Mathematical Biosciences and Engineering

http://www.aimspress.com/journal/mbe

MBE, 20(1): 624–655.
DOI: 10.3934/mbe.2023029
Received: 31 July 2022
Revised: 29 September 2022
Accepted: 08 October 2022
Published: 13 October 2022

Research article

Probabilistic machine learning for breast cancer classification

Anastasia-Maria Leventi-Peetz[1]*and **Kai Weber**[2]

[1] Federal Office for Information Security, Postfach 200363, 53133 Bonn, Germany
[2] inducto Daten- und Informationssysteme GmbH, 84405 Dorfen, Germany

* **Correspondence:** Email: leventi@bsi.bund.de; Tel: +4922895825877.

Abstract: A probabilistic neural network has been implemented to predict the malignancy of breast cancer cells, based on a data set, the features of which are used for the formulation and training of a model for a binary classification problem. The focus is placed on considerations when building the model, in order to achieve not only accuracy but also a safe quantification of the expected uncertainty of the calculated network parameters and the medical prognosis. The source code is included to make the results reproducible, also in accordance with the latest trending in machine learning research, named *Papers with Code*. The various steps taken for the code development are introduced in detail but also the results are visually displayed and critically analyzed also in the sense of explainable artificial intelligence. In statistical-classification problems, the decision boundary is the region of the problem space in which the classification label of the classifier is ambiguous. Problem aspects and model parameters which influence the decision boundary are a special aspect of practical investigation considered in this work. Classification results issued by technically transparent machine learning software can inspire more confidence, as regards their trustworthiness which is very important, especially in the case of medical prognosis. Furthermore, transparency allows the user to adapt models and learning processes to the specific needs of a problem and has a boosting influence on the development of new methods in relevant machine learning fields (transfer learning).

Keywords: Bayesian neural networks; decision boundary; explainable artificial intelligence; machine learning; model uncertainty; posterior predictive check; prior probability; probabilistic programming; scatter plot; variational posterior distribution

1. Introduction

Artificial Intelligence has been revolutionized by deep learning algorithms and their outstanding performance in handling difficult tasks such as image processing, object detection, speech recognition, medical image analysis, computer aided diagnosis, tissue recognition and classification

and other significant examples in the area of medical and healthcare problems [1–3]. Despite their success, classical deep learning algorithms have also limitations: models are prone to overfitting which adversely effects their generalization capabilities and they tend to be overconfident about their predictions which can lead to dramatic consequences in case of silent failures in sensitive applications like medical diagnosis, autonomous driving, finance and so on. Bayesian neural networks (BNN) is a paradigm developed to mitigate this risk. Comparisons between classical statistical models and artificial neural networks (ANN) have been already performed in earlier medical data studies and confirmed advantages of Bayes networks as to other ANN, like better insight into the structure of the prediction problem, efficient prediction by parsimonious modeling, avoidance of data overfitting and a clear perception of the relationships involved [4, 5]. The parameters of BNN are assigned probability distributions instead of single values or point estimates and they offer a better quantification of the uncertainty of their predictions [6]. Furthermore, they allow the distinction between epistemic and aleatoric uncertainty, that is, between the uncertainty due to the lack of data, or lack of knowledge and the uncertainty due to inherently random effects, like the stochastic dependency between instances and outcomes [7, 8]. Aleatoric uncertainty refers to the irreducible part of uncertainty, whereas epistemic uncertainty can be reduced through additional information by the observation of more training data or by improving the model, for example, by extending the description of instances by additional features. Stochastic artificial neural networks incorporate stochastic components which are either stochastic activations or stochastic weights, like in the case of BNN. This allows them to simulate multiple possible models $\boldsymbol{\theta}$ with their associated probability distribution $p(\boldsymbol{\theta})$. Training of BNN can be seen as a special case of ensemble learning which qualitatively corresponds to the aggregate prediction of an ensemble of different neural networks, trained on the same dataset. BNN are uncertainty-aware by construction and their training via Bayesian inference leads to learning parameters of distributions instead of weights directly. Bayesian inference requires the specification of prior distributions which quantify the pre-data uncertainty about parameter values. Quite often there is available expert knowledge which can get integrated into the model in the form of the so-called informative prior distributions [9]. If prior knowledge is considered vague or is missing then noninformative prior distributions are employed, the default priors which for discrete parameters are normal distributions. To the advantages of BNN there belong the possibility of the user to discard highly uncertain predictions as well as the guarantee of getting a higher accuracy for the remaining predictions. Furthermore the user is offered the opportunity to identify unfamiliar patterns in the data and out-of-distribution input samples, that could be corrupted observations or data belonging to different domains.

Detailed steps will be described for the development of a probabilistic neural network in a reduced parameter space. This is achieved by applying a graph-based dimensionality reduction method to create an optimal low-dimensional embedding for the structured data that the model has to fit. The input data will be projected on a two-dimensional space using the *Uniform Manifold Approximation and Projection for Dimension Reduction* (UMAP) technique, so that the model building and training will be effectively performed in the embedded two-dimensional space [10, 11]. Solutions found in embedded spaces after applying UMAP are expected to be of the same quality as in original space because the projection preserves local data relationships and global data geometric structure. The use of UMAP is already established in life sciences research for the analysis of high dimensionality data [12]. It is known to offer faithful representation of the original data topology in low dimensions

and insightful visualizations. The motivation of this work is to paradigmatic reduce the complexity of a medical prediction problem by embedding it into a lower-dimensional space so as to simplify model development and training while enabling the visual examination and explanation of model properties and results. The trained model will be investigated especially to identify the classification decision boundary and the parameter regions corresponding to high and low prediction uncertainty, respectively. The visual representation highly assists the analysis. The results will be transformed back to the original high-dimensional data space and conclusions drawn. A short flow diagram is shown in Figure 1. To our knowledge this approach is novel.

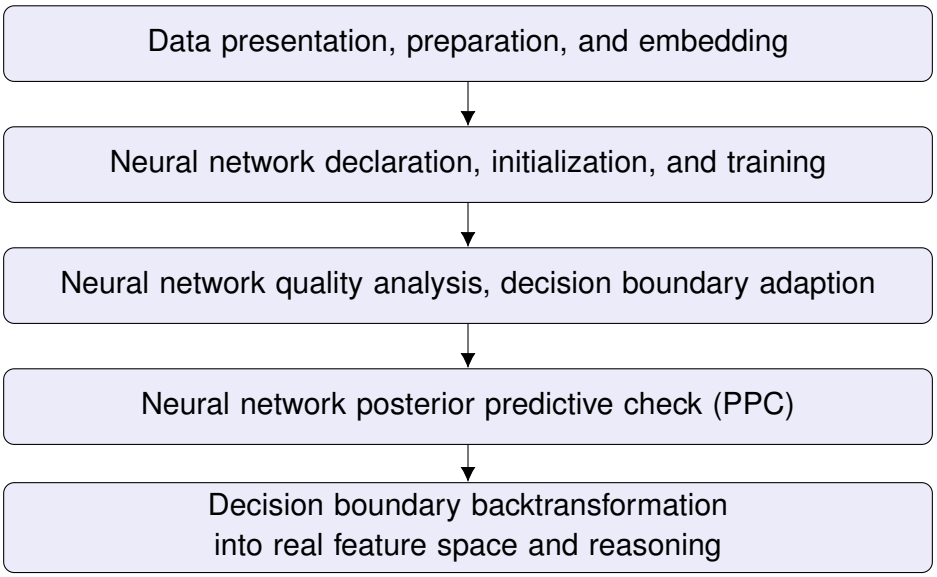

Figure 1. Flow chart of the probabilistic machine learning procedure.

In the the next section, first the data description, preparation as well as their visual presentation in the embedded space will be performed. The building and training of the model will follow with evaluation and interpretation of model interim and final parameters. Quality metrics of the neural network will be calculated and graphically depicted in dependence of the decision boundary, whose value will be optimized. In the third section, the uncertainty of the model prediction probability will be calculated for a space segment containing and surrounding the model input points. The model decision boundary will be transformed back to the original high-dimensional space and its trace through the original data scatter plots will be highlighted. Conclusions regarding the thus derived changes of the data class separation will be drawn. No similar approach was found in the literature to graphically spot regions in parameter space for which the model's predictions are loaded with uncertainty associated with their proximity to the model's decision boundary. Predictions for instances belonging to these regions would need to undergo additional and independent examination. Conclusions, limitations and ideas for further research will be discussed in the last section.

2. Building a Bayesian neural network using ADVI in PyMC3

For the implementation of the here described statistical model there will be used PyMC3, an open source probabilistic programming (PP) framework written in Python, that uses Theano to compute gradients [13]. PyMC3 allows model specification directly in Python code. This is an advantage because the lack of a domain specific language allows for great flexibility and direct interaction with the model. For the training of the model, a recently developed effective variational inference algorithm will be used, the automatic differentiation variational inference (ADVI), developed by Kucukelbir et al. [14] which is already included as a separate class in PyMC3 [13, 15]. The variational posterior distribution is assumed to be spherical Gaussian, whereby the means and standard deviations of the variational posterior are referred to as variational parameters. In this Bayesian network example, reasons will be discussed to underline that overall accuracy alone is not always the appropriate metric for the calibration of classification models. Considerations about the optimization of the model's decision boundary will be reviewed in detail. The visualizations support a better understanding of the model's development and functionality and indicate means and ways to interactively customize the procedure.

2.1. The data and their preparation

To accomplish the task of building and training the model, certain Python libraries have to be imported which must have been previously installed on the system. Most of the libraries are well known for developers of machine learning (ML) models, the UMAP library being an exception. UMAP can be used for visualization similarly to t-SNE but also for general non-linear dimension reduction [10, 11, 16]. UMAP is a robust tool for exploratory data analysis, a common use case when reducing to two dimensions for visualization purposes.

The data set for this example is taken from scikit-learn [17]. It is a copy out of the UC Irvine Machine Learning Repository, the Breast Cancer Wisconsin (Diagnostic) dataset. The data features are computed from digitized images of fine needle aspirate (FNA) of breast mass. They describe characteristics of the cell nuclei present in the image. The attributes of the data are identification number (ID) and diagnosis, M for malignant, B for benign. For this work, a Jupyter Notebook has been created which allows to run code in a Python interactive window of a web browser. The possibility to interactively modify the code, write comments, create graphics, document the results, and debug makes the usage of Jupyter Notebooks an advantageous and portable solution. The computational environment description and commands employed in the Jupyter Notebook are listed in the supplement. The data description output and the commands to retrieve it are given in the supplement in Listing 26 and 2. There follows the code showed in Listing 3 for storing the data into data frames. Only the first ten columns of the original data, i. e., only the captured mean-value features will be used and renamed for the model (see Listing 4). With the command: `data.info()` data information can be extracted (shown in Listing 27). The command: `data.head()` yields the first lines from the beginning of the data which appears as in Table 1.

In Table 2 there are listed the 10 real-valued features of the data, as they have been delivered by the command: `data.describe().T`. In the eleventh row, called "Target", the diagnosis is listed: 1 for *malign* and 0 for *benign*.

The creation of data scatter plots is a valuable diagnostic step to examine if and how the data points

Table 1. The first lines of the data in frames.

	Radius	Texture	Perimeter	Area	Smoothness	Compactness	Concavity	Concave Points	Symmetry	Fractal Dim	Target
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	0
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	0
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	0
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744	0
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883	0

Table 2. Features and Statistical data description.

	count	mean	std	min	25%	50%	75%	max
Radius	569	14.127	3.524	6.981	11.700	13.370	15.780	28.110
Texture	569	19.290	4.301	9.710	16.170	18.840	21.800	39.280
Perimeter	569	91.969	24.299	43.790	75.170	86.240	104.100	188.500
Area	569	654.889	351.914	143.500	420.300	551.100	782.700	2501.000
Smoothness	569	0.096	0.014	0.053	0.086	0.096	0.105	0.163
Compactness	569	0.104	0.053	0.019	0.065	0.093	0.130	0.345
Concavity	569	0.089	0.080	0.000	0.030	0.062	0.131	0.427
Concave Points	569	0.049	0.039	0.000	0.020	0.034	0.074	0.201
Symmetry	569	0.181	0.027	0.106	0.162	0.179	0.196	0.304
Fractal Dim	569	0.063	0.007	0.050	0.058	0.062	0.066	0.097
Target	569	0.627	0.484	0.000	0.000	1.000	1.000	1.000

cluster together, if there exist unexpected gaps in the data or if there are outlier points. Listing 5 shows the definition of a function to generate such a plot. The graphic in Figure 2 shows a matrix of plots indexed by the features of the data set. On the diagonal there are depicted histogram plots of the data distribution. The data for the two target values 0 and 1 (benign and malign) are shown in different colors (blue and red, respectively). The lower triangle of the matrix shows data-point scatter plots for pairs of features indexed by the respective row and column. The target values are again indicated by color. The upper triangle of the matrix shows the cross-correlation coefficient of Pearson, visualized with its value and a colored circular area. The values lie between 1 (for linear correlation) and -1 (linear anti-correlation) with colors ranging from reddish to bluish. The absolute correlation value determines the size of the circular area. The command: `data_overview_plot(data, 'Target')` produces the graphic seen in Figure 2.

It is obvious from Figure 2, that "Radius" and "Perimeter" are linearly correlated (100 %). Therefore, the feature "Perimeter" can be dropped. Similarly, "Radius" and "Area" are 99 % correlated. However, "Area" and "Radius" should be expected to be quadratically related. Both features will be kept. Also "Concave Points" and "Concavity" seem to be highly correlated (92 %). But both these variables will be kept because both are known to be practically significant. The two parameters are essential in the biology of tumor cells, so even if a statistical benefit can't be seen because 92 % of their variation is the same, the remaining 8 % could contain patterns able to boost the ML model's ability. Finally the model will be trained with 9 features. A further necessary step for the data preparation is the scaling of the data. Feature scaling is used to normalize the range of the values of data features and is necessary in order to avoid bias and improve the model's accuracy. In scaling it

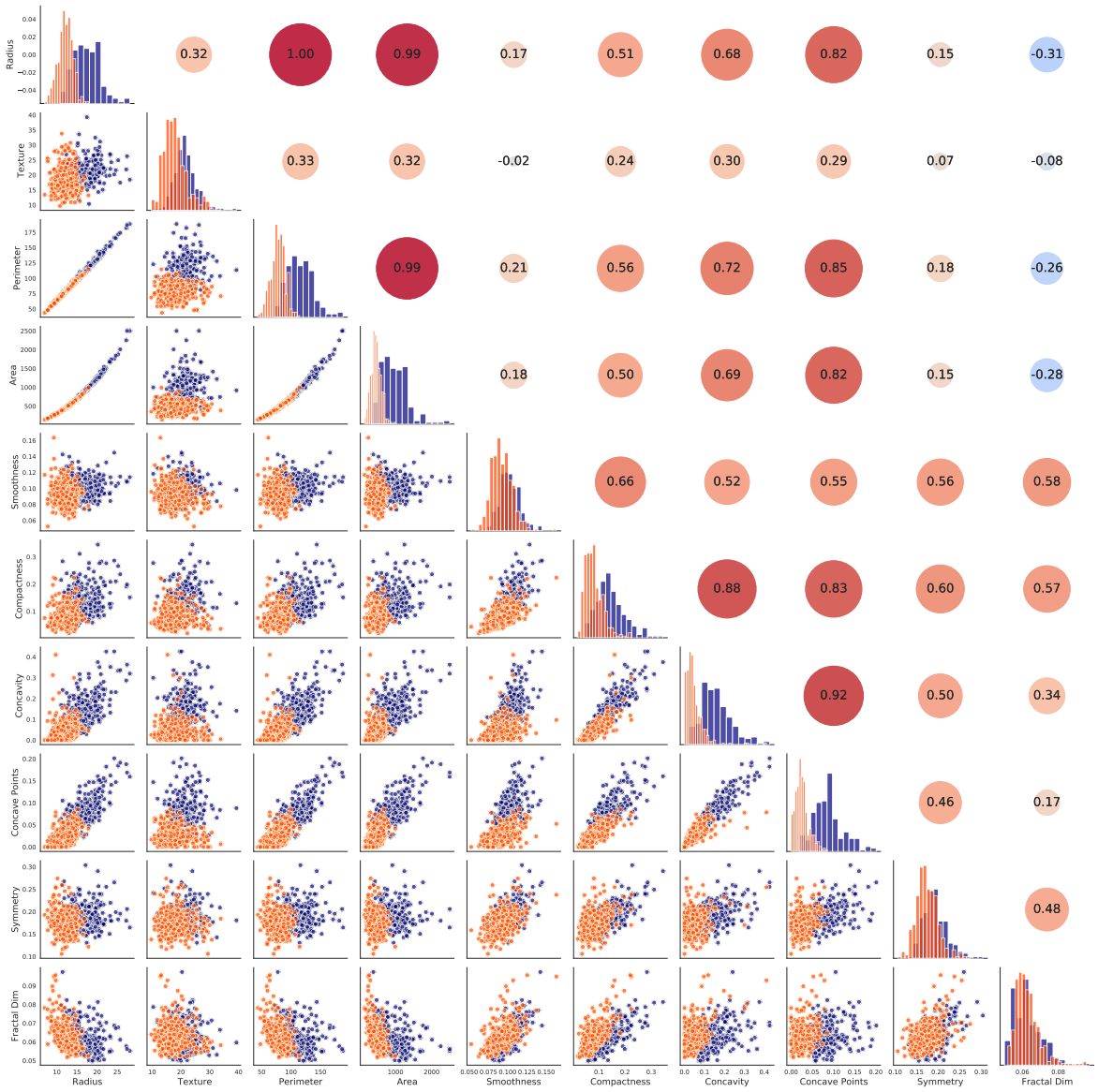

Figure 2. Matrix of scatter plots, cross-correlation indicators and histograms indexed by the features of the data set. The two classes are colored: red for *malign*, blue for *benign*.

is important to not leak information from the test data set into the trainings data set (See scikit-learn documentation about *Common pitfalls and recommended practices*). In the present example a standard scaler is used to rescale all features, resulting in a mean value 0 and standard deviation 1 for the values of each feature. To further proceed into data consolidation and understanding, UMAP will be applied to reduce the dimensionality of the dataset. Since UMAP by default reduces the feature dimension to two, visualization and interpretation of a fitted ML becomes more feasible. After reduction of the dimensions, a scaling step is again necessary. The scalers and the UMAP transformer will be combined in a scikit-learn pipeline, described in Listing 6.

After dropping the feature “Perimeter”, the data will be splitted into training and test data (see

Listing 7). There follows the fitting of the dimension reduction and scaling pipe with the training data, and the actual reduction and scaling (see Listing 8). The implementation of a plot function is helpful in order to visualize the data embedded into the two-dimensional reduced feature space (Listings 9 and 10).

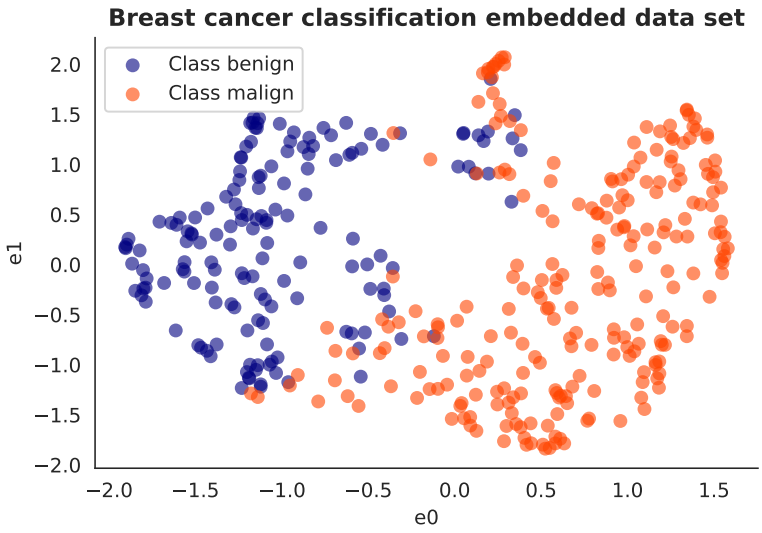

Figure 3. Scatter plot of training data instances in the two-dimensional embedded space after an UMAP data dimension reduction. The values of the two (new) embedded features lie on the embedded space axes: e0 and e1 respectively.

The graphic in Figure 3 shows a good separation of the target classes in the embedded space. This implies that the dimension reduction facilitates the model building. It has to be noted that no original feature can be assigned to any of the two embedded features of the reduced data instances, created by the UMAP projection. The two new features are synthesized out of the original ones, they implicitly integrate properties of all the real-valued features without exclusively corresponding to any of them. All data point instances will get mapped into the two-dimensional UMAP-space. Then the model building and analysis will be conducted in the reduced UMAP-space to make visualization and analysis easier and through a backtransformation of the results into the original feature representation space, some special conclusions will be finally drawn.

2.2. Defining, initializing and training the neural network

The neural network will be built with four layers. The number of the nodes in the first layer corresponds to the number of features of the input data. In this case we have two features in the embedded space after the UMAP projection and therefore two neurons in the first layer. There follow two hidden layers, the first with four neurons, the second with two. Finally the fourth layer is a single neuron perceptron to perform logistic regression, delivering a classification probability lying between 0 and 1, compare Figure 4. A network with one hidden layer is considered capable to approximate any function that contains a continuous mapping from one finite space to another. With two hidden layers, a network can represent an arbitrary decision boundary to arbitrary accuracy with rational

activation functions and can approximate any smooth mapping to any accuracy. Too few or too many

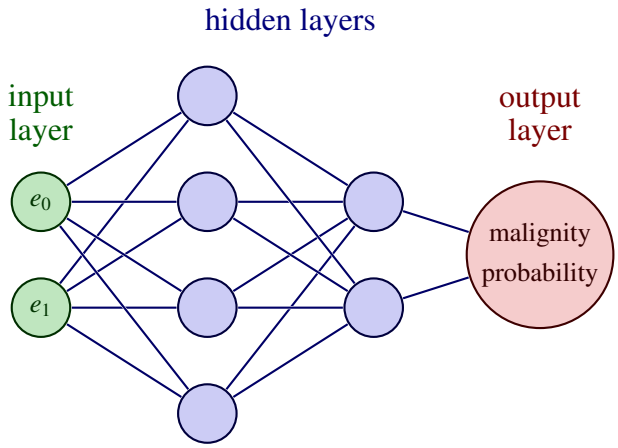

Figure 4. Architecture of the Bayesian neural network in embedded space.

neurons lead to underfitting and overfitting respectively. However, the selection of the network architecture is mostly performed empirically, decided by trial and error [18]. In Listing 11 a Python class for a Bayesian neural network will be defined together with help functions to support diagnostics of both the network, the training, and the results. The `tanh` activation function has been selected for the hidden layers. For each layer, prior distributions have to be defined for all network weights. Standard normal distributions, with mean 0 and standard deviation 1, have been applied. For the perceptron of the last layer, a sigmoid activation function has been used [19]. The final outputs are modeled using a Bernoulli Likelihood function so that results can also be values between 0 and 1.

The network can be declared and trained now with the code as displayed in Listing 12. In Figure 5 there is depicted the minimization process for the network fitting by plotting the negative evidence lower bound (ELBO) over the iteration number in order to assess the convergence [20].

It is obvious that there is noise in the convergence. The training dataset has about 400 data points only. Hence, the network finds it challenging to train itself and account for all the complexities while having only 400 data points as support. Thus, the course of the optimization of the ELBO couldn't run smoother.

2.3. Interpreting the weights of the model - TracePlot

A simple method to generate a visualization of the weight distributions is by building a trace plot of the model parameters (see Listing 13).

In Figure 6 the results are similar to Gaussian distributions which aligns well with the prior assumptions and initial distributions. Both output distributions `w_2_out` seem to have a statistically significant difference from 0. This means that all network components have transformations with statistically significant weights, and therefore, their effect is prominent on the final result.

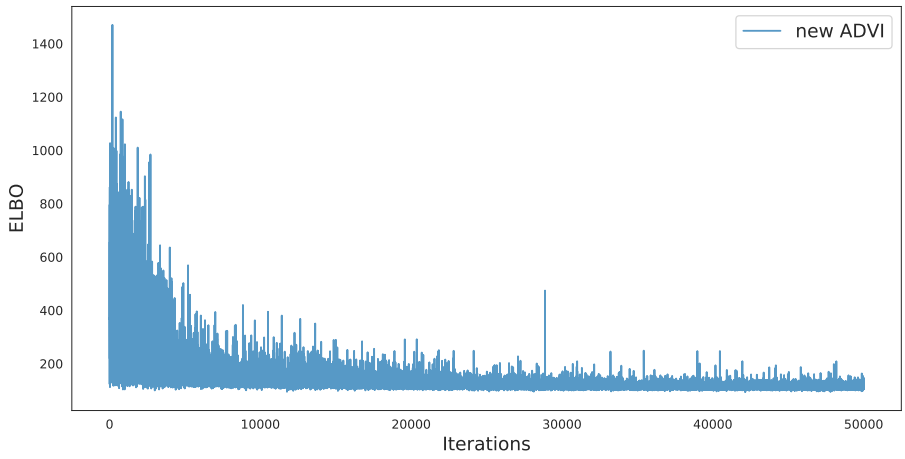

Figure 5. The ELBO optimization has taken about 10 seconds and finished with an Average Loss of 112.05.

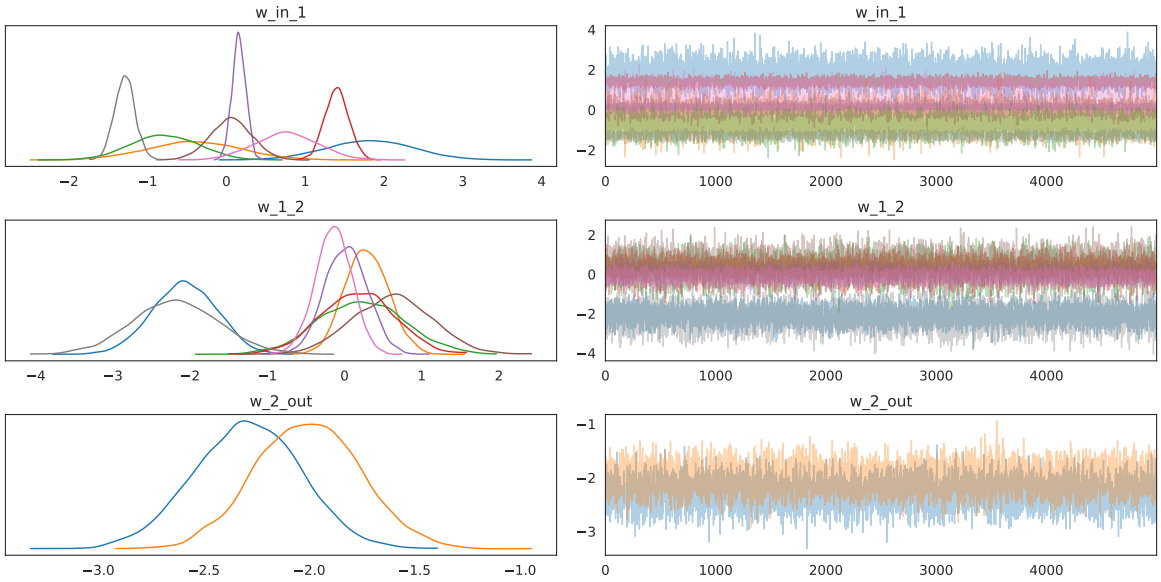

Figure 6. *Left panel*: Network weight distributions as calculated by the network: `w_1_2` are the distributions of the weights for the connections of the neurons between the first hidden and the second hidden layer. The total number of these weights is 4×2. The weights `w_in_1` represent the connections of the two neurons of the input layer to the neurons of the first hidden layer. Their number is also $2 \times 4 = 8$. The last set of weights: `w_2_out`, is a pair of distributions corresponding to the connections of the two neurons of the second hidden layer to the perceptron node. *Right panel*: Sampled weight values during the iterations.

2.4. *Optimizing the decision boundary*

First, samples will be created using the trained network to produce several predictions for each input data point. The posterior distributions of the network parameters are used for the evaluation of the prediction's distribution (see Listing 14). The mean of the distribution is taken as the classification result. Then, the percentage of the prediction error in dependence of the classification decision boundary will be plotted (see `plot_errors()` in Listing 11 and 15) as shown in Figure 7. The classification decision boundary for the model should be set to a value at which the false negative rate and the total error rate become both very small.

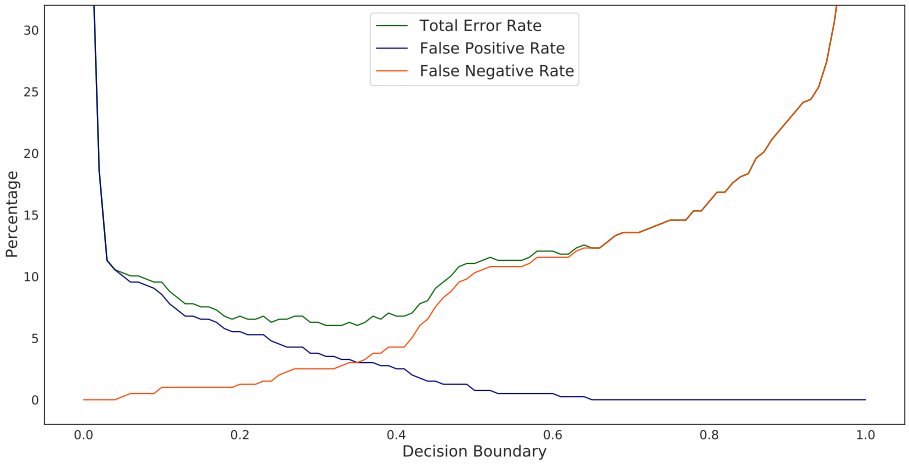

Figure 7. Total error rate, false positive rate, and false negative rate over the classification decision boundary for the training data.

The decision boundary is the threshold over which the result changes from a benign to a malign prediction. Logistic regression places the threshold at the value 0.5 which intuitively appears to be a plausible choice. At a very low decision boundary, the number of false positives is very high and every case would then be classified as malignant. As the decision boundary is raised, the first false negatives appear at a value close to 0.05. Further raising its value causes the false negative rate to slowly increase until the point corresponding to the minimum of the total error rate is reached. At this point the false negative rate is about 3 %. Beyond that point, the false negative rate rises dramatically. Focusing on as low as possible false negative rate as the highest requirement for the model, a good decision boundary lies at around 0.15. The Bayes ML model delivers also estimations of the accuracy of the error rates, see Listing 16 for the calculation of the deviation of the false negative rate, depicted in Figure 8.

The accuracy of the network for the training data and for a decision boundary set to 0.15 is 92.46 %. This accuracy is calculated as equal to 100 minus the percental total error. As is obvious from Figure 8, the standard deviation of the calculated false negative rate is also small at the selected decision boundary which is a desired result. If the number of false negatives would be the only criterion of importance, it would be easy to set the decision boundary equal to 0, but this wouldn't be useful anymore since the model would be telling every patient that they have malignant cancer, and the algorithm would be useless. Hence, the quality of the predictions depends on the cautious implementation of the chosen metric for the model.

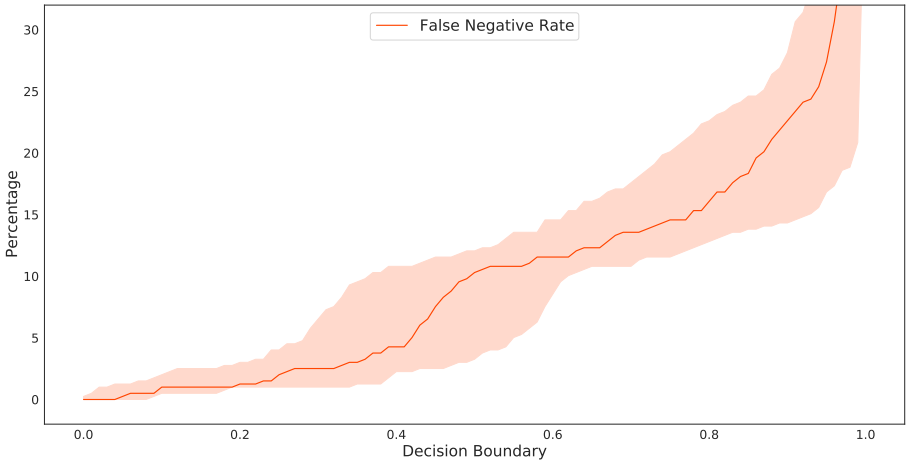

Figure 8. False negative rate in dependence of the decision boundary value and its standard deviation (reddish colored region) calculated with the training data.

The above procedures were repeated also for the test data: draw samples, display their error rates, and calculate the accuracy (code in Listing 17), see Figure 9. The values of the total error rate form a short minimum plateau which begins at 0.14 and ends at 0.18. Placing the decision boundary at 0.15 delivers a limit that combines a low total error rate with a low false negative rate. This confirms the chosen decision boundary.

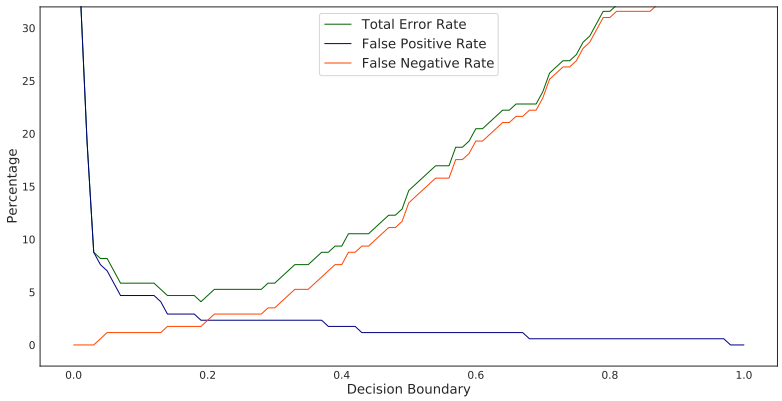

Figure 9. Total error rate, false positive rate, and false negative rate over the classification decision boundary, now for the test data.

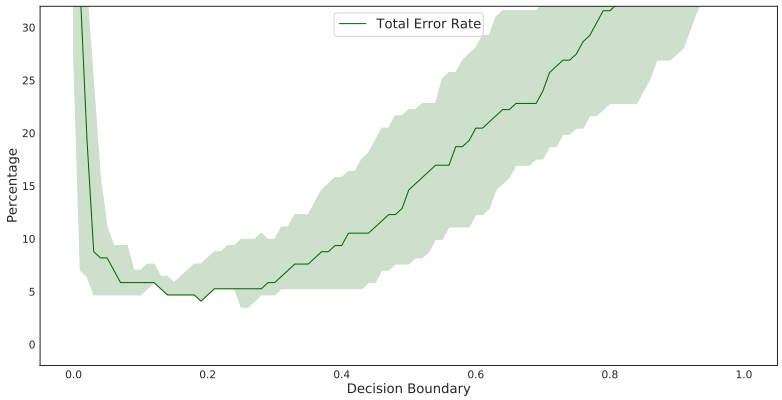

Figure 10. Total error rate in dependence of the decision boundary for the test data and its standard deviation (lightly colored region).

Again, the advantage of the Bayes ML model is that it delivers an estimation of the variability of the accuracy in dependence of the decision boundary, marked as the color shaded region around the values plot of the total and the false negative error rates respectively, see Figure 10 and Figure 11 (code in Listing 18).

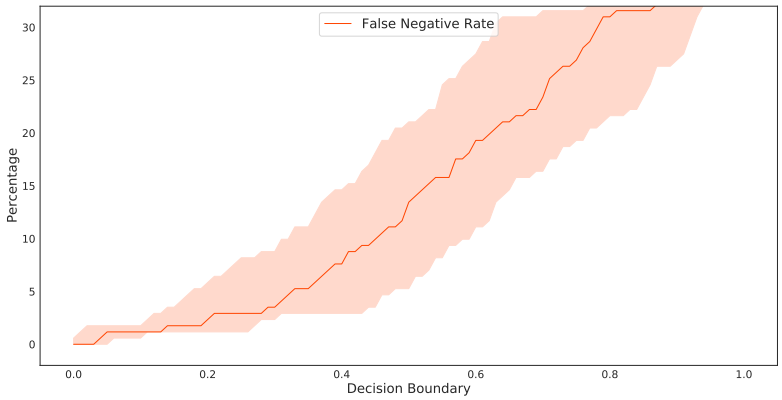

Figure 11. False negative rate in dependence of the decision boundary for the test data and its standard deviation (lightly colored region).

The accuracy of the network for the test data and for a decision boundary of 0.15 is 95.32 %. This accuracy is calculated as equal to 100 minus the percental total error.

The predictions of the embedded model with a decision boundary of 0.15 can also be plotted in a scatter plot, colored by the target value. The code in Listing 19 produces the results as depicted in Figure 12.

In Figure 13 the wrong model predictions are colored green (code in Listing 20).

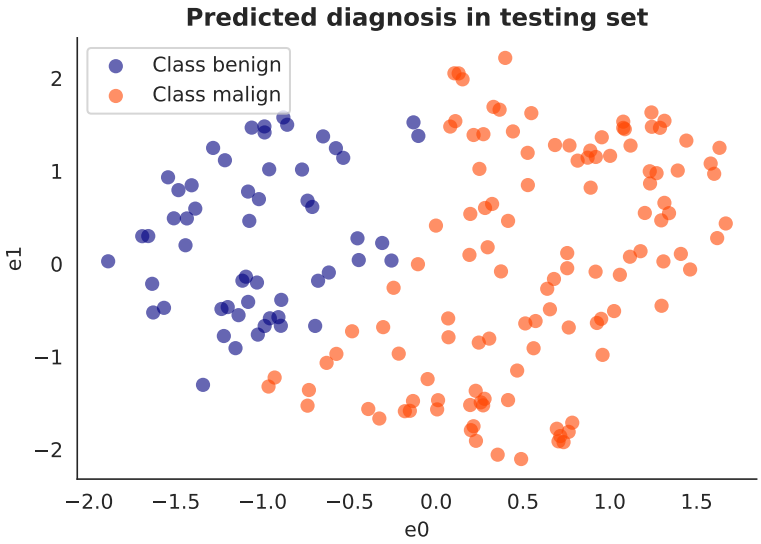

Figure 12. Test data scatter plot. e0 and e1 are the embedded space coordinate axes.

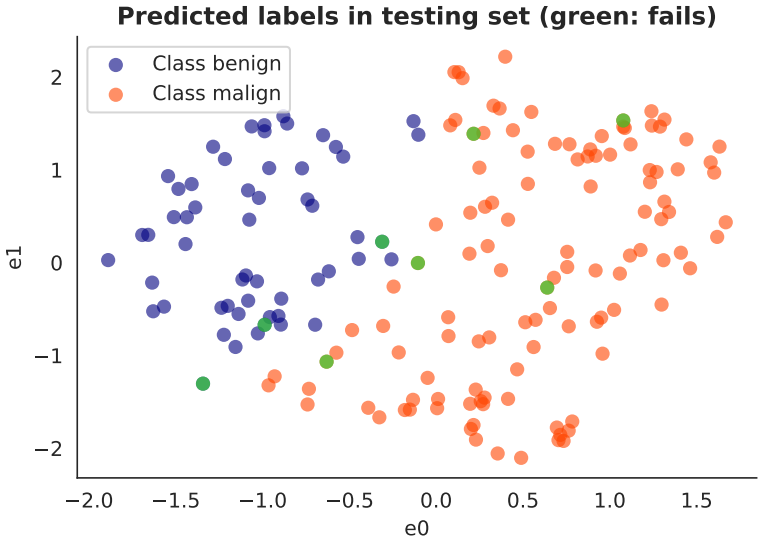

Figure 13. Test data scatter plot of the embedded model, e0 and e1 are the embedded space coordinate axes. Green points indicate wrong classifications.

3. What the classifier has learned

It is interesting to extend the examination of the trained network. To this purpose, the class probability predictions (posterior predictive check, PPC) on a grid over the embedding space around the data points will be evaluated. This code and a helper function, implemented to visualize the probability surface, are given in Listing 21. Using the helper function in the way described in

Listing 22, produces a visualization of the classification probability over an area around the data points in the reduced space, as shown in Figure 14.

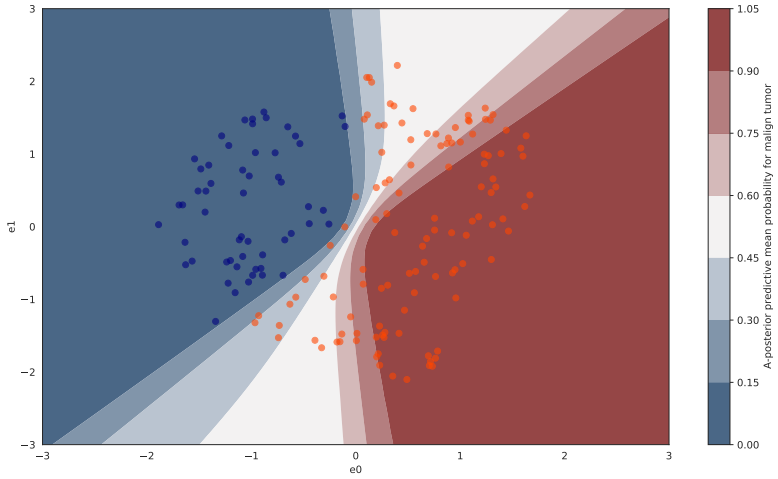

Figure 14. The model classification probability surface, overlaid with the scatter plot of the test data, colored according to the predicted target value. The dark blue (dark red) are regions of instance cases that are with high probability benign (malign), see also text. e0 and e1 are the coordinate axes of the embedded space.

One can also use the standard deviation of the posterior predictive probability, to visualize the uncertainty of the model predictions. Listing 23 contains the code for calling the visualization helper function for the creation of the according graphic of Figure 15.

The wrong predictions can also be visualized with a different color in the above plot, using the visualization helper function as in Listing 24. The resulting graphic is depicted in Figure 16.

In the contour plot of Figure 16, the colors indicate the value of the standard deviation of the model's PPC. The darker the surface background color, the higher the uncertainty of the prediction for points lying on it. However, some of the false predictions for the test data (4 of the green points) lie in regions of low uncertainty. But, 3 of these 4 points are very close to the decision boundary. This shows that one has to be careful also with confident model predictions, especially if the test points lie in the vicinity of the decision boundary. This also indicates that additional training data in this parameter region could be helpful for the model.

3.1. Backtransformation of the decision boundary isoline to the initial feature space

The decision boundary can get clearly identified as an isoline in the contour plot of the probability surface in Figure 14, along which the background color, representing the probability values, changes at the value of 0.15 from dark blue to a lighter blue. This is evident from the color bar on the right side of Figure 14. Geometrical points of the isoline can get extracted from the contour plot and projected back into the high-dimensional feature model space. In this way, properties of the embedded two-features model can be transformed to statements about properties of predictions in the original model space.

The code for the isoline extraction, its backtransformation into the initial high-dimensional feature

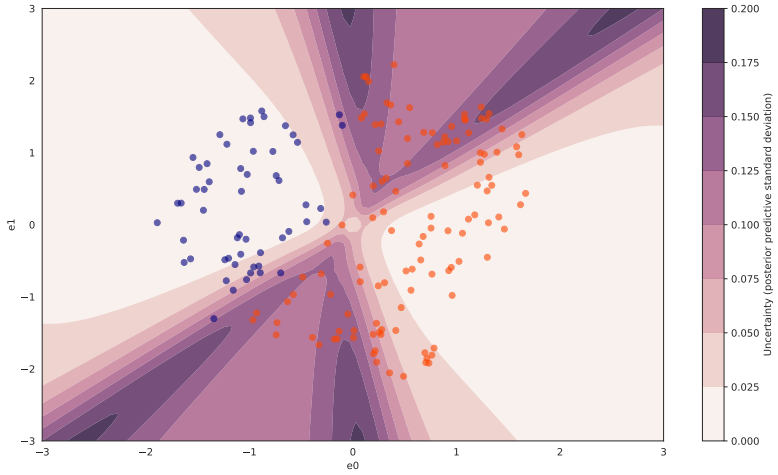

Figure 15. The standard deviation of the PPC of the model, overlaid with a scatter plot of the test data set, colored according to the predicted target value (blue for benign, red for malign). Points on light background are connected with low prediction uncertainty and vice versa. The colorbar on the right indicates the varying uncertainty using colors ranging from light beige to dark lilac. e0 and e1 are the embedded space coordinates to which the values of the two embedded features respectively belong.

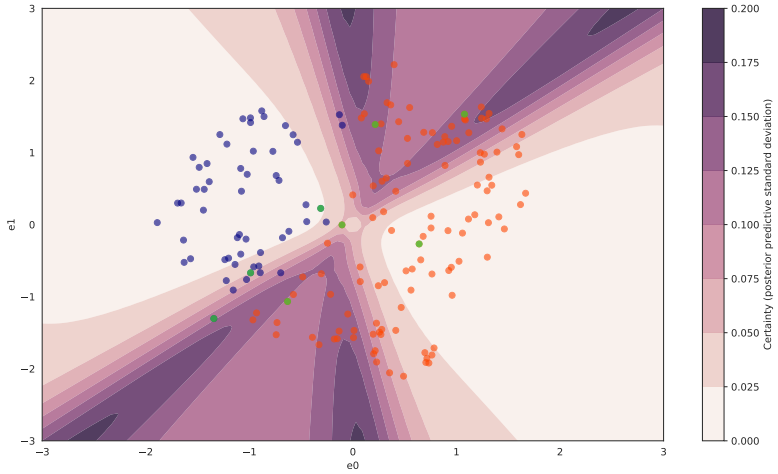

Figure 16. The standard deviation of the PPC of the model, overlaid with a scatter plot of the test data set, colored according to the predicted target value (compare Figure 15). False predictions are colored green (see also text). e0 and e1 are the coordinate axes of the embedded space which the values of the embedded features respectively lie on.

space, and the creation of a scatter-plot matrix, including points that correspond to the isoline of the model decision boundary can be found in Listing 25. The resulting matrix of scatter plots is shown in Figure 17 and the points originating from the decision boundary are depicted in green color. The

backtransformation of the decision boundary from the embedded into the initial high-dimensional feature space highlights the vicinity of real model inputs in which prediction failure awareness must be particularly high. This could serve as an additional assistance for the user because it has the benefit of visualization. Visualizations directly in the high-dimensional space are not possible. The target separation in the scatter plots becomes complicated when the influence of all features gets implicitly accounted for. In the vicinity of the model decision boundary (green points), additional case evidence and expert advice should be necessary to independently confirm the model's predictions.

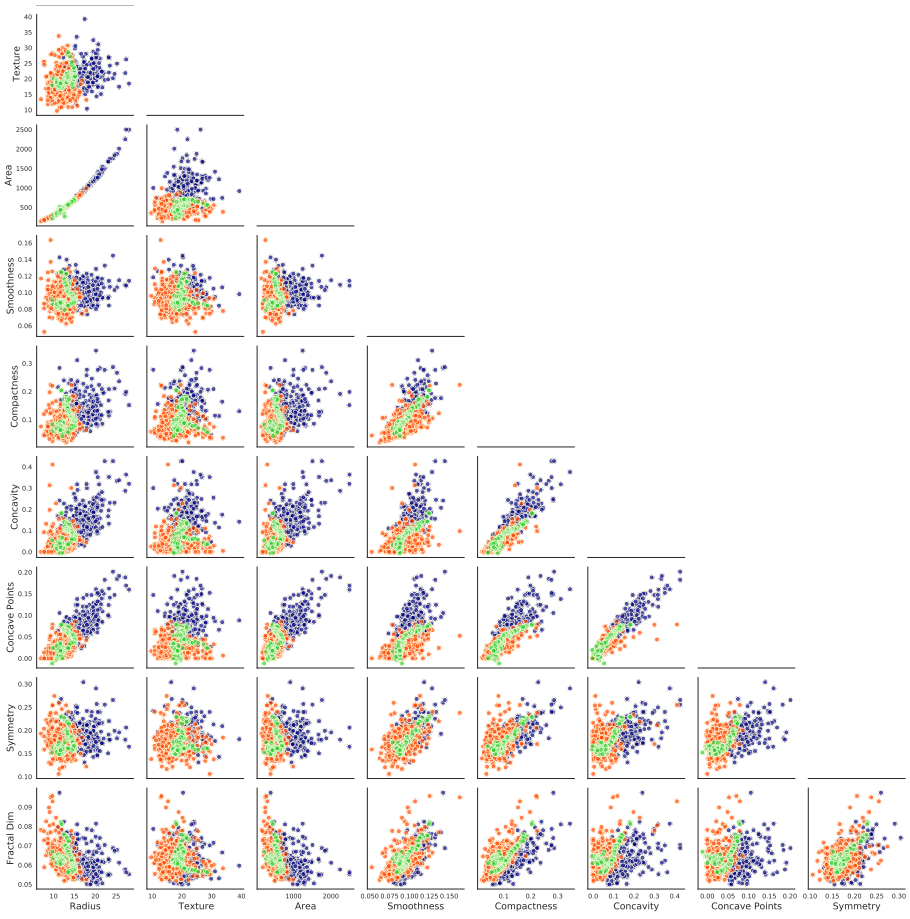

Figure 17. Matrix of scatter plots of the whole data set, indexed by the initial model features. The instances are colored according to the target value (compare Figure 2) . The points marked green represent points of the embedded model decision boundary after their backtransformation into the original feature space (see text).

4. Conclusions, limitations and future research

A Bayesian model has been created and trained in an embedded space, so that properties and objects of the model could be visually presented and analyzed at all development stages. A concise description of the computational aspects involved with source code listings have been included

(Appendix). For the designation of the model's decision boundary, which is the cutoff value for the logistic regression, the cost of false negative predictions has been given priority, considered to be much higher than the cost of false positives. A non-intuitive decision threshold for the classification was thus created, the plausibility of which could be justified by subsequent prognosis tests of the model. The focus of the discussion has been laid on the fact that the model calibration is not a purely technical exercise and solving this exercise does not necessarily always deliver intuitive results. Particular considerations about the exact purpose of the model are required in advance so that the main concerns of specialists, who would make use of the model, are met by design. The correct specification of prior distributions of parameters is often a challenging problem. A recent example is that of the high false positive rate of SARS-CoV-2 infections, delivered by a Bayesian analysis model which was due to a low base rate estimation (prevalence in the population) [21]. BNN priors are often determined to be mixed Gaussian densities with zero mean but different variances, where the setting of parameter priors is heuristic, and there are more than two hyperparameters that need to be tuned in this case. Priors should be chosen in a way that they reflect beliefs about the model parameters $\boldsymbol{\theta}$ before seeing the data. It is not trivial to map the subjective beliefs of the practitioner unambiguously onto tractable probability distributions [22, 23]. In Bayesian deep learning, it is therefore a common practice to choose a (seemingly) uninformative prior, such as a standard Gaussian. This trend is considered troubling by some experts, because choosing a bad prior can have detrimental consequences for the whole inference process [21, 24]. The prior can have a strong influence on the posterior, often forcing the probability mass onto arbitrary subspaces of the parameter space [24, 25]. This means that a standard Gaussian prior is not as uninformative as assumed but forces the posterior mass onto a thin subspace which in most cases does not reflect any useful prior knowledge and can severely bias the inference. The classification threshold set far away from the default value of 0.5 could imply that the data is imbalanced [26]. However, following the reasoning of minimization of the error rates of interest, one can define a decision threshold leading to a safe prognosis also in this case. Our future work will be dedicated to further developing the method of embedding and backtransformation using deeper Bayesian networks and train them with larger than in this example data sets and more data features. Also the adequate integration of informed priors based primarily on expert knowledge supplemented with very limited data, like often the case, will be a focus of our future investigations [27]. A rigorous technical development and analysis of a Bayes neural network, supported by open source code and by visualizations through dimensionality reduction is the main contribution of this work. The presented problem complexity reduction and backtransformation of model properties from the reduced to the original real feature space with the task to assess critical input data regions of a multi-parametric decision problem is a new approach in the ML context. The results suggest that a human-in-the-loop is indispensable for ML applications especially in the medical sector [28, 29]. Finally, one should not only improve the quality of model predictions but also advance the transparency of the multi-modal explainability of BNN [30]. Trustworthiness of neural networks achieved through transparency and explainability is the ultimate task of our work.

Conflict of interest

The authors declare there is no conflict of interest.

References

1. Q. Xia, Y. Cheng, J. Hu, J. Huang, Y. Yu, H. Xie, J. Wang, Differential diagnosis of breast cancer assisted by S-Detect artificial intelligence system, *Math. Biosci. Eng.*, **18** (2021), 3680–3689. https://doi.org/10.3934/mbe.2021184
2. X. Zhang, Z. Yu, Pathological analysis of hesperetin-derived small cell lung cancer by artificial intelligence technology under fiberoptic bronchoscopy, *Math. Biosci. Eng.*, **18** (2021), 8538–8558. https://doi.org/10.3934/mbe.2021423
3. S. Biswas, D. Sen, D. Bhatia, P. Phukan, M. Mukherjee, Chest X-Ray image and pathological data based artificial intelligence enabled dual diagnostic method for multi-stage classification of COVID-19 patients, *AIMS Biophys.*, **8** (2021), 346–371. https://doi.org/10.3934/biophy.2021028
4. T. Ayer, J. Chhatwal, O. Alagoz, C. E. Kahn, R. W. Woods, E. S. Burnside, Comparison of logistic regression and artificial neural network models in breast cancer risk estimation, *Radiographics*, **30** (2010), 13–21. https://doi.org/10.1148/rg.301095057
5. T. van der Ploeg, M. Smits, D. W. Dippel, M. Hunink, E. W. Steyerberg, Prediction of intracranial findings on CT-scans by alternative modelling techniques, *BMC Med. Res. Methodol.*, **11** (2011). https://doi.org/10.1186/1471-2288-11-143
6. L. V. Jospin, W. L. Buntine, F. Boussaïd, H. Laga, M. Bennamoun, Hands-On Bayesian neural networks – A tutorial for deep learning users, *IEEE Comput. Intell. Mag.*, **17** (2022), 29–48. https://doi.org/10.1109/MCI.2022.3155327
7. A. A. Abdullah, M. M. Hassan, Y. T. Mustafa, A review on Bayesian deep learning in healthcare: Applications and challenges, *IEEE Access*, **10** (2022), 36538–36562. https://doi.org/10.1109/ACCESS.2022.3163384
8. E. Hüllermeier, W. Waegeman, Aleatoric and epistemic uncertainty in machine learning: An introduction to concepts and methods, *Mach. Learn.*, **110** (2021), 457–506. https://doi.org/10.1007/s10994-021-05946-3
9. G. Deodato, Uncertainty modeling in deep learning, Variational inference for Bayesian neural networks, 2019. Available from: http://webthesis.biblio.polito.it/id/eprint/10920.
10. H. Huang, Y. Wang, C. Rudin, E. P. Browne, Towards a comprehensive evaluation of dimension reduction methods for transcriptomic data visualization, *Commun. Biol.*, **5** (2022). https://doi.org/10.1038/s42003-022-03628-x
11. L. McInnes, J. Healy, J. Melville, UMAP: Uniform manifold approximation and projection for dimension reduction, preprint, arXiv: 1802.03426. https://doi.org/10.48550/arXiv.1802.03426
12. A. Diaz-Papkovich, L. Anderson-Trocmé, S. Gravel, A review of UMAP in population genetics, *J. Hum. Genet.*, **66** (2021), 85–91. https://doi.org/10.1038/s10038-020-00851-4
13. J. Salvatier, T. V. Wiecki, C. Fonnesbeck, Probabilistic programming in Python using PyMC3, *PeerJ Comput. Sci.*, **2** (2016), e55. https://doi.org/10.7717/peerj-cs.55
14. A. Kucukelbir, D. Tran, R. Ranganath, A. Gelman, D. M. Blei, David M., Automatic differentiation variational inference, *J. Mach. Learn. Res.*, **18** (2017), 1–45. https://doi.org/10.48550/arXiv.1603.00788

15. M. Krasser, Variational inference in Bayesian neural networks. Available from: http://krasserm.github.io/2019/03/14/bayesian-neural-networks/.
16. L. van der Maaten, G. Hinton, Visualizing data using t-SNE, *J. Mach. Learn. Res.*, **9** (2008), 2579–2605.
17. F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, et al., Scikit-learn: Machine learning in Python, *J. Mach. Learn. Res.*, **12** (2011), 2825–2830.
18. Heaton Research, The number of hidden layers. Available from: https://www.heatonresearch.com/2017/06/01/hidden-layers.html.
19. T. Wood, What is the Sigmoid Function?, 2020. Available from: https://deepai.org/machine-learning-glossary-and-terms/sigmoid-function.
20. A. K. Dhaka, A. Catalina, M. R. Andersen, M. Magnusson, J. H. Huggins, A. Vehtari, Robust, accurate stochastic optimization for variational inference, in *Advances in Neural Information Processing Systems* (eds. H. Larochelle et al.), Curran Associates, Inc., **33** (2020), 10961–10973.
21. A. Gelman, B. Carpenter, Bayesian analysis of tests with unknown specificity and sensitivity, *J. R. Stat. Soc. Ser. C*, **69** (2020), 1269–1283. https://doi.org/10.1111/rssc.12435
22. D. Lunn, C. Jackson, N. Best, A. Thomas, D. Spiegelhalter, *The BUGS book: A practical introduction to Bayesian analysis*, Chapman and Hall/CRC, Boca Raton, USA, 2012.
23. D. B. Rizzo, M. R. Blackburn, Harnessing expert knowledge: Defining Bayesian network model priors from expert knowledge only–prior elicitation for the vibration qualification problem *IEEE Syst. J.*, **13** (2019), 1895–1905. https://doi.org/10.1109/JSYST.2019.2892942
24. A. M. Stefan, D. Katsimpokis, Q. F. Gronau, E. J. Wagenmakers, Expert agreement in prior elicitation and its effects on Bayesian inference, *Psychon. Bull. Rev.*, 2022. https://doi.org/10.3758/s13423-022-02074-4
25. V. Fortuin, Priors in Bayesian deep learning: A review, *Int. Stat. Rev.*, 2022. https://doi.org/10.1111/insr.12502
26. C. Esposito, G. A. Landrum, N. Schneider, N. Stiefl, S. Riniker, GHOST: Adjusting the decision threshold to handle imbalanced data in machine learning, *J. Chem. Inf. Model.*, **61** (2021), 2623–2640. https://doi.org/10.1021/acs.jcim.1c00160
27. Q. Kim, J.-H. Ko, S. Kim, N. Park, W. Jhe, Bayesian neural network with pretrained protein embedding enhances prediction accuracy of drug-protein interaction, *Bioinformatics*, **37** (2021), 3428–3435. https://doi.org/10.1093/bioinformatics/btab346
28. S. Budd, E. C. Robinson, B. Kainz, A Survey on active learning and human-in-the-Loop deep learning for medical image analysis, *Med. Image Anal.*, **71** (2021), 102062. https://doi.org/10.1016/j.media.2021.102062
29. V. S. Bisen, What is Human in the Loop Machine Learning: Why & How Used in AI?, 2020. Available from: https://medium.com/vsinghbisen/what-is-human-in-the-loop-machine-learning-why-how-used-in-ai-60c7b44eb2c0.
30. K. Bykov, M. M. C. Höhne, A. Creosteanu, K. R. Müller, F. Klauschen, S. Nakajima, et al., Explaining Bayesian neural networks, preprint, arXiv: 2108.10346. https://doi.org/10.48550/arXiv.2108.10346

Appendix

Computational environment

Hardware: desktop computer with eight core AMD Ryzen 7 5700G, 64 GB RAM, 1 TB SSD.
Operating Systems: GNU/Linux Debian bookworm and Ubuntu 20.04.5 LTS.
Versions of the used Python modules:

```
Python implementation: CPython
Python version:        3.9.12
IPython version:       7.31.1

numpy:      1.19.5
pandas:     1.1.5
matplotlib: 3.3.4
seaborn:    0.11.2
sklearn:    0.23.2
pymc3:      3.11.5
scipy:      1.7.3
theano:     1.1.2
umap:       0.5.3
```

The fitting of the Bayesian neural network takes around 10s after the theano modules have been pre-compiled. To run the complete Python notebook takes around 110 seconds.

Code listings

Listing 1. Importing the necessary libraries.

```
import random
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import pandas as pd
import pymc3 as pm
import theano
import theano.tensor as T
import sklearn as skl
from sklearn import (datasets, preprocessing, pipeline, model_selection)
import umap
```

Listing 2. Printing the data description with scikit-learn.

```
data = skl.datasets.load_breast_cancer()
print(data.DESCR)
```

Listing 3. Load data into data frames.

```
X, y = skl.datasets.load_breast_cancer(return_X_y=True, as_frame=True)
```

Listing 4. Extraction of the first ten columns.

```
orig_columns = X.columns[:10].to_list()
features = ['Radius', 'Texture', 'Perimeter', 'Area',
            'Smoothness', 'Compactness', 'Concavity',
            'Concave␣Points', 'Symmetry', 'Fractal␣Dim']
data = X[orig_columns].rename(columns=dict(zip(orig_columns, features)))
data['Target'] = y
```

Listing 5. Create data scatter plots and correlation.

```
colpal = { 0: "navy", 1: "orangered", 2: "limegreen" }

def data_overview_plot(data, target):
    def corr_dot(_x, _y, **kws):
        corr_r = kws['corr_df'].loc[_x.name,_y.name]
        corr_text = f"{corr_r:2.2f}"
        ax = plt.gca()
        ax.set_axis_off()
        font_size = 20
        ax.annotate(corr_text, [.5, .5,],  xycoords="axes␣fraction",
                    ha='center', va='center', fontsize=font_size)
        marker_size = abs(corr_r) * 10000
        ax.scatter([.5], [.5], marker_size, [corr_r], alpha=0.6,
                   cmap="coolwarm", vmin=-1, vmax=1, transform=ax.transAxes)
    g = sns.PairGrid(data, hue=target, diag_sharey=False, palette=colpal)
    g.map_lower(plt.scatter, s=60, edgecolor='xkcd:off␣white', alpha=0.7)
    g.map_diag(plt.hist, bins='fd', alpha=0.7)
    g.map_upper(corr_dot, corr_df=data.corr(method='pearson'))
```

Listing 6. Data preprocessing function definition.

```
reducer_pipe = skl.pipeline.Pipeline(
      [('pre-scale', skl.preprocessing.StandardScaler()),
       ('UMAP', umap.UMAP(random_state=42)),
       ('post-scale', skl.preprocessing.StandardScaler())])
```

Listing 7. Splitting the data into trainings and test data set.

```
data2 = data.drop(['Perimeter'], axis=1)
features = list(data2.columns)
features.remove('Target')
X = data2[features].values
Y = data2['Target'].values
```

```
X_train_o, X_test_o, Y_train, Y_test = skl.model_selection.train_test_split(
      X, Y, test_size=0.3)
```

Listing 8. Reducing the data dimensions.

```
reducer = reducer_pipe.fit(X_train_o)
X_train = reducer.transform(X_train_o)
X_test = reducer.transform(X_test_o)
```

Listing 9. Plot function for data scatter plot.

```
def data_scatter_plot(X_t, ft_names, pred, title, types, Y_t=None):
  fig, ax = plt.subplots()
  ax.scatter(X_t[pred == 0, 0], X_t[pred == 0, 1],
             color=colpal[0], label=f"Class␣{types[0]}", alpha=0.6)
  ax.scatter(X_t[pred == 1, 0], X_t[pred == 1, 1],
             color=colpal[1], label=f"Class␣{types[1]}", alpha=0.6)
  if Y_t is not None:
      ax.scatter(X_t[pred != Y_t, 0], X_t[pred != Y_t, 1],
                 color=colpal[2], alpha=0.7)
      title = "␣".join([title, "(green:␣fails)"])
  sns.despine()
  ax.legend()
  ax.set(xlabel=ft_names[0], ylabel=ft_names[1])
  ax.set_title(fontweight="bold", label=title);

ft_names = ('e0', 'e1')
types = ('benign', 'malign')
```

Listing 10. Code for creating the embedded data scatter plot.

```
data_scatter_plot(
      X_train, ft_names, Y_train,
      "Breast␣cancer␣classification␣embedded␣data␣set", types)
```

Listing 11. Neural Net Python Class.

```
from pymc3.theanof import set_tt_rng, RandomStream
import scipy

class advi_neural_network():
    def __init__(self):
        self.neural_network = None
        self.inference = None
        self.approx = None
        self.sample_proba = None
        self.samples_draw = None
```

```
def build_network(self, X_train, Y_train):
    in_non = X_train.shape[1]
    hid1_non = 4
    hid2_non = 2
    initial_weights_1 = np.random.randn(in_non, hid1_non).astype(floatX)
    initial_weights_2 = np.random.randn(hid1_non, hid2_non).astype(floatX)
    initial_weights_p = np.random.randn(hid2_non).astype(floatX)
    with pm.Model() as self.neural_network:
        features = pm.Data('ann_input', X_train)
        output = pm.Data('ann_output', Y_train)
        prior_1 = pm.Normal('w_in_1', 0 , sigma=1,
                            shape=(in_non, hid1_non),
                            testval=initial_weights_1)
        prior_2 = pm.Normal('w_1_2', 0, sigma=1,
                            shape=(hid1_non, hid2_non),
                            testval=initial_weights_2)
        prior_perceptron = pm.Normal('w_2_out', 0, sigma=1,
                            shape=(hid2_non,),
                            testval=initial_weights_p)
        layer_1 = pm.math.tanh(pm.math.dot(features, prior_1))
        layer_2 = pm.math.tanh(pm.math.dot(layer_1, prior_2))
        perceptron = pm.math.sigmoid(pm.math.dot(layer_2,
                                                 prior_perceptron))
        likelihood = pm.Bernoulli('out', perceptron, observed=output,
                                  total_size=Y_train.shape[0])

def fit(self, iterations=30000, more_replacements=None):
    set_tt_rng(RandomStream(42))
    with self.neural_network:
        self.inference = pm.ADVI()
        self.approx = pm.fit(n=iterations,
                             method=self.inference,
                             more_replacements=more_replacements)

def plot_advi(self, old_ADVI=False):
    plt.figure(figsize=(12,6))
    plt.plot(self.inference.hist, label='new␣ADVI', alpha=.75)
    if old_ADVI:
        plt.plot(self.inference.hist-self.approx.hist,
                 label='deviation␣from␣old␣ADVI', alpha=.75)
    plt.legend(fontsize=15)
    plt.ylabel('ELBO', fontsize=15)
```

```
        plt.xlabel('Iterations', fontsize=15)
        plt.show()

    def traceplot(self):
        self.trace = self.approx.sample(draws=5000)
        pm.plot_trace(self.trace)

    def sample_probabilities(self):
        x = T.matrix('X') # create symbolic input
        n = T.iscalar('n') # number of samples
        x.tag.test_value = np.empty_like(X_train[:10])
        n.tag.test_value = 100
        sample_proba = self.approx.sample_node(
                    self.neural_network.out.distribution.p,
                    size=n,
                    more_replacements={self.neural_network['ann_input']: x})
        self.sample_proba = theano.function([x, n], _sample_proba)

    def draw_samples(self, X_t, sample_size=500):
        if self.sample_proba is None:
            self.sample_probabilities()
        self.samples_draw = self.sample_proba(X_t, sample_size)

    def accuracy(self, Y_t, decision_boundary):
        try:
            sample_proba_mean = self.samples_draw.mean(0)
        except AttributeError:
            print('No samples drawn; try calling `draw_samples()` first.')
            return
        pred = sample_proba_mean > decision_boundary
        return 100 * (Y_t == pred).mean()

    def true_positives(self, sample_values, Y_t, decision_boundary):
        pred = sample_values > decision_boundary
        return 100 * ((pred == 1) & (Y_t == 1)).mean()

    def true_negatives(self, sample_values, Y_t, decision_boundary):
        pred = sample_values > decision_boundary
        return 100 * ((pred == 0) & (Y_t == 0)).mean()

    def false_positives(self, sample_values, Y_t, decision_boundary):
        pred = sample_values > decision_boundary
        return 100 * ((pred == 1) & (Y_t == 0)).mean()
```

```
def false_negatives(self, sample_values, Y_t, decision_boundary):
    pred = sample_values > decision_boundary
    return 100 * ((pred == 0) & (Y_t == 1)).mean()

def fail_rate(self, sample_values, Y_t, decision_boundary):
    pred = sample_values > decision_boundary
    return 100 * (pred != Y_t).mean()

def plot_sensitivity_specificity(self, Y_t, filename=''):
    try:
        sample_proba_mean = self.samples_draw.mean(0)
    except AttributeError:
        print('No samples drawn; try calling `draw_samples()` first.')
        return

    boundaries = np.linspace(0,1,101)
    malign_p = (Y_t == 1).mean()
    sensitivity_num = np.array(
              [self.true_positives(sample_proba_mean, Y_t, i) / malign_p
               for i in boundaries])
    benign_p = (Y_t == 0).mean()
    specificity_num = np.array(
              [self.true_negatives(sample_proba_mean, Y_t, i) / benign_p
               for i in boundaries])
    accuracy_num = 100 - np.array(
              [self.fail_rate(sample_proba_mean, Y_t, i)
               for i in boundaries])

    plt.figure(figsize=(20,10))
    plt.plot(boundaries, sensitivity_num,
             label='Sensitivity', color=colpal[1])
    plt.plot(boundaries, specificity_num,
             label='Specificity', color=colpal[0])
    plt.plot(boundaries, accuracy_num,
             label='Accuracy', color='darkgreen')
    plt.xlabel('Decision Boundary', fontsize=20)
    plt.xticks(fontsize=16)
    plt.ylabel('Percentage', fontsize=20)
    plt.yticks(fontsize=16)
    plt.ylim([25,105])
    plt.legend(fontsize=20, loc='lower center')
    if len(filename) > 0:
```

```
            plt.savefig(filename+'.pdf', bbox_inches='tight')
        plt.show()

    def plot_errors(self, Y_t):
        try:
            sample_proba_mean = self.samples_draw.mean(0)
        except AttributeError:
            print('No samples drawn; try calling ‘draw_samples()‘ first.')
            return
        boundaries = np.linspace(0,1,101)
        false_positive_num = np.array(
                    [self.false_positives(sample_proba_mean, Y_t, i)
                     for i in boundaries])
        false_negative_num = np.array(
                    [self.false_negatives(sample_proba_mean, Y_t, i)
                     for i in boundaries])
        total_error_num = false_positive_num + false_negative_num
        plt.figure(figsize=(20,10))
        plt.plot(boundaries, total_error_num,
                 label='Total Error Rate', color='darkgreen')
        plt.plot(boundaries, false_positive_num,
                 label='False Positive Rate', color=colpal[0])
        plt.plot(boundaries, false_negative_num,
                 label='False Negative Rate', color=colpal[1])
        plt.xlabel('Decision Boundary', fontsize=20)
        plt.xticks(fontsize=16)
        plt.ylabel('Percentage', fontsize=20)
        plt.yticks(fontsize=16)
        plt.ylim([-2,32])
        plt.legend(fontsize=20, loc='upper center')
        plt.show()
        print('Minima of total error rate at: '
              f'{boundaries[np.argmin(total_error_num )]:.2f}')

    def plot_error_with_sd(self, Y_t, error_type='total error'):
        try:
            sample_proba_mean = self.samples_draw.mean(0)
        except AttributeError:
            print('No samples drawn; try calling ‘draw_samples()‘ first.')
            return
        if error_type == 'total error':
            descr = 'Total Error Rate'
            err_meth = self.fail_rate
```

```
            color = 'darkgreen'
        elif error_type == 'false positive':
            descr = 'False Positive Rate'
            err_meth = self.false_positives
            color = colpal[0]
        elif error_type == 'false negative':
            descr = 'False Negative Rate'
            err_meth = self.false_negatives
            color = colpal[1]
        else:
            print('Unknown error type')
            return
        sample_proba_sd = self.samples_draw.std(0)
        boundaries = np.linspace(0,1,101)
        fails_num = np.array([err_meth(sample_proba_mean, Y_t, i)
                              for i in boundaries])
        fn_plus_sd = np.array(
                        [err_meth(sample_proba_mean+sample_proba_sd, Y_t, i)
                         for i in boundaries])
        fn_minus_sd = np.array(
                        [err_meth(sample_proba_mean-sample_proba_sd, Y_t, i)
                         for i in boundaries])
        fails_num_upp = np.amax([fails_num, fn_plus_sd, fn_minus_sd], axis=0)
        fails_num_low = np.amin([fails_num, fn_plus_sd, fn_minus_sd], axis=0)
        plt.figure(figsize=(20,10))
        plt.plot(boundaries, fails_num, label=descr, color=color)
        plt.fill_between(boundaries, fails_num_low, fails_num_upp,
                         color=color, alpha=0.2)
        plt.xlabel('Decision Boundary', fontsize=20)
        plt.xticks(fontsize=16)
        plt.ylabel('Percentage', fontsize=20)
        plt.yticks(fontsize=16)
        plt.ylim([-2,32])
        plt.legend(fontsize=20, loc='upper center')
        plt.show()
```

Listing 12. Network initialization and fitting

```
bc2_network = advi_neural_network()
bc2_network.build_network(X_train, Y_train)
bc2_network.fit(iterations=50000)
bc2_network.plot_advi()
```

Listing 13. Trace plot for fitted neural network.

```
bc2_network.traceplot()
```

Listing 14. Drawing of prediction samples.

```
bc2_network.draw_samples(X_train, sample_size=1000)
```

Listing 15. Error variation in dependence of the decision boundary.

```
bc2_network.plot_errors(Y_train)
```

Listing 16. False negative rate deviation in dependence of the decision boundary.

```
bc2_network.plot_error_with_sd(Y_train, error_type='false␣negative')
print(f'Accuracy:␣{bc2_network.accuracy(Y_train,␣0.15):.2f}␣%')
```

Listing 17. Sample draws, error and accuracy calculations for the test data (see Figure 9).

```
bc2_network.draw_samples(X_test, sample_size=1000)
bc2_network.plot_errors(Y_test)
print(f'Accuracy:␣{bc2_network.accuracy(Y_test,␣0.15):.2f}')
```

Listing 18. Plot errors with indication of standard deviation.

```
bc2_network.plot_error_with_sd(Y_test)
bc2_network.plot_error_with_sd(Y_test, error_type='false␣negative')
```

Listing 19. Scatter plot of predictions for the test data.

```
pred = bc2_network.samples_draw.mean(0) > 0.15
data_scatter_plot(X_test, ft_names, pred,
                  "Predicted␣diagnosis␣in␣testing␣set", types)
```

Listing 20. Scatter plot of predictions for the test data indicating fails.

```
data_scatter_plot(X_test, ft_names, pred,
                   "Predicted␣labels␣in␣testing␣set", types, Y_t=Y_test)
```

Listing 21. Generating class probability predictions in an area around the data points in the reduced space and a visualization helper function.

```
xv_lim = (-3., 3.)
yv_lim = (-3., 3.)
grid = pm.floatX(np.mgrid[xv_lim[0]:xv_lim[1]:100j, yv_lim[0]:yv_lim[1]:100j])
grid_2d = grid.reshape(2, -1).T

bc2_network.draw_samples(grid_2d, sample_size=1000)
ppc = bc2_network.samples_draw

def plot_ppc_with_data(grid, ppc, X_t, pred, xv_lim, yv_lim,
                       ft_names, cbar_label,
                       certainty=False, fails=False, Y_t=None,
                       filename=''):
```

```
    fig, ax = plt.subplots(figsize=(14, 8))
    if not certainty:
        ppc_vals = ppc.mean(axis=0)
        cmap = sns.diverging_palette(250, 12, s=85, l=25, as_cmap=True)
    else:
        ppc_vals = ppc.std(axis=0)
        cmap = sns.cubehelix_palette(light=1, as_cmap=True)
    contour = ax.contourf(grid[0], grid[1], ppc_vals.reshape(100, 100),
                          cmap=cmap, alpha=0.9)
    ax.scatter(X_t[pred == 0, 0], X_t[pred == 0, 1],
               color=colpal[0], alpha=0.6)
    ax.scatter(X_t[pred == 1, 0], X_t[pred == 1, 1],
               color=colpal[1], alpha=0.6)
    if fails:
        ax.scatter(X_t[pred != Y_t, 0], X_t[pred != Y_t, 1],
                   color=colpal[2], alpha=0.7)
    cbar = plt.colorbar(contour, ax=ax)
    ax.set(xlim=xv_lim, ylim=yv_lim,
           xlabel=ft_names[0], ylabel=ft_names[1])
    cbar.ax.set_ylabel(cbar_label)
    if len(filename) > 0:
        plt.savefig(filename+'.pdf', bbox_inches='tight')
    return contour
```

Listing 22. Plot the model classification probability surface.

```
ppc_contour = plot_ppc_with_data(
      grid, ppc, X_test, pred, xv_lim, yv_lim, ft_names,
      f"A-posterior␣predictive␣mean␣probability␣for␣malign␣tumor")
```

Listing 23. Function call to produce a surface plot of the standard deviation of the PPC, overlaid with the scatter plot of the test data predictions.

```
plot_ppc_with_data(
      grid, ppc, X_test, pred, xv_lim, yv_lim, ft_names,
      "Uncertainty␣(posterior␣predictive␣standard␣deviation)",
      certainty=True)
```

Listing 24. Function call to produce a surface plot of the standard deviation of the PPC, overlaid with the scatter plot of the test data predictions. Wrong predictions are marked in green.

```
plot_ppc_with_data(
      grid, ppc, X_test, pred, xv_lim, yv_lim, ft_names,
      "Certainty␣(posterior␣predictive␣standard␣deviation)",
      certainty=True, fails=True, Y_t=Y_test)
```

Listing 25. Isoline extraction, its backtransformation into the initial high-dimensional feature space, and the creation of a scatter-plot matrix, including points that correspond to the isoline of the model decision boundary.

```
indx = np.where(abs(ppc_contour.levels - 0.15) < 0.001)[0][0]
points = np.array(
    [_p for _p, _c in
     ppc_contour.collections[indx].get_paths()[0].iter_segments()])
lstidx = np.where(abs(points[:,1] - 3.0) < 0.001)[0][0]
cl_points = points[:lstidx,:]
cl_points_o = reducer.inverse_transform(cl_points)
cl_data = pd.DataFrame(cl_points_o, columns=features)
cl_data['Target'] = pd.Series(np.full(cl_data.shape[0], 3.0))
data3 = pd.concat([data2, cl_data], ignore_index=True)
grf1 = sns.PairGrid(data3, hue='Target', corner=True, palette=colpal)
grf1.map_lower(plt.scatter, s=60, edgecolor='xkcd:off␣white', alpha=0.7)
```

Code outputs

Listing 26. The data description from scikit-learn.

```
Breast cancer wisconsin (diagnostic) dataset
--------------------------------------------
**Data Set Characteristics:**
    :Number of Instances: 569
    :Number of Attributes: 30 numeric, predictive attributes and the class
    :Attribute Information:
        - radius (mean of distances from center to points on the perimeter)
        - texture (standard deviation of gray-scale values)
        - perimeter
        - area
        - smoothness (local variation in radius lengths)
        - compactness (perimeter^2 / area - 1.0)
        - concavity (severity of concave portions of the contour)
        - concave points (number of concave portions of the contour)
        - symmetry
        - fractal dimension ("coastline approximation" - 1)

        The mean, standard error, and "worst" or largest (mean of the three
        worst/largest values) of these features were computed for each image,
        resulting in 30 features.  For instance, field 0 is Mean Radius, field
        10 is Radius SE, field 20 is Worst Radius.

        - class:
```

```
            - WDBC-Malignant
            - WDBC-Benign

    :Summary Statistics:
    ===================================== ====== ======
                                           Min    Max
    ===================================== ====== ======
    radius (mean):                        6.981  28.11
    texture (mean):                       9.71   39.28
    perimeter (mean):                     43.79  188.5
    area (mean):                          143.5  2501.0
    smoothness (mean):                    0.053  0.163
    compactness (mean):                   0.019  0.345
    concavity (mean):                     0.0    0.427
    concave points (mean):                0.0    0.201
    symmetry (mean):                      0.106  0.304
    fractal dimension (mean):             0.05   0.097
    radius (standard error):              0.112  2.873
    texture (standard error):             0.36   4.885
    perimeter (standard error):           0.757  21.98
    area (standard error):                6.802  542.2
    smoothness (standard error):          0.002  0.031
    compactness (standard error):         0.002  0.135
    concavity (standard error):           0.0    0.396
    concave points (standard error):      0.0    0.053
    symmetry (standard error):            0.008  0.079
    fractal dimension (standard error):   0.001  0.03
    radius (worst):                       7.93   36.04
    texture (worst):                      12.02  49.54
    perimeter (worst):                    50.41  251.2
    area (worst):                         185.2  4254.0
    smoothness (worst):                   0.071  0.223
    compactness (worst):                  0.027  1.058
    concavity (worst):                    0.0    1.252
    concave points (worst):               0.0    0.291
    symmetry (worst):                     0.156  0.664
    fractal dimension (worst):            0.055  0.208
    ===================================== ====== ======

    :Missing Attribute Values: None
    :Class Distribution: 212 - Malignant, 357 - Benign
    :Creator:  Dr. William H. Wolberg, W. Nick Street, Olvi L. Mangasarian
    :Donor: Nick Street
```

```
    :Date: November, 1995

This is a copy of UCI ML Breast Cancer Wisconsin (Diagnostic) datasets.
https://goo.gl/U2Uwz2

Features are computed from a digitized image of a fine needle
aspirate (FNA) of a breast mass.  They describe
characteristics of the cell nuclei present in the image.
```

Listing 27. The data information of the data to be modelled.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 11 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Radius          569 non-null    float64
 1   Texture         569 non-null    float64
 2   Perimeter       569 non-null    float64
 3   Area            569 non-null    float64
 4   Smoothness      569 non-null    float64
 5   Compactness     569 non-null    float64
 6   Concavity       569 non-null    float64
 7   Concave Points  569 non-null    float64
 8   Symmetry        569 non-null    float64
 9   Fractal Dim     569 non-null    float64
 10  Target          569 non-null    int64
dtypes: float64(10), int64(1)
memory usage: 49.0 KB
```