UHASSELT KNOWLEDGE IN ACTION

Maastricht University

Faculty of Sciences
School for Information Technology
Master of Statistics and Data Science

Master's thesis

Development of an Explainable AI Model for Predicting Patient Outcomes in Radiotherapy for Cervical Cancer

Guus Spenkelink
Thesis presented in fulfillment of the requirements for the degree of Master of Statistics and Data Science, specialization Data Science

SUPERVISOR :
Prof. dr. Dirk VALKENBORG
Prof. dr. Inigo BERMEJO DELGADO

SUPERVISOR :
Peter BOSMAN
Tanja ALDERLIESTEN

2024
2025

Faculty of Sciences

School for Information Technology

Master of Statistics and Data Science

Master's thesis

Development of an Explainable AI Model for Predicting Patient Outcomes in Radiotherapy for Cervical Cancer

Guus Spenkelink

Thesis presented in fulfillment of the requirements for the degree of Master of Statistics and Data Science, specialization Data Science

SUPERVISOR :

Prof. dr. Dirk VALKENBORG

Prof. dr. Inigo BERMEJO DELGADO

SUPERVISOR :

Peter BOSMAN

Tanja ALDERLIESTEN

Contents

Abstract

Locally advanced cervical cancer is the most frequently diagnosed cancer in women aged 40-50 years, making it an important medical problem. In this study, Bayesian networks (BNs) are learned and evaluated as predictive models for treatment outcomes of cervical cancer patients. BNs model relationships between random variables, and their compactness and intuitive graphical representation make them suitable for eXplainable AI (XAI) – a field of research focused on methods that allow for human intellectual oversight over AI algorithms. XAI is particularly important in the healthcare field due to the high stakes involved in clinical decision-making. BNs were learned using DBN-GOMEA, a state-of-the-art self-discretizing BN learning algorithm for discrete and continuous data powered by an evolutionary algorithm. The dataset contains clinical inputs and outcomes for 280 patients treated for locally advanced cervical cancer, and this is the first time DBN-GOMEA is applied to a real-world dataset. Three different outcomes were modelled: distant metastasis-free survival time, overall survival time, and 12-month metastasis-free survival. These models were evaluated and compared to classical survival analysis. The BN for metastasis-free survival time and overall survival time showed indications of predictive power but performed similar to or worse than Cox proportional hazards models as measured by Harrell's concordance index, potentially due to shortcomings of the model for censored survival data. The BN for 12-month metastasis-free survival (binary classification) on the other hand showed promising results, with a test ROC area under curve (AUC) estimated to be 0.79 (sd=0.14), which is higher than the Cox regression AUC. Furthermore, an explorative study into the relationship between dose-volume histogram (DVH) metrics and toxicity was performed. DVH metrics can be used to measure the amount of unwanted treatment radiation dose in critical organs. No clear indicators of correlation were found. Counterintuitively, the occurrence of toxicity became more common over time whereas DVH metrics improved, a trend that is not yet understood and requires further investigation.

Introduction

Locally advanced cervical cancer is the most frequently diagnosed cancer in women aged 40-50 years [1], and the morbidity due to the cancer as well as treatment impacts many aspects of their lives. The standard of treatment is external beam radiotherapy (EBRT) with concurrent chemotherapy and brachytherapy (BT) [2]. EBRT and BT are two types of radiation treatment where the radiation source is placed outside and inside the patient respectively. In radiotherapy, patients receive personalized treatment plans where the goal is to give a prescribed amount of radiation dose to the tumor, while sparing the surrounding healthy tissue and critical organs. However, it can be challenging to achieve a low amount of radiation dose to critical organs, especially those situated close to the tumor, which can lead to treatment-related toxicity. Therefore, an *explainable* yet flexible model that reveals how patient and treatment factors influence outcomes such as survival as well as toxicity would be valuable for personalized clinical decision-making.

Bayesian networks (BNs) are a potential solution – they are flexible models and have good properties for explainability as they tend to be compact, and captured relationships can be directly visually inspected by domain experts [3]. This thesis explores the use of BNs as a predictive model for treatment outcomes of patients treated for locally advanced cervical cancer. BNs will be trained using Discretizing Bayesian Network Gene-pool Optimal Mixing Evolutionary Algorithm (DBN-GOMEA) [3], a state-of-the-art evolutionary algorithm. The goal of the research will be to explore the feasibility of this algorithm for learning BNs on the cervical cancer dataset. To benchmark the performance, the test performance of the BNs will be compared to that of classical models.

Explainable AI (XAI) is a field of research focused on methods that allow for human intellectual oversight over AI algorithms. This thesis will focus on the *model interpretability* aspect of XAI. In contrast to a black-box model, an XAI model gives insight into the reasoning behind its predictions, making them more understandable and transparent. This is especially important in the field of healthcare where clinical decisions carry high stakes. Explainable models are allow clinicians to critically evaluate and challenge the model's recommendations, rather than relying on opaque predictions. This also aligns with modern regulations such as the EU's *AI Act* [4], which emphasizes transparency and interpretability and makes human oversight a requirement for high-risk AI systems such as those used in healthcare.

The BN learning algorithm DBN-GOMEA has been developed by the Evolutionary Intelligence group which supervised this work, and has been thoroughly tested in a simulation study by Ha et al. [3]. This thesis is the first time this algorithm has been used to learn BNs from a real-world dataset.

The dataset studied in this research was previously analyzed by Horeweg et al. [1] using classical methods such as Cox proportional hazards models and Kruskal-Wallis H-tests. Their research will be used to inform feature selection. Since then, the dataset has grown by about 80% as more patients have been treated. Horeweg et al. identified relevant factors that influence different types of survival outcomes. About 70% of patients experienced toxicity; one risk factor for vaginal toxicity was identified, but no statistically significant predictors for bladder, rectal, bowel and bone toxicity were found. However, DVH (dose-volume histogram) metrics for EBRT plans, which quantify the amount and distribution of radiation dose into the critical organs as well as tumor volumes, were not included in the analysis. In this thesis, new features will be generated by calculating DVH metrics for EBRT plans and the relationship between them and toxicity outcomes will be explored, with a focus on urinary incontinence, urinary urgency and urinary frequency. These toxicities were flagged by a clinician as a common cause of discomfort for patients. Since radiation dose to critical organs can be adjusted in the treatment preparation process, it is important to study and understand dose thresholds for toxicity.

As stated before, this thesis is the first time DBN-GOMEA has been used for learning BNs for a real-life dataset. By applying this technique to real-world data and benchmarking it against classical models, this study aims to contribute to methodological insight for the field of medical XAI.

Research questions

The goal of this study can be summarized into the following three research questions:

1. Can we learn a Bayesian Network as a predictive XAI model for clinical censored and uncensored survival data, and what are the limitations?
2. How do the performance and results of the XAI model compare to a classical survival analysis?
3. Do new features (DVH metrics) carry predictive power with respect to toxicity outcomes, in particular urinary incontinence, urinary urgency and urinary frequency?

The first two questions are the core of this research, and the third research question is more exploratory in nature.

Ethical thinking, societal relevance and stakeholder awareness

Locally advanced cervical cancer is a relevant medical and societal problem as it is the most frequently diagnosed cancer for women aged 40-50 years old, and many patients suffer from treatment-related toxicity in the form of urinary incontinence, urinary urgency and/or urinary frequency, reducing their quality of life. This makes it important to study the factors that lead to treatment success as well as complications.

In the clinic, doctors rely on models to make treatment decisions. For example, in radiotherapy, doctors use dose-toxicity models from research literature to estimate the probability of treatment-related toxicity based on the radiation dose absorbed by critical organs. The usefulness of the model is limited by the quality of the model.

This thesis researches the potential of Bayesian Networks as a model for treatment outcomes of locally advanced cervical cancer. The potential of BNs lies in their combination of flexibility and explainability. This could make it a better model than either classical methods, due to its higher flexibility, or black-box type machine learning models, due to its better explainability. In the medical domain, XAI becomes especially important and is ethically advantageous to black-box models, as it allows clinicians to critically evaluate and challenge the model's recommendations.

In the field of radiotherapy, the use of AI algorithms has increased tremendously in the past years, with especially deep learning-based algorithms being used for applications such as the automatic segmentation of organs at risk, synthetic CT generation, deep learning-based automated treatment planning and outcome prediction [5] [6]. However, there are concerns, as well as ethical and legal challenges in adopting AI into clinical practice including patient privacy, the healthcare provider's accountability for errors, developer responsibility for transparency and bias mitigation, and the trustworthiness of the algorithms [7]. XAI offers a path forward. The importance of using explainable methods may still increase as the EU's *AI Act* [4], effective since Augst 1st 2024, emphasizes transparency and interpretability and mandates human oversight for high-risk AI systems such as those used in healthcare. This means that explainability is not only ethically responsible, but also legally required in certain contexts.

The dataset used in this study is a real-world clinical dataset, collected retrospectively and anonymized to protect patients' privacy. Despite the anonymization, the data is still sensitive as it contains CT and MRI scans of patients. All data handling occurred on a designated secure system and the data was not allowed to be copied or moved to any other systems.

Several stakeholders are impacted by this study. **Clinicians** will be the ones using the model to make treatment decisions, and the explainability is especially important for them because if the XAI's reasoning is medically sensible, they can more easily trust and rely on its outcomes. **Patients** will benefit from improved predictive power as this leads to better treatment decisions and thus better quality of care. A more directly involved stakeholder is the **research group** supervising this work. They have developed DBN-GOMEA and have an XAI research line, which means they are interested in studying the potential of this method for predicting clinical outcomes. The group is part of the Leiden University Medical Center and collaborates closely with clinicians in developing explainable AI tools.

Data

Description

The dataset examined in this work is an extension of [1]. The dataset consists of anonymized data from 280 patients who were treated for locally advanced cervical cancer between 2008 and 2021. Two types of data were available: tabular data and DICOM data.

Tabular data

The tabular data has 327 columns with features that can be categorized into four groups:

- Patient features – demographic and medical history data such as age, sex, presence of diabetes, and the presence of other malignancies;
- Diagnosis features – characterization of the patient's disease such as the tumor size at diagnosis, tumor histology, the classification of the tumor, and whether lymph nodes are affected;
- Treatment features – details of the treatment, such as the administered radiation dose, number of brachytherapy needles used, and whether the patient received chemotherapy;
- Clinical outcome features – clinical outcomes such as overall survival, distant metastasis-free survival, as well as different types of toxicities, when they occurred and their severity.

Data completeness is >98% for most features used in our analyses (among others age, overall survival time, histology, tumor classification, comorbidity, lymph node involvement). However, tumor size is only 95% complete, and the presence of diabetes and other malignities are only 78% complete.

DICOM data

DICOM is a medical imaging data standard [8]. In radiotherapy, the DICOM format allows storing of patient-specific treatment plans, the radiation dose distribution in the patient and organ segmentations. The DICOM data in this study consists of:

- CT and/or MRI images;
- radiotherapy treatment plans for both external beam radiotherapy as well as brachytherapy;
- organ and tumor segmentations;
- radiation dose distributions for external beam radiotherapy and brachytherapy.

Figure 1 gives an example of a visualization of a radiation dose distribution calculated on a CT.

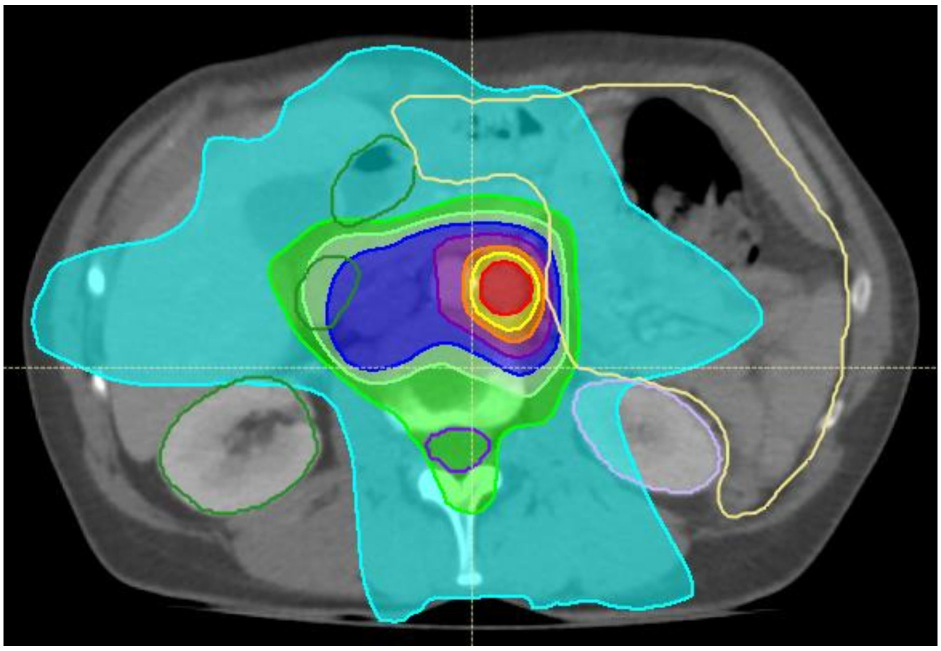

Figure 1. An example of DICOM data. Shown here is once slice of a CT scan (greyscale) with a 3D dose distribution indicated the color-graded surfaces. Some organ segmentations are also shown here, for example the two kidneys segmented by a green and purple line, and the bowel segmented by the large yellow contour on the right. This figure also illustrates critical organ sparing: the kidneys receive lower dose than the tissue between and outside of the kidneys, this is because the treatment plan was designed in such a way to spare these critical organs.

The available DICOM data was not a curated selection for this study, but rather extracted in bulk from the hospital's medical imaging data archive. As a result, the dataset contained all DICOM records associated with the 280 patients, including a considerable amount of surplus data. Significant effort was undertaken to go through the 280 patients and identify the relevant data suitable for analysis.

The DICOM data is used only to engineer additional features that quantify the quality of the external beam radiation treatment plan: the DVH metrics. To calculate DVH metrics, the simulated patient dose distribution and organ segmentations are required. Two examples of DVH metrics are:

- *Bladder V40 relative volume*: the relative volume of the bladder segmentation receiving a total dose of 40 Gray or more, with *Gray* being the unit for measuring absorbed radiation dose;
- *Kidney D_{mean}*: the mean radiation dose in the (left or right) kidney segmentation volume.

The DVH metrics are calculated for all segmented organs as well as the tumor volume. These new features are appended to the tabular dataset for further analysis.

Sufficient DICOM data to calculate DVH metrics was only available for 157 out of 280 patients (56 percent). This poor rate of completeness is due to two reasons:

- Before a change of policy in 2015, radiation dose distributions were not saved in the hospital's PACS[1] system. For some of these patients treated before 2015, the dose was recoverable, and was recalculated and added to the dataset. This was done by a researcher for research project. For other patients, the data is missing;
- Multi-center treatment: a considerable number of patients received external beam radiation therapy in another clinic. In these cases, the DICOM dose distributions, treatment plans and segmentations are missing.

[1] *Picture Archiving and Communication System*; the system used for storing, maintaining and distributing medical images in the clinic

Exploratory data analysis

Exploratory data analysis was performed to better understand the dataset through summary tables and visualizations. Table 1 summarizes key features in the dataset regarding the patients, diagnoses, and treatments. Note that the population is rather young for a cancer population, that all patients received external beam radiotherapy and brachytherapy, and the majority of patients also received chemotherapy.

Characteristics	**N (% of total)**
Age – *median*	*55 years*
Comorbidity	158 (56%)
Tumor size in mm – *mean (SD)*	*49.5 (17.1)*
Lymph node involvement	146 (52%)
Histological type	
Squamous cell carcinoma	232 (83%)
Adenocarcinoma	35 (13%)
Adenosquamous carcinoma	8 (3%)
Other	4 (1%)
FIGO stage	
$\leq$2a	77 (28%)
2b – 4	203 (72%)
External beam radiotherapy	280 (100%)
Brachytherapy	280 (100%)
Chemotherapy	218 (78%)
Treatment *time in days*	*42.9*

Table 1. Summary of important features

The treatment outcomes overall survival and distant metastasis-free survival are summarized in Table 2. Note that in the analysis of distant metastasis-free survival, death before metastasis is not seen as an event but rather as censoring.

Outcome	**1-year survival**	**3-year survival**	**5-year survival**	**Mean survival**
Overall survival	93%	75%	62%	6.3 years
Distant metastasis-free survival	90%	76%	71%	8.1 years

Table 2. Summary of treatment outcomes

Histograms of overall survival time and metastasis-free survival time are shown in Figure 2.

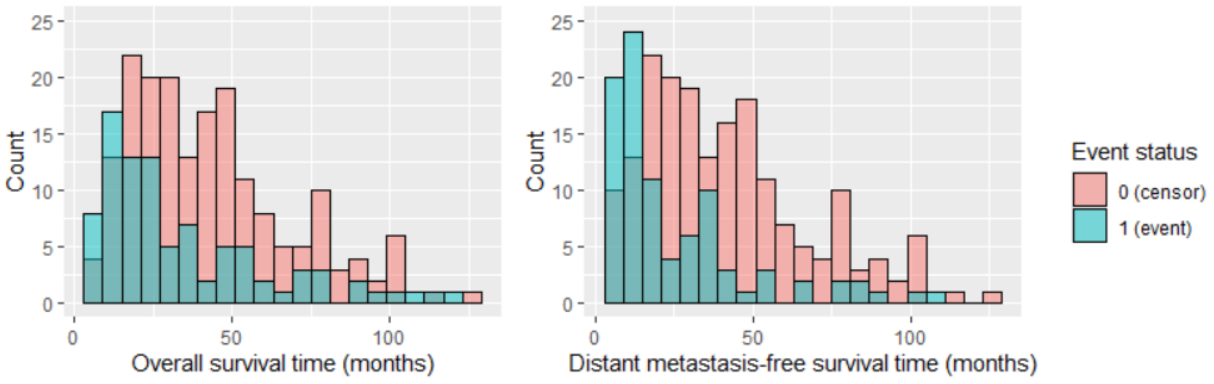

Figure 2. Histograms of overall survival time (left) and distant metastasis-free survival time (right) grouped by event status where 0=censoring and 1=event. The bin width is 6 months.

In prior work on the same dataset [1], tumor size was found to be the most significant risk factor for overall survival as well as distant metastasis-free survival. A scatterplot for overall survival time as a function of tumor size is shown in Figure 3. This figure shows clearly that survival past 50 months (4.2

years) was not observed for any patients with a tumor size greater than about 60 mm, whereas patients with smaller tumors did often survive longer.

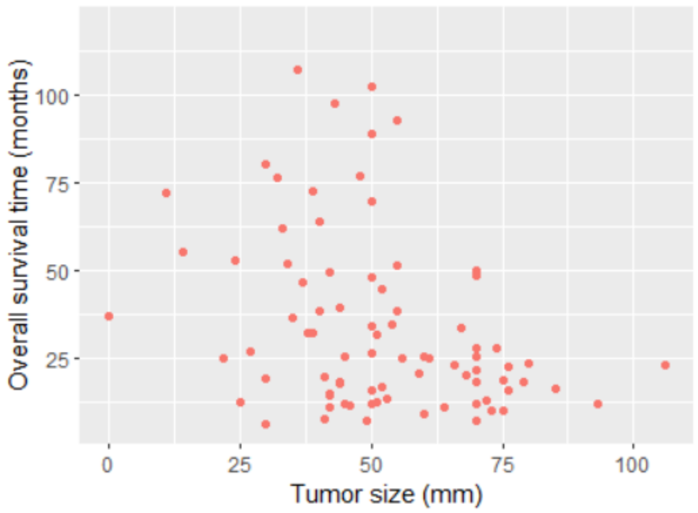

Figure 3. Scatterplot of non-censored overall survival time versus tumor size

DVH metrics versus toxicity

Throughout the course of data collection (2008 – 2021), external radiotherapy treatment techniques used in the clinic have evolved, allowing better sparing of critical organs. As a result, the values of DVH metrics have gone down notably (Figure 4). For all six metrics shown, a lower value is desirable as it indicates less radiation dose to the critical organs.

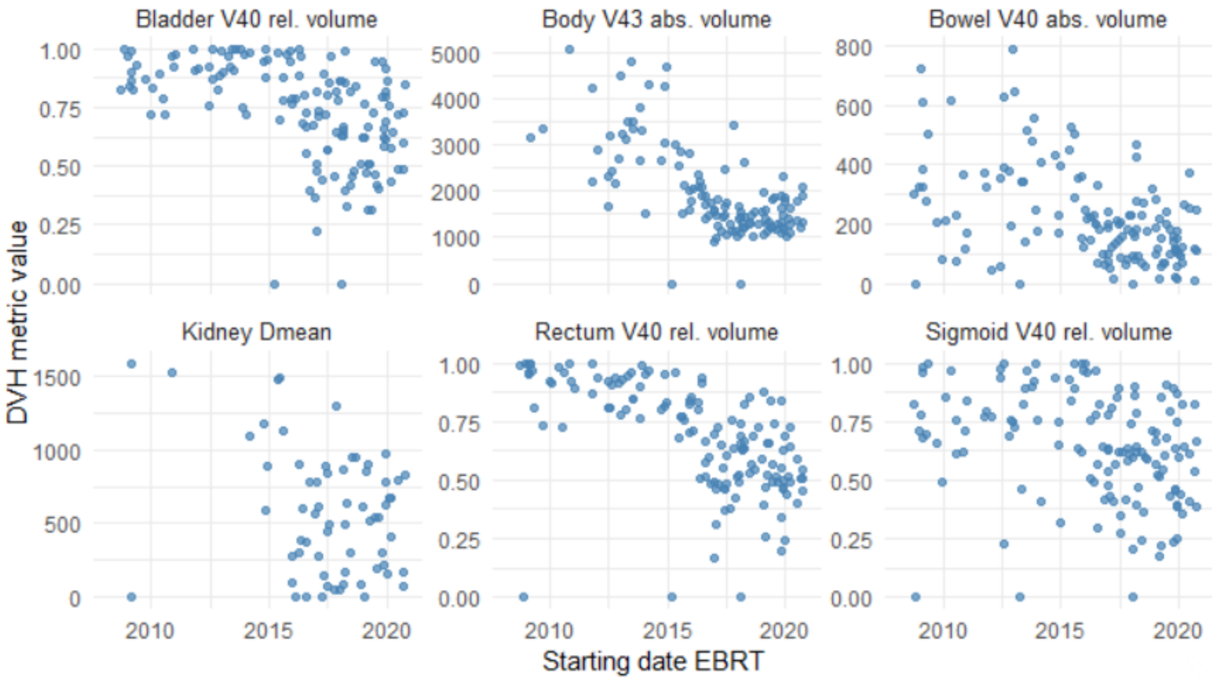

Figure 4. Evolution of several DVH metric values over time. Each dot represents a patient's treatment plan

One might then expect to see an improvement in treatment-related toxicity. For example, a lower dose to the bladder might result in a lower degree of urinary toxicity such as incontinence.

Toxicity events are recorded by date as well as by grade 1 through 4, with grade 4 being the most severe case of toxicity. For radiation-related toxicities, grade 2 toxicities are of particular interest according to an involved clinician. Table 3 shows the frequencies of the most common grade 1 and 2 toxicities. For most toxicity types, there are less than ten events, potentially making it difficult to model.

	Frequency	
Toxicity type	**grade 2**	**grade 1**
Urinary frequency	2	36
Spinal fracture	2	0
Fecal incontinence	3	15
Vaginal dryness	3	13
Pelvic fracture	3	5
Vaginal hemorrhage	4	35
Chronic kidney disease	5	3
Constipation	6	6
Urinary incontinence	6	15
Proctitis	7	7
Diarrhea	15	34
Abdominal pain	23	40
Vaginal stenosis or stricture	26	63

Table 3. Most common grade 1 and grade 2 toxicities, ordered by the frequency of grade 2 toxicity

Table 4 shows the frequency of the urinary toxicities of interest for all 4 grades.

	Frequency			
Toxicity type	**Grade 1**	**Grade 2**	**Grade 3**	**Grade 4**
Urinary frequency	36	2	0	0
Urinary urgency	39	1	0	0
Urinary incontinence	39	6	2	0

Table 4. Frequency of three types of urinary toxicity, per grade

The most common of these across all grades is urinary incontinence. Figure 5 shows a boxplot of the grade of urinary incontinence against the DVH metric associated with radiation dose to the bladder. Although there are only 5 datapoints for grade 2 and 1 datapoint for grade 3 toxicity, the apparent trend is the opposite from the expected, i.e. patients with higher grades of toxicity have a lower value for *bladder V30 rel. volume*; the relative volume of the bladder receiving 30 Gray of dose or more. This trend will be explored further in the Results section.

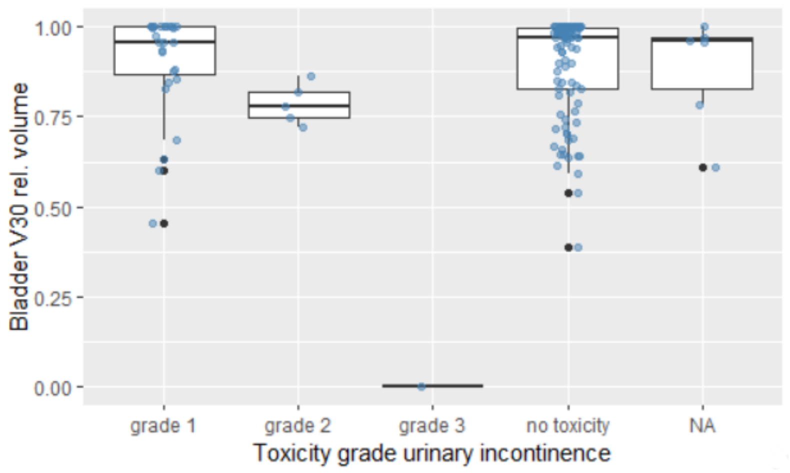

Figure 5. Boxplot of bladder V30 relative volume versus the grade of urinary incontinence

Methodology

The main model under study is a BN, learned using a state-of-the-art learning algorithm called DBN-GOMEA. The first subsection will briefly introduce the concepts of BNs and the learning algorithm.

To answer the research questions, three types of analyses are performed: time-to-survival event analysis, fixed-time survival classification, and the DVH metrics versus toxicity exploration. These will be described in the second and later subsections.

Discrete Bayesian networks and DBN-GOMEA

A BN consists of a directed acyclic graph (DAG), with each node representing a feature, and a conditional probability table associated with each node. Figure 6 shows an example of such a network.

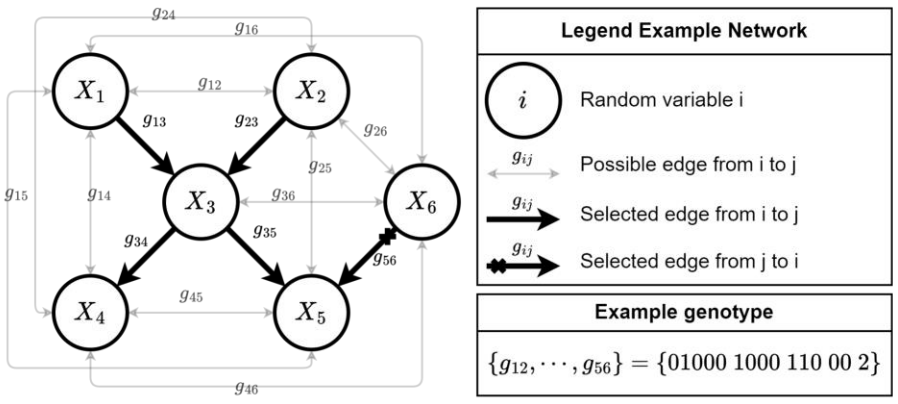

Figure 6. An example of a Bayesian network [3]. All possible edges are shown in grey. An example of edges representing a BN is shown in black

In Figure 6, all possible edges in the DAG are shown in grey, labeled $g_{12}, g_{13}, g_{14}, \dots, g_{56}$ for the edge between nodes X_1 and X_2, X_1 and X_3, X_1 and X_4 and so forth. The selected edges are shown in black, which for this example can be represented by the "genotype" $\{g_{12}, \dots, g_{56}\} = \{01000\ 1000\ 110\ 00\ 2\}$. In a genotype, all possible edges g_{ij} between nodes are represented by a number, with 0 meaning no edge, 1 meaning an edge from i to j, and 2 meaning an edge from j to i.

The directed edges represent the direct conditional dependencies between variables. The conditional probability table of node X_3 is conditionally dependent on its parents, X_1 and X_2, but not for instance its child X_4. The BN is essentially a factorization of the full joint probability distribution, as the joint probability distribution over all random variables $X_1, \dots, X_n$ can be written as:

$$P(X_1, X_2, \dots, X_n) = \prod_{i=1}^{n} P(X_i | Parents(X_i))$$

Where $Parents(X_i)$ denotes the immediate parent nodes of node X_i.

The process of learning BNs from data is also known as *structure learning*. The technique used in this study is called DBN-GOMEA, which employs an evolutionary algorithm[2] called Gene-pool Optimal Mixing Evolutionary Algorithm (GOMEA). GOMEA a model-based evolutionary algorithm that uses

[2] Evolutionary algorithms starts with a so-called generation consisting of multiple individuals with random genotypes, here represented by strings of 1s and 0s. The algorithm then selects individuals from this generation based on their fitness score, crosses their genomes, and applies random mutations, resulting in a next generation. This process is inspired by evolution through natural selection (survival of the fittest) and is repeated for many generations until a specified time point or number of evaluations is reached.

domain knowledge of the optimization problem to improve the performance and scalability of the optimization. This domain knowledge can be learned from the dataset during optimization, or, when possible, be supplied a priori [9]. DBN stands for Discretizing Bayesian Network because the algorithm can jointly discretize real-valued features during structure learning.

To run the learning algorithm, the variables to be used as nodes in the BN need to be specified. DBN-GOMEA optimizes the selection of edges between the nodes and the discretization of continuous features to maximize the density-based log-likelihood of the network defined by equation (1), where G is the Bayesian network and $\boldsymbol{x_i} \in \mathbb{R}^N$ is a sample i from a training data set of size n. The data is normalized to [0,1] for the computation of the densities to make it invariant to data range. The fitness function also includes a weighted complexity penalty as defined in equation (2), which punishes the number of parent discretizations $|Parents(X_i)|$, the number of discretizations of each node $|X_i|$ and the number of observations n (BIC penalty [10]). The resulting fitness function is shown in equation (3), with $\lambda \geq 0$ being the weight of the complexity term.

$$LL(\boldsymbol{X}, G) = \prod_{i=1}^{n} \log\left(f_{density}(\boldsymbol{x_i})\right) \tag{1}$$

$$C(G) = \sum_{i=1}^{N} |Parents(X_i)| \cdot (|X_i| - 1) \cdot \log\left(\frac{n}{2}\right) \tag{2}$$

$$fitness(\boldsymbol{X}, G) = LL(\boldsymbol{X}, G) - \lambda \cdot C(G) \tag{3}$$

Two important hyperparameters were tuned in this study:

- The discretization policy for continuous variables
    - DBN-GOMEA-EF: Equal-Frequency. Continuous variables are discretized into 2 – 9 bins, with each bin containing the same amount of observations
    - DBN-GOMEA-EW: Equal-Width. Continuous variables are discretized into 2 – 9 bins of equal width
    - DBN-GOMEA-BD: Bayesian score discretization: continuous variables are discretized into 2 – 9 bins, and the break points are jointly learned over all continuous variables [11]. This is a more involved and time consuming algorithm than the other two and has been shown in a simulation study to obtain more accurate network structures than EF and EW for networks with only a few nodes [3]. Unlike EF and EW, BD is a post-structure learning discretization method, meaning it is applied *after* the network architecture is learned through optimization of the density log-likelihood
- The complexity penalty (λ)
    - a number greater than 0 that defines the weight of the complexity term in the fitness function.

Survival time-to-event

For the outcomes *metastasis-free survival* and *overall survival*, a BN is learned. Two reference models are also optimized/fitted for comparison: a Cox proportional hazards model and a random survival forest [12].

DBN-GOMEA

BNs are learned from a training set consisting of 70% of the data. All three discretization policies (EF, EW, BD) of DBN-GOMEA are used. The weight of the complexity term in the score function is tuned using 5-fold cross validation on the training set. From the cross-validation results, the combination of discretization policy and complexity penalty resulting in the best validation C-index is selected, and a final model is trained on the full training set and evaluated on the test set (30% of the full dataset). The survival outcomes are represented by two nodes in the network. One representing the time-to-event in months, and another representing the type of event: 0 in for censoring and 1 if the event occurs (for example, death in the case of overall survival). Modeling censored survival data using discrete BNs has been described in literature [13], but it is not a well-established technique.

Reference models

A Cox proportional hazards model is fitted and evaluated using 5-fold cross validation on the full dataset. For each fold, the model is fitted to 80% of the data and evaluated on the remaining 20%. The final result is the average C-index over five folds.

As a modern ML survival model, a random survival forest is trained and evaluated using 5-fold cross validation on the full dataset, the same way as the Cox proportional hazards model. The random survival forest is trained using 1000 trees, 'impurity' variable importance mode, a log-rank splitting rule, a minimal node size to split at of 3, and a number of variables to split at in each node equal to the rounded down square root of the number of variables.

Model evaluation

Models are evaluated and compared using *Harrell's concordance index* [14] [15], from now on referred to as C-index, a common metric for evaluating the performance of survival models that can be computed for a wide range of models. The evaluation of C-index of the final model on the test set is bootstrapped with 1000 samples to obtain the standard deviation.

To calculate the C-index for the BNs, the mean survival time is predicted for each test observation given all other features and a survival event indicator equal to 1. This mean predicted survival time is used to compare pairs of observations and determine whether the predictions are concordant. The algorithm to calculate the C-index is described by Ishwaran et al [12].

Feature selection

This study is the first time that DBN-GOMEA is used on real-world data, so we start with a univariate "toy problem" of modelling *overall survival time* versus *tumor size*. From literature it is known that overall survival is correlated with tumor size, and the simplicity of the model will allow for easy sanity-checking of the results.

After the toy problem, full models are trained for distant metastasis-free survival and overall survival.

For the full models, the same set of features are used for all types of models. That is, the BN trained for overall survival has the same features as the Cox proportional hazards model for overall survival. Feature selection is copied from the study by Horeweg et al [1] on the same data, where candidate risk factors for metastasis-free survival and overall survival were selected by experts prior to analysis, and included in the final multivariate model if the P-value for that feature in a univariate Cox regression model was below 0.10. The model for metastasis-free survival has four regressors, the model for overall survival has seven regressors.

Fixed-time survival classification

For censored time-to-event data, the BN has fundamental limitations as it is not designed to work with censored data, which is expected to introduce bias and compromise the results. For that reason, a fixed-time survival classification analysis is performed as well. Instead of modelling time-to-event (e.g. metastasis-free survival time), the survival status at a fixed point in time is predicted. For example, 12-month metastasis-free survival. This is a binary classification problem, and for an early time point such as 12 months, not many patients have been lost to follow-up yet, meaning the effect of censoring is limited.

BNs are trained for 12-month metastasis-free survival (9% missingness due to censoring).

Just like the survival time-to-event analysis, 5-fold cross validation on the training set is used to optimize hyperparameters for the BN. The performance of the final model is evaluated on the test set.

Reference models

As a classical reference model, a Cox proportional hazards model is fitted and evaluated using 5-fold cross validation on the full dataset. For each fold, the model is fitted to 80% of the data and evaluated on 20%. The Cox PH model parameters were used to predict risk scores at 12 months on the test set, allowing for evaluation. The final cross-validated result is the average over the five folds.

Model evaluation

Since this is a classification problem, the ROC area under curve (AUC) is calculated. The ROC AUC is related to the concordance index [16], and is also a number between 0 and 1. An AUC of 0.5 on the test set means is equivalent to random guessing, and any number greater than that indicates some predictive power.

Feature selection

The 12-month metastasis-free survival status is represented by a single node in the BN. The selection of regressors is identical to the feature selection described for the survival time-to-event models for distant metastasis-free survival.

DVH metrics versus toxicity

The last research question is aimed at investigating whether the DVH metrics carry any predictive power with respect to treatment outcomes.

Generation of DVH metric features

The calculation of DVH metric features is a domain-specific task, which was performed in the radiotherapy *treatment planning system* (TPS) which can import the DICOM-files containing treatment plans, dose distributions, CT scans and organ segmentations for the patients in the study. The software allows for automation using Python scripting and has its own API. The API can be used to extract the desired DVH metrics for any organ if the names of the organ segmentations are known. A Python script was written that loops over all suitable patients in the study, and based on the names used for organ segmentation, extracts the required DVH metrics and writes them to a text file. From here, the new features are copied into the tabular data.

Statistical analysis / exploration

In the first part of the exploration, a quick scan across all combinations of toxicities and DVH metrics is performed. Kruskal-Wallis H-tests are performed to explore univariate relations between a continuous DVH metric (such as the dose to bladder) and the categorical toxicity outcome (such as urinary incontinence: no toxicity / grade 1 / grade 2 / grade 3 / grade 4).

The second part of the exploration focuses on three toxicities of specific interest: urinary incontinence, urinary urgency and urinary frequency. These were combined into a singly binary feature, *urinary_tox_binary*. A value of 0 means none of the toxicities has occurred for this patient during follow-up, and 1 means any of the toxicities of any grade has occurred during follow-up. Logistic regression was used to explore univariate and multivariate relationships of this toxicity with relevant DVH metrics and potential confounders, using a 5% significance level ($\alpha = 0.05$).

This investigation into DVH metrics and toxicity is purely explorative. Using exploratory data analysis techniques, trends in the data were explored, giving ideas for the direction of future research.

Computation

DBN-GOMEA is written in C++ and is run from a Linux server at the Centrum Wiskunde & Informatica (CWI) with the following specifications: Dual AMD EPYC 7282, 16-Core Processors (64 logical CPUs) with 256 GB RAM.

The training data needs to be supplied in the form of txt files, and the algorithm also writes its results to txt files.

Three self-written Python scripts are included in the appendix:

- generate_input_data.py to convert the tabular dataset into the txt files needed for DBN-GOMEA. This script can also be instructed to split the data into a train and a test set, and split the training set into a specified number of folds for cross-validation.
- analyze_bn_probabilities.py further processes the model training output by performing tasks such as calculating the C-index, calculating ROC AUC, bootstrapping, and plotting the Bayesian network.
- collect_experiment_results.py to collect all results into a single CSV file suitable for visualization.

Experiments such as cross validation were performed by writing a Linux bash script that runs all necessary scripts in order; it starts by generating input data, then compiles the C++ program, then runs DBN-GOMEA in a nested loop over a range of complexity penalty values and discretization policies. After all training is done, evaluation of all the solutions is performed in another nested loop using analyze_bn_probabilities.py. Finally, the results are collected into a single csv file with collect_experiment_results.py.

The for-loops in the Linux bash script are parallelized. The server allows up to 64 processes to run in parallel, speeding up the computation time considerably.

Results

Univariate toy problem

Table 5 shows the training and test results for a BNs learned for overall survival versus tumor size, without any cross validation. This network has three nodes: *tumor size, survival time, survival event*. The BN learning algorithm self-discretizes the continuous features *tumor size* and *survival time* into an appropriate number of bins. The columns of Table 5 represent the following:

• discretization	- the discretization policy used
• complexity penalty	- the weight of the complexity term in the fitness function
• networks	- the resulting network structures represented as a genotype
• instantiations	- the number of bins for each of the three nodes
• fitness train	- the value of the fitness function on the training set
• C-index train/test	- the value of the C-index on the training/test set

Discretization	Complexity penalty	Networks	Instantiations	Fitness train	C-index train	C-index validation
Equal width	0	122/111	8-8-2	167	**0.66**	**0.61**
Equal width	0.1	222/211	8-8-2	167	**0.66**	**0.61**
Equal width	0.2	122	8-5-2	139	0.65	0.58
Equal width	0.3	201/202	4-5-2	127	0.59	0.53
Equal width	0.4	000	4-9-2	123	0.41	0.46
Equal width	0.5	000	4-9-2	120	0.41	0.46
Equal width	0.6	000	4-9-2	116	0.41	0.46
Equal frequency	0	112/221/222	9-9-2	221	**0.66**	0.54
Equal frequency	0.1	111/112/221/222	9-9-2	179	**0.66**	0.54
Equal frequency	0.2	221/222	9-8-2	138	**0.66**	0.54
Equal frequency	0.3	001	8-9-2	118	0.52	0.46
Equal frequency	0.4	000	8-9-2	113	0.51	0.44
Equal frequency	0.5	000	8-9-2	109	0.51	0.44
Equal frequency	0.7	000	8-7-2	101	0.56	0.44
Equal frequency	1	000	8-4-2	91	0.56	0.48
Bayesian score	0	210	2-2-2	58	0.61	0.58
Bayesian score	0.1	210	2-2-2	56	0.61	0.58
Bayesian score	0.2	210	2-2-2	55	0.61	0.58
Bayesian score	0.4	210	2-2-2	52	0.61	0.58
Bayesian score	0.6	210	2-2-2	50	0.61	0.58
Bayesian score	0.7	210	2-2-2	48	0.61	0.58
Bayesian score	0.8	000	2-2-2	47	0.5	0.5
Bayesian score	1	000	2-2-2	46	0.5	0.5

Table 5. Training and test results for the Bayesian network of overall survival versus tumor size.

The results show that for all discretization policies, the complexity of the network structure and the number of discretization bins decreases as the complexity penalty is increased. For a high enough complexity penalty, the algorithm always returns a '000' network structure, representing a network with independent nodes. For both the training and test set, the C-index seems to be the highest for a complexity penalty equal to 0. The highest training and validation C-indices are 0.66 and 0.61. This suggests that some predictive power is obtained, although without a measure of uncertainty it isn't clear whether this is statistically significantly different from 0.5.

Interestingly, the Bayesian Score discretization results returns the same network structure (*'210'*) and instantiations (*'2-2-2'*) for a wide range of complexity penalty values. This network is shown in Figure 7.

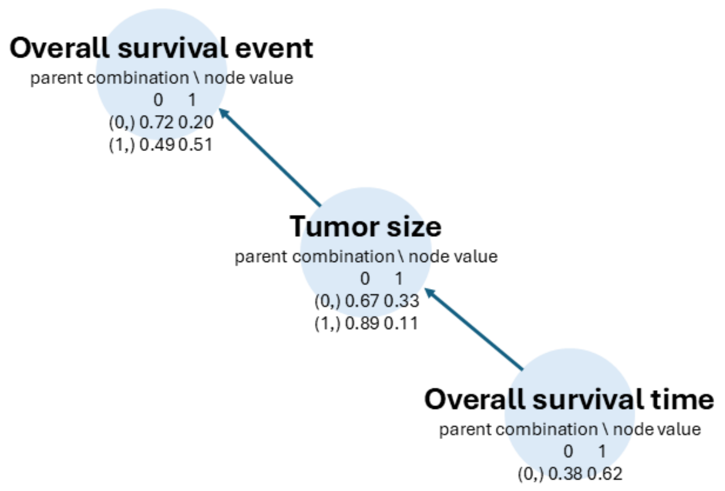

Figure 7. Bayesian network trained on toy problem using a Bayesian score-type discretization policy

This network can be used to sanity-check the result because the continuous nodes are both split into only two bins, allowing for easy intuitive evaluation. The conditional probability table learned for the node representing tumor size can be read from Figure 7, and is also shown below in Table 6.

P(tumor size | survival time)	Small tumor	Large tumor
Short survival	0.67	0.33
Long survival	0.89	0.11

Table 6. Conditional probability table for tumor size in the network shown in Figure 7

From this table, an odds ratio of $\frac{(0.89/0.11)}{(0.67/0.33)} \approx 3.99$ is obtained. In other words, small tumors are 4 times more common in long survivors compared to short survivors, which makes intuitive sense and is consistent with results from Horeweg et al [1].

The discretization boundaries learned are $65.5\,mm$ for tumor size and 24 months for survival time. Comparing this to the scatter plot of overall survival time and tumor size in Figure 3, the boundary for tumor size looks to be in a good position to separate the patients with long survival times from the ones with shorter survival.

As a final sanity check, a log-rank test is used to find the optimal boundary for tumor size by sweeping over a range of boundary positions for a binary tumor size feature and minimizing the P-value. The survival time is used as a continuous survival feature, and the tumor size is discretized into two bins. The log-rank test tests whether there is a statistically significant difference in survival between the small tumor and large tumor groups. The results in Figure 8 show that the biggest difference in survival between the two tumor size groups is obtained for a boundary position between 60 and 70 mm. The minimum is around 67 mm with a p-value of 10^{-5}. This matches very well with the discretization boundary of 65.5 mm that was found by DBN-GOMEA.

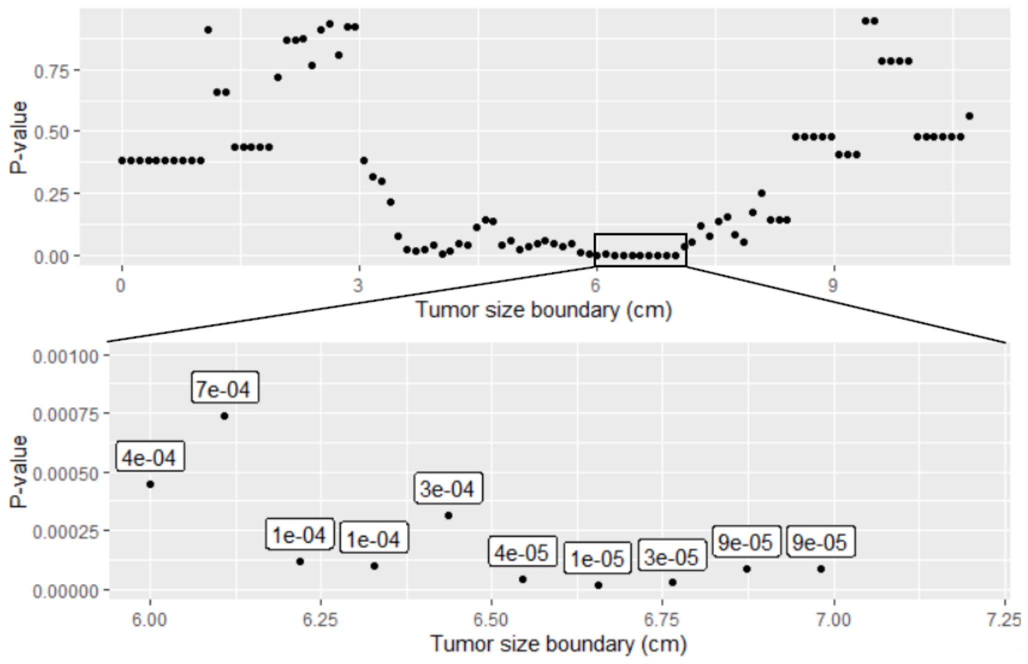

Figure 8. Log-rank test for overall survival versus tumor size, swept over a range of discretization boundary positions to split tumor size into a binary variable (bottom plot is a zoomed-in version of top plot, with labels showing the p-values)

From this it can be concluded that DBN-GOMEA generated a model with reasonable results; the modelled direction of the effect and the proposed discretization are consistent with a classical survival model that does properly handle censored data.

Time-to-event survival analysis

Metastasis-free survival

BNs for metastasis-free survival time were trained using cross validation. The networks have six nodes: *survival time*, *survival event* (0/1); *tumor size*; *FIGO stage* (≤2a vs 2b-4); *nodal involvement* (N0 vs N+); and *treatment time* (days).

The nodes *survival time*, *tumor size* and *treatment time* are continuous features and will be discretized by DBN-GOMEA.

Figure 9 shows the training and validation results.

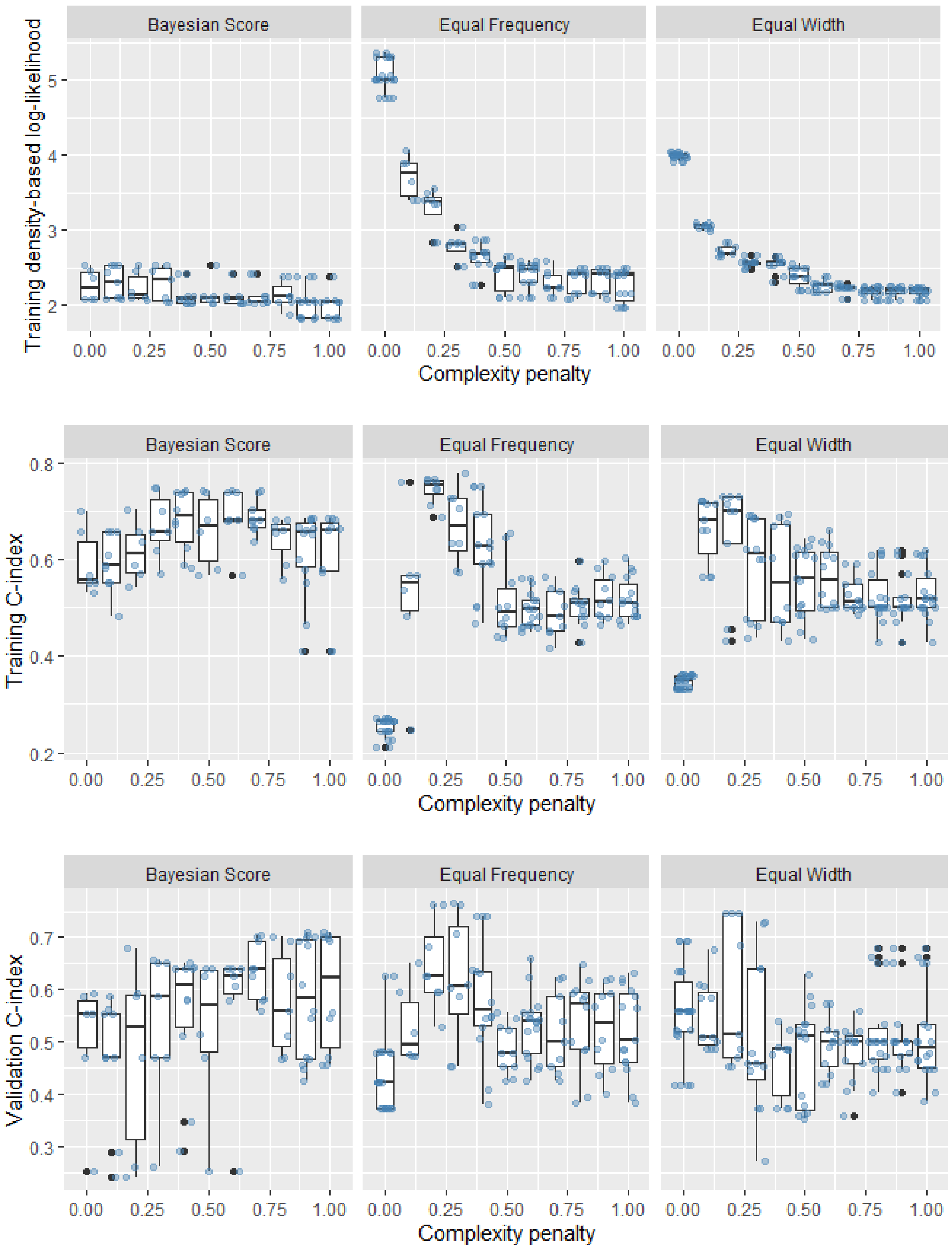

Figure 9. Hyperparameter tuning through cross validation of Bayesian networks for metastasis-free survival. The three rows of plots show training log-likelihood, training C-index and validation C-index as a function of the complexity penalty. The three columns represent the three types of discretization policies.

The first two rows of plots show the training density log-likelihood and training C-index. As expected the training log-likelihood decreases with an increasing complexity penalty as the complexity penalty reduces overfitting to the training set. In contrast, Bayesian discretization does not directly optimize the log-likelihood, unlike the other two methods, and so it does not show this trend. The training C-index regresses to 0.5 for a high enough complexity penalty, indicating underfitting. Interestingly, a complexity penalty of 0.0 can lead to a training C-index well below 0.5, indicating a *discordant* model. This behavior will be explored further in the discussion section.

The final row of plots shows the validation C-index, which is used to select the final model. Based on these plots, two models with the highest average validation C-index are selected: a Bayesian discretization model with a complexity penalty of 0.6, and an Equal-Frequency model with complexity penalty 0.2. Two models were selected as they both performed equally well on the validation set. No Equal-Width model was selected due to the considerably lower validation C-index.

After retraining on the full training data and evaluating on the test set, the final BN model results are summarized in Table 7.

Model	**Discretization**	**Complexity penalty**	**C-index training**	**C-index test (SD)**
Bayesian network	Equal-Frequency	0.2	0.75	0.58 (0.07)
Bayesian network	Bayesian discr.	0.6	0.69	0.63 (0.08)

Table 7. Bayesian Network final test results for metastasis-free survival

The Bayesian score discretization performed best on the test set with a test C-index of 0.63, indicating that on average, for 63% of all pairs of observations in the test set, the model predicted the correct ordering of survival time.

The results for the reference models are shown in Table 8.

model	**C-index training (SD)**	**C-index validation (SD)**
Cox proportional hazards	0.68 (0.02)	0.64 (0.02)
Random survival forest	0.90	0.70 (0.14)

Table 8. Cross validation results for reference models for metastasis-free survival

Comparing these results to the BN results, it can be concluded that the BN with Bayesian discretization performs comparable to the Cox proportional hazards model in terms of its training and test C-index, although with greater confidence intervals. The random survival forest seems to outperform the other models, but with a large confidence interval.

Overall survival

BNs for overall survival time have been trained using cross validation. The networks have 9 nodes: *survival time*, *survival event* (0/1); *age*; *diabetes*; *previous other malignancy*; *tumor size*; *FIGO stage* ($\leq$2a vs 2b-4); *nodal involvement* (N0 vs N+); and *treatment time* (days).

The nodes *survival time*, *age*, *tumor size* and *treatment time* are continuous features and will be discretized by DBN-GOMEA.

Figure 10 shows the training and validation results.

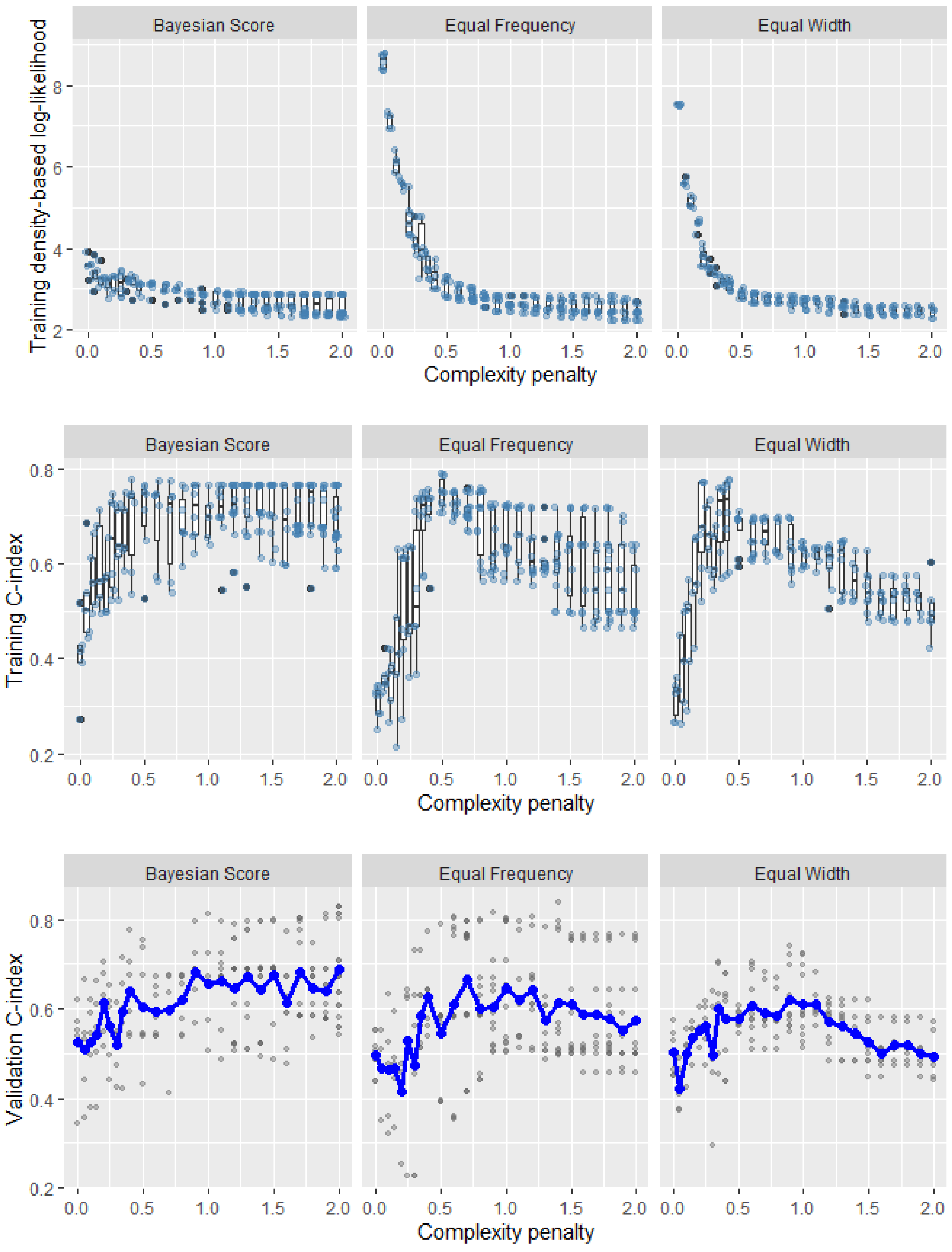

Figure 10. Hyperparameter tuning through cross validation of Bayesian networks for overall survival. The three rows of plots show training log-likelihood, training C-index and validation C-index as a function of the complexity penalty. The three columns represent the three types of discretization policies.

The training results in the first two rows of plots show a similar pattern to the metastasis-free models in Figure 9. The validation C-index in the last row shows a very high degree of variability, and a 'slower' trend with respect to complexity penalty. The higher number of nodes for the same amount of training data perhaps makes the model more susceptible to overfitting, requiring a higher complexity penalty to combat it.

From these results, the final selected models are Equal-Frequency with a complexity penalty of 0.7 and Bayesian Discretization with a complexity penalty of 1.7. The results on the test set, together with the reference models for comparison, are shown in Table 9.

Model	Discretization	Complexity penalty	C-index train	C-index test (SD)
Bayesian network	Equal-Frequency	0.7	0.69	0.60 (0.07)
Bayesian network	Bayesian discr.	1.7	0.49	0.50 (0.01)
Cox PH	-	-	-	0.66 (0.11)
Random survival forest	-	-	-	0.65 (0.15)

Table 9. Final BN test results and cross-validated reference model results for overall survival

On the test set, the Bayesian discretization BN showed no predictive power at all. The equal-frequency BN shows some indication of predictive power, but with a lower test C-index than the Cox PH model as well as the random survival forest.

Fixed-time survival classification

12-month metastasis free survival

BNs for 12-month metastasis-free survival status have been trained using cross validation. The networks have 6 nodes: *12-month survival status* (0/1); *tumor size*; *FIGO stage* (≤2a vs 2b-4); *nodal involvement* (N0 vs N+); and *treatment time* (days).

The nodes *tumor size* and *treatment time* are continuous features and will be discretized by DBN-GOMEA.

Figure 11 shows the training and validation results.

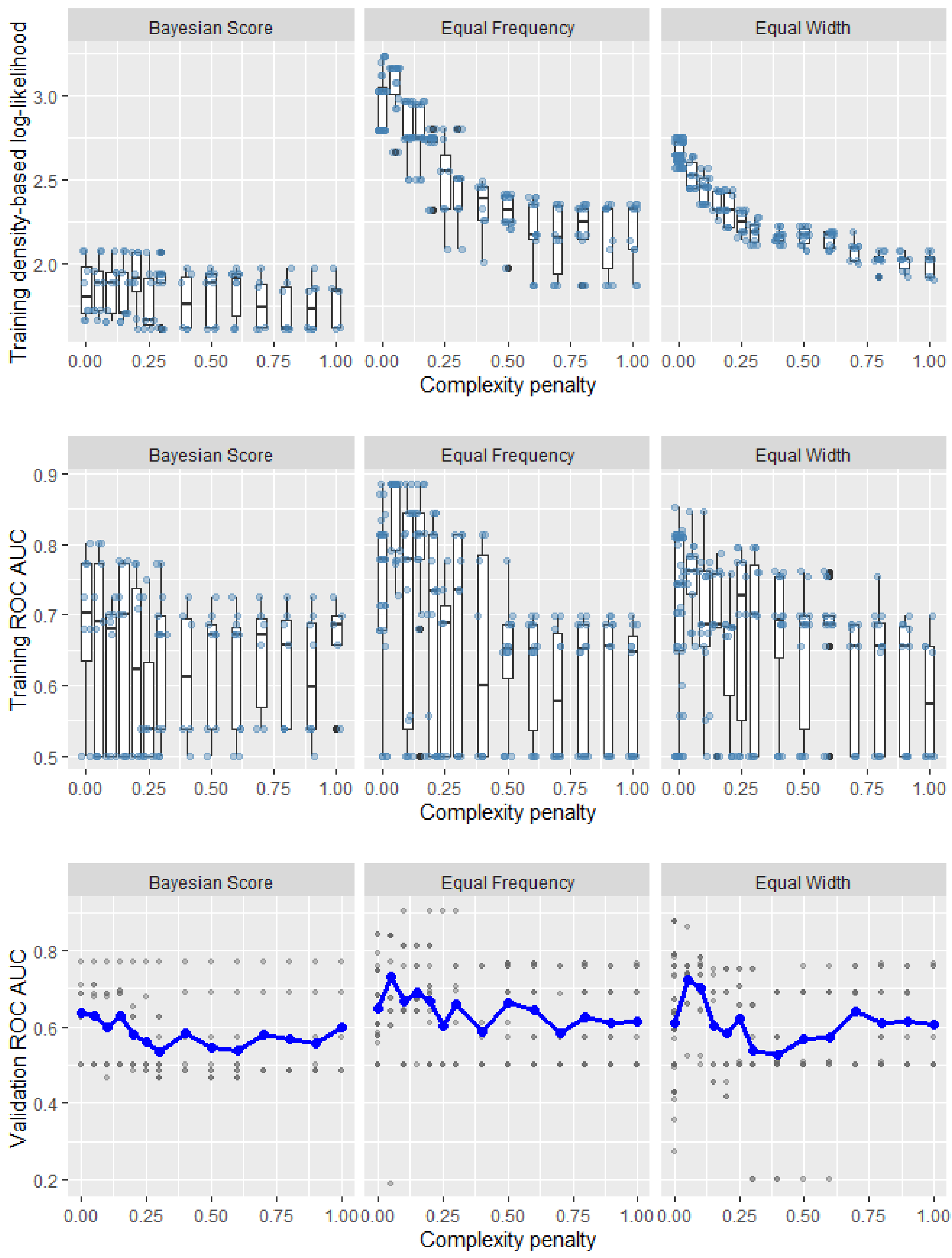

Figure 11. Hyperparameter tuning through cross validation of Bayesian networks for 12-month metastasis-free survival. The three rows of plots show training log-likelihood, training C-index and validation C-index as a function of the complexity penalty. The three columns represent the three types of discretization policies.

Compared to modelling of continuous survival time rather than a fixed-timepoint survival, the training and validation ROC shows a great deal of variability. The final selected model is *Equal-Frequency* with a 0.05 complexity penalty.

This model's performance on the test set after retraining on the full training set is presented in Table 10.

Model	Discretization	Complexity penalty	AUC training	AUC test (SD)
Bayesian network	Equal-Frequency	0.05	0.75	0.90 (0.04)
Cox PH	-	-	0.76	0.74 (0.12)

Table 10. Final model results for 12-month metastasis-free survival

For the bootstrapped AUC on the test set, a very high value of 0.90 was obtained, even considerably higher than the training AUC. While this is evaluated on an unseen test set without any cherry picking, it is thought unlikely that this is a true measure of the model's performance on unseen data. Since the training AUC is lower, it is likely that the model just happened to fit very well to the test set. This is possible in a low-sample, high variance setting like this.

Post-hoc analysis

Figure 11 already showed high variability in AUC across different folds used in cross-validation. This variability in AUC for a single complexity penalty is largely due to the fact that for different folds, different network structures were learned with varying performance on test data. For a fixed complexity penalty, the random seed used in splitting the training/test set was found to strongly affect which network structure was learned, which in turn strongly affects the AUC ROC. The validation sets are rather small and contain only 0s and 1s, most likely unbalanced as well, so the evaluation result is subject to high variability.

To get a more realistic approximation of test performance than 0.90, repeated holdout validation is used: the process of training a final model with complexity penalty 0.05 and then bootstrapping the performance on the test was repeated for 30 different seeds for the random split of training and test data, resulting in 91 unique learned "final" networks (many of them equivalent). For all these solutions, the bootstrapped "test" AUC and its variability is plotted as a function of the training AUC in Figure 12. The error bars represent the standard deviation obtained from bootstrapping for a fixed network structure. The scatter of many datapoints represent different network structures learned for different random splits between training and test set. The plot reveals a strong correlation between the training and test performance; networks that score well on training tend to also score well on the test set, most likely indicating that it is simply a better network for the data.

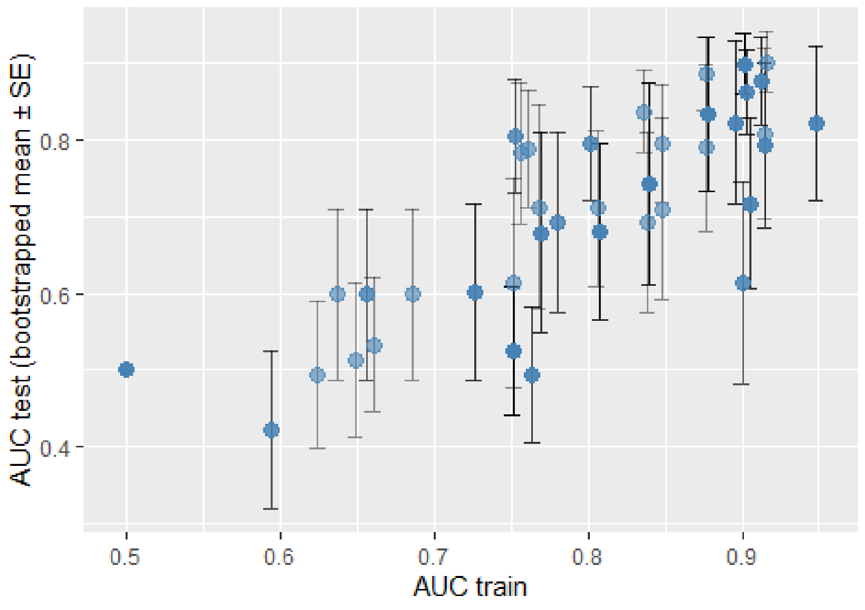

Figure 12. Bootstrapped mean and standard deviation for test AUC versus train AUC, obtained for 30 different random splits of train and test set.

Choosing the best network from this collection would lead to an upwards biased estimate of the test performance. To get a better estimate of the model's test set performance, the network architecture should not be selected based on test-set performance, but on a validation set performance. That is, choose the *network structure* that performed best in cross validation on the validation set, fix this architecture and optimize the discretization boundaries on the full training set, and evaluate the performance on the test set. However, the current implementation of DBN-GOMEA does not allow to fix the network architecture and will always learn a new architecture from scratch. The repeated holdout validation provides a solution here: we can identify the network architecture(s) that worked best in cross validation, and then find matching networks among 1000 repeated random holdout runs and average their test set performance. The best network from was chosen by pooling solutions from all 5 folds and their validation AUC, and choosing the network(s) with the highest validation AUC. Note that DBN-GOMEA can learn multiple equivalent network structures in a single run because equivalent network structures have the same fitness score.

In cross-validation (shown earlier in Figure 11), the following network architectures performed equally well, with a training AUC of 0.89 and a validation AUC of 0.84, one of which is plotted in Figure 13:

{1111110201}, {**1121220201**}, {2221110101}, {2111110201}, {2111110101}, {2211110101}, {1211210101}, {2211210101}, {2121120201}, {1221220201}

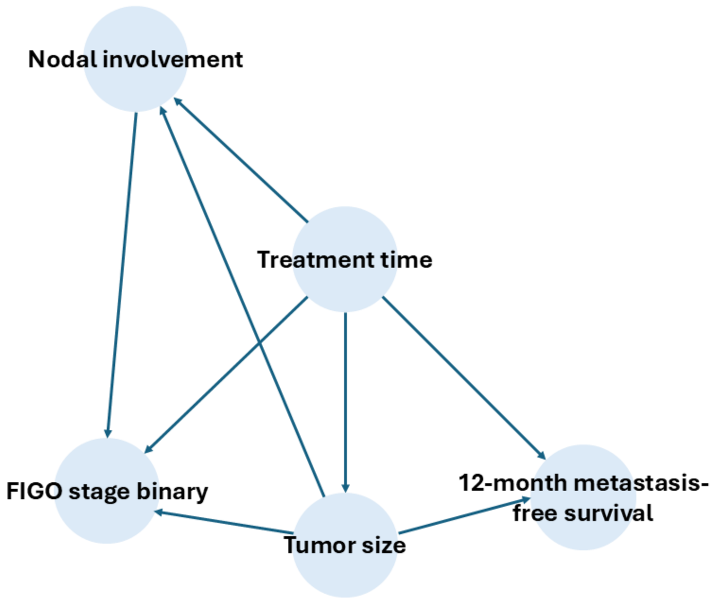

Figure 13. Graphical representation of one of the networks with the best performance in cross-validation (genotype {1121220201})

Finding these networks among 1000 evaluations with different seeds gives the **bold** results in table Table 11. This table also shows the average performance over all networks, rather than only those selected.

Model	Discretization	Complexity penalty	AUC training	AUC test (SD)
Bayesian network – averaged result over *all* networks	Equal-Frequency	0.05	0.78	0.73 (0.08)
Bayesian network – average over best networks from CV	**Equal-Frequency**	**0.05**	**0.83**	**0.79 (0.09)**

Table 11. Post-hoc analysis for 12-month metastasis-free survival

This estimate is higher than the AUC obtained from the Cox PH model, although it might still be biased upwards because there is overlap between the test data used by the many random splits and the training data used for cross validation that informed the choice of network architecture.

DVH metrics versus toxicity

Finally, the influence of the DVH metrics on toxicity is explored. A quick scan for strong univariate correlations between all combinations of DVH metric features and toxicity features was performed. The top 15 results by p-value out of more than 350 combinations are shown in Table 12.

DVH metric feature	Toxicity feature	P-value
Sigmoid V30 rel. volume	**toxicity grade vaginal dryness**	**0.003**
Body V43 abs. volume	**toxicity grade chronic kidney disease**	**0.004**
Body V36 abs. volume	toxicity grade chronic kidney disease	0.006
Sigmoid V40 rel. volume	toxicity grade vaginal dryness	0.009
Bowel V30 abs. volume	toxicity grade vaginal dryness	0.018
Rectum V40 rel. volume	**toxicity grade abdominal infection**	**0.022**
Bowel V15 abs. volume	toxicity grade vaginal dryness	0.023
Bowel V40 abs. volume	toxicity grade abdominal pain	0.023
Bladder V40 rel. volume	toxicity grade chronic kidney disease	0.024
Body V36 abs. volume	toxicity grade constipation	0.025
Bowel V40 abs. volume	toxicity grade vaginal dryness	0.025
Body V36 abs. volume	toxicity grade urinary frequency	0.026
Body V43 abs. volume	toxicity grade urinary frequency	0.026
Bladder V30 rel. volume	toxicity grade chronic kidney disease	0.028
Rectum V30 rel. volume	toxicity grade urinary incontinence	0.030

Table 12. Kruskal-Wallis H-test results with highest p-values for all combinations of DVH metric feature and toxicity feature

The combinations marked with bold text are shown as box plots in Figure 14.

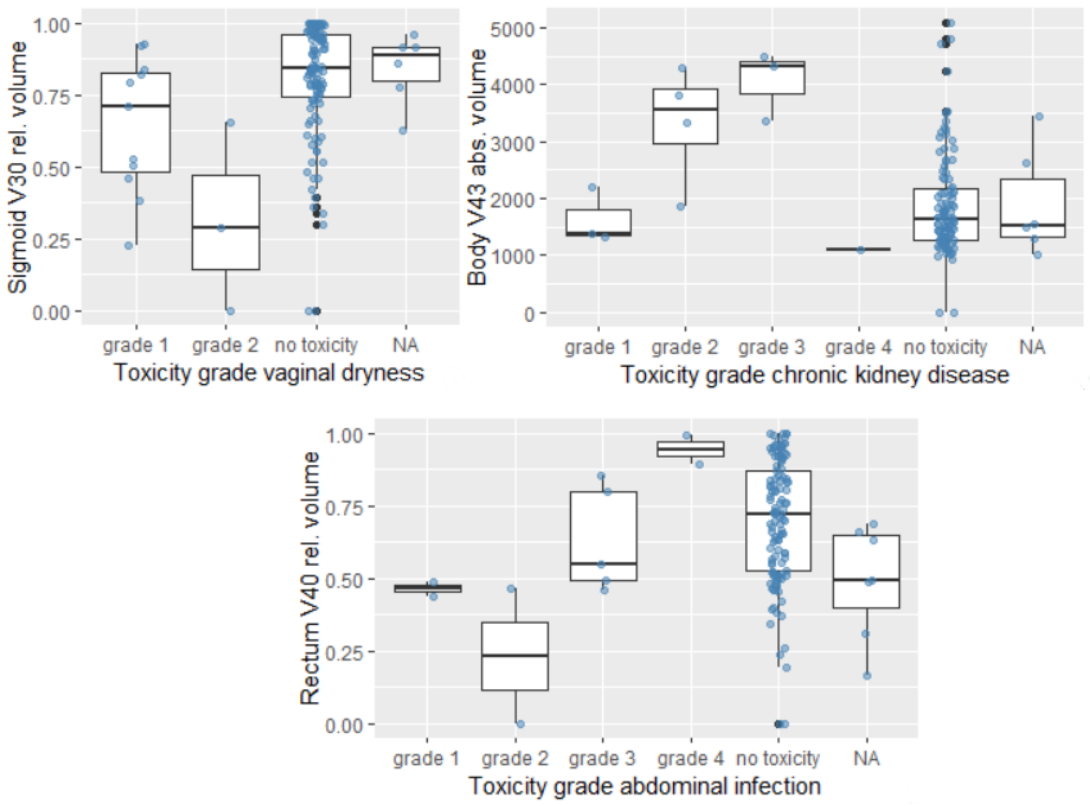

Figure 14. Boxplots for the pairs of DVH-metric and toxicity feature marked bold in Table 12

Urinary toxicity

Exploratory data analysis results shown earlier in Figure 5 revealed a counter-intuitive negative univariate correlation between bladder V30 and urinary incontinence. The Kruskal-Wallis H-test also shows this as a negative coefficient for *Bladder V30 rel. volume* (p=0.02).

The apparently negative correlation may well be due to confounders, it is therefore important to know which other factors are correlated with urinary toxicity. When running univariate logistic regression tests for *urinary_tox_binary* versus every other column in the data, it was found that various date-related features (such as the treatment starting date, the data entry date, etc.) show the strongest correlations, with p-values between 10^{-10} and 10^{-6}. The sign of the correlations indicate that over time, these urinary toxicities became more common in the study population.

In Figure 4 it was shown that after a treatment protocol change in 2015, DVH metrics were improved. Figure 15 shows a comparison of urinary toxicity-free survival curves of patients treated before versus after this protocol change, confirming that the patients before 2015 had better survival outcomes.

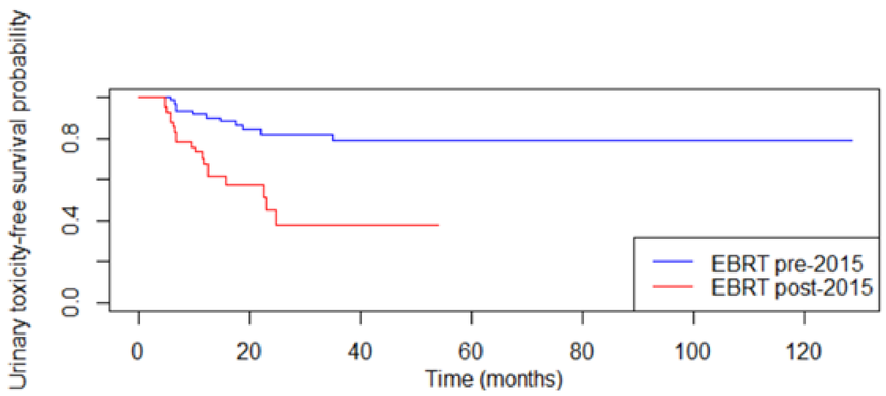

Figure 15. Kaplan-Meier curves for combined urinary toxicity-free survival, comparing two groups of patients: those treated before a protocol change in 2015 and those after

This surprising result coincides with an improvement in DVH metrics, which potentially makes it difficult to demonstrate a positive effect of the better DVH metrics.

Besides date, other factors that have a strong univariate correlation with *urinary_tox_binary* are *age*, *history of urological disease*, *gynaecological surgery*, *EQD2$_{tot}$ ICRU bladder* and *FIGO-classification* of the tumor. *EQD2$_{tot}$ ICRU bladder* is a measure for the dose to the bladder from brachytherapy, rather than external beam radiotherapy. A multivariate logistic regression model for *urinary_tox_binary* and these features gives the results shown in Table 13.

Regressor	**Estimate**	**Std.error**	**Z-value**	**Pr(>|z|)**
(Intercept)	15.73	1481.00	0.011	0.99
Gynaecological surgery	2.55	1.22	2.092	0.04*
History of urological disease	-18.36	1481.00	-0.012	0.99
EQD2$_{tot}$ ICRU bladder	-0.004	-0.001	-2.875	0.004**
Bladder V40 rel. volume	1.95	1.36	1.428	0.15
EBRT after 2015 protocol change	2.49	0.83	2.996	0.003**
FIGO stage 2b – 4	0.71	0.70	1.017	0.31

Table 13. Logistic regression results for outcome urinary_tox_binary. A positive coefficient estimate indicates that a higher value of that regressor is correlated with a higher rate of toxicity.

When correcting for these confounders, the *Bladder V40 rel. volume* DVH metric is no longer statistically significant. The binary date variable (*EBRT after 2015 protocol change)* is statistically significant ($p = 0.003$), as well as the brachytherapy dose given to the bladder, with an unexpected negative sign indicating that higher dose corresponds to less combined urinary toxicity.

Discussion

Interpretation and drawbacks

The main focus of this study is to train Bayesian networks using DBN-GOMEA on a real-world clinical dataset and evaluate predictive power. BNs are of interest because of their potential for explainable AI. Investigating how to present models effectively lies outside the scope.

In a full multivariate network trained for metastasis-free survival, a test C-index of 0.63 with a standard deviation of 0.08 was obtained for DBN-GOMEA, which was comparable to a Cox regression model, although with higher variability. The full network trained for overall survival resulted in a test C-index of 0.60 with a standard deviation of 0.07, a slightly lower estimate than Cox PH. Despite higher flexibility of the Bayesian network, it was not able to achieve a higher C-index on an unseen test set.

A fundamental drawback of learning BNs for survival data is that BNs are not designed to properly handle censored data. By introducing one node for survival time and one node for survival event (0=censor, 1=event), it was possible to mimic a survival model, but in this way censored and non-censored patients are likely modeled as two separate groups and results can be expected to be biased. Another drawback is that the continuous survival-time feature is discretized into 9 bins, causing a loss of information. It is possible that these fundamental shortcomings are why the BNs for survival time do not manage to outperform a simple Cox regression model on unseen test data, despite being more flexible. Results for a random survival forest – an ML model that can handle censored data – indicate that the Cox regression model might be outperformed in terms of C-index with the right model, although due to large confidence intervals this cannot be stated with certainty. Note that random survival forests are not as suited for XAI as BNs; it has drawbacks with regards to explainability due to it being an ensemble method.

To circumvent the issue of censored data, a fixed-timepoint analysis was performed by learning BNs for 12-month metastasis-free survival. These results seem promising although they suffer from a high degree of variability, and the evaluation on the fixed test set resulted in an overly optimistic test AUC of 0.90, considerably higher than the training AUC. A post-hoc analysis using repeated holdout validation resulted in a better approximation of test set performance showing good results, but these might contain some bias. This is because the test sets in repeated holdout partially overlap with the training data used in cross validation to select the complexity penalty and the best-performing network structures. This is a rather mild form of leakage, because in each repetition, the test set is still held out from training, and the model parameters are kept fixed; no additional hyperparameter tuning or selection is performed on repeated holdouts. Despite the potential bias, the obtained estimate (an AUC of 0.79) is deemed more useful than the overly optimistic performance on the fixed test set.

In an earlier simulation study on DBN-GOMEA [3], the sample size of the training set was found to be important to get accurate results. In the simulation study, data was generated from known BNs. DBN-GOMEA was applied on this data to evaluate whether it could correctly reobtain the ground truth BNs. The accuracy and sensitivity achieved by DBN-GOMEA was found to increase with an increasing sample size and did not plateau even past 50,000 samples. The real-world dataset of 280 patients in comparison is small and results from the simulation study suggest that this will likely limit the effectiveness of the learning algorithm.

Low training C-index

In Figure 9, a *training* C-index for metastasis-free survival as low as 0.2 – 0.3 was calculated when no complexity penalty is applied. This is a surprising result because a non-predictive model should result in a C-index of 0.5. A value below 0.5 indicates a discordant model that is more likely to make predictions in the wrong direction; this is unexpected especially on the training set.

To check why this is the case, the predicted mean survival time was plotted as a function of true survival times for both a discordant model (penalty = 0.0) and a concordant model (penalty = 0.2). Figure 16 shows that the discordant model (top row) produces many zero-valued predictions. This in itself would not result in a discordant model, but the histogram on the top right shows that zero-valued predictions are disproportionally assigned to patients with longer observed survival times. This makes the model discordant. The histograms show that the majority of survival times are in fact short so it is possible that a model that produces too many zero-valued survival times can produce a high log-likelihood. During training, the log-likelihood is maximized, not the C-index. This discordant effect does not hold up on a validation set, indicating it results from spurious correlations in the training set.

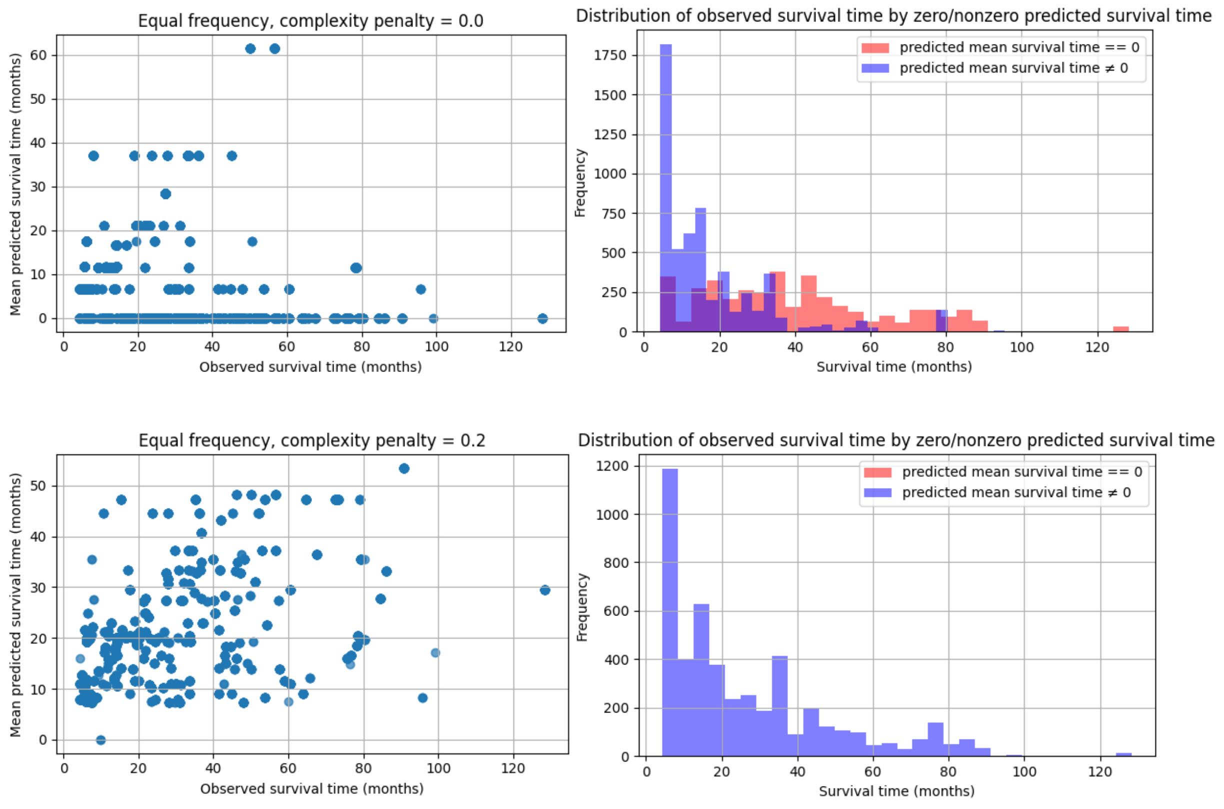

Figure 16. Left: scatterplot of BN-predicted mean metastasis-free survival time versus observed survival time. Right: histogram of observed metastasis-free survival times, grouped by whether the predicted survival time is zero or non-zero. Top: results for discordant model with complexity penalty 0.0, resulting in a training C-index of 0.2 – 0.3. Bottom: results for concordant model with complexity penalty 0.0, resulting in a training C-index of 0.75.

DVH metrics versus toxicity

Exploratory data analysis of DVH metrics and urinary toxicity revealed an apparent paradox in the data: while DVH metrics did improve over time, meaning critical organs received less dose, the occurrence of toxicity actually increased. The increase in toxicity over time is not understood and outside the scope of this research.

Some notable drawbacks of the exploratory analysis and possible explanations of counterintuitive are:

- Radiotherapy treatment trade-offs. A decrease in the dose to one critical organ may increase dose to another unsegmented organ not captured by a DVH metric;
- Incomplete treatment information. Clinically, brachytherapy and external beam radiotherapy are considered cumulatively and both influence toxicity. However, only external radiotherapy DVH metrics were considered here. An improvement of DVH metrics for external beam radiotherapy

can create room for the brachytherapy plan to increase the dose to the critical organs, potentially offsetting the gains.

- Metric limitations. DVH metrics are limited in that they describe a 3D dose distribution with a single number. A highly inhomogeneous dose distribution with strong "hotspots" can still obtain a good value for V30. The bladder in particular is interesting because it is a hollow organ filled with urine. Only the dose to the bladder wall could lead to toxicity, yet DVH metrics include the entire volume.
- Need for clinical insight. Understanding why toxicity seems to increase over time requires collaboration with a clinician to narrow the scope and make the problem more manageable.
- Small amount of data. The full dataset consists of 280 patients, but dose distributions are only known for 158 patients.

Ideas for future research

DBN-GOMEA for learning BNs has been shown before to perform well in a simulation study with self-generated data. For further research into the application of this technique on real-life datasets, it is recommended to use a benchmark dataset with more observations (preferably >500) and a classification target rather than censored survival times, allowing a fair comparison with modern ML methods and clearer performance evaluation.

The BN results for time-to-event data were similar or worse than Cox PH model results. Modelling survival-time this using two BN-nodes (time and event indicator) has fundamental flaws which could partially explain poor test performance. Kraisangka et al. [17] have studied BN interpretations of Cox models that are able to handle censored data. However, these networks were generated based on Cox regression coefficients and not learned from data. The training data will have to be remapped in order to learn a BN this way; it is not clear whether this is possible but it would be worth investigating.

In contrast to the BN results for time-to-event data, results for binary classification of 12-month metastasis-free survival status seem promising and warrant further study by improving the process of model selection through cross-validation. The post-hoc analysis performed can still lead to optimistic estimates, this problem can be solved by nested cross-validation and/or by reprogramming DBN-GOMEA.

In this study, cross-validation was used to find the optimal value of the complexity penalty based on the validation C-index. Then, a new Bayesian network was completely retrained using this value of complexity penalty. Rather than retraining the whole network, a better approach is to fix the network structure from the best validation C-index and only optimize the conditional probability tables and perhaps discretization boundaries. It is highly recommended to apply such an analysis to the results in Figure 10, as the individual datapoints for validation C-index show three 'levels' of performance: around 0.5, around 0.6 and around 0.8. It might be that the network structures with a validation C-index of 0.8 perform much better on the unseen test set than what was achieved now.

This study uses the same feature sets as prior work on the same data. However, the full dataset contains many more features that were not considered for the models. It is worth looking into feature selection techniques to potentially find a better set of features and obtain more predictive power.

The explainability of the BN was not yet explored in this study and will require further research, focusing on interpretable model presentation and ways of quantifying and communicating feature importance.

For research into the relationship between DVH metrics and toxicity, due to the large scope and the limitations of BNs for modeling survival data, it is recommended to separate this research question from the research into BNs for explainable AI. That is, to use established state-of-the-art ML techniques for survival data, for example random survival forests [12], deep-learning based methods [18] or gradient-boosting methods [19] to model the seemingly complex relationship between DVH metrics and toxicity.

Conclusions

BNs were learned on the clinical dataset using Discretizing Bayesian Network Gene-pool Optimal Mixing Evolutionary Algorithm (DBN-GOMEA), targeting three different outcomes: metastasis-free survival time, overall survival time, and 12-month metastasis-free survival.

DBN-GOMEA was compared with Cox-Regression. DBN-GOMEA for **metastasis-free survival time** scored a test C-index of 0.63 (sd=0.08) and for **overall survival time** scored a test C-index of 0.60 (sd=0.07). These results likely indicate some predictive power, but the performance is comparable to or perhaps worse than the Cox regression model results.

For **12-month metastasis-free survival, DBN-GOMEA** does not suffer from fundamental issues regarding censored data, and better results were obtained. Using cross-validation for model selection on the training set and then evaluating performance on a fixed test set resulted in an unrealistically high test AUC of 0.90 (sd=0.04). Post-hoc analysis was performed to get a better estimate of the test performance, which returned an AUC of 0.79 (sd=0.09). However, this score might still be biased. The scores obtained are as good as or better than Cox regression. DBN-GOMEA shows great promise to be used as a predictive model for this data. More work is needed to improve the model selection phase, and to exploit the explainability of the network.

The relationship between urinary **DVH metrics and toxicity** was explored. No clear indicators for a correlation was found based on exploratory box plots and logistic regression models. Furthermore, the data suggests a complicated relationship between DVH metrics and toxicity. DVH metrics improved over time, yet occurrences of urinary toxicity became more common. An in-depth analysis incorporating clinical knowledge to identify all relevant confounders and trade-offs between treatment modalities would be required to understand this problem in further research.

References

[1] N. Horeweg, C. L. Creutzberg, E. C. Rijkmans, M. S. Laman, L. A. Velema, V. L. Coen, T. C. Stam, E. M. Kerkhof, J. R. Kroep, C. D. De Kroon en R. A. Nout, „Efficacy and toxicity of chemoradiation with image-guided adaptive brachytherapy for locally advanced cervical cancer," *International Journal of Gynecological Cancer,* vol. 29, nr. 2, pp. 257-265, 2 2019.

[2] J. C. Dimopoulos, P. Petrow, K. Tanderup, P. Petric, D. Berger, C. Kirisits, E. M. Pedersen, E. Van Limbergen, C. Haie-Meder en R. Pötter, „Recommendations from Gynaecological (GYN) GEC-ESTRO Working Group (IV): Basic principles and parameters for MR imaging within the frame of image based adaptive cervix cancer brachytherapy," *Radiotherapy and Oncology,* vol. 103, nr. 1, pp. 113-122, 4 2012.

[3] D. M. F. Ha, „Learning Discretized Bayesian Networks with GOMEA".

[4] E. P. a. Council, *Artificial Intelligence Act,* OJ L, 2024/1689, 12.7.2024, 2024.

[5] M. Kawamura, T. Kamomae, M. Yanagawa, K. Kamagata, S. Fujita, D. Ueda, Y. Matsui, Y. Fushimi, T. Fujioka, T. Nozaki, A. Yamada, K. Hirata, R. Ito, N. Fujima, F. Tatsugami, T. Nakaura, T. Tsuboyama en S. Naganawa, *Revolutionizing radiation therapy: the role of AI in clinical practice,* vol. 65, Oxford University Press, 2024, pp. 1-9.

[6] K. Sheng, *Artificial intelligence in radiotherapy: a technological review,* vol. 14, Higher Education Press, 2020, pp. 431-449.

[7] C. Hurkmans, J. E. Bibault, K. K. Brock, W. van Elmpt, M. Feng, C. David Fuller, B. A. Jereczek-Fossa, S. Korreman, G. Landry, F. Madesta, C. Mayo, A. McWilliam, F. Moura, L. P. Muren, I. El Naqa, J. Seuntjens, V. Valentini en M. Velec, *A joint ESTRO and AAPM guideline for development, clinical validation and reporting of artificial intelligence models in radiation therapy,* vol. 197, Elsevier Ireland Ltd, 2024.

[8] National Electrical Manufacturers Association, *NEMA PS3 / ISO 12052, Digital Imaging and Communications in Medicine (DICOM) Standard,* Rosslyn, VA, USA.

[9] D. Thierens en P. A. Bosman, *Optimal Mixing Evolutionary Algorithms,* ACM, 2013, pp. 617-624.

[10] G. Schwarz, *Estimating the Dimension of a Model,* vol. 6, 1978, pp. 461-464.

[11] Y.-C. Chen, T. A. Wheeler en M. J. Kochenderfer, „Learning Discrete Bayesian Networks from Continuous Data," 2017.

[12] H. Ishwaran, U. B. Kogalur, E. H. Blackstone en M. S. Lauer, „Random survival forests," *Annals of Applied Statistics,* vol. 2, nr. 3, pp. 841-860, 9 2008.

[13] P. Shahmirzalou, M. J. Khaledi, M. Khayamzadeh en A. Rasekhi, „Survival analysis of recurrent breast cancer patients using mix Bayesian network," *Heliyon,* vol. 9, nr. 10, 10 2023.

[14] F. Harrell, R. Califf, D. Pyor, K. Lee en R. Rosati, „Evaluating the yield of medical tests," *J. Amer. Med. Assoc.,* vol. 247, p. 2543–2546, 1982.

[15] T. Therneau en E. Atkinson, „Concordance,” 2024.

[16] P. J. Heagerty en Y. Zheng, „Survival Model Predictive Accuracy and ROC Curves,” 2005.

[17] J. &. D. M. J. Kraisangka, „A Bayesian network interpretation of the Cox's proportional hazard model,” *International Journal of Approximate Reasoning,* vol. 103, p. 195–211, 2018.

[18] J. L. Katzman, U. Shaham, A. Cloninger, J. Bates, T. Jiang en Y. Kluger, „DeepSurv: Personalized treatment recommender system using a Cox proportional hazards deep neural network,” *BMC Medical Research Methodology,* vol. 18, nr. 1, 2 2018.

[19] Y. Chen, Z. Jia, D. Mercola en X. Xie, „A gradient boosting algorithm for survival analysis via direct optimization of concordance index,” *Computational and Mathematical Methods in Medicine,* vol. 2013, 2013.

Appendix (software code)

Code was written in Python to generate input data for the training algorithm, analyze the results, and collect the results into a single CSV file. Generate_input_data.py and collect_experiment_results.py were fully written by me. For the analysis I built on an existing script analyze_bn_probabilities.py. Below I only attached the classes and functions that I wrote, and left out the code written by my supervisor.

Git was used with a private repository for version control which also includes the C++ code for the training algorithm (DBN-GOMEA).

The DBN-GOMEA code was written by my supervisor, it is not attached here as I did not work on it.

```python
# this print-out contains only the classes and functions in analyze_bn_probabilities.py that I wrote
# code written by my supervisor before the start of my thesis is left out

@dataclass
class AnalysisResults:
    """class for storing analysis training and test results such as likelihood and concordance indices."""
    discretization_policy: str
    complexity_factor: float
    seed: str
    fold_nr: float
    network_str: str
    instantiations_str: str
    boundaries_str: str
    fitness_train: float
    log_l_mass_train: float
    log_l_density_train: float
    log_l_mass_test: float
    log_l_density_test: float
    c_index_train: float
    c_index_test: float
    bootstrap_mean_c_index_test: float
    bootstrap_stde_c_index: float
    bootstrap_all_c_indices: str
    str_probabilities: str

    def save_results_to_file(self, fig, output_path, solution_index):
        # if results.csv already exists, remove it
        filepath = os.path.join(output_path, f'result.csv')
        if solution_index == 0 and os.path.exists(filepath):
            print(f"File {filepath} already exists and was cleared")
            os.remove(filepath)
        # append analysis results to csv file (a new row for each solution)
        with open(os.path.join(output_path, f'result.csv'), 'a', newline='') as f:
            df = pd.DataFrame([self])
            df = df.applymap(lambda x: str(x).replace('\n', '\\n') if isinstance(x, str) else x)
            f.write(df.to_csv(sep=';', index=False, header=(solution_index == 0)))

        # save string of conditional probabilities to file
        with open(os.path.join(output_path, f'{self.network_str}_str_probabilities.txt'), 'w') as f:
            f.write(self.str_probabilities)

        # save figure of network to file
        fig.savefig(os.path.join(output_path, f'{self.network_str}_bn_graph.png'), format='png')


def get_bin_index(value, boundaries):
    # get bin index given a value and the boundary values
    for i in range(len(boundaries)):
        if value <= boundaries[i]:
            return i
    return len(boundaries)  # if the value is larger than the last boundary


def compute_likelihood(data_eval_filepath: str,
                       data_train_path: str,
                       dict_p_conditional: dict,
                       count_info: MSS_Solution_counts,
                       solution: MO_Solution
                       ):
    """
    compute log-likelihood for model defined by dict_p_conditional, count_info and solution, given the evaluation data
    in data_eval_filepath.
    :param data_eval_filepath: path to evaluation dataset
    :param data_train_path: path to training dataset, used to find lower/upper limits for lower/upper discretization
bins
    :param dict_p_conditional:
    :param count_info:
    :param solution:
    :return: dict containing log-likelihood based on probability mass, and log-likelihood based on probability density
    """
    df_eval = pd.read_csv(data_eval_filepath, sep='\s+', header=None)
    df_train = pd.read_csv(os.path.join(data_train_path, 'data.txt'), sep='\s+', header=None)

    # determine data min/max for edge bin volume estimation (needed later)
    data_train_min = df_train.min()
    data_train_max = df_train.max()

    log_lik_mass = 0  # initialize log-likelihood (probability mass-based)
    log_lik_density = 0  # initialize log-likelihood normalized by density
    for index, row in df_eval.iterrows():
        for node_index, node_dict in dict_p_conditional.items():
            likelihood_found_for_node = False
            parent_nodes = count_info.parent_matrix[node_index]  # parent nodes of current node
            # find conditional probability matching the combination of current node value and parent node values
            for (parent_comb, node_dict_node_value), probability in node_dict.items():
                row_node_value = get_node_value(row, node_index, solution)
                if node_dict_node_value == row_node_value:
                    if parent_nodes:
                        row_parent_node_values = \
                            [get_node_value(row, parent_node_idx, solution) for parent_node_idx in parent_nodes]
                        if list(parent_comb) == row_parent_node_values:
                            bin_width = compute_bin_width(node_index, row_node_value, solution.boundaries,
                                                          data_train_min, data_train_max,
                                                          solution.number_of_instantiations)
                            log_lik_density = log_lik_density + math.log(probability / bin_width)
                            log_lik_mass = log_lik_mass + math.log(probability)
                            likelihood_found_for_node = True
                    else:  # node has no parent nodes, so no need to match the parent node values
                        bin_width = compute_bin_width(node_index, row_node_value, solution.boundaries,
                                                      data_train_min, data_train_max,
                                                      solution.number_of_instantiations)
                        log_lik_density = log_lik_density + math.log(probability / bin_width)
                        log_lik_mass = log_lik_mass + math.log(probability)
                        likelihood_found_for_node = True

            if likelihood_found_for_node is False:
                log_lik_mass = log_lik_mass - np.inf
                log_lik_density = log_lik_density - np.inf

    # normalize log-likelihood by number of observations
    log_lik_mass = log_lik_mass / len(df_eval)
    log_lik_density = log_lik_density / len(df_eval)
    return {"log_lik_mass": log_lik_mass, "log_lik_density": log_lik_density}


def compute_bin_width(node_index, node_value, boundaries, data_min, data_max, number_of_instantiations):
    """
    computes the width of the bin for a single node (not joint volume).
    normalized between [0, 1] based on extents of training data.
    """
    boundaries_of_node = boundaries[node_index]

    if boundaries_of_node:  # continuous node
        if node_value == 0:
            left = data_min[node_index]
            right = boundaries_of_node[0]
        elif node_value == len(boundaries_of_node):
            left = boundaries_of_node[-1]
            right = data_max[node_index]
        else:
            left = boundaries_of_node[node_value - 1]
            right = boundaries_of_node[node_value]

        data_range = data_max[node_index] - data_min[node_index]
        if data_range == 0:
            return 1.0
        else:
            return (right - left) / data_range
    else:  # discrete node
        number_of_bins = number_of_instantiations[node_index]
        return 1.0 / number_of_bins


def compute_concordance_index(eval_set_filepath: str,
                              training_set_filepath: str,
                              dict_p_conditional: dict,
                              count_info: MSS_Solution_counts,
                              node_info,
                              solution: MO_Solution,
                              surv_time_col_name: str,
                              surv_event_col_name: str
                              ):
    """
    compute C-index according to algorithm described in https://arxiv.org/pdf/0811.1645
    (2008 paper on random survival forests)
    :param eval_set_filepath: filepath of evaluation set to calculate C-index for
    :param training_set_filepath: filepath of training set (only used to find the min and max survival times)
    :param dict_p_conditional:
    :param count_info:
    :param node_info:
    :param solution:
    :param surv_time_col_name: column name of the survival time column
    :param surv_event_col_name: column name of the survival event column
    :return: float containing the concordance index
    """
    df_eval = pd.read_csv(eval_set_filepath, sep='\s+', header=None)
    df_train = pd.read_csv(os.path.join(training_set_filepath, 'data.txt'), sep='\s+', header=None)
    count_tot = 0
    count_concordant = 0
    for index_i, row_i in df_eval.iterrows():
        for index_j, row_j in df_eval.iterrows():
            if index_j > index_i:
                surv_time_col_idx = [i for i, col_name in enumerate(node_info[0]) if col_name == surv_time_col_name][0]
                surv_event_col_idx = [i for i, col_name in enumerate(node_info[0]) if col_name == surv_event_col_name][
                    0]
                Ti = {"surv_time": row_i[surv_time_col_idx], "event": row_i[surv_event_col_idx]}
                Tj = {"surv_time": row_j[surv_time_col_idx], "event": row_j[surv_event_col_idx]}

                # omit pair if shorter time is censored
                shorter_time = min(Ti["surv_time"], Tj["surv_time"])
                if (Ti["surv_time"] == shorter_time and Ti["event"] == 0) or \
                        (Tj["surv_time"] == shorter_time and Tj["event"] == 0):
                    continue

                # omit pair if Ti = Tj and both are censored
                if Ti["surv_time"] == Tj["surv_time"]:
                    if Ti["event"] == 0 and Tj["event"] == 0:
                        continue

                # count pair i,j as permissible
                count_tot += 1

                # determine central time values of each survival time bin
                surv_time_bin_centers = {}  # central time value for each survival time bin
                surv_time_bin_centers[0] = (min(df_train[surv_time_col_idx]) + solution.boundaries[surv_time_col_idx][
                    0]) / 2
                for i in range(len(solution.boundaries[surv_time_col_idx]) - 1):
                    surv_time_bin_centers[i + 1] = (solution.boundaries[surv_time_col_idx][i] +
                                                    solution.boundaries[surv_time_col_idx][i + 1]) / 2
                surv_time_bin_centers[len(solution.boundaries[surv_time_col_idx])] = \
                    (solution.boundaries[surv_time_col_idx][-1] + max(df_train[surv_time_col_idx])) / 2

                # predict mean T
                t_mean_predict_i = get_mean_t(dict_p_conditional, row_i, surv_time_col_idx, surv_event_col_idx,
                                              count_info, solution, surv_time_bin_centers)
                t_mean_predict_j = get_mean_t(dict_p_conditional, row_j, surv_time_col_idx, surv_event_col_idx,
                                              count_info, solution, surv_time_bin_centers)

                # check if predictions for i,j are concordant
                if Ti["surv_time"] < Tj["surv_time"]:
                    if t_mean_predict_i < t_mean_predict_j:
                        count_concordant += 1
                    elif t_mean_predict_i == t_mean_predict_j:
                        count_concordant += 0.5
                elif Ti["surv_time"] > Tj["surv_time"]:
                    if t_mean_predict_i > t_mean_predict_j:
                        count_concordant += 1
                    elif t_mean_predict_i == t_mean_predict_j:
                        count_concordant += 0.5
                elif Ti["surv_time"] == Tj["surv_time"]:  # matching survival times, both are deaths
                    if t_mean_predict_i == t_mean_predict_j:
                        count_concordant += 1
                    elif t_mean_predict_i != t_mean_predict_j:
                        count_concordant += 0.5

    return count_concordant / count_tot


def bootstrap_c_index(
        eval_set_filepath: str,
        training_set_filepath: str,
        dict_p_conditional: dict,
        count_info: MSS_Solution_counts,
        node_info,
        solution: MO_Solution,
        surv_time_col_name: str,
        surv_event_col_name: str,
        nr_bootstraps: int):
    """
    bootstrap the calculation of C-index on the evaluation set, using nr_bootstraps samples
    """
    # load eval set
    df = pd.read_csv(eval_set_filepath)

    random_num_gen = np.random.default_rng(seed=0)
    c_indices = []

    for i in range(nr_bootstraps):
        # resample with replacement
        bootstrap_df = df.sample(n=len(df), replace=True, random_state=random_num_gen.integers(1e9))

        # save temporarily for evaluation
        bootstrap_file = "bootstrap_sample.csv"
        bootstrap_df.to_csv(bootstrap_file, index=False)

        c_index = compute_concordance_index(
            bootstrap_file,
            training_set_filepath,
            dict_p_conditional,
            count_info,
            node_info,
            solution,
            surv_time_col_name,
            surv_event_col_name
        )
        print(f'bootstrap {i} c-index = {c_index}')
        c_indices.append(c_index)

    c_indices = np.array(c_indices)
    mean_c_index = c_indices.mean()
    std_error = c_indices.std(ddof=1)

    return mean_c_index, std_error, c_indices


def bootstrap_roc_auc(
        eval_set_filepath: str,
        dict_p_conditional: dict,
        count_info: MSS_Solution_counts,
        node_info,
        solution: MO_Solution,
        surv_status_col_name: str,
        nr_bootstraps: int):
    """
    bootstrap the calculation of ROC AUC on the evaluation set, using nr_bootstraps samples
    """
    # load eval set
    df = pd.read_csv(eval_set_filepath)

    random_num_gen = np.random.default_rng(seed=0)
    aucs = []

    for i in range(nr_bootstraps):
        # resample with replacement
        bootstrap_df = df.sample(n=len(df), replace=True, random_state=random_num_gen.integers(1e9))

        # save temporarily for evaluation
        bootstrap_file = "bootstrap_sample.csv"
        bootstrap_df.to_csv(bootstrap_file, index=False)

        auc = compute_roc_auc(
            bootstrap_file,
            dict_p_conditional,
            count_info,
            node_info,
            solution,
            surv_status_col_name
        )
        print(f'bootstrap {i} AUC = {auc}')
        aucs.append(auc)

    aucs = np.array(aucs)
    mean_auc = np.nanmean(aucs)  # nanmean skips np.nan values in aucs
    std_error = np.nanstd(aucs, ddof=1)

    return mean_auc, std_error, aucs


def get_mean_t(dict_p_conditional, row, surv_time_col_idx, surv_event_col_idx, count_info, solution,
               surv_time_bin_centers):
    """
    get mean predicted survival time for observation specified by row, for BN with survival time and survival event
nodes
    :return: predicted mean survival time as a float
    """
    p_cond = {}
    for time_bin_idx in range(len(surv_time_bin_centers)):
        p_cond[time_bin_idx] = 1  # initialize conditional probablity given the values
        for node_index in set(dict_p_conditional.keys()):
            probability_found_for_node = False
            parent_nodes = count_info.parent_matrix[node_index]  # parent nodes of current node
            for node_idx, node_dict in dict_p_conditional.items():
                if node_index == node_idx:  # index of current node
                    # find conditional probability matching the combination of node and parent node values of row
                    for (parent_comb, node_dict_node_value), probability in node_dict.items():
                        row_node_value = get_node_value_conditional_on_survival(row,
                                                                                node_idx,
                                                                                solution,
                                                                                surv_time_col_idx,
                                                                                surv_event_col_idx,
                                                                                time_bin_idx,
                                                                                surv_event=1)
                        if node_dict_node_value == row_node_value:
                            if parent_nodes:
                                node_dict_parent_node_values = [
                                    get_node_value_conditional_on_survival(
                                        row,
                                        parent_node_idx,
                                        solution,
                                        surv_time_col_idx,
                                        surv_event_col_idx,
                                        time_bin_idx,
                                        surv_event=1) for parent_node_idx in parent_nodes]
                                if list(parent_comb) == node_dict_parent_node_values:
                                    p_cond[time_bin_idx] = p_cond[time_bin_idx] * probability
                                    probability_found_for_node = True
```

```python
                            else:  # node has no parent nodes, so no need to match the parent node values
                                p_cond[time_bin_idx] = p_cond[time_bin_idx] * probability
                                probability_found_for_node = True
            if probability_found_for_node is False:
                p_cond[time_bin_idx] = p_cond[time_bin_idx] * 0.0
    p_cond = {k: v / sum(p_cond.values()) if sum(p_cond.values()) > 0 else v for k, v in p_cond.items()}
    t_mean = sum([prob * surv_time_bin_centers[index] for index, prob in p_cond.items()])
    return t_mean


def compute_roc_auc(eval_set_filepath: str,
                    dict_p_conditional: dict,
                    count_info: MSS_Solution_counts,
                    node_info,
                    solution: MO_Solution,
                    surv_status_col_name: str
                    ):
    """
    for a fixed-timepoint survival prediction, compute ROC AUC
    :return: float containing the concordance index
    """
    surv_statuses_true = []
    surv_probabilities_pred = []
    df_eval = pd.read_csv(eval_set_filepath, sep='\s+', header=None)
    for index, row in df_eval.iterrows():
        surv_status_col_idx = [i for i, col_name in enumerate(node_info[0]) if col_name == surv_status_col_name][0]
        surv_statuses_true.append(row[surv_status_col_idx])
        surv_probabilities_pred.append(
            get_surv_status_probability(dict_p_conditional, row, surv_status_col_idx, count_info, solution))

    try:
        roc_auc = roc_auc_score(surv_statuses_true, surv_probabilities_pred)
    except ValueError:  # roc cannot be calculated if only one class exists in y_true
        roc_auc = np.nan
    print(f'surv_statuses_true = {surv_statuses_true}\n'
          f'surv_probabilities_pred = {surv_probabilities_pred}\n'
          f'roc_auc = {roc_auc}')
    return roc_auc


def get_surv_status_probability(dict_p_conditional, row, surv_status_col_idx, count_info, solution):
    """
    predict the probability of fixed-time survival status for a classifier BN for observation 'row'
    :return: probability as float
    """
    prob = 1  # initialize conditional probablity given the values
    parent_nodes = count_info.parent_matrix[surv_status_col_idx]  # parent nodes survival status node
    for (parent_comb, node_dict_node_value), probability in dict_p_conditional[surv_status_col_idx].items():
        if node_dict_node_value == 1:  # survival status = 1
            if parent_nodes:
                node_dict_parent_node_values = [
                    get_node_value_conditional_on_survival(
                        row,
                        parent_node_idx,
                        solution,
                        surv_time_col_idx=None,
                        surv_event_col_idx=surv_status_col_idx,
                        time_bin_idx=None,
                        surv_event=1) for parent_node_idx in parent_nodes]
                if list(parent_comb) == node_dict_parent_node_values:
                    prob = prob * probability
            else:  # node has no parent nodes, so no need to match the parent node values
                prob = prob * probability
    print(f'probability of survival status 1 for row {list(row)} is: {prob}')
    return prob


def get_node_value(
        row,
        node_idx,
        solution):
    """
    get node value of node specified by node_idx in observation 'row', given a specified survival time and status
    if this is a continuous node, use solution.boundaries to find the correct bin index
    :param row: list containing node values
    :param node_idx: node index for which to retrieve the node value
    :param solution: solution containing discretization
    :return: node value
    """
    if solution.boundaries[node_idx]:  # node is continuous; get bin index
        return get_bin_index(row[node_idx], solution.boundaries[node_idx])
    else:  # node is discrete
        return row[node_idx]


def get_node_value_conditional_on_survival(
        row: list,
        node_idx,
        solution,
        surv_time_col_idx=None,
        surv_event_col_idx=None,
        time_bin_idx=None,
        surv_event=None
):
    """
    get node value of node specified by node_idx in observation 'row', given a specified survival time and status
    if this is a continuous node, use solution.boundaries to find the correct bin index
    if the node is a survival time or survival event indicator,
    :param row: list containing node values
    :param node_idx: node index for which to retrieve the node value
    :param solution: solution containing discretization
    :param surv_time_col_idx: column index of row representing survival time
    :param surv_event_col_idx: column index of row representing survival event
    :param time_bin_idx: survival time value (bin index)
    :param surv_event: indicator of survival event, 0 (censored) or 1
    :return: node value
    """
    if node_idx == surv_event_col_idx:
        return surv_event
    if node_idx == surv_time_col_idx:
        return time_bin_idx
    return get_node_value(row, node_idx, solution)


```

```python
import pandas as pd
import numpy as np
import datetime as dt
import os
import argparse
from sklearn.model_selection import KFold
import copy

"""
Function to convert csv-file of EXAMINE data into txt files for CBN-GOMEA (single objective)

Usage:
python generate_input_data.py --column_names <arg1> <arg2> <arg3> --data_path <path> --export_path <path>

optional arguments:
test_set_ratio (generate a test set)
num_folds (generate k-folds)
seed_train_test_split (specificy the seed for the train/test split)
"""


def df_to_str(df) -> str:
    df_string = df.to_string(header=False, index=False)
    return '\n'.join(' '.join(line.split()) for line in df_string.split('\n'))  # ensure single-spacing


class InputDataConverter:
    def __init__(self, file_path, column_names, pat_ids=None, test_set_ratio=None, pre_discretize_features=None,
                 seed_train_test_split=123):
        na_values = [-99, -98, -97, -96, "01-01-2996", "##USER_MISSING_96##", "#VALUE!"]
        self.df = pd.read_csv(file_path, sep=';', na_values=na_values, encoding='latin1')
        self.column_names = column_names
        self.pat_ids = pat_ids
        self.test_set_ratio = test_set_ratio
        self.pre_discretize_features = pre_discretize_features
        self.df_test = None
        self.seed_train_test_split = seed_train_test_split if seed_train_test_split else 123

        self.prepare_data(self.column_names)

    def prepare_data(self, column_names):
        # convert datatype for numeric columns
        for col in self.numeric_columns:
            if col in self.df:
                self.df[col] = pd.to_numeric(self.df[col], errors='coerce')

        def convert_to_int(timestamp):
            if pd.isna(timestamp):
                return np.nan
            else:
                return timestamp.value

        # convert datatype for date columns
        for col in self.date_columns:
            if col in self.df:
                # try:
                self.df[col] = pd.to_datetime(self.df[col], dayfirst=True)
                self.df[col] = self.df[col].apply(convert_to_int) / 2.62974383e15  # in months since 1970  #8.64e13(days
)

        # add survival features
        self.df['survival_time'] = self.df['overl_date'] - self.df['PA_datum']
        self.df['followup_time'] = self.df['last_followup_date'] - self.df['PA_datum']
        self.df['treatment_time'] = self.df['stop_brachy'] - self.df['start_EBRT']
        self.df['meta_free_time'] = self.df['meta_date'] - self.df['PA_datum']
        self.df['pelvic_fail_free_time'] = self.df['pelvic_failure_date'] - self.df['PA_datum']
        self.df['PALN_fail_free_time'] = self.df['PAO_date'] - self.df['PA_datum']

        # replace negative time differences with NaN
        self.df['survival_time'] = self.df['survival_time'].apply(lambda x: np.nan if x < 0 else x)
        self.df['followup_time'] = self.df['followup_time'].apply(lambda x: np.nan if x < 0 else x)
        self.df['treatment_time'] = self.df['treatment_time'].apply(lambda x: np.nan if x < 0 else x)
        self.df['meta_free_time'] = self.df['meta_free_time'].apply(lambda x: np.nan if x < 0 else x)
        self.df['pelvic_fail_free_time'] = self.df['pelvic_fail_free_time'].apply(lambda x: np.nan if x < 0 else x)
        self.df['PALN_fail_free_time'] = self.df['PALN_fail_free_time'].apply(lambda x: np.nan if x < 0 else x)

        # combine event + censoring data into one time column and event indicator
        # overall survival
        self.df['overall_survival_time'] = self.df.apply(
            lambda row: row['followup_time'] if pd.isna(row['survival_time']) else row['survival_time'],
            axis=1
        )
        self.df['overall_survival_event'] = self.df['survival_time'].apply(lambda x: 1 if pd.notna(x) else 0)
        # fixed endpoint,overall survival at 6, 12, 18 etc months (1, 0, or None if censored)
        for months in [6, 12, 18, 24, 75]:
            self.df[f'overall_surv_{months}mo'] = self.df.apply(
                lambda row: (
                    1 if row['overall_survival_time'] >= months else
                    0 if row['overall_survival_time'] < months and row['overall_survival_event'] == 1 else
                    None
                ),
                axis=1
            )
        # metastasis-free survival
        self.df['meta_free_survival_time'] = self.df.apply(
            lambda row: row['followup_time'] if pd.isna(row['meta_free_time']) else row['meta_free_time'],
            axis=1
        )
        self.df['meta_free_survival_event'] = self.df['meta_free_time'].apply(lambda x: 1 if pd.notna(x) else 0)
        # fixed endpoint, meta-free survival at 6, 12, 18 etc months (1, 0, or None if censored)
        for months in [6, 12, 18, 24, 75]:
            self.df[f'meta_free_surv_{months}mo'] = self.df.apply(
                lambda row: (
                    1 if row['meta_free_survival_time'] >= months else
                    0 if row['meta_free_survival_time'] < months and row['meta_free_survival_event'] == 1 else
                    None
                ),
                axis=1
            )
        # pelvic failure-free survival
        self.df['pelvic_fail_survival_time'] = self.df.apply(
            lambda row: row['followup_time'] if pd.isna(row['pelvic_fail_free_time']) else row['pelvic_fail_free_time'],
            axis=1
        )
        self.df['pelvic_fail_survival_event'] = self.df['pelvic_fail_free_time'].apply(lambda x: 1 if pd.notna(x) else 0)
        # PALN failure-free survival
        self.df['PALN_survival_time'] = self.df.apply(
            lambda row: row['followup_time'] if pd.isna(row['PALN_fail_free_time']) else row['PALN_fail_free_time'],
            axis=1
        )
        self.df['PALN_survival_event'] = self.df['PALN_fail_free_time'].apply(lambda x: 1 if pd.notna(x) else 0)

        # combine factor levels
        figo_mapping = {
            'IB1': '<=2a', 'IB2': '<=2a', 'IB3': '<=2a',
            'II': '<=2a', 'IIA': '<=2a', 'IIA1': '<=2a', 'IIA2': '<=2a',
            'IIB': '2b-4', 'III': '2b-4', 'IIIA': '2b-4', 'IIIB': '2b-4',
            'IIIC1': '2b-4', 'IIIC2': '2b-4', 'IVA': '2b-4', 'IVB': '2b-4'
        }

        # apply the mapping to create a new column
        self.df['FIGO_binary'] = self.df['FIGO_final'].map(figo_mapping)

        # create new binary column for urinary toxicity, combining urinary urgency, frequency and incontinence
        self.df['urinary_tox_binary'] = (
            self.df[['tox_urinary_incontinence_date', 'tox_urinary_urgency_date', 'tox_urinary_frequency_date']]
                .notna()
                .any(axis=1)
                .astype(int)
        )

        # subset data
        if self.pat_ids:
            self.df = self.df[self.df['Participant Id'].isin(self.pat_ids)]
        self.df = self.df[column_names]

        # remove rows with missing values
        print(f'\nmissing values per column: \n{self.check_missing()}\n')
        rows_with_nas = sum([col['row_indices'] for col in self.check_missing()], [])
        self.df.drop(rows_with_nas, axis=0, inplace=True)
        print(f'dropped rows {rows_with_nas}\n')

        print('mapping of categorical variables:')
        # factorize non-numeric/non-date columns to 0, 1, 2 etc.
        for col in self.df.columns:
            if col not in self.numeric_columns and col not in self.date_columns and col not in self.survival_columns:
                codes, uniques = pd.factorize(self.df[col], sort=True)
                msg = f'{col} - #unique values: {len(uniques)} {list(uniques)} -> {[int(i) for i in sorted(pd.unique(codes))]}'
                self.df[col] = codes
                print(msg)

        # split test and training set
        if self.test_set_ratio:
            self.df_test = self.df.sample(frac=self.test_set_ratio, random_state=self.seed_train_test_split)
            self.df = self.df.drop(self.df_test.index)  # training set

    def discretize_continuous_data(self, col_name: str, breaks: list):
        """
        discretizes a continuous feature in the dataframe into bins (0, 1, 2, ...)
        according to the given breakpoints.
        """
        # adding a negative infinity as the lower bound and positive infinity as the upper bound
        # to handle edge cases for values below the first breakpoint and above the last one.
        self.df[col_name] = pd.cut(self.df[col_name], bins=[float('-inf')] + breaks + [float('inf')],
                                   labels=range(len(breaks) + 1), right=False)

    def get_input_data(self) -> str:
        df_string = self.df.to_string(header=False, index=False)
        return '\n'.join(' '.join(line.split()) for line in df_string.split('\n'))  # ensure the string is single-
spaced

    def get_test_data(self) -> str:
        return df_to_str(self.df_test)

    def get_expert_solution(self) -> str:
        num_variables = len(self.column_names)/2*(len(self.column_names)-1)
        text = num_variables * '0'
        return text

    def get_geninfo(self) -> str:
        return f'Run index: 0, \nSeed number: 0, \nNumber of samples: {len(self.df)}, \nNetwork index: 0, \nGeneration
method: Random,\nPython version: 3.12.3 | packaged by conda-forge | (main, {dt.datetime.now().strftime("%b %d %Y, %H:%M
:%S")}) [MSC v.1938 64 bit (AMD64)],'

    def get_info(self) -> str:
        text = ''
        max_col_length = max(len(col) for col in self.column_names)
        for col in self.column_names:
            dtype = self.get_datatype_from_column_name(col)
            num_bins = self.get_num_bins_from_column_name(col)
            spaces_needed = max_col_length - len(col)
            text += f'{col}{" " * spaces_needed}\t - {dtype} - {num_bins}\n'
        return text

    def get_optimal_solution(self) -> str:
        pass

    def get_datatype_from_column_name(self, column_name):
        if column_name in self.numeric_columns or column_name in self.date_columns:
            return 'Continuous'
        else:
            return 'Discrete  '

    def get_num_bins_from_column_name(self, column_name):
        if column_name in self.numeric_columns or column_name in self.date_columns:
            return 9  # number of bins in ground truth is not known so we use 9 for now
        else:
            return len(self.df[column_name][self.df[column_name] >= 0].unique())

    def check_missing(self):
        missing_data = {}

        for col in self.df.columns:
            missing_count = self.df[col].isna().sum()
            missing_rows = self.df[self.df[col].isna()].index.tolist()
            missing_data[col] = {'count': missing_count, 'row_indices': missing_rows}

        return pd.Series(missing_data)

    numeric_columns = [
        "leeftijd start EBRT", "lengte", "gewicht", "tumorgrootte_bv",
        "fracties_EBRT", "fractiedosis_EBRT", "totaaldosis_EBRT",
        "fractiesboost_level2_EBRT", "fractiedosisboost_level2_EBRT",
        "fractiesboost_level3_EBRT", "fractiedosisboost__level3_EBRT",
        "PAOfracties_EBRT", "PAOfractiedosis_EBRT", "PAOfracties_niveau2_EBRT",
        "PAOfractiedosis_niveau2_EBRT", "PAOdosis_EBRT", "chemo_dosis_plan",
        "chemo_aantal_cycli", "chemo_dosis_cyclus", "chemo_aantal_cycli_ontvangen",
        "chemo_dosis_totaal", "hyperthermie_cycli", "fracties_brachy",
        "fracties_brachy_uitgevoerd", "Aantal_brachyfracties_LUMC",
        "Aantal_brachyapplicaties_LUMC", "Aantal_brachyplannen_LUMC",
        "fractie_1_blaasvulling", "fractie_2_blaasvulling", "fractie_3_blaasvulling",
        "fractie_4_blaasvulling", "applicatie_1_OvoidRe",   "applicatie_1_OvoidLi", "applicatie_1_OvoidHoekRe",
        "applicatie_3_tandemlengte", "applicatie_3_tandemHoek", "EQD2_tot_GTV_res_D98", "EQD2_tot_HR_CTV_D90",
        "EQD2_tot_HR_CTV_D98", "EQD2_tot_HR_CTV_D50", "EQD2_tot_IR_CTV_D98",
        "EQD2_tot_point_A_R", "EQD2_tot_point_A_L", "EQD2_tot_GTV_N1",
        "EQD2_tot_GTV_N2", "EQD2_tot_GTV_N3", "EQD2_tot_GTV_N4",
        "EQD2_tot_GTV_N5", "EQD2_tot_bladder_D01cc", "EQD2_tot_bladder_D2cc",
        "EQD2_tot_bowel_D2cc", "EQD2_tot_rectum_D01cc", "EQD2_tot_rectum_D2cc",
        "EQD2_tot_sigmoid_D01cc", "EQD2_tot_sigmoid_D2cc",
        "EQD2_tot_ICRU_rectovagina", "EQD2_tot_ICRU_bladder",
        "EQD2_tot_vag_5mm_R", "EQD2_tot_vag_5mm_L", "EQD2_tot_PIBS_2cm_plus",
        "EQD2_tot_PIBS_2cm", "EQD2_tot_PIBS_2cm_min",
        "survival_time", "followup_time", "treatment_time", "overall_survival_time",
        "meta_free_time", "meta_free_survival_time",
        "PALN_fail_free_time", "PALN_survival_time",
        "pelvic_fail_free_time", "pelvic_fail_survival_time",
        "bowel_V15_abs_vol", "bowel_V30_abs_vol", "bowel_V40_abs_vol",
        "bladder_V30_rel_vol", "bladder_V40_rel_vol", "rectum_V30_rel_vol",
        "rectum_V40_rel_vol", "sigmoid_V30_rel_vol", "sigmoid_V40_rel_vol",
        "body_V10_abs_vol", "body_V43_abs_vol", "body_V36_abs_vol",
        "body_V50_abs_vol", "total_PB_vol", "total_PB_V10_vol",
        "total_PB_V20_vol", "total_PB_V40_vol", "kidney_Dmean",
        "EBRT_PIBS", "EBRT_PIBS_2cm"
    ]

    date_columns = [
            "datum_dataverzameling", "start_EBRT", "einde_EBRT", "start_brachy", "stop_brachy",
            "PA_datum", "overl_date", "last_diseasestatus_date", "last_followup_date", "lokaal_failure_date",
            "pelvic_failure_date",
            "locoreg_date", "vagina_date", "PAO_date", "meta_date", "tox_abdominal_pain_date",
            "tox_abdominal_infection_date", "tox_diarrhea_date", "tox_constipation_date", "tox_flatulence_date",
            "tox_fecal_urgency_date", "tox_Fecal_incontinence_date", "tox_proctitis_date", "tox_anal_hemorrhage_date",
            "tox_rectal_hemorrhage_date", "tox_colonic_hemorrhage_date", "tox_sigmoid_hemorrhage_date",
            "tox_small_bowel_hemorrhage_date", "tox_anal_stenosis_date", "tox_rectal_stenosis_date",
            "tox_colonic_stenosis_date",
            "tox_sigmoid_stenosis_date", "tox_small_bowel_stenosis_date", "tox_graad_small_bowel_fistula",
            "tox_small_bowel_fistula_date", "tox_colonic_fistula_date", "tox_rectal_sigmoid_date",
            "tox_rectal_fistula_date",
            "tox_anal_fistula_date", "tox_urinary_fistula_date", "tox_vaginal_fistula_date",
            "tox_gastrointestinal_disorders_other_date", "tox_urinary_urgency_date", "tox_urinary_frequency_date",
            "tox_urinary_incontinence_date", "tox_urinary_cystitis_noninfective_date", "tox_hematuria_date",
            "tox_urinary_tract_obstruction_date", "tox_acute_kidney_injury_date", "tox_chronic_kidney_disease_date",
            "tox_other_renal_and_urinary_disorders_date", "tox_vaginal_dryness_date", "tox_vaginal_discharge_date",
            "tox_vaginal_stenosis_or_stricture_date", "tox_vaginal_hemorrhage_date", "tox_vaginal_other_date",
            "tox_hip_fracture_date", "tox_pelvic_fracture_date", "tox_spinal_fracture_date"
        ]

    survival_columns = [
        "overall_survival_event",
        "meta_free_survival_event",
        "PALN_free_survival_event",
        "pelvic_fail_survival_event"
    ]


def parse_arguments():
    """
    parse command line arguments and return them
    """
    parser = argparse.ArgumentParser(description="Generate input data")
    parser.add_argument('--column_names', nargs='+', required=True, help="List of column names")
    parser.add_argument('--data_path', required=True, help="Path to data file")
    parser.add_argument('--export_path', required=True, help="Path to export data")
    parser.add_argument('--test_set_ratio', help="Optional, number between 0-1 to specify split for training and test.")
    parser.add_argument('--pre_discretize_features', help="Optional, apply discretization of select features specified in script")
    parser.add_argument('--num_folds', type=int, help="Number of folds for k-fold cross validation")
    parser.add_argument('--seed_train_test_split', type=int, help="seed for train/test set split")
    return parser.parse_args()


def main():
    args = parse_arguments()

    column_names = args.column_names
    data_path = args.data_path
    export_path = args.export_path
    test_set_ratio = float(args.test_set_ratio)
    pre_discretize_features = args.pre_discretize_features
    seed = args.seed_train_test_split

    pat_ids = None
    idc = InputDataConverter(data_path, column_names, pat_ids, test_set_ratio, pre_discretize_features, seed)
    input_data = idc.get_input_data()
    geninfo = idc.get_geninfo()
    info = idc.get_info()

    # K-Fold Cross Validation
    if args.num_folds and args.num_folds > 1:
        kf = KFold(n_splits=args.num_folds, shuffle=True, random_state=42)
        for fold, (train_idx, val_idx) in enumerate(kf.split(idc.df)):
            # create fold directory if it doesn't exist
            export_path_fold = os.path.join(export_path, f'fold{fold}', 'data', 'cervix')
            os.makedirs(export_path_fold, exist_ok=True)

            idc_fold = copy.deepcopy(idc)
            idc_fold.df = idc.df.iloc[train_idx]
            idc_fold.df_test = None
            df_val = idc.df.iloc[val_idx]

            # write input files
            input_data_fold = idc_fold.get_input_data()
            geninfo_fold = idc_fold.get_geninfo()
            info_fold = idc_fold.get_info()
            with open(os.path.join(export_path_fold, 'cervix_run0.txt'), 'w') as f:
                f.write(input_data_fold)
            with open(os.path.join(export_path_fold, 'cervix_run0_geninfo.txt'), 'w') as f:
                f.write(geninfo_fold)
            with open(os.path.join(export_path_fold, 'cervix_run0_info.txt'), 'w') as f:
                f.write(info_fold)

            # create validation set folder if it doesn't exist and write validation data to text file
            val_data_str = df_to_str(df_val)
            os.makedirs(os.path.join(export_path_fold, 'val_set'), exist_ok=True)
            with open(os.path.join(export_path_fold, 'val_set', 'cervix_run0_valset.txt'), 'w') as f:
                f.write(val_data_str)

    else:  # no k-fold cross validation
        # create cervix folder if it doesn't exist
        os.makedirs(os.path.join(export_path, 'cervix'), exist_ok=True)

        # write input files
        with open(os.path.join(export_path, 'cervix', 'cervix_run0.txt'), 'w') as f:
            f.write(input_data)
        with open(os.path.join(export_path, 'cervix', 'cervix_run0_geninfo.txt'), 'w') as f:
            f.write(geninfo)
        with open(os.path.join(export_path, 'cervix', 'cervix_run0_info.txt'), 'w') as f:
            f.write(info)

    if test_set_ratio:
        test_data = idc.get_test_data()
        # create test_set folder if it doesn't exist and write test data to text file
        os.makedirs(os.path.join(export_path, 'cervix', 'test_set'), exist_ok=True)
        with open(os.path.join(export_path, 'cervix', 'test_set', 'cervix_run0_testset.txt'), 'w') as f:
            f.write(test_data)


if __name__ == "__main__":
    main()

```

```python
import os
import argparse
import pandas as pd
import datetime

"""
script to collect seperate result.csv files into a single one
the final merged result.csv file is created into a new folder

result.csv files are created by analyze_bn_probabilities.py in seperate folders, for example when looping over
complexity penalty. When provided with a list of complexity penalty values, this script will construct the paths
for each individual result.csv file, and read them all into a single result.csv file.
"""


def parse_arguments():
    """
    parse command line arguments and return them
    """
    parser = argparse.ArgumentParser(description="Collect experiment results")
    parser.add_argument('--path_root', required=True, help="path to results.csv")
    parser.add_argument('--complexity_strs', nargs='+', required=True, help="strings representing complexity functions (0p0, 0p1, ...)")
    parser.add_argument('--discr_strs', nargs='+', required=True, help="strings to discretization types (ew, ef, bs)")
    parser.add_argument('--seeds', nargs='+', required=True, help="seed values for the algorithm")
    parser.add_argument('--num_folds', help="number of folds for cross validation")
    return parser.parse_args()


def main():
    args = parse_arguments()

    path_root = args.path_root
    complexity_strs = args.complexity_strs
    discr_strs = args.discr_strs
    seeds = args.seeds
    num_folds = int(args.num_folds) if args.num_folds else None

    if num_folds:  # kfold cross-validation
        filepaths = [os.path.join(path_root+f'{complexity_str}_{discr_str}_seed{seed}_fold{fold}/analysis_output/', 'result.csv') for
                     discr_str in discr_strs for complexity_str in complexity_strs for seed in seeds for fold in range(num_folds)]
    else:          # no cross validation
        filepaths = [os.path.join(path_root + f'{complexity_str}_{discr_str}_seed{seed}/analysis_output/', 'result.csv') for
                     discr_str in discr_strs for complexity_str in complexity_strs for seed in seeds]

    dataframes = []
    for filepath in filepaths:
        data = pd.read_csv(filepath, sep=';')#, header=None)
        dataframes.append(data)

    df_total = pd.concat(dataframes)
    print(df_total)

    # append analysis results to csv file (a new row for each solution)
    path_total = path_root+'all'+datetime.date.today().strftime("%Y%m%d")
    if not os.path.exists(path_total):
        os.makedirs(path_total)
    with open(os.path.join(path_total, 'result.csv'), 'w', newline='') as f:
        df_total = df_total.applymap(lambda x: str(x).replace('\n', '\\n') if isinstance(x, str) else x)
        f.write(df_total.to_csv(sep=';', index=False, header=True))


main()

```