. The dataset also contains socio-economic indicators like marital status, education status, personal income and household income.

The third retrospective dataset, a breast cancer dataset from Sormland, was not used for experiments, because it is very similar to the BcBase dataset, but only contains 232 samples.

All datasets contain information about medical interventions being done. Analysing the model outputs depending on whether an intervention has been done or not, will be the foundation to the suggestion of interventions for other patients.

## 2.2 Target Variables for Training

Variables that are suitable to be used as target variables should yield information about something that is not easily measurable in a real-world scenario. An example for this is medical data that can only be collected in the future and are not yet available. ASCAPE will examine on hypothetical changes in the patient's medical data to propose treatments and interventions. Since such a simulated medical file is not linked to a real patient, information about quality of life can only be estimated by an AI model. The ORB dataset contains LISAT-11 QoL scores at the time of the diagnosis and three different times relative to the date of diagnosis at months 36, 60 and 120. The dataset will therefore be used to train models to predict future LISAT-11 QoL scores. It is split into a variety of datasets with the naming scheme ORB-*n*-*m* containing all variables up to month *n* and the QoL score in month *m*. These datasets are then used to train predictive models for a LISAT-11 QoL score in month *m* based on the data available until month *n.* For instance, ORB-30-120 will be used to predict the LISAT-11 QoL score for month 120 based on all variables that are available up until the follow-up in month 30. The same scheme is used to create the experiment datasets ORB-30-36, ORB-30-60, ORB-30-120, ORB-54-60, ORB-54-120 and ORB-108-120, respectively. The BcBase dataset does not have QoL scores but contains information whether medications to treat pain, anxiety, insomnia, or depression were given to the patient. This dataset is then used to create datasets to train binary classifiers which estimate if a patient has one of these conditions. The classification scores of the trained models can be used for risk assessment. The datasets were created by dropping all but one medication variable and keeping all other variables to obtain the datasets BcBase-Anxiety, BcBase-Pain, BcBase-Insomnia and BcBase-Depression.

These datasets (6 ORB-*m*-*n* and 4 BcBase-*m*) were used for all following AI-related procedures to have a common ground when comparing performance and results. The training datasets and all subsequently derived datasets and models were stored in a common repository to ensure traceability and quality control over datasets, used algorithms and obtained trained models.

On these datasets the following data pre-processing actions are implemented: dates transformation, one-hot encoding, and missing values imputation. In the end, 4 new datasets are obtained from the BcBase dataset, representing binary classification

problems, and 6 new datasets are obtained from the Orebro one, representing regression problems:

1. BcBase-Anxiety (18988 rows x 99 columns) – the model is trained on 97 features and it should predict if the patient will have anxiety problems after the breast cancer treatment. The output is 0 if the patient will not suffer from anxiety or 1 otherwise.
2. BcBase-Depression (18988 rows x 99 columns) – the model is trained on 97 features and it should predict if the patient will have depression problems after the breast cancer treatment. The output is 0 if the patient will not suffer from depression or 1 otherwise.
3. BcBase-Insomnia (18988 rows x 99 columns) – the model is trained on 97 features and it should predict if the patient will have insomnia problems after the breast cancer treatment. The output is 0 if the patient will not suffer from insomnia or 1 otherwise.
4. BcBase-Pain (18988 rows x 99 columns) – the model is trained on 97 features and it should predict if the patient will have pain problems after the breast cancer treatment. The output is 0 if the patient will not suffer from pain or 1 otherwise.
5. ORB-30-36 (1138 rows x 98 columns) – the model is trained on 96 features measured till month 30 and it should predict the quality of life after 36 months from the beginning of the prostate cancer treatment. The output is a number between 11 and 66.
6. ORB-30-60 (1042 rows x 98 columns) – the model is trained on 96 features measured till month 30 and it should predict the quality of life after 60 months from the beginning of the prostate cancer treatment. The output is a number between 11 and 66.
7. ORB-30-120 (610 rows x 98 columns) – the model is trained on 96 features measured till month 30 and it should predict the quality of life after 120 months from the beginning of the prostate cancer treatment. The output is a number between 11 and 66.
8. ORB-54-60 (1024 rows x 126 columns) – the model is trained on 124 features measured till month 54 and it should predict the quality of life after 60 months from the beginning of the prostate cancer treatment. The output is a number between 11 and 66.
9. ORB-54-120 (610 rows x 126 columns) – the model is trained on 124 features measured till month 54 and it should predict the quality of life after 120 months from the beginning of the prostate cancer treatment. The output is a number between 11 and 66.
10. ORB-108-120 (610 rows x 160 columns) – the model is trained on 158 features measured till month 108 and it should predict the quality of life after 120 months from the beginning of the prostate cancer treatment. The output is a number between 11 and 66.

# 3 Privacy-aware Data Curation

The first methods for the preparation of the datasets for the machine learning algorithms were to impute empty data points and to create data sets from applying differential privacy to ensure privacy. The resulting datasets were all reused then for the subsequent training algorithms again to allow for their comparison.

## 3.1 Training of Models for Missing Value Imputation and Evaluation

### 3.1.1 Completing Datasets by Missing Value Imputation

All the training datasets from section 2.2 contained a number of missing values. Since certain machine learning algorithms are not applicable on data with missing values, we had to perform additional transformations of our datasets.

First, we removed all the instances that had not had a value for the target attribute – such instances cannot be used for training nor testing of our predictive models. After that, we performed imputation of missing values within the rest of the instances.

For the missing values imputation, we used two algorithms from a Python library scikit-learn [1]: simple imputer and iterative imputer. Simple imputer is a rather simple approach, which fills missing values within a single attribute by using the mean of the attribute's existing values. Iterative imputer is a bit more complex: it creates a regression model for each attribute and then fills missing values of an attribute with predictions obtained from the attribute's regression model. A regression model for a single attribute is fit on instances with non-empty value for that attribute. After filling all the missing values, iterative imputer repeats the whole procedure for predefined number of times.

For the missing values imputation with iterative imputer, we do not use a dataset's target attribute. If the target attribute were used, the regression models of the iterative imputer would be fit with target values, which must not happen since it would introduce erroneous dependence between predictors and target attribute.

Finally, we applied these two missing values imputation algorithms on each dataset described in section 2.2, resulting with two variants of each dataset: 1) simple imputer variant, and 2) iterative imputer variant. These datasets are now ready for the model training.

Implementational details regarding missing values imputation can be found in Deliverable D3.1 – Cancer-care predictive analytics and decision-making services: proof of concept demonstration.

### 3.1.2 Dataset preparation for cross-validation

The AI techniques considered in the next sections will be compared to each other by cross-validation. To make cross validation results of different experiments

comparable, we created 10 folds within each dataset, and then used these folds in all the experiments. Before creating the folds, the order of dataset instances was randomized, and then stratification was applied.

For each dataset, the last column contains values from 0 to 9, representing the folds to which the instances are assigned. Therefore, from each of the 10 datasets, 10 groups were created, groups that were used accordingly to the *10-fold cross-validation* method. This method implies splitting the dataset into 10 groups and for each unique group, the group is considered to be the test dataset, and the remaining groups are taken as the training dataset. Hence, there are 10 new datasets formed from the main dataset, on which the models are trained and tested. After this, an aggregation of the outputs is made, and the evaluation metrics are computed and reported.

## 3.2 Differential Privacy

Differential privacy (DP) is a main privacy preserving technique used within ASCAPE project. It represents a sophisticated privacy protection technique which does not require any insights into the structure of knowledge of attackers, nor does it require the reorganization or restructuring of the dataset. Differential privacy is based on the idea that the outcome of the query posed to protected database is essentially equally likely independent of whether any individual joins or refrains from joining the database. In such a way the private data about a particular patient is protected since the system returns the result with the same probability whether a particular patient was involved in the analysis or not.

Differential privacy is not a unique algorithm, but a methodology which can be used for developing a wide variety of algorithms. The general idea of DP is explained in the following definition: a randomized algorithm $A$ is $\varepsilon$-differentially private if for all neighbouring databases $D_1$ and $D_2$ (databases which differ only in one row) and for all sets $\Omega$ of possible outputs the following condition holds:

$$Pr[\mathrm{A}(\mathrm{D}_1) \in \Omega] \leq e^{\varepsilon} \cdot Pr[A(D_2) \in \Omega]$$

The parameter $\varepsilon$ is called the privacy budget. This parameter controls the level of privacy of the algorithm $A$. Smaller values of $\varepsilon$ mean stronger privacy. The value 0 represents total privacy, but the usability of such an algorithm is none since in that case algorithm represents pure randomness.

There are several mechanisms to implement DP, but the most common are: Laplace mechanism, Gaussian mechanism and Exponential mechanism. Laplace mechanism is most commonly used for numeric types of data, and it consists of adding Laplacian noise (a noise which follows Laplace distribution) to the data model. Furthermore, in machine learning algorithms noise could be added in different ways: a) to the training dataset, b) to the prediction model itself (for example in edge weights in neural network), and c) to the predicted results of the model. Within ASCAPE project, first option will be applied as the simplest one, but still sufficiently general and robust.

As expected, more noise means more privacy, but a lot of noise could lead to the reduction of model performance/accuracy. So, the main challenge in introduction of privacy preserving techniques in machine learning models is the optimal balance between privacy and the utility of models. This section will present some initial results of applying DP to machine learning models on retrospective datasets in ASCAPE framework. The implementation of relevant DP component which is used for generation of presented results is described in Deliverable D3.1 – Cancer-care predictive analytics and decision-making services: proof of concept demonstration.

The DP analysis presented here was performed on the ORB-30-36 dataset, which predicts the LISAT-11 QoL score in the month 36 on the basis of data available up to month 30. The LISAT-11 QoL score is a number in the interval 11-66 so the following regression methods were applied (as explained in sections 4.3 and 4.5):

- LINEAR: linear regression
- RIDGE: ridge regression
- LASSO: lasso regression
- ELASTICN: elastic net regression
- KRIDGE: kernel ridge regression
- SVM: support vector machine regression
- RF: random forest regression
- KNN: K-nearest neighbours regression
- ADAB: AdaBoost regression
- TFNN: Tensor Flow Neural Network

The original ORB-30-36 dataset with already imputed missing values is treated with the DP component which adds a specified amount of noise. The amount of noise is expressed through the ε parameter. The noise is added only on numerical features while the categorical features, Boolean features, one-hot-encoding features and class features stayed untouched. The following values of DP parameter (ε) were used for experiments: 0.1, 0.2, 0.5, 1, 2, 5, 10, 20, 50 and 100.

All mentioned regression models were trained with each of these modified datasets (datasets with added noise). For the evaluation of trained models four evaluation metrics were used (as explained in section 4.4):

- MAE: mean absolute error
- MSE: mean squared error
- R2: coefficient of determination
- P: Person's correlation coefficient

Figure 1 shows the values of four evaluation metrics for Linear regression model for all values of $\varepsilon$ DP parameter (blue series). The red line shows the performance of corresponding metrics of Linear regressor trained with original data (without added noise). The x-axis represents the value of parameter $\varepsilon$, while the y-axis represents the value of corresponding evaluation metrics: a) MAE, b) MSE, c) R2, and d) P.

For the first two metrics (MAE and MSE) lower values means a better performance of the model, while for the other two (R2 and P) higher values indicate better performance. In all four cases the performances of the model generated from DP datasets with the values of ε lower than 1 (or 0.5 in MSE and R2 cases) are worse than the model generated on dataset without DP. This behaviour is expected since lower values of $\varepsilon$ means more noise, and more noise negatively influences the performance of the model. However, somewhat unexpected behaviour is noticed in the cases where $\varepsilon > 1$, where DP models perform even better than no-DP models. Surely, these improvements are not so significant, but they are present. By observing these results, and the results for the other regression methods it is evident that all four evaluation metrics show similar behaviour for a particular model from the perspective of DP. So, in future results we will show only MAE measure, while the other measures behave similarly.

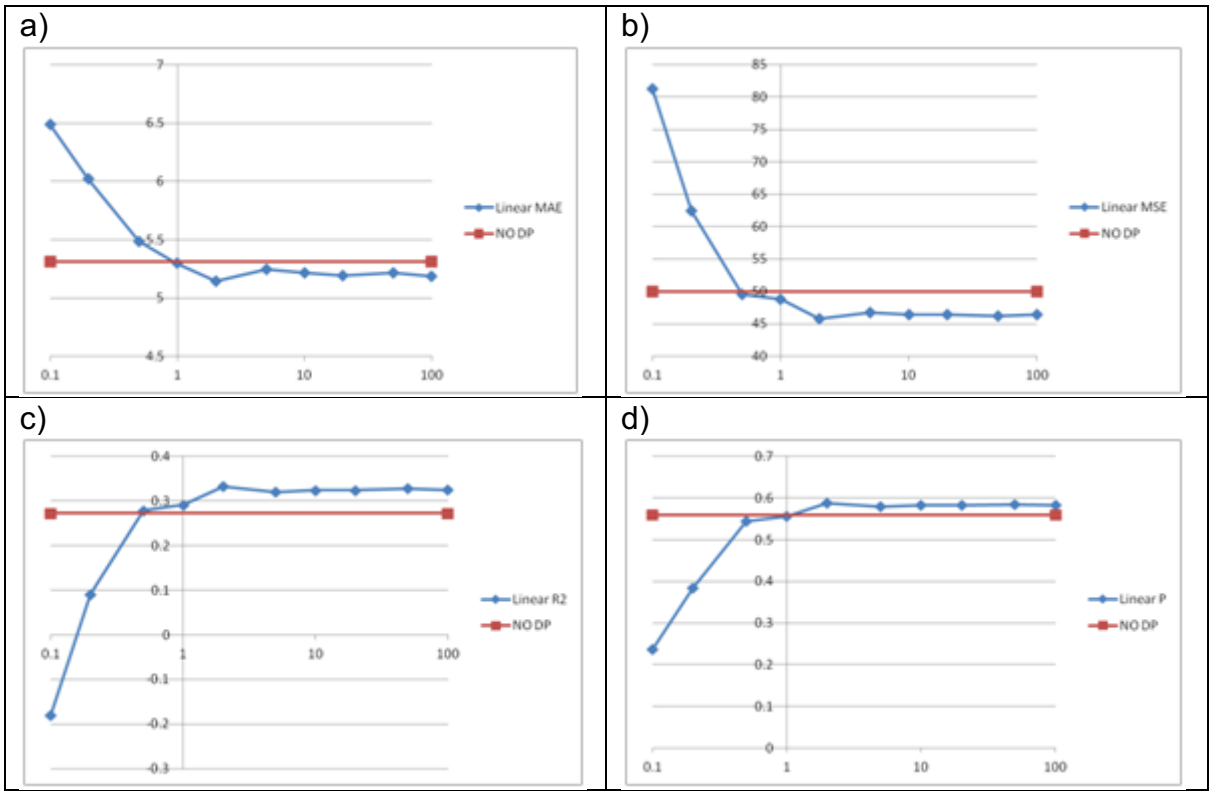


*Figure 1. The values of four evaluation metrics: a) MAE, b) MSE, c) R2, and d) P; for Linear regression model trained with datasets with added noise.*

Table 1 shows MAE values for all considered regression methods and for all values of DP parameter ε which is applied on training dataset. The last row represents MAE value for all regression models trained with data without noise addition. This value can be considered as the baseline value for each regressor. For better understanding, the MAE values of four characteristic regression models (LINEAR, LASSO, RF and TFNN) are shown visually in Figure 2 (p. 17).

*Table 1. MAE values for all conducted regression methods and all values of DP parameter ε.*

| ε | LINEAR | RIDGE | LASSO | ELASTICN | KRIDGE | SVM | RF | KNN | ADAB | TFNN |
|---|---|---|---|---|---|---|---|---|---|---|
| **0.1** | 6.486 | 6.451 | 6.587 | 6.588 | 6.471 | 6.519 | 6.146 | 6.699 | 6.211 | 7.044 |
| **0.2** | 6.018 | 5.966 | 5.855 | 5.860 | 6.017 | 6.519 | 5.971 | 6.714 | 6.130 | 6.402 |
| **0.5** | 5.486 | 5.438 | 5.334 | 5.350 | 5.465 | 6.519 | 5.422 | 6.705 | 5.548 | 6.356 |
| **1** | 5.295 | 5.254 | 5.181 | 5.212 | 5.257 | 6.519 | 5.273 | 6.724 | 5.440 | 6.102 |
| **2** | 5.144 | 5.101 | 5.093 | 5.130 | 5.136 | 6.519 | 5.107 | 6.721 | 5.312 | 5.995 |
| **5** | 5.244 | 5.175 | 5.088 | 5.125 | 5.224 | 6.519 | 5.190 | 6.718 | 5.300 | 5.621 |
| **10** | 5.217 | 5.129 | 5.092 | 5.129 | 5.181 | 6.519 | 5.097 | 6.714 | 5.288 | 5.840 |
| **20** | 5.193 | 5.103 | 5.086 | 5.123 | 5.160 | 6.519 | 5.081 | 6.717 | 5.346 | 6.033 |
| **50** | 5.215 | 5.097 | 5.090 | 5.126 | 5.150 | 6.519 | 5.035 | 6.716 | 5.342 | 6.165 |
| **100** | 5.185 | 5.095 | 5.090 | 5.126 | 5.133 | 6.519 | 4.942 | 6.720 | 5.297 | 6.062 |
| **No DP** | **5.311** | **5.100** | **5.089** | **5.126** | **5.147** | **6.519** | **5.033** | **6.720** | **5.362** | **6.035** |

For almost all regression models the same pattern can be observed. For a very small value of ε (e.g., 0.1), the MAE is notably high. After that, MAE significantly drops with an increase of ε up to the value of approximately 1. For the values of ε>1, MAE remains approximately constant on the level of MAE of models trained with No DP data (baseline level).

The main conclusion from this analysis is that the optimal value of DP parameter ε should be somewhere around or below the value 1. For the values of $\varepsilon > 1$ the models do not have significantly higher performance (do not have lower MAE), and the level of privacy is decreased. From the theoretical point of view, according to the definition of DP, this means that the ratio between two probabilities of neighbouring databases will be less than *e*, which represents a significant level of privacy.

Future efforts concerning investigation of DP within ASCAPE project will be focused on examining privacy phenomenon in other retrospective datasets. Moreover, when the prospective data became available, the same evaluation will be performed on those data too. Based on the present evaluation, it is evident that the optimal value of ε should be somewhere around 1 or bellow it. Therefore, the region around value 1 should be also investigated with higher granularity.

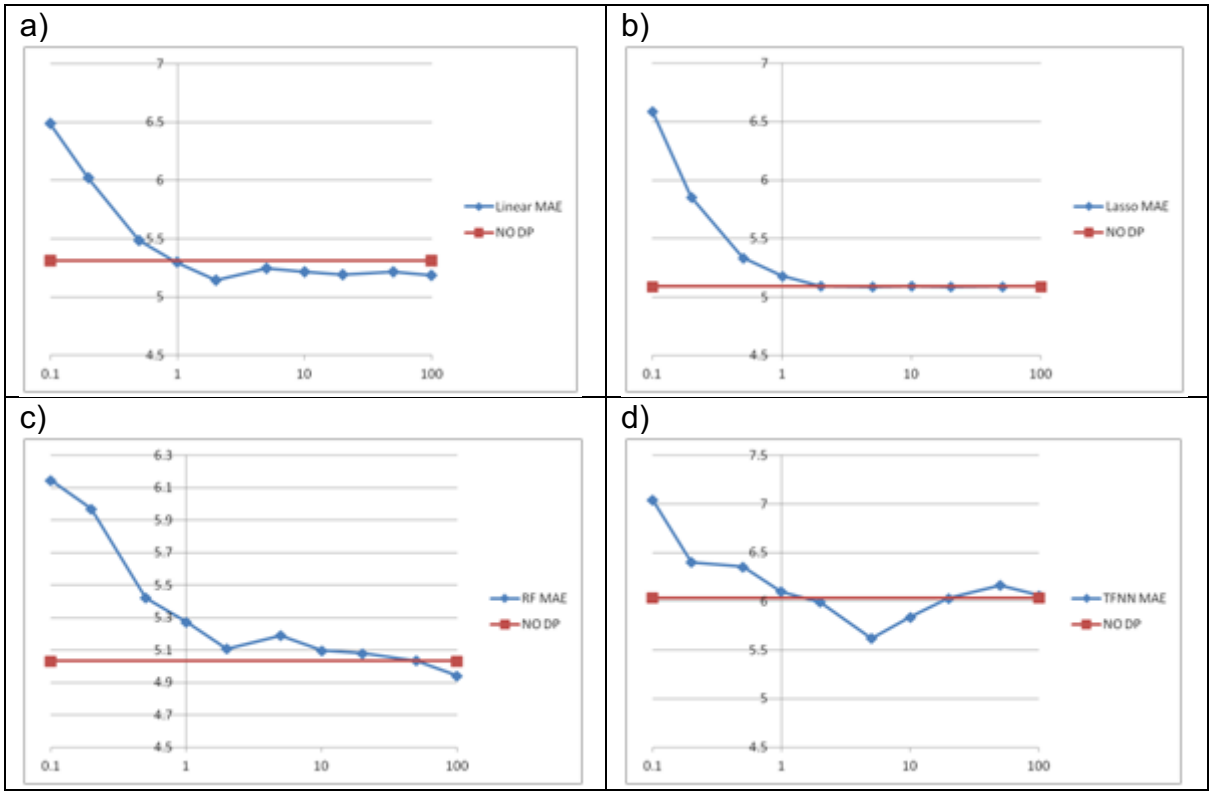


*Figure 2. MAE values of four characteristic regression models (LINEAR, LASSO, RF and TFNN)*

# 4 Model Training and Evaluation

This section presents the algorithms implemented for the different model training styles on the datasets obtained from the procedures in section 3 after missing value imputation respectively with and without differential privacy.

## 4.1 Training of Homomorphic Encrypted AI Models

Homomorphic encryption is used as a privacy measure to enable both training and prediction of machine learning models on encrypted patient data. The HE learning is related to training global models on a collection of homomorphically encrypted patient data. During training, domain-related knowledge is extracted from the data through machine learning models. As the resultant model is encrypted it can be further used for predictions on encrypted patient data.

For experimentation, a variant of a matrix-based homomorphic encryption scheme, called MORE (Matrix Operation for Randomization or Encryption) is employed to enable AI-based data processing on real-data. Following the MORE approach, a matrix-based symmetric secret key is generated upon which a numerical value is encrypted as a matrix and matrix algebra is employed to provide a fully homomorphic behaviour. All operations performed on ciphertext data are matrix-based operations, e.g., addition of plaintext scalars is equivalent to the addition of ciphertext matrices. The considered homomorphic encryption scheme is noise-free (unlimited number of operations can be performed on ciphertext data), non-deterministic (multiple encryptions of the same message and with the same key result in different ciphertexts) and is adapted to directly support floating-point arithmetic. Moreover, it allows a broader spectrum of operations to be performed over encrypted data, including non-linear functions.

Knowing that ultimately AI models break down to a series of repeating blocks of computations that rely on a limited set of simple operations over floating-point numbers and by leveraging the homomorphic property of the MORE scheme, the functionality of AI models can be extended to hold for operations on ciphertext data. Therefore, the MORE homomorphic encryption scheme enables the use of encrypted data for training and testing AI models, without requiring the content of the decrypted data.

Training of HE AI models implies that the training patient data is encrypted with a secret key that is never shared. This ensures that the model has access only to the ciphertext data. Hence, the plaintext data remains private at the Edge Nodes and only the corresponding encrypted form of the data is sent to the Cloud for processing. AI-based models are further trained directly on encrypted data by leveraging the properties of the MORE scheme. This results in a model that can be further applied to provide encrypted predictions, which can only be decrypted at the Edge Nodes.

Based on the considered problems, classical MLP (Multi-Layer Perceptron) models are trained and evaluated on encrypted data, according to the 10-fold cross-validation method. For the moment, the MORE library can operate only with the Stochastic Gradient Descent (SGD) optimizing algorithm [2], which proceeds by iteratively estimating the gradient descent from minibatches. Relevant hyperparameters of the

algorithm are the number of epochs, the cost function, and the learning rate. Hyperparameter optimization, starting from encrypted data, is difficult to achieve because it is typically set upon error analysis, which is itself a ciphertext due to the adopted cryptosystem. As the overall goal of the study was to assess the feasibility of the deep neural network to operate directly on HE data, i.e., demonstrate that the performance does not drop compared to the plaintext setting, we have chosen the hyperparameters through grid search and experimentation on plaintext data. The architectures of the trained models are listed in the below tables. Each table contains the number of layers, the number of neurons, and the activation functions.

1. BcBase-Anxiety – the model's architecture is listed in Table 2. It was trained for 250 epochs. The learning rate was chosen to be 0.05 and the cost function was Binary Cross Entropy.

*Table 2: BcBase-Anxiety model*

| Layer | # neurons | Activation function |
|---|---|---|
| Input | 97 | |
| Fully connected_1 | 50 | Matrix ReLU |
| Fully connected_2 | 50 | Matrix ReLU |
| Output | 1 | Matrix Sigmoid |

2. BcBase-Depression – the model's architecture is listed in Table 3. It was trained for 250 epochs. The learning rate was chosen to be 0.05 and the cost function was Binary Cross Entropy.

*Table 3. BcBase-Depression model*

| Layer | # neurons | Activation function |
|---|---|---|
| Input | 97 | |
| Fully connected_1 | 80 | Matrix ReLU |
| Fully connected_2 | 80 | Matrix ReLU |
| Output | 1 | Matrix Sigmoid |

3. BcBase-Insomnia – the model's architecture is listed in Table 4. It was trained for 100 epochs. The learning rate was chosen to be 0.001 and the cost function was Binary Cross Entropy.

*Table 4. BcBase-Insomnia model*

| Layer | # neurons | Activation function |
|---|---|---|
| Input | 97 | |
| Fully connected_1 | 40 | Matrix ReLU |
| Fully connected_2 | 40 | Matrix ReLU |
| Fully connected_3 | 40 | Matrix ReLU |
| Output | 1 | Matrix Sigmoid |

4. BcBase-Pain – the model's architecture is listed in Table 5. It was trained for 100 epochs. The learning rate was chosen to be 0.001 and the cost function was Binary Cross Entropy.

*Table 5. BcBase-Pain model*

| Layer | # neurons | Activation function |
|---|---|---|
| Input | 97 | |
| Fully connected_1 | 80 | Matrix ReLU |
| Fully connected_2 | 80 | Matrix ReLU |
| Fully connected_3 | 80 | Matrix ReLU |
| Output | 1 | Matrix Sigmoid |

5. ORB-30-36 – the model's architecture is listed in Table 6. It was trained for 100 epochs. The learning rate was chosen to be 0.01 and the cost function was Mean Squared Error.

*Table 6. ORB-30-36 model*

| Layer | # neurons | Activation function |
|---|---|---|
| Input | 96 | |
| Fully connected_1 | 100 | Matrix Tanh |
| Fully connected_2 | 100 | Matrix Tanh |
| Fully connected_3 | 100 | Matrix Tanh |
| Fully connected_4 | 100 | Matrix Tanh |
| Fully connected_5 | 100 | Matrix Tanh |
| Output | 1 | Identity |

6. ORB-30-60 – the model's architecture is listed in Table 7. It was trained for 100 epochs. The learning rate was chosen to be 0.01 and the cost function was Mean Squared Error.

*Table 7. ORB-30-60 model*

| Layer | # neurons | Activation function |
|---|---|---|
| Input | 96 | |
| Fully connected_1 | 100 | Matrix Tanh |
| Fully connected_2 | 100 | Matrix Tanh |
| Fully connected_3 | 100 | Matrix Tanh |
| Fully connected_4 | 100 | Matrix Tanh |
| Fully connected_5 | 100 | Matrix Tanh |
| Output | 1 | Identity |

7. ORB-30-120 – the model's architecture is listed in Table 8. It was trained for 100 epochs. The learning rate was chosen to be 0.01 and the cost function was Mean Squared Error.

*Table 8. ORB-30-120 model*

| Layer | # neurons | Activation function |
|---|---|---|
| Input | 96 | |
| Fully connected_1 | 100 | Matrix Tanh |
| Fully connected_2 | 100 | Matrix Tanh |
| Fully connected_3 | 100 | Matrix Tanh |
| Fully connected_4 | 100 | Matrix Tanh |
| Fully connected_5 | 100 | Matrix Tanh |
| Output | 1 | Identity |

8. ORB-54-60 – the model's architecture is listed in Table 9. It was trained for 100 epochs. The learning rate was chosen to be 0.01 and the cost function was Mean Squared Error.

*Table 9. ORB-54-60 model*

| Layer | # neurons | Activation function |
|---|---|---|
| Input | 124 | |
| Fully connected_1 | 100 | Matrix Tanh |
| Fully connected_2 | 100 | Matrix Tanh |
| Fully connected_3 | 100 | Matrix Tanh |
| Fully connected_4 | 100 | Matrix Tanh |
| Fully connected_5 | 100 | Matrix Tanh |
| Output | 1 | Identity |

9. ORB-54-120 – the model's architecture is listed in Table 10. It was trained for 100 epochs. The learning rate was chosen to be 0.01 and the cost function was Mean Squared Error.

*Table 10. ORB-54-120 model*

| Layer | # neurons | Activation function |
|---|---|---|
| Input | 124 | |
| Fully connected_1 | 100 | Matrix Tanh |
| Fully connected_2 | 100 | Matrix Tanh |
| Fully connected_3 | 100 | Matrix Tanh |
| Fully connected_4 | 100 | Matrix Tanh |
| Fully connected_5 | 100 | Matrix Tanh |
| Output | 1 | Identity |

10. ORB-108-120 – the model's architecture is listed in Table 11. It was trained for 100 epochs. The learning rate was chosen to be 0.01 and the cost function as Mean Squared Error.

*Table 11. ORB-108-120 model*

| Layer | # neurons | Activation function |
|---|---|---|
| Input | 158 | |
| Fully connected_1 | 100 | Matrix Tanh |
| Fully connected_2 | 100 | Matrix Tanh |
| Fully connected_3 | 100 | Matrix Tanh |
| Fully connected_4 | 100 | Matrix Tanh |
| Output | 1 | Identity |

The 10 models have been evaluated on the respective training datasets and the results of the evaluation are presented in section 4.5.3.

## 4.2 Training of Federated Learning AI Models

A federated model is a machine learning model collectively trained by two or more federated learning clients (ASCAPE edge nodes). Each federated learning client has its own dataset to train the model and those training datasets (patient data in our case) are not mutually exchanged, nor aggregated at some central location. Thus, federated learning can be considered as a secure-by-design privacy-preserving machine learning technique. The federated learning process is coordinated and synchronized by a federated learning server. In our case this component runs in the ASCAPE cloud. The main purpose of the federated learning server is to enable the exchange of models between federated learning clients during collective learning. ASCAPE federated models are stored in the ASCAPE cloud, which means that they are also available to ASCAPE edge nodes not participating in collective learning.

Two federated learning modes are present and supported by ASCAPE model learning components: incremental and semi-concurrent federated learning mode. The ASCAPE federated learning process is not predetermined with respect to those modes and the federated learning mode for a particular model is dynamically adapted in time according to the presence of federated learning clients. This means that federated learning in ASCAPE is initiated by federated learning clients and it is not driven by the federated learning server according to some predefined learning scheme.

In the incremental federated learning mode exactly one federated learning client creates or updates a model for a particular QoL indicator or intervention. Let us suppose that a federated learning client *C* wants to update the model for a QoL indicator/intervention *Q* on its dataset *D*. In the first step, *C* connects to the federated learning server to check whether a federated model for *Q* exists in the ASCAPE cloud. If the model for *Q* is not present in the ASCAPE cloud, *C* creates the first instance of the model and sends it to the ASCAPE cloud. Otherwise, *C* retrieves the model for *Q* from the cloud, updates it on *D*, and then sends the updated model back to the

ASCAPE cloud from where it is available to other ASCAPE edge nodes either for further updating or for inference (making predictions).

The ASCAPE federated learning for a particular predictive QoL model always starts in the incremental mode and it may switch to the semi-concurrent learning mode when the federated learning server detects that two or more federated learning clients want to create or update the same model. In the semi-concurrent learning mode, the federated learning server performs the following steps in a loop:

- it waits for all federated learning clients creating/updating the model to send their models after performing one learning round (one iteration in the learning algorithm),
- the received models are averaged into one aggregated model,
- the aggregated model is sent back to the federated learning clients for the next learning round.

The symbiosis between the incremental and semi-concurrent federated learning mode is achieved by the fact that model learning by federated learning clients is done iteratively in learning rounds. To make the switch from the incremental to the semi-concurrent learning mode, a federated learning client currently working in the incremental mode after each learning round checks with the federated learning server whether some other federated learning client wants to create/update the same model. As response from the federated learning server, it either receives a message to continue in the incremental mode or a message to send the model to the server for the federated averaging. In the second case, the client receives the aggregated model from the server and then it continues learning in the semi-concurrent mode.

As already emphasized, the main premise of federated learning is that the learning algorithm operates iteratively in learning rounds. Model parameters are refined with each learning round to better fit the training dataset. Additionally, federated models have to be amenable to some kind of averaging. Neural networks are the most natural choice for federated models for two reasons: (1) neural networks can be easily averaged by averaging edge weights and biases, and (2) they are able to effectively capture non-linear trends and relationships in training data. However, this choice comes with the price of a large parameter space, especially in the case of deep neural networks. Additionally, the learning of effective neural networks often requires computationally demanding fine grained tuning mechanisms for its hyperparameters.

In the most common form, a neural network is a sequence of layers each containing a certain number of neurons or nodes. All neurons from the *k*-th layer are connected to all neurons from the *(k+1)*-th layer via weighted edges. The weights of neural network edges together with biases associated to each node constitute the set of real-valued model parameters that are learned on a given training dataset. The nodes in the first layer are input values for a feed-forward mechanism of the neural network, the last layer contains neural network output values (predictions for input values), while nodes in all other intermediate layers can be considered as hidden variables through which input values are transformed to obtain output values. Let us assume that we have a dataset *D* composed of instances (data points, e.g., one particular patient) in form $< X, y >$ where $X$ is the vector of input variables and $y$ is the vector of output

variables (e.g., $X$ may be variables reflecting patient state and his/her medical conditions, while $y$ in our case is some QoL indicator/intervention). The feed-forward mechanism of the neural network for an input instance *p* (not necessarily from *D*) can be described recursively as

$$x_i^{(l)}(p) = W_i^{(l)} y^{(l-1)}(p) + b_i^{(l-1)}$$
$$y_i^{(l)}(p) = \sigma\left(x_i^{(l)}(p)\right)$$
$$y_i^{(0)}(p) = X_i(p)$$

where $X_i(p)$ is the value of $i$-th feature in $X$ for *p*, $y_i^{(l)}$ is the value of the output of the $i$-th neuron in the $l$-th layer, $x_i^{(l)}$ is the value of the input to the $i$-th neuron in the $l$-th layer, $W_i^{(l)}$ is the vector containing weights of the in-coming edges to the *i*-th neuron in the $l$-th layer, $b_i^{(l)}$ is the value of the bias associated to the $i$-th neuron in the $l$-th layer and $\sigma$ is an activation function (e.g., sigmoid or ReLU). The model parameters (neural network edge weights and biases) are learned by minimizing a loss function on *D*. The loss function quantifies the difference between neural network output values for instances in *D* and their real $y$ values given in *D*. The loss function is minimized in a given number of epochs (one epoch is one learning round considering all instances in *D*), where model parameters are updated after processing batches each containing *B* instances (*B* is the batch size).

The architecture of a neural network is determined by the number of layers and the number of neurons per layer, but also by the type of predictive problem solved by the neural network. In ASCAPE we deal with two different predictive problems: regression (predicting a numerical value, e.g., predicting the LISAT-based QoL index) and $n$-ary classification (predicting a categorical value from a predefined set of $n$ categorical values, e.g., predicting a QoL intervention from a set of possible interventions). Additionally, we have a special case of classification known as binary classification where $n = 2$ (positive and negative category, e.g., the presence or absence of some QoL condition). Thus, in ASCAPE we have three different neural network types:

1. Neural networks for regression. The last layer of such neural networks contains exactly one node activated by the linear function. The mean squared error (MSE) is used as the loss function to determine model parameters.
2. Neural networks for n-ary classification. Here the last layer consists of n nodes activated by the softmax function. This means that a node in the last layer represents the probability of the corresponding category. The categorical cross-entropy is used as the loss function when training neural networks of this type.
3. Neural networks for binary classification. The last layer contains one node activated by the sigmoid function. The value of the output node higher than 0.5 indicates the positive category, while values lower than 0.5 indicate the negative category. Model parameters are determined using the categorical cross-entropy function.

Nodes in hidden layers of all above given neural network types are activated by the ReLU activation function. In our implementation of ASCAPE neural networks, we also

consider various mechanisms to prevent overfitting (dropout and various regularization strategies, e.g., kernel, bias and activation regularization).

The training of federated neural networks is enabled by the ASCAPE edge node core machine learning services that are based on the TensorFlow library [3] (for a detailed description of the ASCAPE edge node core machine learning services please see Deliverable D3.1 – Cancer-care predictive analytics and decision-making services: proof of concept demonstration). We also implemented fully functional prototypes of the federated learning server and clients, which are also described in Deliverable D3.1 – Cancer-care predictive analytics and decision-making services: proof of concept demonstration. However, to foster experimentation with federated machine learning models and enable their evaluation for an arbitrary number of ASCAPE edge nodes, we also developed a federated learning simulator. This simulator is realized in the `fedsim` Python module relying on functionalities provided by `tfnn` module from the ASCAPE edge node core machine learning services (a detailed description of `tfnn` can be found in Deliverable D3.1 – Cancer-care predictive analytics and decision-making services: proof of concept demonstration).

The `fedsim` module defines the following classes:

- `DataSplitter` – a class performing splitting of a dataset into *k* size-balanced parts that can be used either as training or test datasets for simulated ASCAPE edge nodes.
- `ModelFactory` – a base class enabling core functionalities to define and setup a federated TensorFlow neural network models provided by `tfnn` module.
- `RegModelFactory` – a class extending `ModelFactory` used to create a sequence of federated TensorFlow neural networks for regression (one network per simulated ASCAPE edge nodes).
- `ClModelFactory` – a class extending `ModelFactory` enabling the creation of federated TensorFlow neural networks for *n*-ary classification.
- `BinClModelFactory` – similar to the two previous classes, this class realizes the factory for federated TensorFlow neural networks performing binary classification.
- `FedModel` – a base class defining core functionalities when simulating federated model learning.
- `FedIncModel` – a class derived from `FedModel` simulating federated learning in the incremental learning mode for an arbitrary number of ASCAPE edge nodes.
- `FedConModel` – a class derived from `FedModel` simulating federated learning in the semi-concurrent learning mode for an arbitrary number of ASCAPE edge nodes.

```
from loader import CSVDataset
from fedsim import RegModelFactory, FedIncModel
from tfnnie import RAMTFNNRegressor
from evalm import EvalRegressionModel

# load training and test datasets
trainData = CSVDataset("ORB-30-36-Training.csv")
X_train, y_train = trainData.getData()
testData = CSVDataset("ORB-30-36-Test.csv")
X_test, y_test = testData.getData()

# simulate from 1 to 4 edge nodes
for numEdgeNodes in range(1, 5):
    # train federated model
    regModel = RegModelFactory(
        numPredictors=X_train.shape[1],
        hiddenLayersStructure=[40] * 10)
    fedModel = FedIncModel(X_train, y_train, numEdgeNodes, regModel)
    fedNeuralNetwork = fedModel.train(numEpochs=200, batchSize=32)

    # evaluate the model
    model = RAMTFNNRegressor(fedNeuralNetwork)
    em = EvalRegressionModel(model, X_test, y_test)
    print(numEdgeNodes, "edge nodes, MAE = ", em.getMAE())
svc@svcpc:~/ASCAPE/coreML$ python3 fedsimdemo1.py
1 edge nodes, MAE =  6.4087252321496475
2 edge nodes, MAE =  6.397379411005341
3 edge nodes, MAE =  7.120239561637946
4 edge nodes, MAE =  7.022595127071955
svc@svcpc:~/ASCAPE/coreML$
```

*Figure 3. A demonstration of the ASCAPE federated learning simulator module for training a regression model in the incremental federated learning mode on the Orebro dataset for different numbers of edge nodes.*

Figure 3 shows a simple simulator based on `fedsim` module for training a federated neural network doing regression for different number of ASCAPE edge nodes (from one to four). The federated neural network has ten layers, each having 40 nodes. The network is trained from patient data in the ORB-dataset in the incremental learning mode for 200 epochs per simulated node and on batches of size 32. The trained federated model is then evaluated using existing modules from the ASCAPE edge node machine learning services (`RAMTFNNRegressor` is the in-memory inference engine for TensorFlow neural networks performing regression; `EvalRegression-Model` provides various metrics for evaluating regression models and here it is used to compute MAE – the mean absolute error of the model).

An example showing how to simulate federated learning in the semi-concurrent mode is given in Figure 4. Similar to the previous example, one to four ASCAPE edge nodes are simulated, but this time a federated neural network for binary classification is trained on splits obtained from the BcBase dataset.

```
from loader import CSVDataset
from fedsim import BinClModelFactory, FedConModel
from tfnnie import RAMTFNNBinClassifier
from evalm import EvalBinClassificationModel

# load training and test datasets
trainData = CSVDataset("BcBase-Depression-Training.csv")
X_train, y_train = trainData.getData()
testData = CSVDataset("BcBase-Depression-Test.csv")
X_test, y_test = testData.getData()

# simulate from 1 to 4 edge nodes
for numEdgeNodes in range(1, 5):
    # train federated model
    clModel = BinClModelFactory(
        numPredictors=X_train.shape[1],
        hiddenLayersStructure=[40] * 10)
    fedModel = FedConModel(X_train, y_train, numEdgeNodes, clModel)
    fedNeuralNetwork = fedModel.train(numEpochs=200, batchSize=512)

    # evaluate the model
    model = RAMTFNNBinClassifier(fedNeuralNetwork)
    em = EvalBinClassificationModel(model, X_test, y_test)
    print(numEdgeNodes, "edge nodes, accuracy = ", em.getAccuracy())
svc@svcpc:~/ASCAPE/coreML$ python3 fedsimdemo2.py
1 edge nodes, accuracy =  0.7012644889357218
2 edge nodes, accuracy =  0.6949420442571127
3 edge nodes, accuracy =  0.6996838777660696
4 edge nodes, accuracy =  0.6981032665964173
svc@svcpc:~/ASCAPE/coreML$
```

*Figure 4. A demonstration of the ASCAPE federated learning simulator module for training a binary classification model in the semi-concurrent federated learning mode on the BcBase dataset for different numbers of edge nodes.*

## 4.3 Training of Local AI Models

In contrast to federated (global) models, local models are non-collectively learned predictive models, trained considering only datasets available at a particular ASCAPE edge node. An ASCAPE edge node uses its own local models for making predictions instead of the corresponding global models when the global models exhibit a poor performance (low accuracy or high error) on training datasets present on that node. In this way, ASCAPE predictive components will be able to deliver accurate predictions for ASCAPE edge node covering some specific, uncommon and rarely represented cohorts of patients.

Local models are not restricted to iterative optimization algorithms. Secondly, they can be easily retrained in order to avoid potential negative effects of incremental updates, such as catastrophic forgetting (the tendency of the model to "forget" previously learned trends and relationships when updated on new data batches).

Besides neural networks, we consider the following machine learning algorithms for training local models performing classification:

- SVM: support vector machine classifier
- NB: Naïve Bayes classifier
- KNN: K-nearest neighbours' classifier
- DT: Decision-tree classifier
- RF: random forest classifier.

Support vector machine classifiers are based on the idea of using linear models to identify non-linear boundaries of categories. This is achieved by transforming data instances into a new higher-dimensional space using a non-linear mapping. Quadratic programming algorithms are then employed in the higher-dimensional space to determine maximum margin hyperplanes separating instances from different categories. The Naive Bayes classification is a probabilistic classification algorithm. This algorithm returns the most probable category for a given data instance, where category probabilities are computed using conditional probability estimates derived from the training dataset under the assumption that features describing data instances are conditionally independent. KNN is an instance-based (lazy) classifier: the category for a given data instance is determined from categories of the $K$ closest instances from the training dataset determined using some distance function (e.g., Euclidean or Manhattan distance). In our experiments we have set $K$ to 10 (the same $K$ value is also used for KNN regression that is described later). Decision tree classifiers make predictions according to decision trees constructed from training dataset by a divide-and-conquer algorithm guided by some information-theoretic measure (e.g., information gain or Gini impurity). The used information-theoretic measure indicates the best feature to recursively split the training dataset into parts from which subtrees are constructed. A random forest is an ensemble of decision trees learned from bootstrapped samples of the training data. The random forest algorithm employs so-called feature bagging to determine a random subset of features for learning individual decision trees. The category for a given input instance is then the most frequent category derived from decision trees in the ensemble. In our experiments we have used random forest ensembles containing 10 trees (the same applies also for random forest regression).

For regression problems, local models are trained by one of the following machine learning algorithms:

- LINEAR: linear regression
- RIDGE: ridge regression
- LASSO: lasso regression
- ELASTICN: elastic net regression
- KRIDGE: kernel ridge regression

- SVM: support vector machine regression
- RF: random forest regression
- KNN: K-nearest neighbours regression
- ADAB: AdaBoost regression.

The linear regression algorithm determines coefficients of a linear model by minimizing the residual sum of squares between real values of the outcome variable and predictions obtained by the linear approximation. Ridge, Lasso and ElasticNet find a linear model by minimizing the residual sum of squares with incorporated regularization penalties: Ridge uses the L2 regularization penalty, Lasso uses the L1 regularization penalty, while ElasticNet uses both L1 and L2 regularization penalties. Kernel ridge regression performs ridge regression in a space obtained by a non-linear mapping of the training dataset. SVM, RF and KNN are adaptations of the corresponding classification algorithms for regression tasks. AdaBoost is a meta-learning algorithm boosting an arbitrary regression method (decision-tree based regressor in our case) towards data instances having high prediction errors. The algorithm starts by assigning equal weights to all instances in the training dataset. Those weights indicate how hard it is to predict the outcome variable for a particular instance, i.e., instances with higher prediction errors have higher weights. Some base regression method is employed to form an initial regression model. Instances in the training datasets are reweighted according to errors obtained by the trained regression model. Then, a new regression model is learned using a loss function that is pondered with weights. The two steps are repeated either for a fixed number of iterations or until the error of the model becomes acceptable. Predictions by AdaBoost are made considering all sequentially trained regression models. As the baseline for evaluating above-mentioned regression models we use the so-called DUMMY regression model. The DUMMY model always predicts the same value: the mean of the outcome variable computed from the training dataset.

The training of local machine learning models is enabled by the ASCAPE edge node machine learning services based on the scikit-learn library. The implementation of relevant modules is described in Deliverable D3.1 – Cancer-care predictive analytics and decision-making services: proof of concept demonstration.

## 4.4 Evaluation Methodology and Metrics

For the evaluation of models' performance, we used 10-fold cross validation as described in section 3.1.

In order to assess the performance of models from different aspects, we used various measures. For the problem of binary classification we used: accuracy ($ACC$), F1 score ($F1$), precision of the positive class ($Prec^+$), recall of the positive class ($Rec^+$), precision of the negative class ($Prec^-$), recall of the negative class ($Rec^-$), macro-averaged precision ($Prec$), and macro-averaged recall ($Rec$). All of the mentioned measures are listed below, where $tp$ stands for true positives, $tn$ for true negatives, $fp$ for false positives, and $fn$ for false negatives.

$$ACC = \frac{tp + tn}{tp + tn + fp + fn}$$
$$F1 = \frac{2tp}{2tp + fp + fn}$$
$$Prec^{+} = \frac{tp}{tp + fp}$$
$$Rec^{+} = \frac{tp}{tp + fn}$$
$$Prec^{-} = \frac{tn}{tn + fn}$$
$$Rec^{-} = \frac{tn}{tn + fp}$$
$$Prec = \frac{Prec^{+} + Prec^{-}}{2}$$
$$Rec = \frac{Rec^{+} + Rec^{-}}{2}$$

On the other side, for the evaluation of the regression models we used: mean absolute error ($MAE$), mean squared error ($MSE$), coefficient of determination ($R2$), and the Person's correlation coefficient ($PC$). All the measures are presented below, where $n$ is the number of dataset instances, $x_i$ is the target attribute value of the $i$-th instance, $y_i$ is the predicted value of the target attribute of the $i$-th instance, $\bar{x}$ is the mean of the target attribute's values, and $\bar{y}$ is the mean value of all predicted values for the target attribute.

$$MAE = \frac{\sum_{i=1}^{n} |x_i - y_i|}{n}$$

$$MSE = \frac{\sum_{i=1}^{n} (x_i - y_i)^2}{n}$$

$$SS_{tot} = \sum_{i=1}^{n} (x_i - \bar{x})^2, \qquad SS_{res} = \sum_{i=1}^{n} (x_i - y_i)^2, \qquad R2 = 1 - \frac{SS_{res}}{SS_{tot}}$$

$$PC = \frac{\sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n} (x_i - \bar{x})^2} \sqrt{\sum_{i=1}^{n} (y_i - \bar{y})^2}}$$

In order to evaluate performance of different missing values imputation algorithms, we trained regressors on datasets obtained by both the algorithms, and then we compared their performance. Namely, we wanted to verify whether some missing values imputation algorithm helps to achieve better performance of regressors.

The influence of differential privacy techniques on model's performance is evaluated by training models on different data: 1) on original dataset, 2) on datasets that are

generated with different values of $\varepsilon$ DP parameter. The comparison of all these models suggests how DP influences predictions.

Similarly, performance of homomorphic encrypted models is compared with performance of models without homomorphic encryption.

## 4.5 Evaluation Results

At the current stage of project realization, we do not have compatible datasets (datasets containing the same features) coming from different data sources (different clinics corresponding to different ASCAPE edge nodes) to train and evaluate real federated (global) models. Thus, the focus of this deliverable, from the ASCAPE edge node perspective, is on the training and evaluation of local models and simulated federated models. Since simulated federated models are trained and evaluated on the same amount of data as local models, it is natural to expect that locally-trained models (local models) will exhibit better performance than decentralized models (simulated federated models).

### 4.5.1 Evaluation of locally-trained models

The evaluation of locally-trained binary classification models on the BcBase datasets is summarized in Table 12 to Table 15**.** Each table shows accuracy ($ACC$), F1 score, macro-averaged precision ($Prec$), macro-averaged recall ($Rec$), precision and recall of the positive class ($Prec^+$ and $Rec^+$) and precision and recall of negative class ($Prec^-$ and $Rec^-$) for the corresponding BcBase dataset. To the positive class belongs patients experiencing anxiety, depression, insomnia and pain after cancer diagnosis, respectively per dataset, while patients without those symptoms are in the negative class (again, respectively per dataset). The evaluation was performed on a PC with Intel® Core™ i5-2320 CPU @ 3.00GHz × 4, 12 GB RAM memory, and the Ubuntu operating system.

SVM has the highest accuracy on three out of four BcBase datasets: BcBase-Anxiety, BcBase-Depression and BcBase-Pain. However, it can be observed that this classification model on those datasets has zero precision and zero recall for the positive class. In other words, SVM totally fails for patients experiencing anxiety, depression and pain after diagnosis since it dominantly predicts the negative class (no negative QoL related symptoms). Thus, the highest accuracy of SVM for those three datasets is the consequence of the class imbalance in the BcBase datasets (approximately 70% of the patients belong to the negative class and 30% to the positive class). Therefore, accuracy is a biased measure towards the negative class and it should not be used to compare different classification models. F1 score in the case of the BcBase datasets is more adequate measure to compare different models since it is harmonic mean of macro-averaged precision and macro-averaged recall (i.e., it takes into account precision and recall of both classes). Not surprisingly, the SVM exhibits the lowest F1 score on those datasets where it has the highest accuracy. KNN has the lowest F1 score on BcBase-Insomnia.

NB has the highest F1 score on three out of four BcBase datasets (anxiety, depression and insomnia). The best model on the fourth BcBase dataset (pain) is DT, but the F1 score of NB on this dataset is very close to the F1 score of DT. Therefore, it can be concluded that NB is the best performing locally trained model for the BcBase datasets. For all models on all BcBase datasets we have, we identified that they better perform for the negative class than for the positive class.

*Table 12. Evaluation of binary classification models on BcBase-Anxiety.*

| | $ACC$ | $F1$ | $Prec$ | $Rec$ | $Prec^+$ | $Rec^+$ | $Prec^-$ | $Rec^-$ |
|---|---|---|---|---|---|---|---|---|
| RF | 0.673 | 0.484 | 0.535 | 0.514 | 0.366 | 0.112 | 0.704 | 0.916 |
| SVM | 0.698 | 0.411 | 0.349 | 0.500 | 0.000 | 0.000 | 0.698 | 1.000 |
| **NB** | **0.629** | **0.552** | **0.553** | **0.552** | **0.379** | **0.354** | **0.728** | **0.749** |
| KNN | 0.682 | 0.458 | 0.529 | 0.507 | 0.357 | 0.066 | 0.701 | 0.949 |
| DT | 0.583 | 0.511 | 0.511 | 0.511 | 0.317 | 0.329 | 0.705 | 0.693 |

*Table 13. Evaluation of binary classification models on BcBase-Depression.*

| | $ACC$ | $F1$ | $Prec$ | $Rec$ | $Prec^+$ | $Rec^+$ | $Prec^-$ | $Rec^-$ |
|---|---|---|---|---|---|---|---|---|
| RF | 0.677 | 0.473 | 0.524 | 0.509 | 0.342 | 0.093 | 0.706 | 0.924 |
| SVM | 0.702 | 0.413 | 0.351 | 0.500 | 0.000 | 0.000 | 0.702 | 1.000 |
| **NB** | **0.566** | **0.534** | **0.543** | **0.551** | **0.345** | **0.514** | **0.741** | **0.588** |
| KNN | 0.688 | 0.462 | 0.536 | 0.509 | 0.366 | 0.067 | 0.706 | 0.951 |
| DT | 0.589 | 0.515 | 0.515 | 0.515 | 0.318 | 0.333 | 0.712 | 0.697 |

*Table 14. Evaluation of binary classification models on BcBase-Insomnia.*

| | $ACC$ | $F1$ | $Prec$ | $Rec$ | $Prec^+$ | $Rec^+$ | $Prec^-$ | $Rec^-$ |
|---|---|---|---|---|---|---|---|---|
| RF | 0.538 | 0.526 | 0.535 | 0.532 | 0.526 | 0.394 | 0.544 | 0.671 |
| SVM | 0.541 | 0.533 | 0.539 | 0.537 | 0.529 | 0.427 | 0.549 | 0.647 |
| **NB** | **0.555** | **0.554** | **0.555** | **0.554** | **0.539** | **0.529** | **0.570** | **0.580** |
| KNN | 0.521 | 0.502 | 0.516 | 0.514 | 0.503 | 0.343 | 0.529 | 0.686 |
| DT | 0.516 | 0.515 | 0.515 | 0.515 | 0.497 | 0.500 | 0.534 | 0.531 |

*Table 15. Evaluation of binary classification models on BcBase-Pain.*

| | $ACC$ | $F1$ | $Prec$ | $Rec$ | $Prec^+$ | $Rec^+$ | $Prec^-$ | $Rec^-$ |
|---|---|---|---|---|---|---|---|---|
| RF | 0.698 | 0.486 | 0.554 | 0.518 | 0.386 | 0.098 | 0.722 | 0.937 |
| SVM | 0.714 | 0.417 | 0.357 | 0.500 | 0.000 | 0.000 | 0.714 | 1.000 |
| NB | 0.530 | 0.517 | 0.553 | 0.564 | 0.333 | 0.643 | 0.773 | 0.484 |
| KNN | 0.699 | 0.457 | 0.528 | 0.506 | 0.338 | 0.055 | 0.717 | 0.957 |
| **DT** | **0.604** | **0.522** | **0.522** | **0.522** | **0.316** | **0.333** | **0.727** | **0.712** |

The results of the evaluation of locally-trained regression models on the ORB datasets are shown in Table 16 to Table 21 (one table per dataset, the best performing models are bolded). Each table presents the mean absolute error (MAE), the mean squared error (MSE), the coefficient of determination (R2) and the Person's correlation coefficient (PC) for the examined models, including also the DUMMY model as the

baseline. The Person's correlation coefficient between real target values and values predicted by DUMMY is undefined (NaN) since DUMMY always predicts the same value. The best model (the lowest MAE and MSE, the highest R2 and PC) for ORB-30-36 is RF. For all others ORB datasets, the best performing model is LASSO. KNN is the worst performing model on all ORB datasets (predictions made by this model are even more erroneous than predictions made by DUMMY). Excluding KNN, all others considered models exhibit smaller prediction errors (and, consequently, higher correlations with real values) compared to DUMMY except in one case: DUMMY is better than linear regression on ORB-30-120. The prediction errors of the best performing model are in the range [4.84, 6.47], which is an acceptable level of prediction errors taking into account that the target variable (the LISAT QoL index) is in the range [11, 66].

*Table 16. Evaluation of regression models on ORB-30-36.*

| | $MAE$ | $MSE$ | $R2$ | $PC$ |
|---|---|---|---|---|
| DUMMY | 6.541 | 68.579 | -0.001 | NaN |
| LINEAR | 5.311 | 49.914 | 0.273 | 0.559 |
| RIDGE | 5.100 | 44.150 | 0.358 | 0.604 |
| LASSO | 5.089 | 45.001 | 0.346 | 0.592 |
| ELASTICN | 5.126 | 46.245 | 0.328 | 0.581 |
| KRIDGE | 5.147 | 45.193 | 0.343 | 0.594 |
| SVM | 6.519 | 69.469 | -0.014 | 0.015 |
| **RF** | **5.051** | **43.872** | **0.361** | **0.605** |
| KNN | 6.720 | 71.616 | -0.044 | 0.034 |
| ADAB | 5.376 | 47.606 | 0.306 | 0.569 |

*Table 17. Evaluation of regression models on ORB-30-60.*

| | $MAE$ | $MSE$ | $R2$ | $PC$ |
|---|---|---|---|---|
| DUMMY | 6.890 | 78.402 | -0.001 | NaN |
| LINEAR | 6.129 | 71.126 | 0.086 | 0.429 |
| RIDGE | 5.925 | 66.454 | 0.147 | 0.464 |
| **LASSO** | **5.886** | **62.182** | **0.205** | **0.463** |
| ELASTICN | 5.913 | 62.530 | 0.201 | 0.460 |
| KRIDGE | 5.958 | 67.246 | 0.137 | 0.459 |
| SVM | 6.773 | 80.572 | -0.028 | 0.039 |
| RF | 6.015 | 62.576 | 0.202 | 0.459 |
| KNN | 6.968 | 80.045 | -0.023 | 0.076 |
| ADAB | 6.542 | 67.651 | 0.135 | 0.436 |

*Table 18. Evaluation of regression models on ORB-30-120.*

| | ***MAE*** | ***MSE*** | ***R2*** | ***PC*** |
|---|---|---|---|---|
| DUMMY | 6.909 | 81.277 | -0.004 | NaN |
| LINEAR | 7.003 | 88.397 | -0.113 | 0.287 |
| RIDGE | 6.652 | 76.523 | 0.054 | 0.319 |
| **LASSO** | **6.478** | **72.928** | **0.100** | **0.322** |
| ELASTICN | 6.504 | 73.370 | 0.095 | 0.318 |
| KRIDGE | 6.750 | 78.270 | 0.032 | 0.310 |
| SVM | 6.871 | 82.172 | -0.015 | 0.044 |
| RF | 6.685 | 76.875 | 0.047 | 0.277 |
| KNN | 7.133 | 84.482 | -0.045 | 0.061 |
| ADAB | 6.934 | 80.517 | 0.000 | 0.246 |

*Table 19. Evaluation of regression models on ORB-54-60.*

| | ***MAE*** | ***MSE*** | ***R2*** | ***PC*** |
|---|---|---|---|---|
| DUMMY | 6.890 | 78.402 | -0.001 | NaN |
| LINEAR | 5.238 | 57.528 | 0.264 | 0.620 |
| RIDGE | 5.070 | 54.302 | 0.306 | 0.638 |
| **LASSO** | **4.840** | **45.122** | **0.425** | **0.659** |
| ELASTICN | 4.859 | 45.216 | 0.425 | 0.660 |
| KRIDGE | 5.115 | 56.217 | 0.282 | 0.633 |
| SVM | 6.772 | 80.592 | -0.029 | 0.033 |
| RF | 5.009 | 46.034 | 0.413 | 0.649 |
| KNN | 6.906 | 79.092 | -0.013 | 0.113 |
| ADAB | 5.702 | 52.494 | 0.330 | 0.620 |

*Table 20. Evaluation of regression models on ORB-54-120.*

| | ***MAE*** | ***MSE*** | ***R2*** | ***PC*** |
|---|---|---|---|---|
| DUMMY | 6.909 | 81.277 | -0.004 | NaN |
| LINEAR | 6.899 | 87.935 | -0.107 | 0.319 |
| RIDGE | 6.356 | 72.210 | 0.110 | 0.389 |
| **LASSO** | **6.180** | **68.472** | **0.157** | **0.405** |
| ELASTICN | 6.216 | 68.754 | 0.153 | 0.405 |
| KRIDGE | 6.492 | 75.187 | 0.073 | 0.370 |
| SVM | 6.859 | 81.953 | -0.013 | 0.059 |
| RF | 6.357 | 69.279 | 0.142 | 0.405 |
| KNN | 7.128 | 84.079 | -0.043 | 0.070 |
| ADAB | 6.612 | 74.280 | 0.075 | 0.325 |

*Table 21. Evaluation of regression models on ORB-108-120.*

| | ***MAE*** | ***MSE*** | ***R2*** | ***PC*** |
|---|---|---|---|---|
| DUMMY | 6.909 | 81.277 | -0.004 | NaN |
| LINEAR | 6.524 | 74.276 | 0.088 | 0.466 |
| RIDGE | 5.977 | 64.100 | 0.215 | 0.517 |
| **LASSO** | **5.437** | **55.220** | **0.324** | **0.574** |
| ELASTICN | 5.448 | 55.277 | 0.323 | 0.576 |
| KRIDGE | 6.155 | 67.746 | 0.169 | 0.496 |
| SVM | 6.875 | 82.072 | -0.014 | 0.067 |
| RF | 5.635 | 57.028 | 0.299 | 0.556 |
| KNN | 7.033 | 82.971 | -0.025 | 0.115 |
| ADAB | 5.777 | 57.502 | 0.294 | 0.551 |

Since LASSO is the best performing model on the ORB datasets, we have computed how much it is better than DUMMY. The results are shown in Figure 5. It can be observed that LASSO improvements over DUMMY are significant for short term QoL predictions (30-36, 54-60, 108-120) ranging from 20% to 30% in the reduction of MAE scores. For medium term QoL predictions (30-60, 54-120) the improvements are between 10% and 15%. As expected, the lowest improvement is for long term QoL predictions on ORB-30-120 where the reduction of MAE scores is slightly higher than 5%.

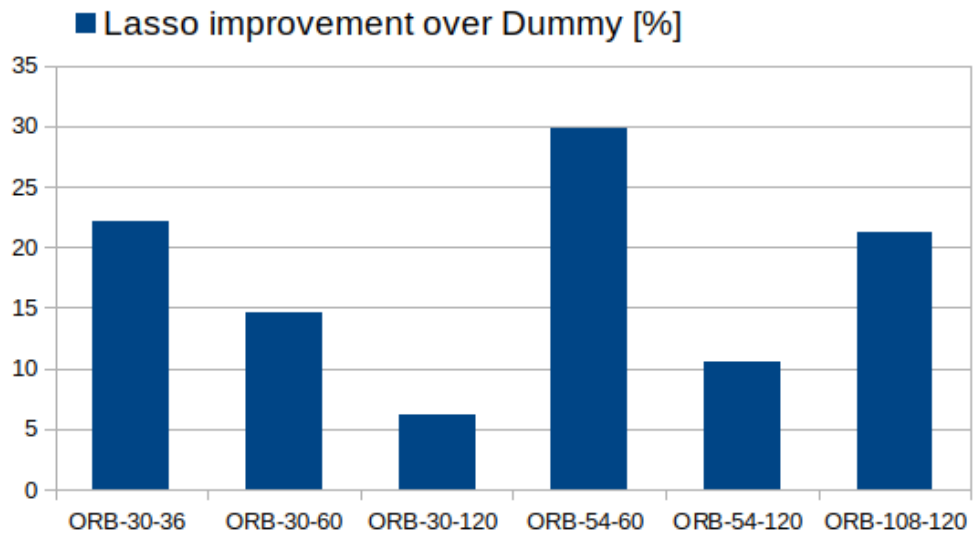


*Figure 5. Lasso improvement over Dummy on ORB datasets in percentages reflecting the reduction of MAE scores.*

We also examined how the missing value inference algorithm affects the accuracy of examined classification and regression models. The comparison of F1 scores for binary classifiers trained after missing value inference performed by iterative and simple missing value imputers is given in Table 22 (p. 36). As depicted, differences in F1 scores are almost absent. We also see a similar result for regression models. Figure 6 shows MAE scores for the best performing model on ORB datasets (LASSO) and it can be seen that differences between the iterative and simple imputer are

insignificant. Therefore, we conclude that missing value inference for BcBase and ORB datasets by simple methods can be as effective as missing value inference done by regression methods.

*Table 22. Comparison of F1 scores of binary classifiers trained on BcBase datasets obtained after missing value inference by the iterative (Iter) and simple (Sim) missing value imputer.*

| | **Anxiety** | | **Depression** | | **Insomnia** | | **Pain** | |
|---|---|---|---|---|---|---|---|---|
| | Iter | Sim | Iter | Sim | Iter | Sim | Iter | Sim |
| RF | 0.484 | 0.488 | 0.473 | 0.472 | 0.526 | 0.517 | 0.486 | 0.484 |
| SVM | 0.411 | 0.411 | 0.413 | 0.413 | 0.533 | 0.533 | 0.417 | 0.417 |
| NB | 0.552 | 0.553 | 0.534 | 0.529 | 0.554 | 0.552 | 0.517 | 0.520 |
| KNN | 0.458 | 0.458 | 0.462 | 0.459 | 0.502 | 0.501 | 0.457 | 0.455 |
| DT | 0.511 | 0.529 | 0.515 | 0.518 | 0.515 | 0.519 | 0.522 | 0.524 |

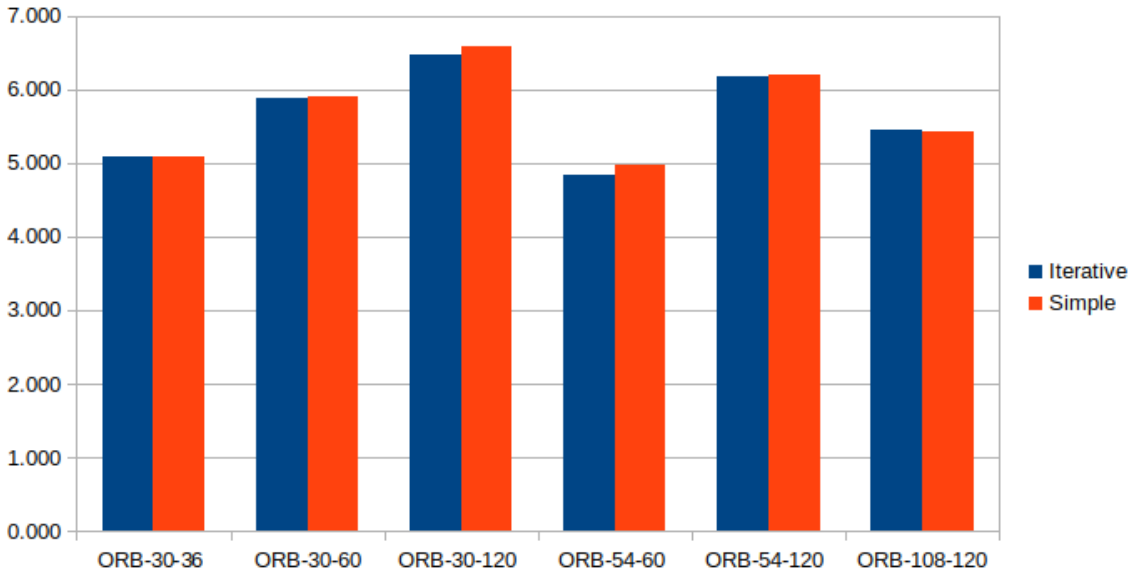


*Figure 6. Comparison of MAE for Lasso regressors (the best performing model on ORB datasets) trained on ORB datasets obtained after missing value inference by the iterative and simple missing value imputer.*

### 4.5.2 Evaluation of simulated federated models

In experiments with simulated federated models, we have used one neural network architecture for the BcBase datasets and second for the ORB dataset. A preliminary investigation, in which we have varied the number of hidden neural network layers between 1 and 10 and the batch size in the set {16, 64, 128, 256, 512}, showed that swallow neural networks (a small number of hidden layers) trained with a large batch size are more suitable for the BcBase datasets, while deeper neural networks (a larger number of hidden layers) trained with a small batch size result with better predictive models for the ORB datasets. We have simulated from 2 to 4 ASCAPE edge nodes training models in both incremental and semi-concurrent federated learning mode. Performance metrics for simulated federated models were obtained in the same way as for locally-trained local models (by the 10-fold cross validation procedure). The

training and evaluation process was conducted on a PC with Intel® Core™ i5-2320 CPU @ 3.00GHz × 4, 12 GB RAM memory, and the Ubuntu operating system.

The architecture of examined TensorFlow-based neural networks for federated binary classification models on the BcBase datasets consists of 4 hidden layers each having 20 nodes. Neural networks were trained in 200 epochs per simulated ASCAPE edge node with batch size equal to 512. The Adam algorithm [4] (a stochastic gradient descent method based on adaptive estimation of first-order and second-order moments) was used as the optimizer of neural network weights and biases with the learning rate set to 0.001. Due to the presence of the class imbalance in the BcBase datasets, we have incorporated class weights in the neural network training process. The weight of class $C$ ($C$ is either the positive or the negative class) was set to $(1 \,/\, |C|)(T \,/\, 2)$, where $|C|$ is the total frequency of class $C$ in the training dataset (or a part of the dataset assigned to a simulated ASCAPE edge node) and the scaling by $T \,/\, 2$ ($T$ is the total number of instances in the training dataset) was introduced to keep the value of the loss function at the same magnitude as without weighting. The comparison of F1 scores of local and simulated federated binary classification models on the BcBase datasets is presented in Table 23. TFNN denotes a local TensorFlow-based neural network binary classification model, while INC-*k* and CON-*k* are simulated federated TensorFlow-based neural network binary classification models trained in the incremental (INC) and semi-concurrent (CON) learning mode for *k* simulated edge nodes.

For the BcBase-Anxiety, Depression and Insomnia datasets, we have identified that simulated federated models are significantly better than the worst performing local model (SVM and KNN depending on the dataset). The F1 scores of simulated federated models are close to F1 scores of NB which is the best performing local model for those three BcBase datasets. On the other hand, simulated federated models trained on the BcBase-Pain dataset have higher F1 scores than the F1 score of the best performing local model on that dataset (DT). It is also important to emphasize that there are no significant differences in F1 scores of incremental models and semi-concurrent models. Additionally, the performance of federated models does not tend to significantly drop with the number of simulated ASCAPE edge nodes.

*Table 23. F1 scores of local and simulated federated binary classification models on the BcBase datasets.*

| | Anxiety | Depression | Insomnia | Pain |
|---|---|---|---|---|
| Best local | 0.552 | 0.534 | 0.554 | 0.522 |
| Worst local | 0.411 | 0.413 | 0.502 | 0.457 |
| TFNN | 0.438 | 0.530 | 0.540 | 0.542 |
| INC-2 | 0.536 | 0.512 | 0.546 | 0.542 |
| INC-3 | 0.542 | 0.507 | 0.529 | 0.542 |
| INC-4 | 0.539 | 0.515 | 0.538 | 0.532 |
| CON-2 | 0.522 | 0.504 | 0.542 | 0.548 |
| CON-3 | 0.512 | 0.519 | 0.550 | 0.534 |
| CON-4 | 0.530 | 0.509 | 0.542 | 0.545 |

For simulated federated regression models trained on the ORB dataset, we have used the neural network architecture with 10 hidden layers each with 40 neurons. Their training was performed in 200 epochs per simulated ASCAPE edge node. The batch size was equal to 32. The optimization algorithm was Adam with the same learning rate as for classification models. The obtained MAE scores are given in Table 24. For all six datasets, the best local model (LASSO) has lower prediction errors than simulated federated models. There are no large differences between simulated federated models trained in different federated learning modes. In contrast to simulated federated models trained on the BcBase datasets, here we can observe a tendency of increasing errors with the number of simulated ASCAPE edge nodes. Simulated federated models are better than the DUMMY baseline for ORB-30-36, ORB-30-60, ORB-54-60 and ORB-108-120, but worse than DUMMY for ORB-30-120 and ORB-54-120. This result implies that different neural network architectures should be employed for short term and long term QoL predictions. Therefore, our subsequent work will be to examine a wider range of neural network architectures for federated regression and determine architectures providing satisfactory results long-term QoL predictions.

*Table 24. Comparison of MAE scores of local and simulated federated regression models on the ORB datasets.*

| | **30-36** | **30-60** | **30-120** | **54-60** | **54-120** | **108-120** |
|---|---|---|---|---|---|---|
| DUMMY | 6.541 | 6.890 | 6.909 | 6.890 | 6.909 | 6.909 |
| LASSO | 5.089 | 5.886 | 6.478 | 4.840 | 6.180 | 5.437 |
| TFNN | 5.783 | 6.572 | 7.323 | 5.811 | 7.206 | 6.562 |
| INC-2 | 6.012 | 6.775 | 7.488 | 5.931 | 7.188 | 6.625 |
| INC-3 | 6.472 | 6.751 | 7.226 | 5.867 | 7.169 | 6.430 |
| INC-4 | 6.595 | 7.042 | 7.463 | 6.220 | 7.206 | 6.484 |
| CON-2 | 5.881 | 6.705 | 7.444 | 5.904 | 7.193 | 6.463 |
| CON-3 | 6.404 | 6.826 | 7.427 | 5.986 | 7.098 | 6.327 |
| CON-4 | 6.534 | 6.883 | 7.538 | 6.269 | 7.221 | 6.652 |

### 4.5.3 Evaluation of HE-based models

The evaluation of HE-based model is presented in the tables below. For comparison and validation, all AI models were trained on plaintext data as well, according to the 10-fold cross-validation. Hence, the outputs of the plaintext models are compared to the results of the encrypted models after decryption. Next, the performance of the unencrypted and encrypted models is compared.
For the classification problem on the BcBase dataset, the metrics' values are listed inTable 25 to Table 28. The predictions obtained with the unencrypted models and the ones obtained with the encrypted models are identical.

*Table 25. Evaluation of binary classification models on BcBase-Anxiety*

| | $ACC$ | $F1$ | $Prec$ | $Rec$ | $Prec^{+}$ | $Rec^{+}$ | $Prec^{-}$ | $Rec^{-}$ |
|---|---|---|---|---|---|---|---|---|
| **Plaintext** | 0.653 | 0.223 | 0.514 | 0.525 | 0.865 | 0.705 | 0.164 | 0.346 |
| **Encrypted** | 0.653 | 0.223 | 0.514 | 0.525 | 0.865 | 0.705 | 0.164 | 0.346 |

*Table 26. Evaluation of binary classification models on BcBase-Depression*

| | $ACC$ | $F1$ | $Prec$ | $Rec$ | $Prec^+$ | $Rec^+$ | $Prec^-$ | $Rec^-$ |
|---|---|---|---|---|---|---|---|---|
| **Plaintext** | 0.642 | 0.255 | 0.517 | 0.524 | 0.828 | 0.711 | 0.206 | 0.337 |
| **Encrypted** | 0.642 | 0.255 | 0.517 | 0.524 | 0.828 | 0.711 | 0.206 | 0.337 |

*Table 27. Evaluation of binary classification models on BcBase-Insomnia*

| | $ACC$ | $F1$ | $Prec$ | $Rec$ | $Prec^+$ | $Rec^+$ | $Prec^-$ | $Rec^-$ |
|---|---|---|---|---|---|---|---|---|
| **Plaintext** | 0.556 | 0.510 | 0.553 | 0.554 | 0.624 | 0.565 | 0.481 | 0.543 |
| **Encrypted** | 0.556 | 0.510 | 0.553 | 0.554 | 0.624 | 0.565 | 0.481 | 0.543 |

*Table 28. Evaluation of binary classification models on BcBase-Pain*

| | $ACC$ | $F1$ | $Prec$ | $Rec$ | $Prec^+$ | $Rec^+$ | $Prec^-$ | $Rec^-$ |
|---|---|---|---|---|---|---|---|---|
| **Plaintext** | 0.699 | 0.170 | 0.522 | 0.564 | 0.936 | 0.724 | 0.107 | 0.405 |
| **Encrypted** | 0.699 | 0.170 | 0.522 | 0.564 | 0.936 | 0.724 | 0.107 | 0.405 |

For the regression problem, the metrics' values are listed in Table 29 to Table 34, followed by the Bland-Altman plot and the Scatter plot for each Orebro dataset in Figure 7 to **Error! Reference source not found.**. In scatter plots the predicted response of the regression model is plotted against the actual, true response. A perfect regression model has a predicted response equal to the true response, so all the points lie on a diagonal line. Usually, a good model has points scattered roughly symmetrically around the diagonal line. Bland-Altman plot is a scatter plot in which the y axis shows the difference between predicted and ground truth and the x axis represents the average of these measures. The predictions obtained with the unencrypted models and the ones obtained with the encrypted models are identical. For this reason, only one pair of plots for each Orebro dataset is illustrated below.

*Table 29. Evaluation of regression models on ORB-30-36*

| | $MAE$ | $MSE$ | $R2$ | $PC$ |
|---|---|---|---|---|
| **Plaintext** | 6.811 | 75.033 | -0.094 | 0.254 |
| **Encrypted** | 6.811 | 75.033 | -0.094 | 0.254 |

*Table 30. Evaluation of regression models on ORB-30-60*

| | $MAE$ | $MSE$ | $R2$ | $PC$ |
|---|---|---|---|---|
| **Plaintext** | 7.500 | 93.420 | -0.191 | 0.170 |
| **Encrypted** | 7.500 | 93.420 | -0.191 | 0.170 |

*Table 31. Evaluation of regression models on ORB-30-120*

| | $MAE$ | $MSE$ | $R2$ | $PC$ |
|---|---|---|---|---|
| **Plaintext** | 7.694 | 107.079 | -0.318 | 0.120 |
| **Encrypted** | 7.694 | 107.079 | -0.318 | 0.120 |

*Table 32. Evaluation of regression models on ORB-54-60*

| | ***MAE*** | ***MSE*** | ***R2*** | ***PC*** |
|---|---|---|---|---|
| **Plaintext** | 7.145 | 87.820 | -0.120 | 0.297 |
| **Encrypted** | 7.145 | 87.820 | -0.120 | 0.297 |

*Table 33. Evaluation of regression models on ORB-54-120*

| | ***MAE*** | ***MSE*** | ***R2*** | ***PC*** |
|---|---|---|---|---|
| **Plaintext** | 7.779 | 100.775 | -0.240 | 0.186 |
| **Encrypted** | 7.779 | 100.775 | -0.240 | 0.186 |

*Table 34. Evaluation of regression models on ORB-108-120*

| | ***MAE*** | ***MSE*** | ***R2*** | ***PC*** |
|---|---|---|---|---|
| **Plaintext** | 7.424 | 88.873 | -0.094 | 0.306 |
| **Encrypted** | 7.424 | 88.873 | -0.094 | 0.306 |

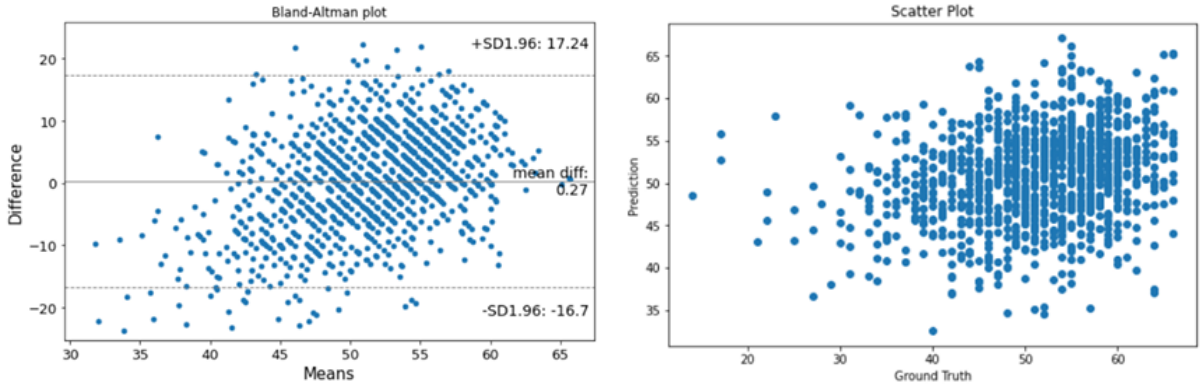


*Figure 7. ORB-30-36 plots*

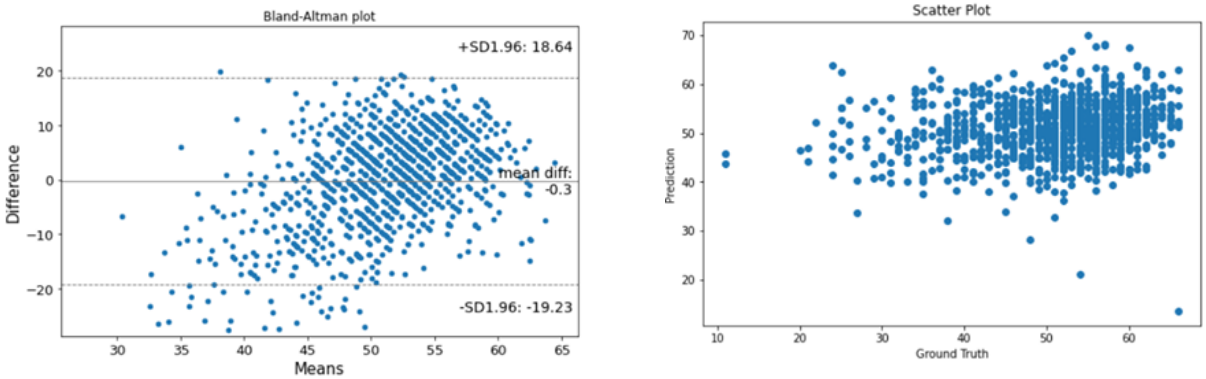


*Figure 8. ORB-30-60 plots*

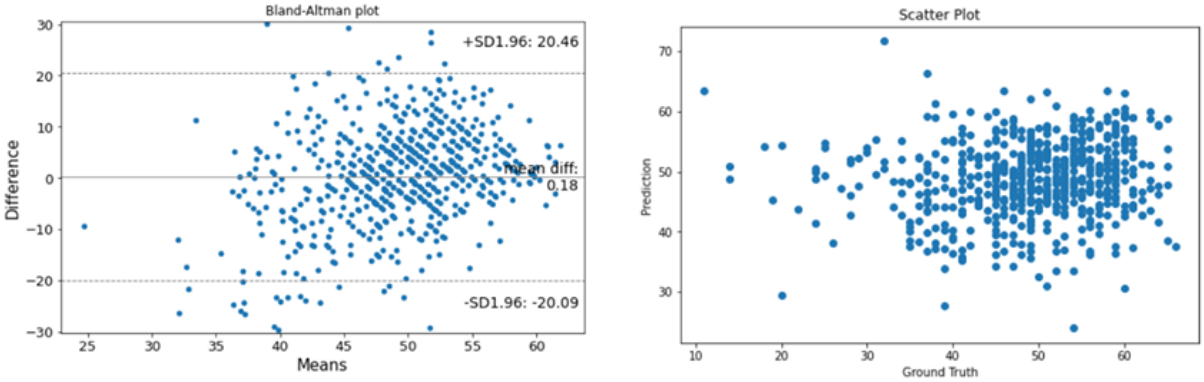


*Figure 9. ORB-30-120 plots*

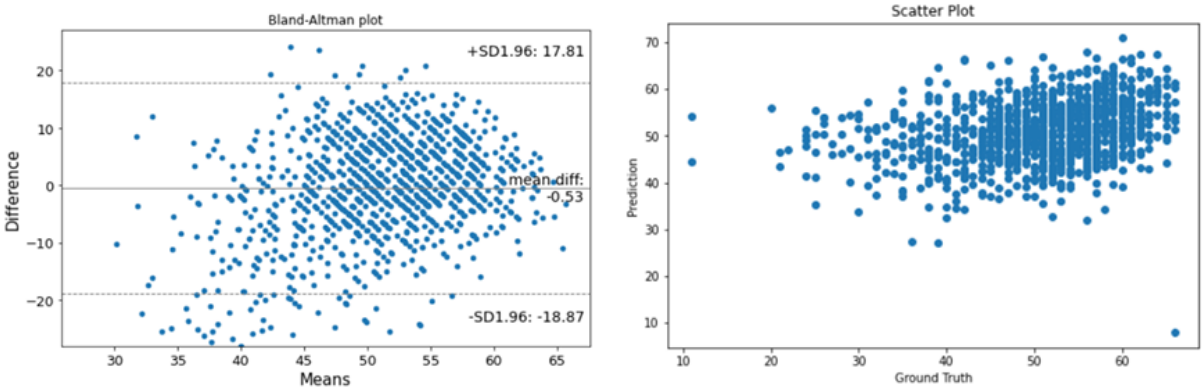


*Figure 10. ORB-54-60 plots*

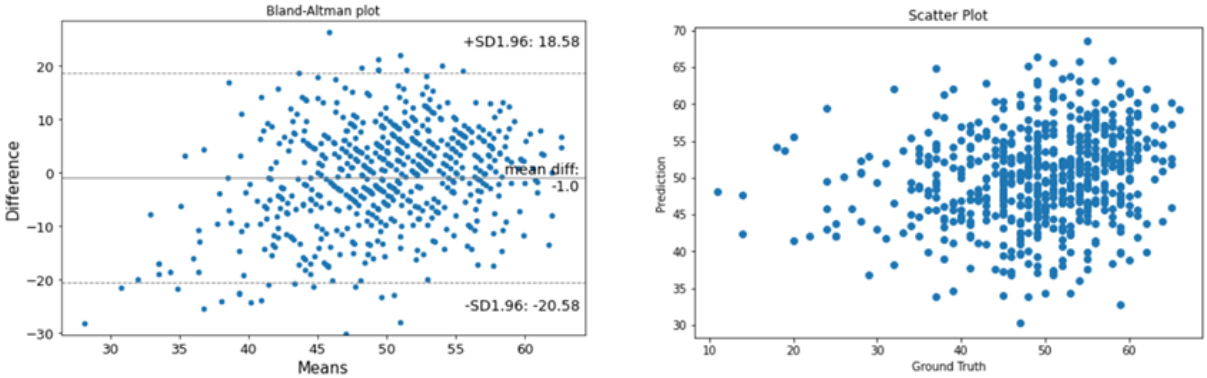


*Figure 11. ORB-54-120 plots*

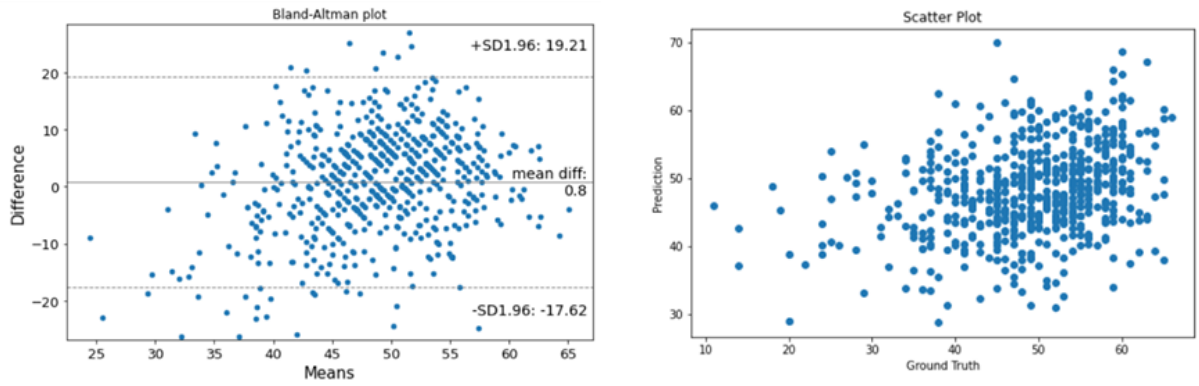


*Figure 12. ORB-108-120 plots*

To investigate the suitability of the MORE encryption scheme in real Machine Learning applications, training and testing are reported. For our experiments, a system with Intel(R) Core (TM) i7-4800MQ CPU @ 2.70GHZ has been used. Considering that the BcBase datasets are similar, only one use-case (*BcBase-Anxiety*) is listed in Table 35. For the same reason, we only report the time for the *ORB-30-36* and *ORB-108-120* datasets. The time reported in the first 2 columns of the table is the mean of the values measured for each fold of the datasets. The other 2 columns contain the mean testing time of the 10 folds for each dataset. On average, the encrypted model took 40 times longer to train and 35 times longer to test than the unencrypted model.

*Table 35. Training and Testing time for the unencrypted and the encrypted model*

| Dataset | Training time on plaintext data (s) | Training time on encrypted data (s) | Testing time on plaintext data (s) | Testing time on encrypted data (s) |
|---|---|---|---|---|
| ***BcBase-Anxiety*** | 448.803 | 18525.69 | 0.049 | 1.764 |
| ***ORB-30–36*** | 67.638 | 2675.353 | 0.018 | 0.622 |
| ***ORB-108–120*** | 31.921 | 1295.513 | 0.008 | 0.282 |

## 4.6 Summary of Model Training Evaluation

In this deliverable we have examined five classification and ten regression machine learning models on BcBase and Orebro retrospective datasets, including also simulated federated machine learning models based on TensorFlow neural networks.

The evaluation of binary classification models on BcBase datasets showed that trained models exhibit relatively high accuracies, but moderate F1 scores (slightly higher than 0.5). The Naive Bayes classifier showed to be the best classification technique for BcBase datasets. The analysis of obtained precision and recall scores per classes revealed that the examined models exhibit satisfactory results for the negative class (patients that do not experience anxiety, depression, insomnia and/or pain after cancer diagnosis), while predictions for the positive class (patients experiencing negative QoL

symptoms) are more error prone. Thus, in our future work we will examine several bootstrapping and data resampling techniques in order to see whether BcBase binary classification models can be improved towards more accurate predictions for the positive class of patients. The analysis of simulated federated models trained on BcBase datasets showed that their performance is comparable to the performance of locally trained models or even better (federated models on the BcBase-Pain dataset have higher F1 scores than the best locally trained binary classification model).

The analysis of regression models on the Orebro retrospective datasets revealed that the Lasso regression model is the best performing model among the examined regression models. This model significantly outperforms the baseline Dummy regression model, giving the reduction in prediction errors ranging from 20-30% for short term QoL predictions, 10-15% for medium term QoL predictions and 5% for long term QoL predictions. The mean absolute error of Lasso ranges from 4.84 to 6.47, which is an acceptable level of prediction errors since the target variable (the LISAT QoL index) ranges from 11 to 66. The analysis of simulated federated models trained on Orebro datasets showed that their performance is comparable to locally trained regression models for short term QoL predictions, but not for long term QoL predictions. Thus, in our future work we will examine different neural network architectures for long term QoL predictions.

The impact of missing value inference to the performance of classification and regression models was also examined. The obtained results indicate that models trained after the simple missing value imputer have F1 scores similar to models trained after the iterative missing value imputer. This result suggests that missing value inference done by simple and time-efficient methods can be as effective as missing value inference done by more sophisticated ML-based methods.

Compared to the TensorFlow models the HE-based models obtained weaker results. This is a consequence of the fact that the C++ HE AI library can operate, for the moment, only with the SGD optimizer. This optimizer is not suitable for complex AI models. Therefore, in future work, we plan to extent the machine learning library and enable the use of advanced optimizer for HE models training and improved neural network architectures. Further experimentation with the HE models will be considered to find better deep learning-based models.

In addition to improving the existing models trained on retrospective datasets, our future work will be mainly focused on the training and evaluation of machine learning models on prospective datasets as they become available to technical AI partners. For both retrospective and prospective datasets, we will also examine machine learning models trained after outlier detection and elimination and models trained after feature selection.

# 5 Explainability Methods and Evaluation

This section reports on the methods experimented with on the models obtained from retrospective datasets to generate explanations. First, different Feature Attribution frameworks are presented in section 5.1 and training of interpretable surrogate models is described in section 5.2. Their evaluations are reported in section 5.3.

## 5.1 Frameworks for Feature Attribution

There are various ways to explain the output of machine learning models. Generally, the objective of an explanation is to provide information about the inference of a prediction. For very complex models, the inner processes that lead to a certain prediction cannot be easily delivered in a form that humans can comprehend. Usually, only the inputs and outputs of an AI model are directly related to real world entities such as physical measurements or human-made definitions and can therefore be easily understood by the user of the ASCAPE platform.

*Feature Attribution* describes the amount each input feature contributes to the prediction of a model. Hereby a scalar real value is assigned to each input feature. A positive number denotes that the feature increased the output, and vice versa. The higher the feature attribution is, the stronger is the influence on the output. The exact scaling of feature attribution value depends on the method used.

Various techniques to calculate feature attribution have been proposed in the last years. The increasing popularity of neural networks accelerated the development of feature attribution techniques to overcome their drawback of being a black-box unpredictable behaviour.

The *inspection*-module of the SciKit-learn package provides a function for permutation importance [1]. However, this method is rather outdated and can only determine the feature attribution for a whole batch or dataset, but not for each sample individually.

Captum [5] is an explainability framework developed for the deep learning library PyTorch. It provides a set of classes for various modern feature attribution methods and techniques to examine individual layers of a neural network. The Feature Attribution methods are Integrated Gradients [6], Gradient SHAP [7], DeepLIFT [8], DeepLIFT SHAP [7], Saliency [9], Input X Gradient [9], Guided Backpropagation and Deconvolution [10], Guided GradCAM [11], Feature Permutation [12], Occlusion [13], Shapley Value Sampling [7], Lime [14] and KernelSHAP [7]. Although providing a broad set of state-of-the-art methods, Captum is heavily tailored to being used with PyTorch, making it difficult to being used with non-deep learning methods.

ELI5 [15] is a class library tailored for use with SciKit-learn, and various gradient boosting libraries like XGBoost and Keras. It has implementations of some more popular methods like LIME [14]and more basic techniques like permutation importance. The format in which explanations are provided are separated by the calculations. ELI5 provides functions to explain model weights and predictions. The

features of ELI5 are a bit behind of those of Captum and it does not seem to be actively developed anymore.

Based on the Paper “A Unified Approach to Interpreting Model Predictions” [7] by S. A. Lundberg and Su-In Lee, a Python explainability framework called SHAP maintained by Lundberg is publicly available. Beyond the basic SHAP implementation from their paper it utilises multiple other state-of-the-art concepts like Integrated Gradients [6], (Linear-)LIFT and DeepLIFT [8] mainly to accelerate the computation of explanations.

The underlying concept of SHAP originates from the game-theoretic concept of Shapley values developed by Lloyd S. Shapley [16]. They mathematically describe the influence on the result of a process for each entry in a set of entities E by evaluating the processes’ output for all possible subsets of E.

The calculation of Shapley values has exponential complexity, which is why SHAP has implementations which make use of the internal structure of the model which reduces the complexity. If the model is not compatible with any of the optimizations, Kernel-Shap can be used, which efficiently approximates the Shapley values for any black-

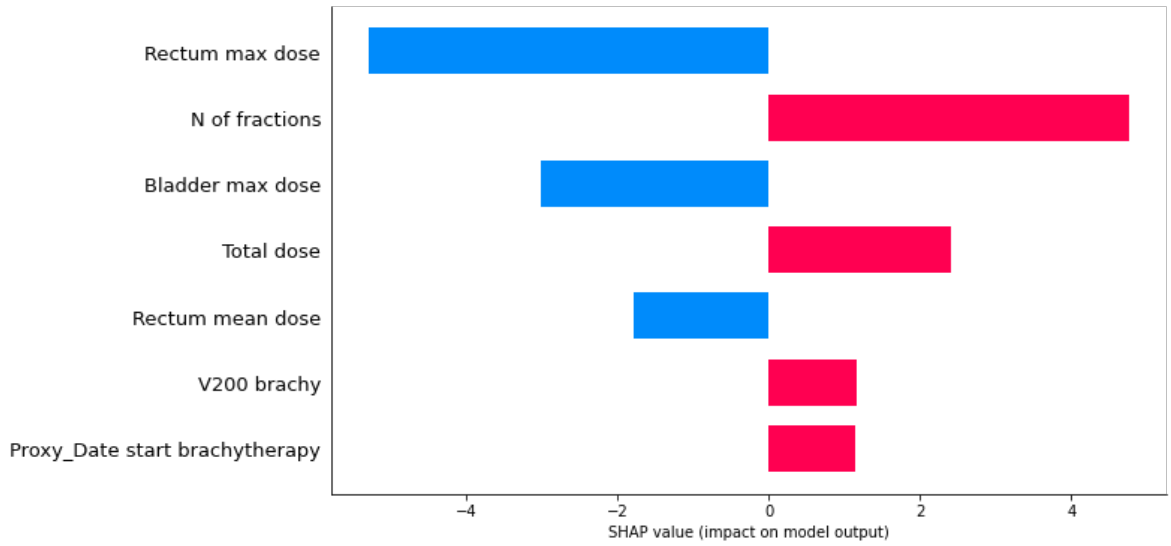


*Figure 13. Bar plot showing the feature attribution of a single input sample.*

box model. Since all modules of SHAP calculate or approximate the same mathematically defined Shapley values for feature attribution, they can be used interchangeably. For ASCAPE, we will therefore integrate SHAP to provide explanations for our model predictions.

## 5.2 Surrogate Models

Since most machine learning models are very complex, have a large parameter space and are therefore not interpretable by design, ASCAPE will provide surrogate models

along with each prediction model (called from here on *regular model*). A surrogate model is a model trained on training data labelled by the outputs of the regular model but is interpretable by design. This way it gets optimized producing similar outputs as the regular model but can easily be examined to provide human-readable rulesets explaining the output.

In ASCAPE, Linear Regression and Decision Trees are used as surrogate trees. The coefficients of a Linear Regressor provide correlations between each input feature and the models' output while treating the inputs feature as stochastically independent. Decision Trees are nonlinear models and generate clear "if … then …"-rules during training. Given a data sample, the decision path can be calculated and visualized as graphic or text-based showing which decision boundaries were used to calculate the surrogates' output. The decision trees can also be visualised and function as interpretable generalization of the regular models. An example of a visualization is shown in Figure 14 on p.47. A text-based decision path for a prediction explained with the surrogate decision tree is shown in Figure 15 on p. 48. For each decision, the inputs value of one variable is checked. If the value is higher or lower than the decision boundary, the next node is checked. (For a value higher than the decision boundary, the right node is used, otherwise the left node is used). A node without subsequent nodes is called a leaf and its value defines the surrogates' output. In the example shown, the decision tree applied the following rule for the input sample:

*"If the QoL at baseline is between 52.2723 and 56.982 and the value for erectile function in month 24 after the diagnosis is below 0.0097, then the expected QoL at month 36 is estimated to be 50.572."*

To create a surrogate model from a regular model, the input data of a training dataset is needed. Theoretically, this dataset must not be the original dataset the regular model was trained on, but it is desirable to ensure the outputs of the surrogate and regular model are as similar as possible. If the regular model is a local model, the surrogate training is very simple. The training dataset without it's labels gets passed into the regular model and the outputs are used as labels for the surrogate training dataset. With this dataset, a linear regression model and a decision tree get trained and are saved along the regular model. If the Edge Node has multiple local models, this process must be repeated for each model individually.

Global models were not or only partly trained with the edge nodes' own dataset. The training datasets from other edge nodes cannot be transferred because of privacy concerns. Therefore, the surrogate model must be trained in a privacy-preserving federated learning procedure similar to the global model.

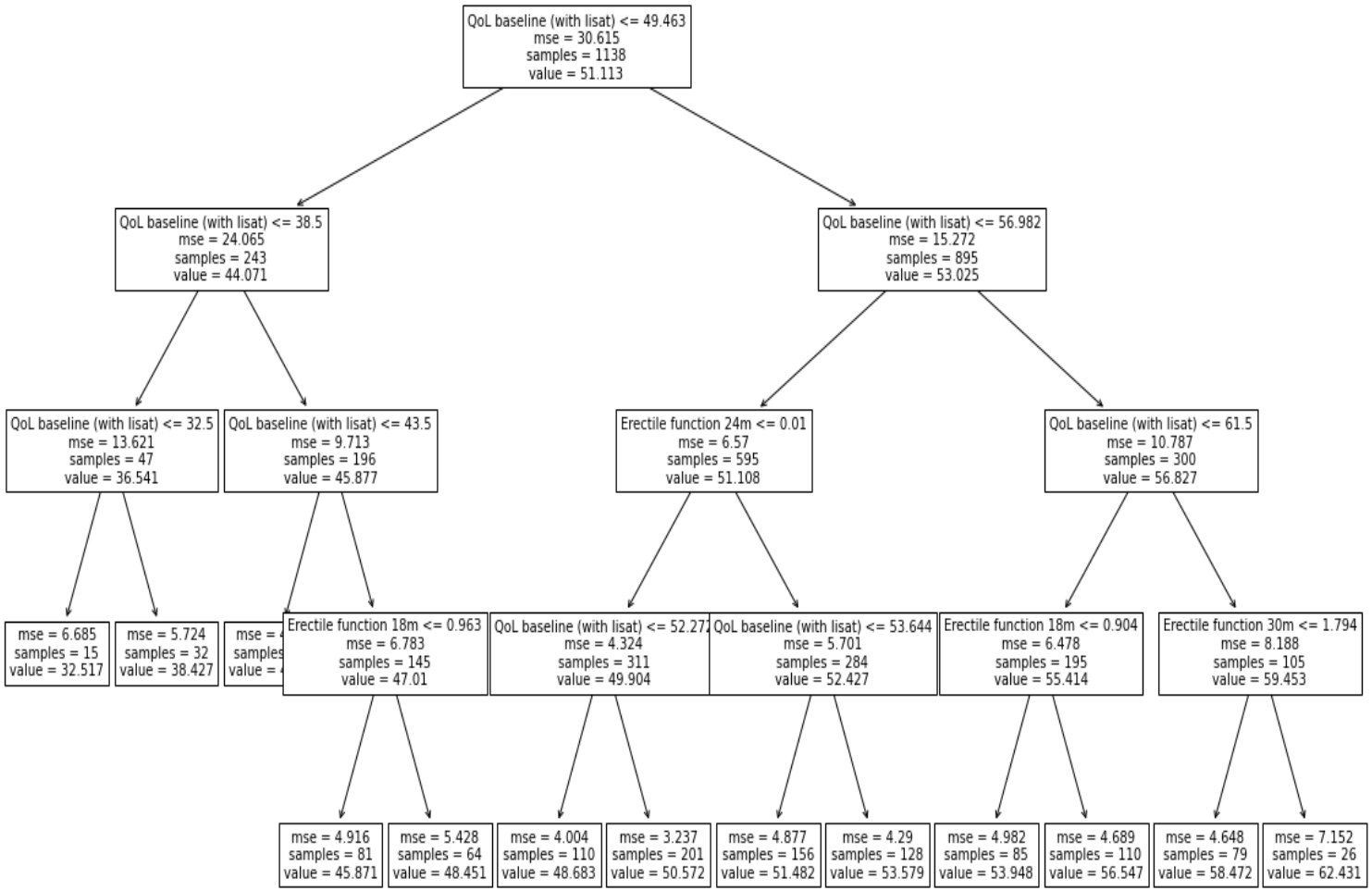


*Figure 14. Visualization of a surrogate decision tree based on a linear regressor trained on the ORB-30-36 dataset*

```
decision node 0   : QoL baseline (with lisat) = 54.0909) > 49.463
decision node 10 : QoL baseline (with lisat) = 54.0909) <= 56.9823
decision node 11 : Erectile function 24m = -0.1227) <= 0.0097
decision node 12 : QoL baseline (with lisat) = 54.0909) > 52.2723
surrogate tree output is 50.5720, actual model output is 49.9321
```

*Figure 15. Example of a text-based description of a decision path extracted from the decision tree in Figure 14.*

For Linear Regression, various methods have been proposed to combine their weights or train them concurrently over gradient descent based on the mean squared error [17] For decision trees, federated learning was mainly explored with random forests and gradient boosted trees.

## 5.3 Evaluation methodology and Results

We evaluated various state of the art methods and frameworks for feature attribution and put focus on a self-explanatory output with a clear mathematical definition. SHAP satisfies these conditions. Evaluating feature attribution methods cannot be done mathematically, since there is no definition of what a correct explanation is. The authors of SHAP conducted an experiment where humans gave feature impact estimations for a simple model predicting a sickness score for three symptoms. They found SHAP being particularly consistent with what a person understanding a model would give as estimation for feature attribution [7].

The performance of surrogate models is critical to use them as a reliable method. Since surrogate models are optimized to predict the outputs of the target model rather than predicting the distribution of the training dataset, they must fulfil two criteria at once. First, their outputs must be close to those of the target model in order to provide explanations for its predictions. Secondly, it must still have a good performance on the original evaluation dataset. This performance can usually only be as good as the performance of the target model.

For the evaluation of the surrogate models we use the models trained and evaluated in section 4 as target models. For each dataset and each target model, we trained a linear regression surrogate model and a decision tree surrogate model. For their evaluation, we used two types of ground truths: For the first evaluation, the outputs of the target model were used as ground truth. For the second evaluation, the actual ground truth of the original retrospective dataset was used (shown in the results with “dataset”). This way, we could examine if the surrogate models were still able to predict the actual QoL scores while also emulating the target model’s predictions. For both evaluations, a dedicated evaluation dataset was used that was not part of the training dataset. For classification tasks, we used accuracy as metric (1 is best, 0 is worst), for regression tasks we used the R2-score (1 is best, 0 is random guessing, below 0 is worse than random guessing).

Table 36 to Table 39 below present the evaluation results of all surrogate models on all datasets. We observed that surrogate training is successful in most cases while

also having accuracies similar to the target models on the original evaluation dataset. Surrogate models trained on linear models seem to be especially consistent in maintaining good performance on both emulating the target models output as well as predicting the actual model.

*Table 36. Evaluation metrics for surrogate decision trees using the BcBase datasets*

| | | BcBase-Anxiety | BcBase-Depression | BcBase-Insomnia | BcBase-Pain |
|---|---|---|---|---|---|
| Surrogate | DT | 0.72 | 0.72 | 0.58 | 0.73 |
| | KNN | 0.95 | 0.96 | 0.72 | 0.96 |
| | NB | 0.91 | 0.87 | 0.86 | 0.87 |
| | RF | 0.91 | 0.93 | 0.69 | 0.94 |
| | TFNN | 0.88 | 0.95 | 0.88 | 0.84 |
| | SVM | 1.00 | 1.00 | 1.00 | 1.00 |
| Dataset | DT | 0.67 | 0.68 | 0.51 | 0.70 |
| | KNN | 0.70 | 0.70 | 0.55 | 0.71 |
| | NB | 0.63 | 0.59 | 0.55 | 0.51 |
| | RF | 0.70 | 0.70 | 0.55 | 0.71 |
| | TFNN | 0.53 | 0.67 | 0.56 | 0.25 |
| | SVM | 0.70 | 0.70 | 0.54 | 0.71 |

*Table 37. Evaluation metrics for surrogate logistic regression using the BcBase datasets*

| | | BcBase-Anxiety | BcBase-Depression | BcBase-Insomnia | BcBase-Pain |
|---|---|---|---|---|---|
| Surrogate | DT | 0.33 | 0.68 | 0.53 | 0.71 |
| | KNN | 0.94 | 0.95 | 0.67 | 0.94 |
| | NB | 0.66 | 0.53 | 0.60 | 0.62 |
| | RF | 0.90 | 0.91 | 0.62 | 0.92 |
| | TFNN | 0.66 | 0.82 | 0.68 | 0.47 |
| | SVM | 1.00 | 1.00 | 0.84 | 1.00 |
| Dataset | DT | 0.30 | 0.70 | 0.52 | 0.71 |
| | KNN | 0.70 | 0.70 | 0.52 | 0.71 |
| | NB | 0.53 | 0.44 | 0.53 | 0.51 |
| | RF | 0.70 | 0.70 | 0.52 | 0.71 |
| | TFNN | 0.43 | 0.69 | 0.53 | 0.65 |
| | SVM | 0.70 | 0.70 | 0.54 | 0.71 |

*Table 38. Evaluation metrics for surrogate decision trees using the ORB datasets.*

| | | ORB-30-36 | ORB-30-60 | ORB-30-120 | ORB-54-60 | ORB-54-120 | ORB-108-120 |
|---|---|---|---|---|---|---|---|
| Surrogate | AdaB | 0.61 | 0.37 | 0.21 | 0.60 | 0.04 | 0.40 |
| | ElasticN | 0.87 | 0.73 | 0.34 | 0.85 | -0.82 | 0.73 |
| | KRidge | 0.65 | 0.16 | -0.01 | 0.53 | -0.05 | 0.16 |
| | KNN | -0.31 | -0.46 | -0.70 | -0.90 | -0.10 | -0.51 |
| | Lasso | 0.84 | 0.74 | 0.34 | 0.90 | 0.00 | 0.62 |
| | Linear | 0.67 | 0.29 | -0.11 | 0.55 | -1.16 | -0.21 |
| | RF | 0.44 | 0.24 | 0.08 | 0.58 | 0.00 | 0.38 |
| | Ridge | 0.72 | 0.37 | 0.11 | 0.59 | 0.03 | 0.49 |
| | TFNN | 0.48 | 0.33 | 0.37 | 0.61 | 0.34 | 0.42 |
| | SVM | 0.74 | 0.76 | 0.68 | 0.81 | 0.45 | 0.50 |
| Dataset | AdaB | 0.33 | 0.11 | 0.04 | 0.24 | 0.04 | 0.18 |
| | ElasticN | 0.36 | 0.21 | 0.06 | 0.37 | -0.07 | 0.25 |
| | KRidge | 0.28 | 0.05 | -0.02 | 0.32 | 0.03 | 0.08 |
| | KNN | -0.01 | -0.01 | -0.06 | -0.08 | 0.01 | -0.03 |
| | Lasso | 0.35 | 0.21 | 0.07 | 0.39 | 0.08 | 0.24 |
| | Linear | 0.30 | 0.11 | -0.04 | 0.29 | -0.47 | -0.09 |
| | RF | 0.30 | 0.13 | 0.04 | 0.40 | 0.01 | 0.25 |
| | Ridge | 0.31 | 0.13 | 0.02 | 0.33 | 0.01 | 0.26 |
| | TFNN | 0.11 | 0.02 | 0.03 | -0.36 | 0.07 | 0.09 |
| | SVM | -0.01 | -0.03 | 0.00 | -0.03 | 0.00 | -0.01 |

*Table 39. Evaluation metrics for surrogate linear regression using the ORB datasets*

| | | ORB-30-36 | ORB-30-60 | ORB-30-120 | ORB-54-60 | ORB-54-120 | ORB-108-120 |
|---|---|---|---|---|---|---|---|
| Surrogate | AdaB | -5.07 | -10.59 | -14.12 | -45.45 | -11.07 | -0.33 |
| | ElasticN | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 |
| | KRidge | 1.00 | 1.00 | 0.37 | 1.00 | 0.46 | 0.38 |
| | KNN | -23.10 | -49.96 | -39.15 | -59.13 | -14.02 | -4.49 |
| | Lasso | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 |
| | Linear | 1.00 | 1.00 | -8.09 | 0.34 | -0.68 | -1.86 |
| | RF | -0.21 | -3.37 | -0.61 | -7.46 | -2.05 | -0.18 |
| | Ridge | 1.00 | 1.00 | 0.62 | 1.00 | 0.74 | 0.62 |
| | TFNN | 0.56 | -1.02 | 0.17 | -2.18 | 0.80 | 0.89 |
| | SVM | -5.76 | -12.73 | -8.41 | -90.55 | -7.83 | -2.68 |
| Dataset | AdaB | -1.56 | -2.10 | -4.30 | -18.63 | -3.63 | -0.05 |
| | ElasticN | 0.39 | 0.24 | 0.12 | 0.43 | 0.21 | 0.33 |
| | KRidge | 0.46 | 0.35 | 0.10 | 0.53 | 0.16 | 0.18 |
| | KNN | -1.36 | -3.08 | -3.22 | -4.15 | -0.64 | -0.30 |
| | Lasso | 0.39 | 0.24 | 0.13 | 0.44 | 0.22 | 0.33 |
| | Linear | 0.46 | 0.36 | -2.58 | 0.18 | -0.27 | -0.96 |
| | RF | -0.11 | -1.81 | -0.29 | -4.99 | -1.05 | -0.09 |
| | Ridge | 0.45 | 0.34 | 0.17 | 0.52 | 0.26 | 0.28 |
| | TFNN | 0.16 | -0.17 | -0.02 | -1.10 | 0.10 | 0.16 |
| | SVM | -0.02 | -0.02 | -0.04 | -0.08 | -0.03 | -0.02 |

When training linear regression surrogates for Ridge Regression, Elastic Net or Lasso Regression, we sometimes achieve a perfect score of 1, because the overall method is the same for both the surrogate and target model. All models are based on a weighting of input features and adding an independent term. Therefore, Ridge Regression, Elastic Net or Lasso Regression yield a surrogate linear regressor with the same coefficients. The surrogate training can be skipped completely, and these models can be used as linear surrogate model directly.

Surrogate decision trees do not achieve a perfect score when trained on a decision tree (see first row of Table 36), because hyperparameters might influence the training, which uses a “greedy” algorithm to build its nodes. To achieve perfect similarity with the target model here, we will instead simply use the target decision tree model as surrogate directly, since it is interpretable by design anyway.

# 6 Simulations and Evaluations

## 6.1 Simulation Targets

Giving patient data into a model will provide an estimation for the variable the model was trained for to predict, that is the result of a QoL questionnaire or conditions affecting the patient's wellbeing like anxiety or pain. If the target variable of the model is a QoL indicator for a future time point, the model itself can give predictions for the future condition of the patient. However, it is not able to directly infer what medical interventions improve the expected QoL.

A central task of ASCAPE is to actively propose medical interventions to the medical staff that are expected to improve the patients QoL. Simulations with the AI models will be used to infer what treatments will improve the expected QoL the most. The results shall contain what treatments are recommended and provide some explanations and metrics to let the medical staff evaluate the proposal.

## 6.2 Simulation Concepts and Techniques

To identify promising treatments and other medical interventions in the dataset, the behaviour of the model can be studied by making changes to the input data and observing the models' predictions. When making a prediction for a QoL indicator, the model takes all input features it was trained on into account. Usually, these input features are not stochastically independent and even if they are, many machine learning methods will process them in a way that each variable's influence on the prediction is dependent on other variables. Therefore, input features cannot be analysed independently to identify promising treatments.

Taking the dependency between multiple input variables into account on the other hand is a computationally expensive task. If the input data has $n$ variables, the complexity of running simulations for every possible combination of input values is $O^n$. This is not feasible for any case occurring in ASCAPE. Therefore, a small subset of the possible input data space must be found that can be used for simulations.

The input variables can be classified into three types: Direct proxy variables indicating if an intervention has taken place or not, variables that can be influenced by interventions (e.g., blood pressure) and static variables that cannot be influenced by interventions (e.g., age or household income). Only the former two types are relevant for simulations.

A simple approach was taken at first and only the proxy indicators for treatments were simulated by being set to 0 or 1. With this, slight changes in the output of models trained on the retrospective Örebro dataset were observed. The use of Feature Attribution methods shows that the treatment proxy variables influence the output of the models just slightly, which is expected because the treatment itself does not yield information about the condition of the patient directly. When a treatment is performed,

it influences other medical data variables, which then influence the patients QoL directly. To analyse the influence of a treatment on medical data, two approaches were implemented and tested:

For the first approach, covariances between each treatment proxy and each variable containing medical data were calculated and normalized with the treatment proxy's standard derivation. This assumes that all variables are normally distributed.

Therefore, a second approach was implemented in which the patients of the training dataset were split into two cohorts for whether they received a treatment or not. This is done for each treatment individually. The expected change of a variable after a treatment was calculated as the difference of its mean value in the two cohorts.
To propose a treatment, the steps as described in the algorithm shown in Figure 16 are as follows: First, the influence of all input variables on the prediction are measured with feature attribution. If desired, only the *k* most influential variables are changed during the simulation. In the second step, the expected change of all selected variables in the first step is calculated for each treatment. Then, for each patient and for each possible treatment, an inference with the model is performed. The input is the patient data with the respective treatment proxy set to 1 and the other variables increased or decreased according to the expected changes of the treatment. Then, the difference of the models' prediction for the current, actual medical data of each patient and the prediction for the treatment is calculated, resulting in the predicted change of QoL when a treatment is performed.

For each patient, the treatment with the highest predicted QoL increase is proposed in the ASCAPE dashboard. This proposition of the treatment is explained with the calculated feature attribution and the predicted QoL increase is shown.

For the prediction of binary indicators denoting if a QoL-related issue will emerge like in the BcBase dataset, the simulation process must be slightly adapted. A models' prediction on such a dataset can only have the values 0 and 1, meaning the output

```
Dataset: d
Model: m
Treatments: t
Patient_sample: s
Changeable medical variables: v

iterate over each t in t:
   feature_attr ← get_attribution(m, s, t)
   treatment_influences ← mean(d_{t=1}) - mean(d_{t=0})
   current_prediction ← m.predict(s)
   s[t] ← 1
   s[v] += treatment_influences
   simulated prediction ← model(s)
   expected_qol_change ← sim_prediction - current_prediciton
Propose t with largest expected_qol_change
```

*Figure 16. Pseudo code outlining the steps used to propose treatments.*

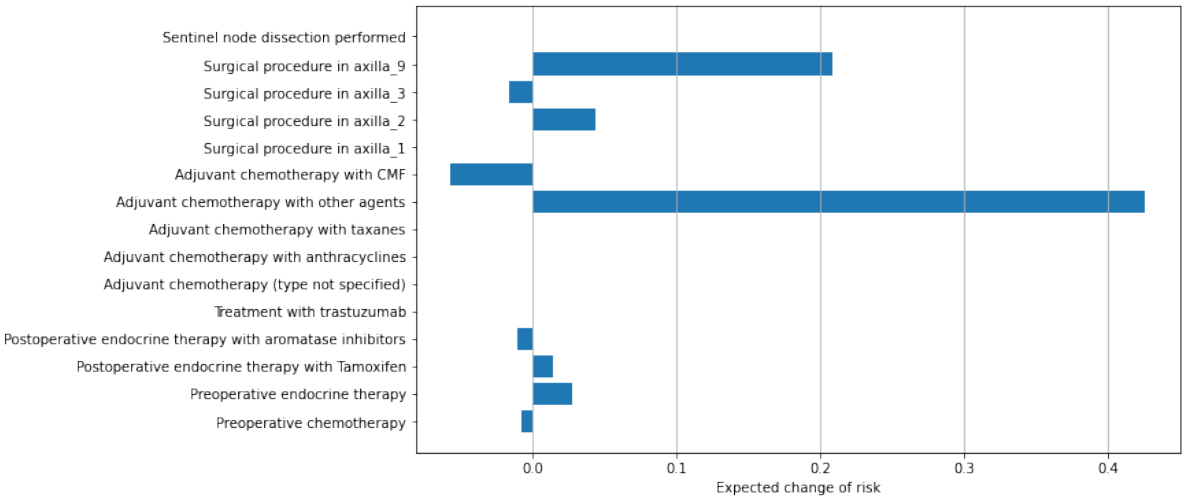


*Figure 17. Example of the expected change of risk of pain evaluating the class probabilities of a Naive Bayes classifier trained with BcBase.*

values are not continuous. Therefore, we instead evaluate the class probabilities and use them do a risk prediction.

Figure 17 shows the expected change in probability of suffering from pain based on a sample simulated with a Naïve Bayes model trained on the BcBase-Pain dataset. Since we want to reduce the risk, “Adjuvant chemotherapy with CMF” is the recommended treatment since it reduces the risk by about 5%.

## 6.3 Summary of Simulation Evaluations

The retrospective datasets are limited in their structure to provide information about how treatments influence a patient’s medical data. A central drawback is that most variables are only collected once. To identify a clear causality between a treatment and other medical data, it would be necessary to collect each variable multiple times before and after the dataset. Since the prospective datasets will also contain regularly collected data, the effects will be easier to identify once that data is available. We will also investigate techniques to identify the influence of treatments more precisely, like splitting the patients into many small cohorts like a matched case-control study [18].
However, it is not easy to evaluate simulations, because for a benchmark a ground truth would be needed. There is no information about the QoL increase after treatments that have not been done yet.

# 7 Conclusions

This deliverable reported on the first experiments with different machine learning methods considered to develop a data driven assistive service to help preserve and possibly improve the quality of life of cancer patients. The experiments were conducted on 10 datasets derived from retrospective datasets with respect to classification and regression and uniform evaluation metrics were fixed. The main findings and derived next steps:

- Missing value inference by simple methods can be as effective as missing value inference done by regression methods.
- The optimal value of the ε parameter of differential privacy performed by using the Laplace mechanism with respect to mean average error and an acceptable level of privacy is around the value 1.
- Regarding classification the best performing locally trained algorithm was the Naïve Bayes classifier, but the overall performance was satisfactory on the negative classes while it was more error-prone for the positive class.
- The simulated federated trained classifier models had comparable and even better performance than the locally trained models.
- In the locally trained setting, the Lasso regression model is the best performing model to predict quality-of-life values and significantly outperformed the baseline Dummy regression model.
- The simulated federated trained regression models were found to be comparable to locally trained regression models for short term QoL predictions, but not for long term QoL predictions. Thus, future work will examine different neural network architectures for long term QoL predictions.
- In comparison to the above used models on non-encrypted data, the models trained on homomorphically encrypted data showed weaker results, which is mainly due to the fact that the current HE library can only operate with the SGD optimizer. Thus, future next steps are concerned to extend the HE library to enable the use of more enhanced optimizers
- For feature attribution as one mechanism for explainability, the SHAP framework provides the mathematically best basis using the game-theoretic concepts of Shapley-values and has been evaluated in previous work to be particularly consistent with feature attribution estimations done by human experts. Next steps will be to investigate together with the clinical partners how sensible the obtained feature estimations are from an expert's perspective.
- To use surrogate models as a second mechanism for explainability, the surrogate model must accurately approximate the target model to explain. In the classification case, trained decision tree and linear regression based surrogate models have accuracies very similar to the target models. In the regression case, linear regression trees surrogate models are also very accurate, while decision tree based surrogate models are less accurate, but still acceptable. The next steps will consist of investigating how useful for a medical expert the information that can be read out of a surrogate model can be in general, as well as for the assessment of the quality-of-life impact for a specific patient.

- A simulation-based approach was implemented to compute treatments ranked with their expected increase/decrease of the quality of life for a specific patient based on systematic variations of key feature identified by feature attribution. The first experiments are promising, but more investigations are required, as the retrospective data contains limited information to assess the impact of treatments on medical conditions of patients and because of lacking ground truth to compare simulation results to.

Overall, the experiments and evaluations performed thus far in the project based on retrospective data in order to assess if and which machine learning methods can result in useful assistance are promising. Key issues to address have been identified and future work will be devoted to these, in order to have the appropriate methods in place to apply as soon as prospectively collected datasets arrive from the ASCAPE pilot sites.

# 8 Bibliography

[1] F. e. a. Pedregosa, “Scikit-learn: Machine learning in Python,” *the Journal of machine Learning research 12,* pp. 2825-2830, 2011.

[2] H. B. Curry, “The method of steepest descent for non-linear minimization problems,” *Quarterly of Applied Mathematics,* pp. 258-261, 1944.

[3] Abadi, M. et. al., “TensorFlow: Large-scale machine learning on heterogeneous systems,” 2015. [Online]. Available: Software available from tensorflow.org.

[4] D. P. Kingma and J. Ba, “Adam: A Method for Stochastic Optimization,” in *3rd International Conference on Learning Representations, ICLR 2015*, San Diego, CA, USA, 2015.

[5] N. Kokhlikyan, V. Miglani, M. Martin, E. Wang, B. Alsallakh, J. Reynolds, A. Melnikov, N. Kliushkina, C. Araya, S. Yan and O. Reblitz-Richardson, “Captum: A unified and generic model interpretability library for PyTorch,” arXiv preprint, arXiv:2009.07896, 2020.

[6] M. Sundararajan, A. Taly and Q. Yan, “Axiomatic Attribution for Deep Networks,” *Proceedings of the 34th International Conference on Machine Learning,* p. 3319–3328, August 2017.

[7] S. M. Lundberg and S.-I. Lee, “A Unified Approach to Interpretable Model Predicitions,” NIPS Proceedings, http://papers.nips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictions, 2017.

[8] A. Shrikumar, P. Greenside and A. Kundaje, “Learning Important Features Through Propagating Activation Differences,” *Proceedings of the 34th International Conference on Machine Learning,* pp. 3145-3153, 2017.

[9] K. Simoyan, A. Vedaldi and A. Zisserman, “Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps,” *arXiv preprint arXiv:1312.6034,* 2013.

[10] J. T. Springenberg, A. Dosovitskiy and M. R. Thomas Brox, “Striving for Simplicity: The All Convolutional Net,” *arXiv preprint arXiv:1412.6806,* 2014.

[11] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh and D. Batra, “Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization,” *arXiv:1610.02391,* 2016.

[12] C. Molnar, Interpretable Machine Learning, https://christophm.github.io/interpretable-ml-book/, 2021.

[13] M. D. Zeiler and R. Fergus, “Visualizing and understanding convolutional networks,” *European conference on computer vision,* pp. 818-833, 2014.

[14] M. T. Ribeiro, S. Singh and C. Guestrin, “"Why Should I Trust You?": Explaining the Predictions of Any Classifier,” *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining,* p. 1135–1144, 2016.

[15] various, “ELI5 Gitlab Repository,” 15 09 2016. [Online]. Available: https://github.com/TeamHG-Memex/eli5. [Accessed 22 01 2021].

[16] L. S. Shapley, “A value for n-person games,” *Contributions to the Theory of Games 2,* pp. 307-317, 1953.

[17] Z. Yuan and Y. Yang, “Combining linear regression models: When and how?,” *Journal of the American Statistical Association,* pp. 1202-1214, 2005.

[18] M. A. de Graaf and e. al, “Matching, an appealing method to avoid confounding?,” *Nephron Clinical Practice,* pp. 315-318, 2011.