MSc Mathematics

Track: Data Science and Optimization

Master thesis

Addressing class imbalance in disease classification problems

by

Marije van Haeringen

April 29, 2022

Supervisor: Prof. Dr. Mark Hoogendoorn

Second examiner: Prof. Dr. Sandjai Bhulai

Department of Mathematics

Faculty of Sciences

VU UNIVERSITY AMSTERDAM

Abstract

Disease classification models are challenged by a class imbalance in the target class, which decreases the performance of standard classifiers. There are two main techniques to deal with imbalanced classes: generating minority class data or using cost-sensitive learning algorithms. Approaches that combine these two techniques have been proposed previously, but research reviewing these approaches in the context of disease classification is currently lacking. This thesis evaluated the individual performance of generative models and cost-sensitive classifiers, as well as the performance of hybrid approaches that combine the two methods. The results showed that imbalanced learning methods improve the performance of disease classification models, and that there is great promise in hybrid approaches.

Title: Addressing class imbalance in disease classification problems
Author: Marije van Haeringen, marijevanhaeringen@gmail.com, 2558833
Supervisor: Prof. Dr. Mark Hoogendoorn
Second examiner: Prof. Dr. Sandjai Bhulai
Date: April 29, 2022

Department of Mathematics
VU University Amsterdam
de Boelelaan 1081, 1081 HV Amsterdam
`http://www.math.vu.nl/`

Popular summary

Machine learning (ML) is a sub-field of artificial intelligence that enables computers to learn specific patterns and relationships from data, without being explicitly programmed. An important subgroup of ML algorithms are concerned with making predictions or classifications through the use of statistical methods. An example of a domain that uses machine learning to solve various problems is health care. An important aspect of health care is finding the right diagnosis as soon as possible. Therefore, ML algorithms are applied in attempts to classify specific diseases. One major challenge of these disease classification algorithms is that there are in general many more healthy individuals than patients, which makes it more difficult for the algorithm to accurately classify the patients. This is known as the class imbalance problem. There are two main approaches of dealing with the class imbalance problem: data generation and cost-sensitive learning. Data generation techniques create synthetic data from the patients, which can be used to train the ML algorithm. In that case, the algorithm can learn from more patient cases, and it is better able to distinguish the patients and healthy individuals. Cost-sensitive learning algorithms are a subgroup of ML algorithms that are able to put more focus on correctly classifying specific cases, such as the patients. The purpose of this thesis is to study which of these two methods (data generation or cost-sensitive learning) is more suited for disease classification. Additionally, we investigated whether combining a data generation method and a cost-sensitive algorithm can give even better results. The methods were compared on two datasets, each of which consisted of patients and healthy individuals. For one of the datasets, the combined method was best able to predict the patient class, while on the other dataset, a cost-sensitive algorithm had the best results. The experiment showed that data generation and cost-sensitive learning are very useful techniques to reduce the effects of class imbalance for disease classification problems. Furthermore, the combined method had the best performance on one of the dataset. However, the results did not indicate that there was one superior approach, but that the most suitable method depends on the characteristics of the dataset. For future research, it is important that the experiment is repeated on a larger range of medical datasets. That way, it can be determined which methods are best suited for specific data characteristics.

Contents

1. Introduction

Machine learning is a sub-field of artificial intelligence that focuses on algorithms that automatically learn specific patterns and relationships from data [1]. Examples include algorithms that can be trained to make classifications or predictions based on statistical methods or can cluster data based on the available features. The usage of machine learning models has drastically increased in the past two decades due to the availability of large amounts of data and growing computational power. These days, machine learning is applied in attempts to solve various complex problems in a wide range of domains, including health care [2, 3, 4]. An important aspect of health care is accurately diagnosing patients in the shortest possible time frame. As such, many research projects concentrate on building machine learning models that can effectively classify diseases. One major challenge of disease classification models is class imbalance [5]. Patients with a certain disease are generally rare compared to healthy subjects, which leads to an asymmetry between the patient and control class. Standard classification models perform worse on such problems, because these algorithms assume balanced class distributions. Furthermore, using performance metrics such as the accuracy can give a false sense of security because the class imbalance is not taken into account [6, 7]. Another challenge is that – in the context of medical diagnosis – false-negative results are often considered to be more harmful than false-positive ones. Therefore, it is important that a disease classification model avoids false-negative results, even at the cost of false-positives [8]. Since traditional classifiers do not account for unequal misclassification costs, these can lead to undesirable results. The combination of unequal costs of misclassification and class imbalance can lead to an even larger decrease in performance.

The most common approach to deal with the class imbalance problem is generating synthetic data from the minority class [8]. Generating minority class data makes it possible to create a balanced dataset for training, and thereby improve the performance of standard classifiers. Research has shown that supplementing the training set with synthetic patient data leads to higher performance on disease classification tasks [9, 10]. One classic method is Synthetic Minority Oversampling Technique (SMOTE), which uses a k-nearest neighbour approach to create new samples [11]. In the last decade, there has been an increasing amount of attention for generating data with deep learning models, for example with variational auto-encoders (VAE) [12] or generative adversarial networks (GAN) [13]. Although these techniques were initially designed to generate continuous image data, it has also shown promising results for generating tabular data containing discrete features [14, 15, 16, 17]. Cost-sensitive learning is another approach that addresses the class imbalance problem by enforcing different misclassification costs for classes using an imbalanced loss function [18]. These loss functions apply different weights to classes, thereby causing the algorithm to focus more on correctly classifying samples with higher weights. Cost-sensitive algorithms are supervised learning models that aim to minimise a cost-sensitive loss function. Many standard classifiers have been adapted to cost-sensitive versions. This includes models such as decision trees [19], extreme gradient boosting trees (XGBoost) [20] and multi-layer perception neural networks [21]. Cost-sensitive models have shown improved performance on classification tasks with medical data, compared to their cost-insensitive counterparts [22].

In addition to using either of the aforementioned methods, one can combine a generative model and a cost-sensitive learning algorithm. Some specific hybrid procedures have been proposed in previous research [18], also for diabetes classification [23]. However, research reviewing different combinations of generative models and cost-sensitive learning algorithms in the context of medical diagnosis is currently lacking [22]. The objective of this thesis is to investigate which imbalanced learning method, that is, data generation method or cost-sensitive algorithm, or combination of methods is most suitable for disease classification. More specifically, the research question of this thesis is: which imbalanced learning method or combination of methods yields the best performance on binary disease classification problems? In the context of this research, the performance is measured by two scores: one punishes false-negative results more severely than false-positives, and the other assumes equal costs of misclassification. In order to answer the research question, an experiment was performed that compares various methods on medical datasets that represent the general population. The following two binary classification problems were selected: (1) predicting congenital hypothyroidism in the Newborn Screening (NBS) dataset and (2) predicting (pre)diabetes in the Diabetes Health Indicator (DHI) dataset. Five different classification algorithms were trained and evaluated per problem: Random Forest, Extreme Gradient Boosting (XGBoost) with cross-entropy loss, and three adaptations of the XGBoost algorithm with cost-sensitive loss functions. All selected classifiers are tree ensemble models, because these classifiers have shown superior performance over other predictive models on classification problems with tabular data [24, 25]. Two generative models were implemented to create synthetic minority class data: SMOTE and VAE for tabular data. The predictive models were fitted on three different training sets: original data, original data combined with a SMOTE generated sample and original data combined with a VAE generated sample. This experimental design makes it possible to investigate the individual performance of the imbalanced learning methods, as well as the performance of the hybrid procedures.

This thesis consists of two parts: the first is concerned with theoretical background, while the second discusses the experiment performed in this research. More specifically, the first part consists of an introduction to supervised learning (Chapter 2), a description of the mathematical basics of tree-based models (Chapter 3), the principles behind data generation methods (Chapter 4) and cost-sensitive learning algorithms (Chapter 5). The second part contains the experimental setup (Chapter 6), a detailed description of the two datasets used in this research (Chapter 7), the results from the experiment (Chapter 8) and finally a discussion that includes the explanations and implications of the results (Chapter 9).

Part I.

Theoretical background

2. Supervised learning

Machine learning concerns giving computers the ability to make decisions from data without being explicitly programmed. Two large subfields of machine learning are supervised learning and unsupervised learning. Supervised learning algorithms aim to predict the response Y using the set of features X. Supervised learning problems can be expressed as the approximation of the conditional distribution $\mathbb{P}_{Y|X}$. If the response Y is a categorical variable taking only a finite number of possible values, this is a classification problem. If Y is numerical, it is a regression problem. In contrast, unsupervised learning aims to find patterns in X without there being a pre-defined response. An unsupervised learning problem can be expressed as the estimation of (properties of) the probability distribution $\mathbb{P}_X$. Since the objective of this thesis is to compare various supervised learning techniques, this section describes the supervised learning task in more detail [26]. The statistical framework of supervised learning models is discussed as well as the training and evaluation of these models on data.

2.1. Statistical framework

Let $Y \in \mathcal{Y}$ be the response variable, $X = (X_1, ..., X_p) \in \mathcal{X}$ the p-dimensional vector of predictor variables and assume that $Y \times X$ are random variables on $\mathcal{Y} \times \mathcal{X}$. The input of the supervised learning algorithm is a training dataset $\mathcal{D} = \{(y_i, \boldsymbol{x}_i)\}_{i=1}^N$ that is assumed to be independently and identically drawn from the joint distribution $\mathbb{P}_{Y,X}$. The goal of the algorithm is to find an accurate predictor function $f : \mathcal{X} \to \mathcal{Y}$ with respect to some loss function $L : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$. The objective of the function f is to predict $y \in \mathcal{Y}$ for some $\boldsymbol{x} \in \mathcal{X}$, in particular values of x that are not contained in the training data. The loss function L measures the accuracy of the prediction $f(\boldsymbol{x})$ with respect to the corresponding y. More specifically, we want to find the f that minimises the expected loss, known as the *risk*, that is defined as

$$R(f) = \mathbb{E}_{Y,X}[L(y, f(\boldsymbol{x}))].$$

The main challenge is that the risk can not be minimised because the underlying distribution $\mathbb{P}_{Y,X}$ is unknown. However, we can consider the empirical distribution $\hat{\mathbb{P}}_{Y,X}$, in which each element from $\mathcal{D}$ occurs with probability $1/n$. The empirical risk $\hat{R}(f)$ is defined as the expectation of the loss with respect to the empirical distribution $\hat{\mathbb{P}}_{Y,X}$ instead of the true distribution $\mathbb{P}_{Y,X}$, and given by

$$\hat{R}(f) = \frac{1}{n}\sum_{i=1}^{n} L(y_i, f(\boldsymbol{x}_i)).$$

The approach of finding f that minimises $\hat{R}$ is called empirical risk minimisation (ERM). The resulting empirical risk minimiser

$$\hat{f} = \arg\min_{f} \hat{R}(f),$$

is an approximation of the true risk minimiser $f^* = \arg\min_f R(f)$. In order to make the problem of finding f feasible, it is important to assume that f belongs to some function class $\mathcal{F}$. Tree models are an example of such a function class, and are discussed in detail in Chapter 3.

2.1.1. Loss function

As discussed, the loss function measures the error made in predicting y by $f(\boldsymbol{x})$ for $(y, \boldsymbol{x}) \in \mathcal{D}$, and the goal is to find f such that this error is as small as possible. Which loss function is most suitable depends on the type of problem. In case of regression, $\mathcal{Y}$ is continuous and the loss function is usually a distance measure. The most common loss function for regression problems is the quadratic loss

$$L(y, f(\boldsymbol{x})) = |y - f(\boldsymbol{x})|^2,$$

in which case the risk is known as the mean squared error $R(f) = \mathbb{E}[|Y - f(X)|^2]$. In case of binary classification ($\mathcal{Y} \in \{0, 1\}$), the default loss function is the binary cross-entropy function given by

$$L(y, p(\boldsymbol{x})) = -y\log(p(\boldsymbol{x})) - (1-y)\log(1-p(\boldsymbol{x})),$$

where $p(\boldsymbol{x}) = \mathbb{P}(Y = 1|X)$. For multiclass classification problems, the binary cross-entropy loss can be extended to the categorical cross-entropy loss. Given that there are K classes, the output of the model is $p(\boldsymbol{x}) = (p_1(\boldsymbol{x}), ..., p_K(\boldsymbol{x}))$ with $p_k(\boldsymbol{x}) = \mathbb{P}(Y = k|X)$, and the true outcome y is split in K binary variables $y = (y_1, ..., y_K)$ that indicate whether the outcome belongs to a particular class. The categorical cross-entropy loss is given by

$$L(y, p(\boldsymbol{x})) = -\sum_{k=1}^{K} y_k \log(p_k(\boldsymbol{x})).$$

There are many other examples of loss functions that may be more suitable for given problems. Most loss functions are convex, because every local minimum of a convex function is equal to the global minimum. This implies that we can find the global minimum of a convex loss function using a local optimisation algorithm. An example of such an algorithm is gradient descent, which is discussed in the context of tree boosting in Chapter 3.

2.2. Training and Evaluation

The goal of supervised learning is to fit a model to the data, in order to make predictions about other unknown cases. It is common practise to split the data into a training and test set prior to fitting the model. The training data is used to estimate the empirical risk minimiser f. In some cases, it is also desired to use to training data to finetune the hyperparameters of the model. The test set is used to evaluate the performance of the fitted model with the optimal hyperparameters, using one or more performance measures. Additionally, the importance of individual features on the performance of the trained model can be investigated. Hyperparameter tuning and feature importance are discussed in more detail in this paragraph, together with a number of performance metrics for regression and classification.

2.2.1. Hyperparameter tuning

Most supervised learning algorithms contain one or more hyperparameters that determine the structure of the model. Examples include the number of trees in a Random Forest or the number of hidden layers in a neural network. It is important to choose the parameter values that yield the best performance given the problem setting and the training data. One can compare the performance of various models with different hyperparameter values in order to choose the optimal values. However, training and evaluating the models on the same dataset can lead to overfitting. Therefore, it is important to set aside a part of the training data as an independent evaluation of the performance: the validation set. Then, models with different parameter values are fitted to the training data, and compared on the validation set. Another approach that is often used for hyperparameter tuning is *k-fold cross validation.* Instead of fitting a model with a specific set of hyperparameters to the train set once, the training data is split in k folds and a model with the same parameters is fit to each fold. The performance of the model on each fold is evaluated on the validation set. This technique gives more insight in the variability of the model performance for a particular choice of parameters, reduces overfitting, and identifies the most suitable set of hyperparameters.

2.2.2. Feature Importance

In addition to evaluating the performance of the model, it is relevant to know which features contribute most to making good predictions. There are different ways to investigate the feature importance, and the most suitable choice depends on the model. For example for linear or logistic regression models, one can compare the size of the coefficients of different features, while for tree models one can measure feature importance using the Gini impurity. One popular method that can be used for all supervised learning models is to calculate the decrease in model performance if one feature is randomly shuffled, known as permutation feature importance [27]. Because the values of a feature are shuffled, there is no relationship anymore between the feature and the target. If this severely decreases the model performance, it indicates that the reshuffled feature is important for predicting the outcome.

2.2.3. Performance metrics for regression

Performance metrics for regression measure the performance as a function of the predicted value $\hat{y} = f(\boldsymbol{x})$ and the true value y. Similar to the loss function, the performance metrics used in regression are often distance metrics. Examples include the *mean absolute error* (MAE) and the *mean squared error* (MSE), given by

$$MAE = \frac{1}{n}\sum_{i=1}^{n} |y_i - \hat{y}_i|,$$

$$MSE = \frac{1}{n}\sum_{i=1}^{n} (y_i - \hat{y}_i)^2.$$

The *R-squared* – also known as the coefficient of determination – is defined as one minus the sum of residual squares divided by the total sum of squares

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2},$$

where $\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$. The R-squared determines to which degree the model can explain the variance in the data, where 0 is the worst possible performance and 1 indicates a perfect model. An advantage of the R-squared metric compared to the MAE or MSE is that it is independent of the unit of the outcome, since it is between 0 and 1. A disadvantage is that adding more features never decreases the R^2, because it assumes that adding more features always increases the variance. Even if the additional features do not have any explanatory value, the R-squared score is greater than or equal to the R^2 of the model without these features. The *adjusted R-squared* score handles this by adding an extra term that accounts for the number of features in the model. It is given by

$$R_a^2 = 1 - \frac{n-1}{n-p-1} \cdot (1 - R^2),$$

where n is the sample size and p the number of features included in the model. The adjusted R-squared is always lower than the R-squared score, and only increases when the added features improve the model more than expected.

2.2.4. Performance metrics for classification

Classification metrics measure the performance of a model using the true target labels and the predictions, which can either be the predicted labels or probabilities. Various performance metrics for binary classification problems, i.e. $\mathcal{Y} = \{0, 1\}$, are described in this paragraph. Most of these measures can be extended to the multiclass case, although that is not discussed here.

The *confusion matrix* visualises the true outcome labels against the predicted labels. Although the confusion matrix on itself is not a performance metric, the entries of the matrix are used as building blocks for other metrics. An example is shown in Figure 2.1. The true positive (TP) and true negative (TN) are the number of positive class ($Y = 1$) respectively negative class samples ($Y = 0$) that were predicted correctly. The false positive (FP) and false negative (FN) are the number of positive respectively negative class samples that were incorrectly predicted.

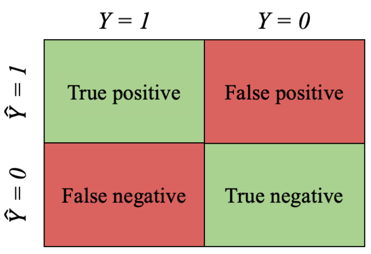


Figure 2.1.: Confusion matrix.

The most simple and intuitive metric for classification is the *accuracy*, which is defined as the percentage of correct predictions

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN}.$$

The accuracy only measures how many samples are classified correctly, and does not look at the respective labels. Therefore, using this metric can be misleading for problems with an unequal class distribution (imbalanced data) or problems in which the cost of misclassification is not equal for the two classes.

The *positive predictive value* (PPV) – also known as the *precision* – quantifies the percentage of true positive samples among all samples that were predicted as positive by the model. It is given by

$$PPV = \frac{TP}{TP + FP}.$$

The PPV only takes into account the positive class. Therefore, measuring the performance with this metric provides information about the type-I errors made, but not about the type-II errors. The *true positive rate* (TPR) – also known as the *recall* – measures the percentage of correctly labelled positive predictions among all positive samples, while the *false positive rate* (FPR) measures the percentage of incorrectly labelled negative predictions among all negative samples. The formulas are as follows:

$$TPR = \frac{TP}{TP + FN},$$
$$FPR = \frac{FP}{FP + TN}.$$

The *receiver operating characteristic* (ROC) curve is a classification metric that measures the performance of a classifier using the predicted probabilities, instead of the predicted class labels. As such, this metric is only suitable for models that predict the probabilities that the input $\boldsymbol{x}$ belongs to each class k, known as probabilistic classifiers. In case of binary classification, a probabilistic classifier predicts the probability that $\boldsymbol{x}$ belongs to the positive class ($y = 1$). The predicted class labels are derived from these probabilities using a classification threshold, by default set to 0.5. The ROC curve displays the performance of a model in terms of the TPR and FPR at all classification thresholds (Figure 2.2).

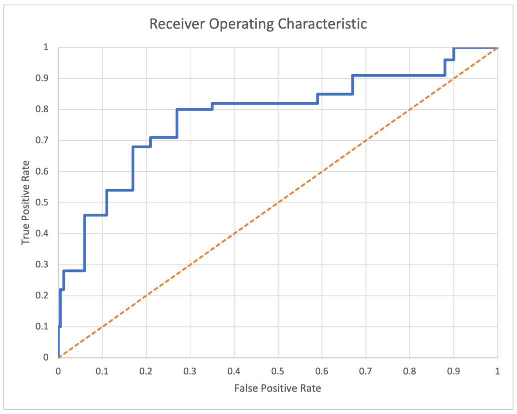


Figure 2.2.: Receiver Operating Characteristic curve.

The ROC curve is especially meaningful in case of unequal cost of misclassification. It informs whether there exists a classification threshold for which it is possible to correctly classify all positive cases while remaining a relatively low false positive rate. Since the ROC curve compares the number of true and false positives as a fraction of the total number of all positive and negative samples, respectively, it is sensitive to class imbalance. More specifically, the ROC curve overestimates the performance of a classifier in case the positive class is much smaller than the negative class. Therefore, the ROC curve should be used with caution in case of class imbalance. The *area under the ROC* (ROC-AUC) can be derived from the ROC curve, and measures the area under the ROC curve on the unit plane. The ROC-AUC score summarises the TPR and FPR at different thresholds in a single measure, which can be useful when comparing different classifiers. A disadvantage is that the ROC-AUC does not account for imbalanced class labels or unequal misclassification costs, and can therefore give a false sense of security.

The *F1 score* is the harmonic mean of the positive predictive value (precision) and the true positive rate (recall), and is given by

$$F_1 = 2 \cdot \frac{PPV \cdot TPR}{PPV + TPR}.$$

A high F1 score corresponds to both a high precision and a high recall. In this metric, the precision and recall are considered equally important. Therefore, it is very useful in cases were the cost of misclassification is equal among the classes, but should be used with caution if this is not the case. The *F-beta score* is an extension of the F1 score that allows to shift the balance between the precision and recall, and is therefore more suitable for problems with unbalanced costs. The F-beta score is the weighted harmonic mean of the PPV and TPR given by

$$F_\beta = (1 + \beta^2)\frac{PPV \cdot TPR}{\beta^2 \cdot PPV + TPR}.$$

The positive parameter β determines the weight of recall, where the score with $\beta < 1$ favours precision and $\beta > 1$ favours recall. To be more precise, in the F-beta score the recall is considered β times more important than the precision.

3. Tree-based models

Tree-based models are supervised learning algorithms that predict the outcome $y \in \mathcal{Y} \subset \mathbb{R}$ based on the feature vector $\boldsymbol{x} \in \mathcal{X} \subset \mathbb{R}^p$ by first partitioning the feature space into rectangles and then assigning a constant output value to each partition. The data is split in various partitions $R_1, ..., R_J$ using a binary decision tree. At each node of the tree, the data is split into two subsets according to the value of one of the features. Each leaf node results in a partition R_j and a constant output value γ_j modelled to R_j, $j = 1, ..., J$. Given that we have such a partition, the model prediction $\hat{y}$ is given by the function

$$T(\boldsymbol{x}, \Theta) = \sum_{j=1}^{J} \gamma_j \mathbb{I}\{\boldsymbol{x} \in R_j\}, \tag{3.1}$$

where the vector of model parameters $\Theta = \{R_j, \gamma_j\}_j$ consists of the partitions and corresponding output values. For each region, the value of γ_j that minimises the loss function given $\boldsymbol{x} \in R_j$ can be computed. Although there are various algorithms for constructing trees, the Classification And Regression Trees (CART) algorithm for constructing trees [26] is discussed in more detail because this method provides a foundation for other models such as random forest and XGBoost. This chapter describes the CART algorithm to construct regression and classification trees, two important tree ensemble methods known as bagging and boosting and several algorithms derived from these methods. These tree ensemble algorithms have shown superior performance on a wide range of classification and regression problems with tabular data [24, 25].

3.1. CART for regression

Suppose $\{(y_n, \boldsymbol{x}_n)\}_{n=1}^N$ is the training data, where $\boldsymbol{x}_n = (x_{n1}, ..., x_{np})$ is the p-dimensional feature vector and $y \in \mathbb{R}$ the output. For regression problems a common choice of loss function is the sum of squares $\sum(y_i - \hat{y}_i))^2$. The tree is build using a top-down, greedy, recursive algorithm that chooses the split with the minimal loss at each step. Starting with all the data, splitting variable t at split point s results in

$$R_1(t, s) = \mathbb{I}\{\boldsymbol{x}_t \leqslant s\} \text{ and } R_2(t, s) = \mathbb{I}\{\boldsymbol{x}_t > s\}.$$

Then, the splitting pair (t, s) with corresponding constant values γ_1, γ_2 that minimises the sum of squares is given by

$$\min_{t,s}\left[\min_{\gamma_1}\sum_{\boldsymbol{x}_n\in R_1(t,s)}(y_n-\gamma_1)^2+\min_{\gamma_2}\sum_{\boldsymbol{x}_n\in R_2(t,s)}(y_n-\gamma_2)^2\right].$$

Notice that $\gamma_i = \text{average}(y_n|\boldsymbol{x}_n \in R_i(t,s))$, $i = 1, 2$, regardless of the choice for (t, s). After finding the best split, the data is partitioned into two subsets and the splitting is repeated for both regions. The splitting process continues until a stopping criteria is met.
It should be noted that the CART algorithm – for both regression and classification – assumes all features $\boldsymbol{x}$ are numerical, and is therefore not able to handle categorical input. Therefore, categorical features should be handled accordingly, for example, by one-hot encoding categorical variables that have no intrinsic order between the classes.

For CART, the preferred criteria is to grow a large tree until some minimum node size is reached. Thereafter, the tree is pruned using the cost-complexity pruning technique, which is explained next. Let a subtree $T \subset T_0$ be any tree that can be obtained by pruning T_0, that is, by collapsing one or more internal nodes. Let $|T|$ be the number of leaf nodes in T, index the leaf nodes by $j = 1, ..., |T|$ and let leaf node j correspond to partition R_j. The cost-complexity criterion is defined as

$$C_\alpha(T) = \sum_{j=1}^{|T|} N_j Q_j(T) + \alpha|T|,$$

where $N_j = \#\{x_n \in R_j\}$ is the size of partition R_m, $\hat{\gamma}_j = \frac{1}{N_j}\sum_{x_n\in R_j} y_n$ is the average outcome for partition R_j, $Q_j(T) = \frac{1}{N_j}\sum_{x_n\in R_j}(y_n - \hat{\gamma}_j)^2$ the error of partition R_j given the prediction $\hat{\gamma}_j$ and α the hyperparameter that controls the size of the tree. By minimising the cost-complexity criterion, a trade-off is made between the predictive accuracy and the size of the tree. More specifically, a larger tree decreases the first term of $C_\alpha(T)$, but increases the second term. The degree to which the size of the tree is penalised is determined by α, where larger values of α lead to smaller trees. The cost-complexity criterion is used to find the subtree $T_\alpha \subseteq T_0$ that minimises $C_\alpha(T)$ for each $\alpha \geqslant 0$. The tuning parameter $\hat{\alpha}$ is optimised using k-fold cross-validation, and the pruned tree $T_{\hat{\alpha}}$ is obtained.

3.2. CART for classification

Consider the case where the outcome y is a categorical variable taking values $1, 2, ..., K$. The CART algorithm described in the previous paragraph remains the same, except for the error function. In case of regression, the mean squared error was used for building the large tree as well as the pruning algorithm. For classification there are three options to measure the error:

- Misclassification rate $1 - \max_k \hat{p}_{jk}$;
- Gini index $\sum_{k=1}^{K} \hat{p}_{jk}(1 - \hat{p}_{jk})$;
- Cross-entropy function $-\sum_{k=1}^{K} \hat{p}_{jk} \ln \hat{p}_{jk}$.

Here $\hat{p}_{jk} = \frac{1}{N_j} \sum_{\boldsymbol{x}_n \in R_j} \mathbb{I}\{y_n = k\}$ is the proportion of class k observations in node j. The Gini index and cross-entropy function are strictly concave as a function of the node probabilities $\hat{p}_{jk}$, and therefore, more sensitive to changes than the misclassification rate. As such, the latter two measures are preferred when growing large trees. Although all three measures could be used as $Q_j(T)$ in the cost-complexity pruning algorithm, the misclassification rate is the most common choice.

3.3. Tree ensemble models

Decision trees are easy to implement and interpreted. However, a large disadvantage of using a single tree as predictor is that they are prone to overfitting; a small change in the training data can lead to major changes in the structure of the tree. Therefore, decision trees are weak learners, meaning that they predict only slightly better than random chance. The idea behind ensemble methods is to combine the predictions of multiple weak learners in order to build a more robust model with lower variance. If the weak learners are trees, these are referred to as tree ensembles. The two main families of ensemble methods – bagging and boosting – are discussed in this paragraph with respect to trees as weak learners, but it should be kept in mind that these methods can be generalised to other weak learners.

3.3.1. Bagging

Bagging was first introduced by Breiman [28], and it is an acronym for bootstrap aggregating. The main idea of bagging is to train a model on many bootstrap samples and averaging the predictions to obtain one model. By bootstrapping, the algorithm imitates having many different training samples, thereby reducing the variance of the overall model. The algorithm is as follows. Let $\{(y_n, \boldsymbol{x}_n)\}_{n=1}^{N}$

be the training data, with $\boldsymbol{x}_n = (x_{n1}, ..., x_{np})$ the predictor variables and $y \in \mathbb{R}$ the response variable. For $b = 1, ..., B$, the following two steps are repeated. First, a bootstrap sample of size N is drawn from the training data with replacement. Second, a tree is fit to the bootstrap sample to obtain the tree parameters Θ_b for the predictor $T(\boldsymbol{x}, \Theta_b)$. Then, in case of regression, the predictor of the bagged model after B iterations is

$$T_{bag}(\boldsymbol{x}) = \frac{1}{B}\sum_{b=1}^{B} T(\boldsymbol{x}, \Theta_b).$$

For classification trees, the output of each $T(\boldsymbol{x}, \Theta_b)$ is the predicted class of $\boldsymbol{x}$. The prediction of all B trees is summarised in the vector $(p_1(\boldsymbol{x}), ..., p_K(\boldsymbol{x}))$, where $p_k(\boldsymbol{x})$ is the proportion of bootstrapped trees that predicted class k, for $k = 1, ..., K$. The prediction of the bagged estimator is then equal to the class that got the most votes, that is,

$$T_{bag}(\boldsymbol{x}) = \arg\max_k p_k(\boldsymbol{x}).$$

Random Forest

An important extension of bagged trees is the Random Forest [27]. In addition to using bootstrap samples to train the weak learners, a random subset of features is selected for each node splitting. This results in a collection of de-correlated trees, since each time different features are available to make the splits. Consequently, the variance of the aggregated model decreases, which results in a more robust model. The intuition behind this is as follows. The trees generated by the bootstrap samples are identically distributed, but not independent. Let ρ be the pairwise variance between the trees, then the variance of the average of the B bootstrap trees is

$$\rho\sigma^2 + \frac{1-\rho}{B}\sigma^2.$$

The second term decreases for growing values of B, and the variance of the average is mainly determined by the pairwise correlation ρ. The random forest aims the reduce ρ by randomly selecting the input variables. Consequently, the variance of the average reduces when a smaller number of variables is considered at each split.

The Random Forest algorithm is as follows. Let $\{(y_n, \boldsymbol{x}_n)\}_{n=1}^{N}$ be the training data, with $\boldsymbol{x}_n = (x_{n1}, ..., x_{np})$ the predictor variables and $y \in \mathbb{R}$ the response variable. For $b = 1, ..., B$, the following two steps are repeated. First, a bootstrap sample of size N is drawn from the training data, in a similar manner as the bagged trees. Second, a tree is fit to the bootstrap sample to obtain the tree parameters Θ_b for the predictor $T(\boldsymbol{x}, \Theta_b)$ by repeating the following until the minimum node size

n_{min} is met. Randomly select $m \leqslant p$ features, select the best splitting variable and split point, and split the node in two child nodes. For regression trees, the predictor of the bagged model after B iterations is equal to

$$T_{rf}(\boldsymbol{x}) = \frac{1}{B}\sum_{b=1}^{B} T(\boldsymbol{x}, \Theta_b).$$

For classification trees, the bagged predictor is given by

$$T_{rf}(\boldsymbol{x}) = \arg\max_k p_k(\boldsymbol{x}),$$

where $p_k(\boldsymbol{x})$ is the proportion of bootstrapped trees that predicted class k, $k = 1, ..., K$.

As explained before, the variance of the average reduces for smaller values of m. The size of the trees is determined by the minimum node size, where a smaller minimum node size corresponds to larger trees. Large trees are more prone to overfitting, and are therefore not desired. It is also possible to set a maximum depth of the tree to make sure that it can not grow larger than a specified number of splits.

3.3.2. Boosting

In contrast to bagging, the weak models that make up the boosting estimator are not constructed independently, but build in series. Starting with an initial model that predicts the target y from the features $\boldsymbol{x}$, the subsequent models are fitted to the error of the previous model. Thus, each learner – except the initial model – aims to predict the target using the prediction error made by the previous model. Then, adding the predictions of all subsequent learners to the initial model yields a more reliable estimator. More specifically, the idea behind boosting is to use an additive expansion of many weak learners to produce a strong predictor. The base learners used in tree boosting are CART regression trees. The goal of tree boosting is to find a predictor of the form

$$\hat{y} = \sum_{m=1}^{M} T(\boldsymbol{x}, \Theta_m), \tag{3.2}$$

that minimises the expected loss of predicting y with $\hat{y}$ defined by some loss function L. This predictor is found with an additive stagewise algorithm. Let $\{(y_n, \boldsymbol{x}_n)\}_{n=1}^{N}$ be the training data, with $\boldsymbol{x}_n = (x_{n1}, ..., x_{np})$ the predictor variables and $y \in \mathbb{R}$ the response variable. First, set an initial constant model $f_0(\boldsymbol{x}) = c$

with $c \in \mathbb{R}$. Then for $m = 1, ..., M$, repeat the following. Calculate the parameters $\Theta_m = (R_{jm}, \gamma_{jm})_j$ that solve the following expression

$$\hat{\Theta}_m = \arg\min_{\Theta} \sum_{i=1}^{N} L(y_i, f_{m-1}(\boldsymbol{x}_i) + T(\boldsymbol{x}_i; \Theta)), \tag{3.3}$$

and set $f_m(\boldsymbol{x}) = f_{m-1}(\boldsymbol{x}) + T(\boldsymbol{x}, \Theta_m)$. Estimating the parameters Θ_m is done using a greedy, top-down recursive algorithm.

Finding the parameters Θ_m is not easy for a arbitrary loss functions, and therefore takes time. In order to make fast algorithms we need approximations of the solution. Gradient tree boosting and XGBoost are two popular examples of algorithms that rely on approximations of Equation (3.3).

Gradient tree boosting

The idea behind gradient tree boosting is to approximate the solution of (3.3) using the steepest descent algorithm. This optimisation algorithm finds a local minimum by taking iterative steps into the opposite direction of the gradient, which is the direction of steepest descent. Recall from Chapter 2 that the loss function is generally convex, which implies that every local minimum is equal to the global. Thus, the steepest descent algorithm is used to approximate the global minimum of the loss function, and thereby corresponding solution of (3.3).

The steepest descent algorithm works as follows. Let the objective be to minimise the loss function $L(f)$ with respect to f, where the loss of predicting y by $f(\boldsymbol{x})$ is given by $L(f) = \sum_{n=1}^{N} L(y_n, f(\boldsymbol{x}_n))$. In terms of numerical optimisation, this is equivalent to

$$\hat{\mathbf{f}} = \arg\min_{\mathrm{f}} L(\mathbf{f}),$$

with $\mathbf{f} = \{f(\boldsymbol{x}_1), ..., f(\boldsymbol{x}_N)\}^T$ the approximate values of $f(\boldsymbol{x}_n)$ for $n = 1, ..., N$. This equation is then solved by the sum of M increment steps

$$\mathbf{f}_M = \sum_{m=1}^{M} \boldsymbol{h}_m, \qquad \boldsymbol{h}_m \in \mathbb{R}^N.$$

Here, $\mathbf{f}_0 = \boldsymbol{h}_0$ is the initial guess, and for each next step m the guess is incremented with $\boldsymbol{h}_m$. The way that the increment vector $\boldsymbol{h}_m$ is computed differs between optimisation methods. The steepest descent algorithm uses $\boldsymbol{h}_m = -\rho_m \boldsymbol{g}_m$ with

$$\boldsymbol{g}_{nm} = \left[\frac{\delta L(y_n, f(\boldsymbol{x}_n))}{\delta f(x_n)}\right]_{f(\boldsymbol{x}_n) = f_{m-1}(\boldsymbol{x}_n)},$$

$$\rho_m = \arg\min_{\rho} L(\mathbf{f}_{m-1} - \rho \boldsymbol{g}_m).$$

Both steepest descent algorithm and forward stagewise boosting are very greedy algorithms, since at each step the solution that gives the maximal reduce is chosen.

The original implementation of gradient tree boosting is known as Multiple Additive Regression Trees (MART) [29], and is described in Algorithm 1 for regression. The algorithm for classification is similar, where we have a categorical target y and a $(K+1)$-dimensional loss function L. For a K-class target, the MART algorithm builds K estimators – one for each class – that can produce probabilities or vote for a class label. The details are described in Algorithm 2.

Algorithm 1 MART algorithm for regression.

```
Require: constant model $f_0(\boldsymbol{x}) = \arg\min_{\gamma} \sum_{n=1}^{N} L(y_n, \gamma)$
 1: for $m = 1, ..., M$ do
 2:     for $n = 1, ..., N$ do
 3:         Compute the pseudo-residuals
                $r_{nm} = -\left[\frac{\delta L(y_n, f(\boldsymbol{x}_n))}{\delta f(\boldsymbol{x}_n)}\right]_{f=f_{m-1}}$
 4:     end for
 5:     Fit regression tree $T$ to $\{r_{nm}\}_{n=1}^{N}$ to find $\{R_{jm}\}_{j=1}^{J_m}$
 6:     for $j = 1, ..., J_m$ do
 7:         Calculate $\gamma_{jm} = \arg\min_{\gamma} \sum_{\boldsymbol{x}_n \in R_{jm}} L(y_n, f_{m-1}(\boldsymbol{x}_n) + \gamma)$
 8:     end for
 9:     Set $f_m(\boldsymbol{x}) = f_{m-1}(\boldsymbol{x}) + \sum_{j=1}^{J_m} \gamma_{jm} \mathbb{I}\{\boldsymbol{x} \in R_{jm}\}$
10: end for
11: $f_M(\boldsymbol{x}) = \sum_{m=1}^{M} T(\boldsymbol{x}, \Theta_m)$ with $\Theta_m = \{R_{jm}, \gamma_{jm}\}_j$
```

Algorithm 2 MART algorithm for classification.

```
Require: constant model $f_0(\boldsymbol{x}) = \arg\min_{\gamma} \sum_{n=1}^{N} L(y_n, \gamma)$
 1: for $k = 1, ..., K$ do
 2:     for $m = 1, ..., M$ do
 3:         for $n = 1, ..., N$ do
 4:             Compute the pseudo-residuals
                    $r_{nkm} = -\left[\frac{\delta L(y_n, f_1(\boldsymbol{x}_i), ..., f_K(\boldsymbol{x}_n))}{\delta f_k(\boldsymbol{x}_n)}\right]_{f=f_{m-1}}$
 5:         end for
 6:         Fit regression tree $T$ to $\{r_{nkm}\}_{n=1}^{N}$ to find $\{R_{jkm}\}_{j=1}^{J_{km}}$
 7:         for $j = 1, ..., J_{km}$ do
 8:             Calculate $\gamma_{jkm} = \arg\min_{\gamma} \sum_{\boldsymbol{x}_n \in R_{jkm}} L(y_n, f_{m-1}(\boldsymbol{x}_n) + \gamma)$
 9:         end for
10:         Set $f_{km}(\boldsymbol{x}) = f_{k(m-1)}(\boldsymbol{x}) + \sum_{j=1}^{J_{km}} \gamma_{jkm} \mathbb{I}\{\boldsymbol{x} \in R_{jkm}\}$
11:     end for
12: end for
13: $f_{kM}(\boldsymbol{x}) = \sum_{m=1}^{M} T(\boldsymbol{x}, \Theta_{km})$ with $\Theta_{km} = \{R_{jkm}, \gamma_{jkm}\}_j$ for $k = 1, ..., K$
```

XGBoost

Extreme gradient boosting (XGBoost) was introduced by Chen and Guestrin [30] as an improvement to the original gradient tree boosting algorithm. The main improvements are better performance, faster training and scalability to high dimensional problems. The XGBoost algorithm has shown state-of-the-art performance on classification and regression tasks with tabular data, compared to other ensemble models and deep learning methods [25]. However, it should be noted that the performance of gradient tree boosting algorithms is very dependent on the hyperparameter values. Leaving parameters at default often leads to worse results, whereas the performance of the Random Forest is more robust under different parameter values [24]. The details of the XGBoost algorithm are discussed below.

XGBoost deviates from the MART algorithm presented in the previous paragraph in a number of ways. Recall that the goal of boosting is to find a sum of trees (3.2) that minimises the expected loss. In order to prevent overfitting, XGBoost implemented a regularised objective to be minimised with respect to $\hat{y}$

$$\mathcal{L} = \sum_{n=1}^{N} L(y_n, \hat{y}_n) + \sum_{m} \Omega(T(\boldsymbol{x}, \Theta_m)), \tag{3.4}$$

where $\Theta_m = \{R_{jm}, \gamma_{jm}\}_j$ are parameters of the m-th tree. The term $\Omega(T)$ penalises the complexity of the trees, and can be written as

$$\Omega(T) = \alpha|T| + \lambda_1||\gamma||_1 + \frac{1}{2}\lambda_2||\gamma||_2^2.$$

with $|T|$ the number of leaves of tree T, γ the vector of constant values corresponding to the regions $\{R_j\}$, and $\alpha, \lambda_1, \lambda_2 \in \mathbb{R}$. The number of terminal nodes of each individual tree are penalised by the parameter α, which is equivalent to the second term of the cost-complexity criterion (Paragraph 3.1). The parameters λ_1, λ_2 correspond to L1 and L2 regularisation of the leaf output γ, respectively. A stagewise boosting approach is used to optimise this expression. Let $\hat{y}_n^{(m-1)}$ be the prediction of the n-th instance of y at the m-th iteration. The goal is to find the tree parameters Θ_m that solve

$$\Theta_m = \arg\min_{\Theta} \sum_{n=1}^{N} L(y_i, \hat{y}_n^{(m-1)} + T(\boldsymbol{x}_n, \Theta)) + \Omega(T(\boldsymbol{x}, \Theta)).$$

While MART uses a first-order approximation, XGBoost has implemented a second-order approximation to quickly optimise this objective. This approximation is given by

$$\mathcal{L} = \sum_{n=1}^{N} [L(y_n, \hat{y}_n^{(m-1)}) + g_n T(\boldsymbol{x}_n, \Theta_m) + \frac{1}{2} h_n T(\boldsymbol{x}_n, \Theta_m)^2] + \Omega(T(\boldsymbol{x}, \Theta_m)),$$

with g_n respectively h_n the first and second order derivatives of the loss function at $\hat{y}^{(m-1)}$:

$$g_n = \frac{\delta L(y_n, \hat{y}^{(m-1)})}{\delta \hat{y}^{(m-1)}} \text{ and } h_n = \frac{\delta^2 L(y_n, \hat{y}^{(m-1)})}{\delta (\hat{y}^{(m-1)})^2}.$$

In addition to the regularised objective, XGBoost uses shrinkage and column subsampling to further prevent overfitting. Shrinkage of the newly added weights with learning rate η reduces the influence of past trees compared to more recent ones. Column subsampling is also used in the Random Forest, where a subset of features is selected for each node splitting.

Finally, XGBoost has several implementations that make the algorithm more applicable for problems with high dimensional data. The goal of tree learning is to find the splitting parameters Θ that build a tree that minimises the objective. In Random Forest and MART, an exact greedy algorithm is used to find these parameters. Although this algorithm is very powerful, it is computationally demanding in high dimensional problems because it enumerates over all possible splits in order to find the best one. To overcome this issue in such settings, XGBoost offers an approximate algorithm that makes gradient boosting possible. First, the algorithm proposes candidate splitting points using the percentiles of the feature distribution. Second, the continuous features are split into buckets by these candidate points and finds the best solution according to the aggregated statistics. Furthermore, it is common in real-world problems for the feature matrix $\boldsymbol{x}$ to be sparse. In XGBoost, each tree node has a default direction for missing data points.

The XGBoost algorithm can be implemented with any twice differentiable loss function. By default, the loss function for regression is the squared loss function and for binary classification the binary cross entropy function (see Chapter 2). However, there are other possible loss functions that may be more suitable in certain situations. A few examples of alternative loss functions are described in Chapter 5 in the context of cost-sensitive learning for imbalanced classification problems.

4. Data generation methods

Real-world classification tasks are often challenged by an asymmetrical class distribution of the target variable, which is known as the *class imbalance problem*. The class imbalance problem occurs frequently in the medical domain because disease cases are usually rare compared to healthy individuals and the cost of misclassifying an ill person is much larger than of a healthy person. Because traditional classifiers assume balanced classes, they generally perform worse on imbalanced data. There are two main approaches of dealing with class imbalanced problems: re-sampling and cost-sensitive learning (Chapter 5) [6]. Re-sampling methods aim to increase the classifiers performance by creating a balanced training sample, which can be obtained by either undersampling or oversampling the data. Undersampling methods eliminate samples from the majority class in order to balance the data, while oversampling methods increase the size of the minority class by generating new samples. Because undersampling leads to smaller sample sizes and possible bias in the data, oversampling is preferred over undersampling in almost all settings. Classic oversampling approaches – such as random oversampling – simply increase the sample by multiplying minority cases, and are therefore sensitive to overfitting [31]. In order to overcome this problem, methods that use a k-nearest neighbour approach to generate synthetic samples were proposed, for example Synthetic Minority Oversampling Technique (SMOTE) [11] or Adaptive Synthetic (ADASYN) sampling approach [32]. More recent techniques aim to use deep learning models to generate synthetic data, including variational auto-encoders (VAE) [12] or generative adversarial networks (GAN) [13]. This section describes the theoretical principles of SMOTE and VAE in terms of tabular data generation, followed by several metrics to evaluate the quality of the generated data.

4.1. SMOTE

The Synthetic Minority Oversampling Technique (SMOTE) was introduced by Chawla et al. [11]. The idea behind SMOTE is to create a new minority sample in the direction of one of the nearest neighbours of a real minority datapoint. The generation of new data works as follows. A real minority class datapoint is chosen, and one of its k-nearest neighbours is randomly selected. The difference between the original datapoint and the selected neighbour is calculated, and multiplied by a random number between 0 and 1. Then, the synthetic datapoint is created by adding this difference to the original sample, which yields a random sample along the line between the sample and its neighbour. These steps are repeated to created a new synthetic dataset of minority class cases, as is visualised in Figure 4.1. Algorithm 3 contains a detailed description of SMOTE.

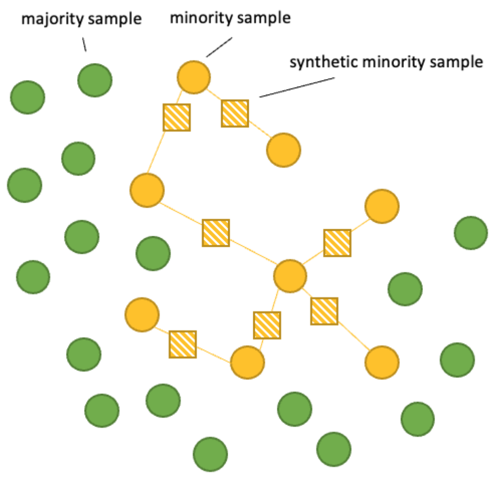


Figure 4.1.: Graphical representation of data generation using SMOTE.

A disadvantage of the SMOTE algorithm is that it can only handle data with continuous variables. Therefore, the inventors proposed a generalisation of SMOTE – called Synthetic Minority Oversampling Technique-Nominal Continuous (SMOTE-NC) – that is able to create synthetic data for mixed datasets of continuous and categorical variables. In order to calculate the k-nearest neighbours of a minority datapoint, SMOTE-NC uses an alternative method to calculate the distance between the datapoints. The median of the standard deviations of the continuous features for the minority class is used as a proxy for the distance between two samples that have different values for a categorical feature. The distance between two datapoints is then the Euclidean distance of the respective continuous variables, plus the median for every categorical variable that differs between the two points. The new synthetic sample is generated, where the continuous variables are calculated in the same manner as the SMOTE algorithm. For the categorical variables, the class that occurs in the majority of the k-nearest neighbours is implemented in the new minority sample.

Algorithm 3 Synthetic Minority Oversampling Technique.

```
 1: $T$ number of minority class samples
 2: $N$ amount of SMOTE in %
 3: $k$ number of nearest neighbours
 4: $p$ number of variables
 5: $Sample[][]$ array for original minority class samples
 6: $Synthetic[][]$ array for synthetic samples
 7: if $N < 100$ then
 8:     Randomise T minority class samples
 9:     $T = (N/100) * T$
10:     $N = 100$
11: end if
12: $N = \text{round}(N/100)$
13: Initialise $newindex = 0$
14: for $i = 1, ..., T$ do
15:     Compute $k$ nearest neighbours of $i$, and save the indices to $nnarray$
16:     Set $N_i = N$
17:     while $N_i \neq 0$ do
18:         Choose a random integer between 1 and $k$, and call it $nn$
19:         for $j = 1, ..., p$ do
20:             Compute $dif = Sample[nnarray[nn]][j] - Sample[i][j]$
21:             Compute a random number between 0 and 1, and call it $gap$
22:             Set $Synthetic[newindex][j] = Sample[i][j] + gap * dif$
23:         end for
24:         Increment $newindex$ by 1
25:         Set $N_i = N_i - 1$
26:     end while
27: end for
```

4.2. Variational Auto-Encoder

The variational auto-encoder (VAE) was proposed by Kingma and Welling [12]. Similar to a standard auto-encoder, the VAE consists of an encoder and decoder network. The encoder network converts the input to a more dense representation, while the decoder network converts the dense variables back to the original input. In contrast to the standard auto-encoder, the space of the dense representations – called the latent space – of the variational auto-encoder is continuous, and therefore allows for random sampling. In this case, the input is encoded as a distribution over the latent space, instead of a single value. Data can be generated by sampling from the latent space and using the decoder to convert it to the dimensions of the original data. Next follows a more technical description of the variational auto-encoder.

The VAE belongs to the family of variational Bayesian models and continuous latent variable models. Here, it is assumed that the data $\{\boldsymbol{x}_i\}_{i=1}^n \subseteq \mathcal{X} = \mathbb{R}^R$ is generated by a random process involving the continuous latent variable $\boldsymbol{z} \in \mathcal{Z} = \mathbb{R}^Q$ with $Q \ll R$. In this process, a vector $\boldsymbol{z}$ is generated from some prior distribution $p_{\boldsymbol{\theta}}(\boldsymbol{x})$ and $\boldsymbol{x}$ is generated by some conditional distribution $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})$. The goal is to estimate the parameters $\boldsymbol{\theta}$ by maximising the marginal likelihood given by

$$p_{\boldsymbol{\theta}}(\boldsymbol{x}) = \int p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z}) d\boldsymbol{z} = \int p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}) p_{\boldsymbol{\theta}}(\boldsymbol{z}) d\boldsymbol{z}.$$

A common problem is that the integral of the marginal likelihood $p_{\boldsymbol{\theta}}(\boldsymbol{x})$ and the true posterior density $p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}) = p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})p_{\boldsymbol{\theta}}(\boldsymbol{z})/p_{\boldsymbol{\theta}}(\boldsymbol{x})$ are intractable. In order to approximate the parameters $\boldsymbol{\theta}$, the posterior inference of $\boldsymbol{z}$ given $\boldsymbol{x}$ and the marginal inference of $\boldsymbol{x}$, a recognition model $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ is introduced as an approximation of the intractable posterior $p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})$. The recognition model $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ is referred to as the encoder, and $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})$ as the decoder. Variational auto-encoders use neural networks to model both the encoder and the decoder. The general structure of a VAE is displayed in Figure 4.2. The VAE was originally proposed as a model to generate continuous image data, but can also be modified to generate tabular data. This section first describes the mathematics behind the traditional variational auto-encoder, and then discusses an implementation of a tabular variational auto-encoder (TVAE).

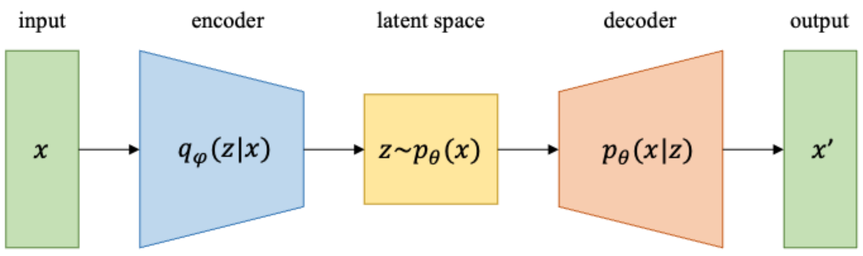


Figure 4.2.: Variational Auto-Encoder.

4.2.1. Mathematical Details

In the VAE model, the encoder and decoder are modelled as neural networks. Therefore, the marginal likelihood and posterior density are intractable. The aim is to maximise the marginal likelihood $p_{\boldsymbol{\theta}}$ indirectly by maximising the variational lower bound of the marginal likelihood at each datapoint $\boldsymbol{x}_i$. The marginal likelihood of the data is a composition of the likelihood over the individual data:

$$\log p_{\boldsymbol{\theta}}(\boldsymbol{x}_1, ..., \boldsymbol{x}_n) = \sum_{i=1}^{n} \log p_{\boldsymbol{\theta}}(\boldsymbol{x}_i).$$

Thus, the marginal likelihood of the data can be maximised by optimising the likelihood over the individual datapoints. The derivation of the variational lower bound is described next.

The goal is to approximate $p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})$ by $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ by minimising the Kullback-Leibler divergence given by

$$KL(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})|p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})) = \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log \frac{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}{p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})}\right].$$

The Kullback-Leibler divergence KL is a distance measure for probability distributions. This expression can be rewritten as follows.

$$\begin{aligned} KL(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})|p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})) &= \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log \frac{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}{p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})}\right], \\ &= \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}[\log q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})] - \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}[\log p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})], \\ &= \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}[\log q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})] - \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}[\log p_{\boldsymbol{\theta}}(\boldsymbol{z}, \boldsymbol{x})] + \log p_{\boldsymbol{\theta}}(\boldsymbol{x}). \end{aligned}$$

Then, re-arranging the terms yields the variational lower bound on $p(x)$ given by

$$\begin{aligned} \log p_{\boldsymbol{\theta}}(\boldsymbol{x}) &= KL(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})|p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})) - \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}[\log q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})] + \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}[\log p_{\boldsymbol{\theta}}(\boldsymbol{z}, \boldsymbol{x})], \\ &\geqslant -\mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}[\log q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})] + \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}[\log p_{\boldsymbol{\theta}}(\boldsymbol{z}, \boldsymbol{x})], \\ &= -\mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}[\log q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})] + \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}[\log p_{\boldsymbol{\theta}}(\boldsymbol{z})] + \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})], \\ &= -KL(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})|p_{\boldsymbol{\theta}}(\boldsymbol{z})) + \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})], \\ &= \boldsymbol{\mathcal{L}}(\boldsymbol{x}; \boldsymbol{\theta}, \boldsymbol{\phi}). \end{aligned}$$

We aim to optimise the lower bound $\boldsymbol{\mathcal{L}}(\boldsymbol{x}; \boldsymbol{\theta}, \boldsymbol{\phi})$ with respect to the parameters $\boldsymbol{\phi}$ and $\boldsymbol{\theta}$. However, differentiating with respect to ϕ is not straightforward because of the second term. The parametrisation trick was proposed as an alternative method to generate samples from $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$. Let z be a continuous random variable drawn from the conditional distribution $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$. Then $\boldsymbol{z}$ can be expressed as a deterministic variable $\boldsymbol{z} = g_{\boldsymbol{\phi}}(\epsilon, \boldsymbol{x})$, where ϵ is an auxiliary variable from an independent distribution $p(\epsilon)$ and g a function with parameter $\boldsymbol{\phi}$. This parametrisation ensures that the Monte Carlo estimate of the expectation with respect to $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ is differentiable. The estimate is given by

$$\int q_{\boldsymbol{\phi}}(\boldsymbol{x}|\boldsymbol{z}) f(\boldsymbol{z}) d\boldsymbol{z} \simeq \frac{1}{L} \sum_{l=1}^{L} f(g_{\boldsymbol{\phi}}(\boldsymbol{x}, \epsilon_l)),$$

where $\epsilon_l \sim p(\epsilon)$ and L the number of Monte-Carlo simulations. Further assumptions on the distribution of the latent space are made based on the nature of the problem at hand. A common choice is the standard normal distribution, which is described in the following paragraph.

Gaussian variational auto-encoder

Assume that the latent space $\mathcal{Z}$ follows a multivariate standard normal distribution, that is,

$$p_{\boldsymbol{\phi}}(\boldsymbol{z}) = \mathcal{N}(\boldsymbol{z}; 0, I_Q),$$

where I_Q is the $Q \times Q$ identity matrix. Since the encoder $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ maps the input from the sample space $\mathcal{X}$ to the latent space $\mathcal{Z}$, it is reasonable to assume that $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ also follows a Gaussian distribution given by

$$q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) = \mathcal{N}(\boldsymbol{z}; \mu(\boldsymbol{x}; \boldsymbol{w}), \operatorname{diag}(\sigma^2(\boldsymbol{x}; \boldsymbol{w}))).$$

Here, the mean $\mu(\boldsymbol{x}; \boldsymbol{w})$ and covariance matrix $\operatorname{diag}(\sigma^2(\boldsymbol{x}; \boldsymbol{w}))$ are predicted by the encoder neural network with weights $\boldsymbol{w}$. In this case, we use the parametrisation

$$z_i = g_\phi(\boldsymbol{x}_i, \epsilon) = \mu^{(i)} + \sigma_i \odot \epsilon \qquad \epsilon \sim \mathcal{N}(0, I).$$

Because $p_{\boldsymbol{\theta}}(\boldsymbol{z})$ and $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ are normally distributed, the Kullback-Leibler divergence of $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ and $p_{\boldsymbol{\theta}}(\boldsymbol{z})$ can be computed analytically. The resulting variational lower bound is

$$\boldsymbol{\mathcal{L}}(\boldsymbol{\theta}, \boldsymbol{\phi}, \boldsymbol{x}_i) \simeq \frac{1}{2}\sum_{j=1}^{J}(1 + \log(\sigma(\boldsymbol{x}_i)_j)^2 - \mu(\boldsymbol{x}_i)_j - \sigma(\boldsymbol{x}_i)_j^2) + \frac{1}{L}\sum_{l=1}^{L}\log p_{\boldsymbol{\theta}}(\boldsymbol{x}_i|z_{i,l}).$$

In practise, the variational auto-encoder is trained in mini batches of the training data. Therefore, it is not necessary to use multiple Monte-Carlo simulations for the second term of the variational lower bound. By setting $L = 1$, the second term – also known as the *reconstruction loss* – reduces to $\log p_{\boldsymbol{\theta}}(\boldsymbol{x}_i|z_i)$.

4.2.2. Generating data

Fitting the VAE model to a training dataset $X \subseteq \mathcal{X}$ yields the trained encoder $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ and decoder $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})$ that minimise the variational lower bound $\boldsymbol{\mathcal{L}}(\boldsymbol{\theta}, \boldsymbol{\phi}, \boldsymbol{x})$. This trained decoder can be used to generate data as follows. First, take a random sample Z from the latent space. For the Gaussian VAE, this would be a random sample from the multivariate standard normal distribution. Then, feed the sample Z to the decoder to generate the synthetic sample X'.

4.2.3. Tabular Variational Auto-Encoder

Generating tabular data poses an extra challenge, because it consists of a mixture of categorical and continuous variables. The categorical variables are often highly imbalanced, making it more difficult for the model to learn the distribution. Furthermore, continuous variables in tabular data often have non-Gaussian or even multi-modal distributions. Therefore, the variational auto-encoder as described above generally does not perform well on tabular data. Xu and Wang [16] introduced the tabular variational auto-encoder (TVAE) model, an adaptation of the standard VAE for tabular data, by applying a mode-specific normalisation step prior to training the model and adjusting the decoder network $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})$. TVAE has shown great performance compared to other deep learning algorithms for tabular data generation [17].

Mode-specific normalisation

The mode-specific normalisation step is designed to increase the performance of the model on non-Gaussian distributions. It is introduced as an alternative to min-max transformation, which can result in vanishing gradients for non-Gaussian distributions. Let $\boldsymbol{x}$ be the training data with n instances and p features, then the jth feature is given by $\boldsymbol{x}_j = (x_{1j}, ..., x_{nj})$ for $j = 1, ..., m$. The matrix $\boldsymbol{x}$ is a mixture of d categorical variables and c continuous variables. The categorical features are represented as one-hot encoded vectors, while the continuous features as numerical values. The normalisation step is performed independently for each continuous feature $\boldsymbol{x}_j$, $j = 1, ..., c$, and consists of the following three steps. First, use a variational Gaussian mixture model to estimate the number of modes m_j and fit a Gaussian mixture to $\boldsymbol{x}_j$. Let $\eta_1, ..., \eta_{m_j}$ be the modes of $\boldsymbol{x}_j$, then the estimated Gaussian mixture is given by

$$\mathbb{P}_{\boldsymbol{x}_j}(x_{ij}) = \sum_{k=1}^{m_j} \mu_k \mathcal{N}(x_{ij}; \eta_k, \phi_k),$$

where x_{ij} is the value of column $\boldsymbol{x}_j$ for instance i, and μ_k respectively ϕ_k the weight and standard deviation of mode η_k. Second, compute the probability of x_{ij} coming from each node using the probability densities

$$\rho_k = \mu_k \mathcal{N}(x_{ij}; \eta_k, \phi_k), \qquad k = 1, ..., m_j.$$

Third, each instance x_{ij} is represented by two variables: the normalised value α_{ij} and the vector β_{ij}. The scalar α_{ij} is the value x_{ij} normalised by the most probable mode η^*, which is selected using the probability densities ρ_k. It is calculated as $\alpha_{ij} = \frac{x_{ij} - \eta^*}{4\phi^*}$, where ϕ^* is the standard deviation of mode η^*. The one-hot encoded vector β_{ij} indicates the most probable mode. After the normalisation is performed on each continuous column, the ith instance of the data is represented as

$$\boldsymbol{x}_i = \alpha_{i1} \oplus \beta_{i1} \oplus ... \oplus \alpha_{ic} \oplus \beta_{ic} \oplus d_{i1} \oplus ... \oplus d_{id}.$$

where d_{ij} is the one-hot encoded vector of the categorical variable $j = 1, ..., d$. Thus, the data after normalisation consists of $2c + d$ variables.

Decoder network

In the TVAE model, the encoder $q_{\phi}(\boldsymbol{z}|\boldsymbol{x})$ is equivalent to the encoder in the Gaussian variational auto-encoder described in the previous paragraph. The output of the decoder is the joint distribution of the reconstructed values $\hat{\alpha}_{ij}, \hat{\beta}_{ij}$ and $\hat{d}_{ij}$, where the former are activated with the *tanh* function and the latter two with the *softmax* function. It is assumed that the $\hat{\alpha}_{ij}$ are normally distributed, whereas the $\hat{\beta}_{ij}$ and $\hat{d}_{ij}$ follow a categorical distribution. The output joint distribution of the decoder network for instance i is given by

$$p_{\theta}(\boldsymbol{x}_i|z_i) = \prod_{j=1}^{c} \mathbb{P}(\hat{\alpha}_{ij} = \alpha_{ij}) \prod_{j=1}^{c} \mathbb{P}(\hat{\beta}_{ij} = \beta_{ij}) \prod_{j=1}^{d} \mathbb{P}(\hat{d}_{ij} = d_{ij}).$$

4.3. Synthetic data quality

It is important to investigate whether the generated data is an accurate representation of the original data. This way, one can quantify how reliable the generated data is, or compare the performance of models with different hyperparameter values. There are various approaches that can be considered, and it is often desired to use a combination of metrics [33, 34]. First of all, one can compare the marginal distributions of the real and synthetic data using a statistical test. For continuous features, a common choice is the two-sided Kolmogorov-Smirnov (KS) statistic that tests whether two samples originate from the same distribution. For categorical variables, one can use the chi-squared statistic or the Kullback-Leibler divergence to test if the real and generated data are from the same discrete distribution.

Although it is important to investigate the marginal distributions, it does not provide any information about the relationships between the variables in the real data versus the synthetic data. More specifically, it is necessary to assess whether the generated data has captured the statistical dependence structure of the real data in order to evaluate the quality of the generated data. One approach is to calculate the pairwise correlation difference (PCD), which is defined as the distance between the correlation matrices of the real and synthetic data given by

$$PCD = ||corr(X_{real}) - corr(X_{synthetic})||,$$

where $corr(X)$ is the Pearson correlation matrix of feature matrix X and $||\cdot||$ some distance measure, for example the $L2$ norm or Frobenius norm. For smaller values of the PCD, the synthetic data was better able to capture the pairwise correlation structure of the real data. The similarity of the underlying structure of the real versus synthetic data can also be investigated using clustering methods. One can combine the real and synthetic data and perform a cluster analysis on the combined data with a fixed number of clusters, for example k-means clustering. Then, the fraction of the real data in each cluster compared to the fraction of real data in the overall dataset can be used as a measure of the similarity of the real and synthetic data. The log-cluster metric is defined as

$$logcluster = \log\left(\frac{1}{k}\sum_{i=1}^{k}\left[\frac{n_j^R}{n_j} - \frac{n^R}{n^R + n^S}\right]^2\right),$$

where k is the number of clusters, n_j the number of samples in cluster j, n_j^R, n_j^S the number of real respectively synthetic samples in cluster j and n^R, n^S the total number of real respectively synthetic samples. Large values of the log-cluster metric suggest that the distributions of the real and synthetic data are different.

Another approach to compare the dependence structure of the real and synthetic data is to evaluate the predictive performance of both datasets. Suppose one variable from the data is selected as a target variable, and the goal is to predict the target using the remaining variables as features. Two predictive models are fitted, one on the real data and the other on the generated data. Then, a suitable performance metric is used to measure the performance of both models on a real data validation set. This setup can also be extended to compare more than two datasets, for example as was done by [16] and [17] to investigate the performance of their proposed models against other generative models.

5. Cost-sensitive learning

The previous section describes the principles behind data generation methods as a way improving a classifier's performance on a class imbalanced dataset. Another approach is adapting the classifier to assign different costs to classes, known as cost-sensitive learning. Classification models that minimise a cost-sensitive loss function are known as cost-sensitive algorithms. Most standard classifiers have imbalanced counterparts, including decision trees [19] and extreme gradient boosting trees (XGBoost) [20]. This research is focused on cost-sensitive extensions of the XGBoost algorithm, due to the excellent performance of XGBoost on classification tasks with tabular data. However, it should be kept in mind that this does not indicate that cost-sensitive XGBoost algorithms are superior to other cost-sensitive models. The impact of minimising an imbalanced loss function may be different among algorithms. This section describes three cost-sensitive loss functions and the implementation of these losses in XGBoost for binary classification.

5.1. Cost-sensitive loss functions

5.1.1. Weighted cross-entropy loss

Recall from Chapter 2 that it is common practise in a K-class classification problem ($K \geqslant 2$) to optimise the cross-entropy loss function given by

$$L(y, p(\boldsymbol{x})) = -\sum_{k=1}^{K} y_k \log(p_k(\boldsymbol{x})),$$

where $p_k(\boldsymbol{x})$ is the probability that $y = k$ given $\boldsymbol{x}$, and y_k the binary variable indicating $y = k$. The *weighted cross-entropy loss* is a cost-sensitive loss function that is directly derived from the cross-entropy loss. It is given by

$$L(y, p(\boldsymbol{x})) = -\sum_{k=1}^{K} \alpha_k y_k \log(p_k(\boldsymbol{x})),$$

with α_k the weight parameter that determines the cost of misclassification for class k, $k = 1, ..., K$. Larger values of α_k cause the algorithm to focus more on classifying class k samples correctly. The weighted cross-entropy loss function for a fixed class k is shown in Figure 5.1. Notice that for larger values of α, the loss decreases slower as p approaches 1.

5.1.2. Focal loss

Another cost-sensitive error function is the *focal loss* function [35]. It was introduced in the context of dense object detection, but can also be applied to other types of classification problems. The idea behind focal loss is an additional error term $[1 - p_k(\boldsymbol{x})]^\gamma$ to the cross-entropy loss, which causes the algorithm to focus on examples that are difficult to classify. For $p_k(\boldsymbol{x})$ close to 0 (i.e. low confidence that it is the correct class) the factor $[1 - p_k(\boldsymbol{x})]^\gamma$ is large, while for $p_k(\boldsymbol{x})$ closer to 1 the factor quickly decays towards 0. The focal loss function is defined as

$$L(y, p(\boldsymbol{x})) = -\sum_{k=1}^{K} y_k [1 - p_k(\boldsymbol{x})]^\gamma \log(p_k(\boldsymbol{x})),$$

where $\gamma \geqslant 0$ is the focus parameter. Setting $\gamma_k > 0$ reduces the loss for well-identified examples and puts more weight on the difficult examples. The focal loss function for a fixed class k is shown in Figure 5.2.

5.1.3. Weighted focal loss

The inventors of the focal loss function also proposed the *weighted focal loss* function, given by

$$L(y, p(\boldsymbol{x})) = -\sum_{k=1}^{K} \alpha_k y_k [1 - p_k(\boldsymbol{x})]^\gamma \log(p_k(\boldsymbol{x})).$$

Here, α_k is an additional penalisation term for misclassification for class k, equivalent to the weight parameter in the weighted cross-entropy loss function. The authors argue that the α-weighted focal loss function has slightly higher accuracy than the unweighted variant.

5.2. Imbalanced XGBoost

Recall from Chapter 3 that the XGBoost algorithm is a gradient tree boosting algorithm that uses a second-order approximation of the loss function to find the tree parameters that minimise the objective. Any twice differentiable convex loss function can be used, and therefore also the weighted cross-entropy and (weighted) focal loss. It is necessary for implementation to provide the first and second-order

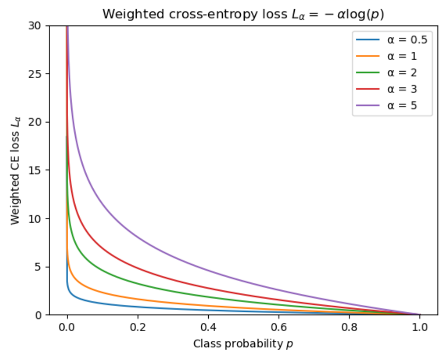


Figure 5.1.: Weighted cross-entropy loss as a function of the class probability for different values of weight parameter α.

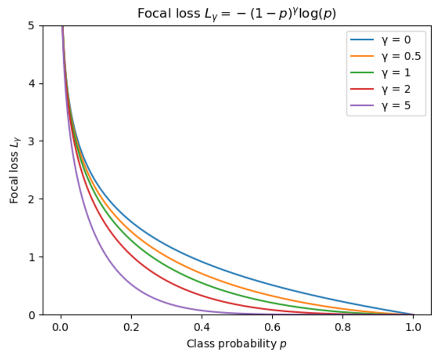


Figure 5.2.: Focal loss as a function of the class probability for different values of focal parameter γ.

derivatives of the loss functions explicitly. The derivations of the gradient and Hessian of the three cost-sensitive loss functions are described next. This is based on the work of Wang et al. [20], in which the weighted cross-entropy and focal loss function for binary classification problems with the XGBoost algorithm were implemented in the `imbalanced-xgb` package.

5.2.1. Weighted cross-entropy loss

The weighted cross-entropy loss function for binary classification is given by

$$L_\alpha(y,p) = -\alpha y\log(p) - (1-y)\log(1-p), \tag{5.1}$$

where $y \in \{0,1\}$ is the target, p the predicted probability that y is equal to 1 and $\alpha > 0$ the relative cost of misclassification. If α is set to 1, it is equivalent to the standard cross-entropy function. For $\alpha > 1$, the cost associated with incorrectly classifying a positive case ($y = 1$) is higher than with a negative case ($y = 0$), while for $\alpha < 1$, misclassification of a negative case has higher costs. It should be noted that the predicted probability p is the raw model output z applied with the sigmoid function given by

$$p = \sigma(z) = \frac{1}{1+\exp(-z)}.$$

It is a known property of the sigmoid function that its derivative with respect to z is equal to $\sigma(z)(1-\sigma(z))$. The gradient and Hessian that are used in the XGBoost algorithm are the derivatives of the loss function with respect to the raw output

z. By the chain rule for derivatives, the gradient can be written as

$$\frac{\delta L_\alpha(y,p)}{\delta z} = \frac{\delta L_\alpha(y,p)}{\delta p} \cdot \frac{\delta p}{\delta z}.$$

It follows from (5.1) that the first term on the right hand side is equal to

$$\frac{\delta L_\alpha(y,p)}{\delta p} = \frac{\alpha y(p-1) + (1-y)p}{p(p-1)}.$$

Furthermore, it follows from the derivative of the sigmoid function that $\delta p/\delta z = p(1-p)$. Then, the gradient of the binary weighted cross-entropy loss is equal to

$$\frac{\delta L_\alpha(y,p)}{\delta z} = \alpha y(p-1) + (1-y)p = -\alpha^y(y-p).$$

The Hessian can be derived in a similar manner, as shown below.

$$\frac{\delta^2 L_\alpha(y,p)}{\delta z} = \frac{\delta^2 L_\alpha}{\delta z \delta p} \cdot \frac{\delta p}{\delta z} = \alpha^y \cdot p(1-p) = \alpha^\gamma p(1-p).$$

5.2.2. Focal loss

The focal loss function for binary classification is given by

$$L_\gamma(y,p) = -y(1-p)^\gamma \log(p) - (1-y)p^\gamma \log(1-p),$$

where the focus parameter γ reduces the loss for examples that are easy to classify and increases the loss on difficult examples. Computing the gradient of the focal loss is done in a similar manner as the weighted cross-entropy loss, by using the chain rule to write the derivative as a product of $\delta L_\gamma/\delta p$ and $\delta p/\delta z$. The first-order derivative of the binary focal loss function with respect to p is given by

$$\frac{\delta L_\gamma(y,p)}{\delta p} = \frac{y(1-p)^\gamma(\gamma p \log(p) + p - 1) - (y-1)p^\gamma(\gamma(p-1)\log(1-p) + p)}{p(1-p)}.$$

Again by multiplying this expression with the derivative of the sigmoid function $\sigma(z)$ with respect to z, it follows that the gradient of the binary focal loss function is equal to

$$\begin{aligned}\frac{\delta L_\gamma(y,p)}{\delta z} =& y(1-p)^\gamma(\gamma p \log(p) + p - 1) - (y-1)p^\gamma(\gamma(p-1)\log(1-p) + p),\\ =& \gamma((p+y-1)(y+(-1)^y p)^\gamma \log(1-y-(-1)^y p))\\ &+ (-1)^y(y+(-1)^y p)^{\gamma+1}.\end{aligned}$$

The authors proposed four shorthand variables to simplify the expression for the derivatives of the focal loss. The variables and their derivatives with respect to p – which are necessary for computation of the Hessian in the next step – are given by

$$\begin{aligned}
\eta_1 &= p(1-p); && \\
\eta_2 &= y + (-1)^y p && \Rightarrow \quad \delta\eta_2/\delta p = (-1)^y; \\
\eta_3 &= p + y - 1 && \Rightarrow \quad \delta\eta_3/\delta p = 1; \\
\eta_4 &= 1 - y - (-1)^y p && \Rightarrow \quad \delta\eta_4/\delta p = -(-1)^y.
\end{aligned}$$

Then, the first-order derivative can be written as

$$\frac{\delta L_\gamma}{\delta z} = \gamma\eta_3\eta_2^\gamma \log(\eta_4) + (-1)^y \eta_2^{\gamma+1}.$$

Finally, the second-order derivative of the binary focal loss is derived as follows.

$$\begin{aligned}
\frac{\delta L_\gamma}{\delta z^2} =& \frac{\delta^2 L_\gamma}{\delta z \delta p} \cdot \frac{\delta p}{\delta z}, \\
=& \Bigg(\gamma \left[\frac{\delta\eta_3}{\delta p}\eta_2^\gamma \log(\eta_4) + \eta_3\gamma\eta_2^{\gamma-1}\frac{\delta\eta_2}{\delta p}\log(\eta_4) + \eta_3\eta_2^\gamma\eta_4^{-1}\frac{\delta\eta_4}{\delta p} \right] \\
& + (-1)^y(\gamma+1)\eta_2^\gamma \frac{\delta\eta_2}{\delta p} \Bigg) \cdot \eta_1, \\
=& \eta_1 \left(\gamma \left[(\eta_2^\gamma + \gamma(-1)^y\eta_3\eta_2^{\gamma-1}) \log(\eta_4) + -(-1)^y\eta_3\eta_2^\gamma\eta_4^{-1} \right] + (\gamma+1)\eta_2^\gamma \right).
\end{aligned}$$

5.2.3. Weighted focal loss

The `imbalanced-xgb` package was extended to include the option for binary classification with weighted focal loss for the research in this thesis. The binary weighted focal loss function is given by

$$L_{\alpha,\gamma}(y,p) = -\alpha y(1-p)^\gamma \log(p) - (1-y)p^\gamma \log(1-p),$$

where the interpretation of α and γ is equivalent to that of the weighted cross-entropy loss and focal loss, respectively. Notice that the derivatives from the binary weighted cross-entropy loss are equal to the derivatives of the standard cross-entropy function times α^y. This is not surprising, because the weight parameter multiplies the loss by α in case $y = 1$. It follows that the gradient and Hessian of the weighted focal loss function are equal to the first and second derivative of the focal loss function times α^y, respectively.

$$\begin{aligned}
\frac{\delta L_{\alpha,\gamma}}{\delta z} &= \alpha^y \left(\gamma\eta_3\eta_2^\gamma \log(\eta_4) + (-1)^y\eta_2^{\gamma+1} \right); \\
\frac{\delta L_{\alpha,\gamma}}{\delta z^2} &= \alpha^y\eta_1 \left(\gamma \left[(\eta_2^\gamma + \gamma(-1)^y\eta_3\eta_2^{\gamma-1}) \log(\eta_4) + -(-1)^y\eta_3\eta_2^\gamma\eta_4^{-1} \right] + (\gamma+1)\eta_2^\gamma \right).
\end{aligned}$$

Part II.

Experiment

6. Experimental setup

The objective of this thesis is to investigate several imbalanced learning methods (generative models or cost-sensitive algorithms) on binary disease classification problems. The following two problems were considered: classifying congenital hypothyroidism in the Newborn Screening (NBS) and predicting (pre)diabetes in the Diabetes Health Indicator (DHI) dataset. For each problem, the analysis consisted of the three steps: data preprocessing, data generation, model training and evaluation. The code that was written to perform the analyses is publicly available on GitHub [36]. This chapter describes the three steps of the experiment, and the two datasets are described in the next chapter.

6.1. Data preprocessing

The data was split in a model train set and test set using a stratified split (67/33) on the target variable with the `scikit-learn` package [37], in order to preserve the prevalence of the minority class in both samples. Missing values were imputed for the train and test set seperately, using RandomForest based imputation from the `missingpy` package [38]. This imputation method was chosen because it handles continuous and categorical variables simultaneously, does not require hyperparameter tuning and has shown superior performance compared to other imputation methods such as k-nearest neighbour imputation [39, 40, 41]. The model train set was used to generate the minority class data and fit the classifiers, while the test set was used to evaluated the performance of the trained classifiers.

6.2. Data generation

Two generative models (SMOTE, TVAE) were used to generate synthetic minority class data to balance the model training set. First, the optimal hyperparameters of the generative models were selected. The model train set was split in a parameter train and validation set (67/33) using a stratified split on the target variable. The parameter train set was used to fit generative models for different parameter values, which were evaluated using the validation set. To maintain the ratio between the required amount of oversampling, each fitted model generated a sample that balances the parameter train set (NBS $n = 1080$; DHI $n = 82,124$). The performance of the models was measured by the quality of the corresponding synthetic samples compared to the minority cases of the validation set, as described in Section 4.3. The quality was determined by comparing the marginal distributions of the continuous and categorical features using the inverted Kolmogorov-Smirnov Distance (KS D) test statistic and p-value of the Chi-Squared (CS) test, respectively. The average inverted KS D-statistic estimates 1 minus the mean maximum distance between the cumulative distribution function of the real and synthetic data over all continuous features. The average p-value of the CS-test indicates the average probability that a categorical column of the real and synthetic data was sampled from the same distribution. Both marginal distribution metrics are between 0 and 1, and higher values indicate a better performance. The similarity of the underlying structure was investigated using the log-cluster metric, and for NBS also the L2 norm of the pairwise correlation difference (PCD-L2). Higher values of these measures suggest differences in the underlying structure of the real and synthetic data. In order to reduce the variability of the log-cluster metric, the clustering analysis was performed 20 times and the log-cluster metric was computed for each iteration. It was not possible to determine pairwise correlations in the DHI dataset because it contained only one continuous variable. Furthermore, the loss on the parameter training set was calculated for the TVAE models as an additional performance measure. Finally, models with the optimal hyperparameter values were fitted on the model training set and used to generate synthetic minority class samples (NBS $n = 1612$; DHI $n = 122,601$).

Table 6.1.: Parameter values that were considered during tuning for the SMOTE-NC and TVAE model per dataset.

Model	**Parameter**	**NBS**	**DHI**
SMOTE-NC	`k_neighbors`	$\{5, 6, ..., 49, 50\}$	$\{5, 6, ..., 49, 50\}$
TVAE	`embedding_dim`	$\{8, 10, ..., 26, 28\}$	$\{4, 6, ..., 18, 20\}$
	`compress_dim/` `decompress_dim`	$\{(16, 16), (32, 32), ..., (160, 160), (192, 192)\}$	$\{(16, 16), (32, 32), ..., (160, 160), (192, 192)\}$
	`batch_size`	$\{20, 30, 40, 50\}$	$\{1000, 2000, 3000, 4000\}$

The SMOTE-NC models were constructed using the `imbalanced-learn` package [42]. There was one hyperparameter available for tuning: the number of nearest neighbours. Each integer between 5 and 50 was considered in order to investigate the performance of the models on a wide range of values (Table 6.1). The TVAE models were constructed with the `TVAESynthesizer` class from the `ctgan` python package [16]. The following parameters were considered for tuning: the structure of the encoder and decoder network, the dimensions of the latent space, batch size and number of training epochs. The possible parameter values for each dataset were selected based on the default values and the number of instances and features per dataset (Table 6.1). For simplicity, only neural networks with two fully connected hidden layers with an equal number of neurons per layer were considered for the encoder. Since the decoder structure mirrors the encoder, it is assumed to have the same structure as the encoder. For both datasets, 10 different network structures were considered, of which the smallest had 16 neurons per layer and the largest 192. Since the latent space has a lower dimension than the original dataset, 11 and 9 integer values, smaller than the number of features, were considered for NBS and DHI, respectively. The batch size was chosen such that it was possible to make at least 5 and not more than 25 batches, in order to keep the calculations computationally feasible.

6.3. Model training and evaluation

Five types of classification models were implemented per dataset: Random Forest, XGBoost with cross-entropy loss, XGBoost with weighted cross-entropy loss, XG-Boost with focal loss and XGBoost with weighted focal loss. Each model is trained and evaluated on three different datasets: model train set, model train set plus SMOTE-NC generated sample (SMOTE-NC model train set) and model train set plus TVAE generated sample (TVAE model train set). First, the hyperparameters of each model were tuned using randomised search 5 fold cross validation on the respective training sets. The F1 score was used as the scoring method for the parameter tuning. Second, classifiers were trained with the optimal parameter values. The performance of the fitted models was evaluated on the test set using the F1 score and F5 score, which is equivalent to the F-beta score for beta equal to 5. McNemar's test was performed to compare the performance of each pair of predictive models, which resulted in 105 tests per problem [43]. Therefore, the p-values were adjusted for multiple testing using Benjamini-Yekutieli correction (FDR= 5%), because independence of the p-values can not be assumed [44].

Random Forest (RF) models were constructed with the `RandomForestClassifier` class from the `sklearn` package [37]. The following hyperparameters were considered for tuning: the number of trees, maximum depth of the individual trees, maximum number of features considered to make a split, and the fraction of data drawn for the bootstrap sample. The XGBoost models with cross-entropy loss (XGB-CE) were build using the `XGBClassifier` class from the `xgboost` package [30]. The customised `imbalanced-xgboost` package (Chapter 5) was used to constructed the XGBoost models with weighted cross-entropy loss (XGB-WCE), focal loss (XGB-F) and weighted focal loss (XGB-WF). The following hyperparameters were optimised for all XGBoost models: learning rate, the number of trees, maximum depth of the individual trees, and maximum number of features considered to make a split. For the cost-sensitive XGBoost variations the weight parameter α and/or focal parameter γ was also included. The hyperparameters and corresponding values for each model are described in Table 6.2. The values were chosen based on existing guidelines, model documentation, and the characteristics of the datasets [20, 30, 37, 45].

Table 6.2.: Parameter values that were considered during tuning for each classifier per dataset.

Model	**Parameter**	**NBS**	**DHI**
RF	`n_estimators`	$\{100, 200, ..., 400, 500\}$	$\{100, 200, ..., 400, 500\}$
	`max_depth`	$\{2, 3, ..., 14, 15\}$	$\{2, 3, ..., 14, 15\}$
	`max_features`	$\{2, 3, ..., 28, 29\}$	$\{2, 3, ..., 20, 21\}$
	`max_samples`	$[0.5, 1]$	$[0.5, 1]$
XGB-CE	`learning_rate`	$[0.01, 0.4]$	$[0.01, 0.4]$
	`num_round`	$\{10, 15, ..., 45, 50\}$	$\{10, 15, ..., 45, 50\}$
	`max_depth`	$\{2, 3, ..., 14, 15\}$	$\{2, 3, ..., 14, 15\}$
	`subsample`	$[0.5, 1]$	$[0.5, 1]$
XGB-WCE	`learning_rate`	$[0.01, 0.4]$	$[0.01, 0.4]$
	`num_round`	$\{10, 15, ..., 45, 50\}$	$\{10, 15, ..., 45, 50\}$
	`max_depth`	$\{2, 3, ..., 14, 15\}$	$\{2, 3, ..., 14, 15\}$
	`subsample`	$[0.5, 1]$	$[0.5, 1]$
	`imbalance-alpha`	$\{1.5, 2.0, ..., 3.5, 4.0\}$	$\{1.5, 2.0, ..., 3.5, 4.0\}$
XGB-F	`learning_rate`	$[0.01, 0.4]$	$[0.01, 0.4]$
	`num_round`	$\{10, 15, ..., 45, 50\}$	$\{10, 15, ..., 45, 50\}$
	`max_depth`	$\{2, 3, ..., 14, 15\}$	$\{2, 3, ..., 14, 15\}$
	`subsample`	$[0.5, 1]$	$[0.5, 1]$
	`focal_gamma`	$\{1.0, 1.5, 2.0, 2.5, 3.0\}$	$\{1.0, 1.5, 2.0, 2.5, 3.0\}$
XGB-WF	`learning_rate`	$[0.01, 0.4]$	$[0.01, 0.4]$
	`num_round`	$\{10, 15, ..., 45, 50\}$	$\{10, 15, ..., 45, 50\}$
	`max_depth`	$\{2, 3, ..., 14, 15\}$	$\{2, 3, ..., 14, 15\}$
	`subsample`	$[0.5, 1]$	$[0.5, 1]$
	`imbalance-alpha`	$\{1.5, 2.0, ..., 3.5, 4.0\}$	$\{1.5, 2.0, ..., 3.5, 4.0\}$
	`focal_gamma`	$\{1.0, 1.5, 2.0, 2.5, 3.0\}$	$\{1.0, 1.5, 2.0, 2.5, 3.0\}$

7. Data

The characteristics of the two datasets that were used to compare the performance of the generative models and cost-sensitive learning algorithms are described in this chapter.

7.1. Newborn Screening

The Newborn Screening (NBS) dataset was constructed from two cohorts of healthy newborns and referrals for possible congenital hypothyroidism (CH), for which measurements were available. The steps that were performed to create the NBS dataset from these two cohorts are described in Appendix A. The NBS dataset contains 30 variables regarding the sex, birth weight, pregnancy duration and CH status of the newborns, and measurements of thyroid hormones (T4, T4 sd, TSH, TBG, T4/TBG ratio), various acylcarnitines (C0, C2, C5, C5DC, C5OH, C6, C8, C10, C10:1, C14, C14:1, C14:2, C16, C16:1, C16OH, C18:1OH) and amino-acid markers (leucine, phenylaline, succinylacetone, tyrosine, valine). The NBS dataset was used to predict the presence of CH using the remaining 29 features. During manual inspection of the distributions of the continuous variables, one outlier was detected and set to missing (C6 = 1.19). A summary of the variables contained in the dataset is shown in Table 7.1.

The NBS dataset contains 3435 newborns, of which 515 (15.0%) were diagnosed with congenital hypothyroidism. The data consists of 28 continuous and 2 binary features, of which one is the target variable. It should be noted that some of the continuous variables in the dataset have relatively low cardinality. Of the 28 continuous variables, 12 have less than 50 unique values in 3436 instances and 5 even less than 10 unique values. The latter were all acylcarnitine measurements that assumed values between 0 and 1. For these variables, the low number of unique values is explained by the fact that the measurement technique can only determine these acylcarnitines to one decimal place. The vast majority of the features do not contain more than 0.5% missing values, with the exception of TBG (1.4%), T4/TBG ratio (1.4%), TSH (2.1%) and succinylacetone (8.1%).

Appendix B.1 contains the distribution plots of the CH cases versus healthy con-

Table 7.1.: The variable type (B: binary, C: continuous), number of unique values, percentage of missing values, minimum, maximum, mean and standard deviation of the 30 variables in the NBS dataset.

Variable	**Type**	**Unique**	**Missing**	**Min**	**Max**	**Mean**	**S.D.**
sex	B	2	0.5 %	0	1	0.57	-
birthweight	C	997	0.0 %	650	6020	3336	600
pregnancy_dur	C	92	0.0 %	176	298	275	14
CH	B	2	0.0 %	0	1	0.15	-
T4	C	132	0.2 %	5	148	57	32
T4_sd	C	77	0.0 %	-4.9	3.8	-1.5	1.8
TSH	C	142	2.1 %	0.2	120.0	13.4	33.8
TBG	C	312	1.4 %	20	387	164	65
T4TBG_ratio	C	2334	1.4 %	2.8	105	25.7	14.3
c0	C	396	0.2 %	2.0	66.0	18.6	8.1
c2	C	56	0.1 %	3	84	16	7
c5	C	4	0.1 %	0.1	0.4	0.12	0.04
c5dc	C	33	0.1 %	0.01	0.60	0.06	0.04
c5oh	C	5	0.1 %	0.1	0.5	0.17	0.06
c6	C	18	0.0 %	0.00	0.48	0.03	0.02
c8	C	22	0.1 %	0.02	0.41	0.04	0.02
c10	C	23	0.1 %	0.02	0.55	0.05	0.03
c101	C	34	0.0 %	0.00	0.52	0.04	0.03
c14	C	48	0.1 %	0.00	0.68	0.16	0.07
c141	C	23	0.1 %	0.02	0.34	0.06	0.03
c142	C	9	0.1 %	0.00	0.11	0.02	0.00
c16	C	79	0.3 %	0.02	8.5	2.48	1.15
c161	C	50	0.1 %	0.02	0.59	0.14	0.08
c16oh	C	8	0.1 %	0.01	0.13	0.01	0.01
c181oh	C	9	0.1 %	0.00	0.10	0.02	0.01
leu	C	257	0.1 %	20	424	151	45
phe	C	128	0.1 %	16	341	49	20
sa	C	106	8.1 %	0.01	1.18	0.25	0.21
tyr	C	244	0.2 %	2	649	86	46
val	C	254	0.1 %	18	401	136	44

trols for the remaining 29 variables. The five features regarding the thyroid hormones show most distinct difference in distribution between the patients and the controls (Figure B.2). This is not surprising, since congenital hypothyroidism is a disorder of the thyroid gland. Consequently, these variables have bi-modal distributions in the study population. The five amino-acid markers have right-skewed distributions, which is most severe for succinylacetone (sa) (Figure B.3). There are small differences in distribution between the cases and controls for succinylacetone, tyrosine and phenylaline. Finally, the sixteen acylcarnitines are right-skewed, and do not indicate substantial differences between the CH and control class (Figure B.4). For the acylcarnitine variables with very low cardinality (C5, C5OH, C14:2, C16OH, C18:1OH), there are one or two dominant classes.

7.2. Diabetes Health Indicators

Diabetes Health Indicators (DHI) is a public dataset that can be used to answer research questions about (pre)diabetes using machine learning, and is available on Kaggle [46]. It was derived from the Behavior Risk Factor Surveillance System (BRFSS), a telephone survey that is performed annually by the Centers for Disease Control and Prevention (CDC) in the United States. The data ($n = 253,680$) consists of various sociodemographic (sex, age, education) and lifestyle variables (body mass index (BMI), smoking, physical activity, fruit consumption, vegetable consumption, heavy alcohol consumption), information about the financial situation (income, health care coverage, not able to afford a doctors visit), medical history (diabetes, high blood pressure, high cholesterol, cholesterol check in the last five years, stroke, coronary heart disease or myocardial infarction, difficulty walking), days of poor mental or physical health in the past 30 days and self-reported general health score. The Diabetes Health Indicator dataset was used to predict the presence of diabetes using the other 21 features. The data was manually inspected for outliers, and none were detected. Table 7.2 contains a summary of the variables in the DHI dataset.

The data consists of $253,680$ subjects, including $35,346$ (13.9%) diabetes patients. There are 1 continuous and 20 categorical features, of which 14 binary and 6 ordinal. Three of the ordinal variables (age, education, income) are binned continuous variables. Notice that there are several binary variables with one dominant class: for 6 variables the dominant class occurs in more than 90% of the subjects. The dataset – as obtained from Kaggle – did not obtain any missing values.

The distributions of the features grouped by the target variable *Diabetes_binary* are displayed in Appendix B.2. The diabetes patients were older, had a lower education and lower income than the healthy controls (Figure B.5). Moreover, more than one third of the control group was in the highest income class, while for the diabetes patients this was only one fifth. The distribution of the only continuous variable (BMI) was right-skewed, and there is a slight shift in distribution between the patient and control class (Figure B.6). Moreover, the diabetes patients smoked more often and had less physical activity. The distribution plots suggest that

Table 7.2.: The variable type (B: binary, C: continuous, O: ordinal), number of unique values, percentage of missing values, minimum, maximum, mean and standard deviation of the 30 variables in the DHI dataset.

Variable	**Type**	**Unique**	**Missing**	**Min**	**Max**	**Mean**	**S.D.**
Diabetes_binary	B	2	0.0 %	0	1	0.14	-
HighBP	B	2	0.0 %	0	1	0.43	-
HighChol	B	2	0.0 %	0	1	0.42	-
CholCheck	B	2	0.0 %	0	1	0.96	-
BMI	C	84	0.0 %	12	98	28.4	6.6
Smoker	B	2	0.0 %	0	1	0.44	-
Stroke	B	2	0.0 %	0	1	0.04	-
HeartDiseaseorAttack	B	2	0.0 %	0	1	0.09	-
PhysActivity	B	2	0.0 %	0	1	0.76	-
Fruits	B	2	0.0 %	0	1	0.63	-
Veggies	B	2	0.0 %	0	1	0.81	-
HvyAlcoholConsump	B	2	0.0 %	0	1	0.06	-
AnyHealthcare	B	2	0.0 %	0	1	0.95	-
NoDocbcCost	B	2	0.0 %	0	1	0.08	-
GenHlth	O	5	0.0 %	1	5	2.51	-
MentHlth	O	31	0.0 %	0	30	3.18	-
PhysHlth	O	31	0.0 %	0	30	4.24	-
DiffWalk	B	2	0.0 %	0	1	0.17	-
Sex	B	2	0.0 %	0	1	0.44	-
Age	O	13	0.0 %	1	13	8.03	-
Education	O	6	0.0 %	1	6	5.05	-
Income	O	8	0.0 %	1	8	6.05	-

diabetes patients had more often high blood pressure, high cholesterol, stroke, heart disease or heart attack and difficulty walking than the control group (Figure B.7). Furthermore, there were very few diabetes patients that did not have a cholesterol check in the last five years. The diabetes patients report poorer general, physical and mental health than the healthy controls (Figure B.8). For days of poor physical or mental health per month, zero is the most dominant class in both groups.

8. Results

The results from the experiment are described seperately for the two disease classification problems. First, the quality of the two synthetic samples is assessed and second, the performance of the predictive models is evaluated.

8.1. Newborn Screening

The newborn screening (NBS) dataset was split in a stratified train and test set consisting of 2302 and 1134 newborns, respectively. The missing values were imputed for both sets seperately. The NBS dataset was not only provided for the purpose of this thesis, but in the context of a specific use case. A detailed description of this use case and its results are given in Appendix D.

8.1.1. Data generation

Two synthetic samples were generated in order to balance the training data, where one was created with SMOTE-NC and the other with TVAE. Each synthetic sample consisted of 1612 CH patients. In order to choose appropriate values of the hyperparameters, the performance of the generative models was evaluated for different parameter values (Appendix C). For SMOTE-NC, the number of nearest neighbours was set to 20. For TVAE, the latent dimension was 14, the encoder and decoder network had 2 hidden layers with 80 neurons per layer and the batch size was set to 30. The quality of the generated samples compared to the CH cases in the test set was evaluated using the inverted KS D-statistic, p-value of the CS test, L2 norm of the pairwise correlation difference and the log-cluster metric (Table 8.1). The Mann-Whitney U test was used to investigate whether the KS D-statistic of the individual features and the iterations of the log-cluster metric differed significantly between the SMOTE-NC and TVAE sample. Under the null hypothesis, the measurements do not differ between the two synthetic samples. It was not possible to perform the Mann-Whitney U test for the other two quality metrics because these consisted of only one measurement. There was no significant difference in average inverted KS D-statistic between the two samples ($p = 0.851$). The inverted KS D-statistic was approximately 0.9, which suggests that the average maximum difference between the cumulative distribution functions of the real and synthetic data was equal to 0.1. The CS test p-value indicates that the probability that the categorical variable of the real CH patients and the synthetic data were sampled from the same distribution is equal to 0.863 and 0.996 for the SMOTE-NC and TVAE sample, respectively. The L2 norm of the pairwise correlation differences suggests that the correlation structure of the continuous variables from the SMOTE-NC sample was more similar to the test set than the TVAE sample. The log-cluster metric was significantly lower for TVAE ($p = 6.80\text{e}{-08}$), suggesting that this sample was better able to capture the underlying clusters in the data. According to the quality metrics, the SMOTE-NC sample was better able to capture underlying structure of the patient data in terms of correlation, while the TVAE sample had a better approximation of the distribution of the categorical variable and the cluster structure. Both synthetic samples performed well on reconstructing the marginal distributions of the continuous variables.

Table 8.1.: The inverted KS D-statistic, CS test p-value, L2 norm of the pairwise correlation differences and mean log-cluster metric of the SMOTE-NC and TVAE generated patient data in the NBS dataset. The standard deviation of the metrics are given between the brackets.

Method	**invKS D-statistic**	**CS p-value**	**PCD-L2**	**Log cluster**
SMOTE-NC	0.901 (0.045)	0.863 (−)	3.617 (−)	−4.598 (0.014)
TVAE	0.896 (0.056)	0.996 (−)	6.715 (−)	−5.549 (0.465)

The inverted KS D-statistic and CS test p-value of the 29 explanatory variables are shown in Figure 8.1. For a number of variables, the distributions in the real patient data and two synthetic samples are discussed in more detail. Both synthetic samples obtained the lowest inverted KS D-statistic on TSH. Further investigation of the distributions of TSH showed that the shape of the real distribution had two major peaks on the far right and left of the distribution. The low score is explained by the underestimation of these peaks in the distributions of the synthetic data (Figure 8.2). The SMOTE-NC model generated values in between the peaks due to the nature of the nearest neighbours algorithm, while TVAE modelled a multi-modal Gaussian distribution around the peaks. For T4, the inverted KS D-statistic was similar for the two samples. However, the resulting shape of the generated distributions depended on the generative models (Figure 8.2). On one hand, the k-nearest neighbour approach caused SMOTE-NC to generate values in between frequently occurring values. This is apparent from the overestimation of the density between the first two peaks. On the other hand, TVAE estimated the number of modes of the distribution and modelled the variable with a (multi-modal) Gaussian distribution. However, the third mode was not detected. For tyrosine, TVAE had a lower score than SMOTE-NC (Figure 8.2).

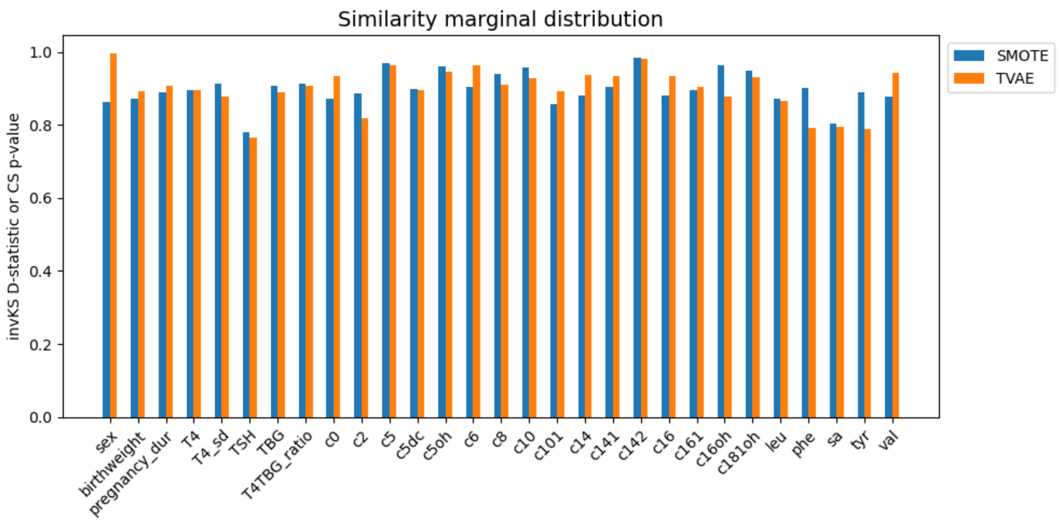


Figure 8.1.: Inverted KS D-statistic or CS test p-value of the two synthetic samples for each of the individual continuous respectively categorical features in the NBS dataset.

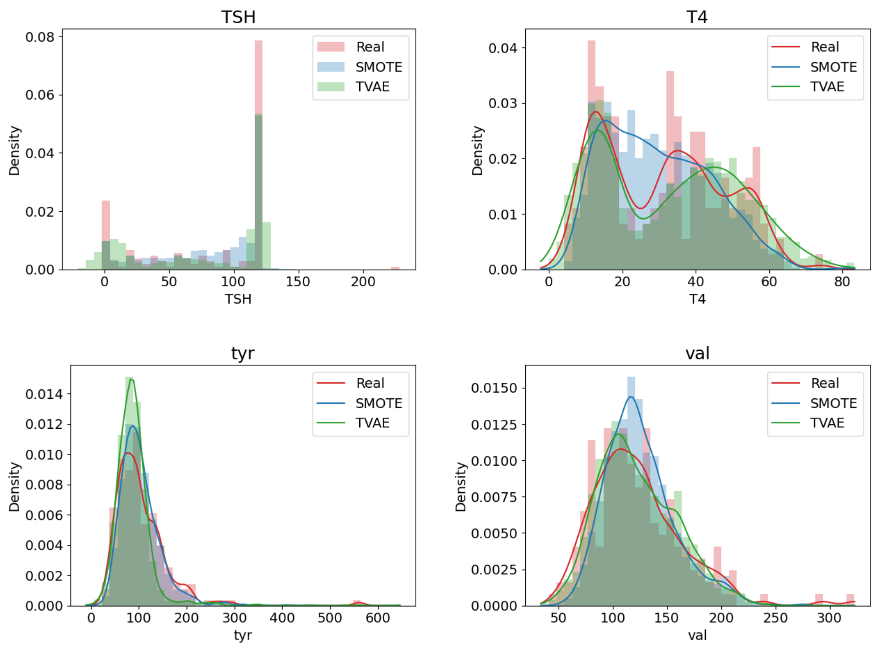


Figure 8.2.: Empirical distribution of TSH (top left), T4 (top right), tyrosine (bottom left) and valine (bottom right) in the real patient data, SMOTE-NC sample and TVAE sample in the NBS dataset.

The lower score is caused by an overestimation of the peak at the expense of underestimation of the right tail. In contrast, TVAE had a higher score than SMOTE-NC for valine, because it was better able to capture the shape of the peak and the left tail of the real distribution (Figure 8.2).

8.1.2. CH prediction

The goal was to predict the binary target variable CH by the remaining 29 features. Five classifiers were trained on three different datasets: the original data, the SMOTE-NC balanced sample and TVAE balanced sample. The optimal hyperparameters were selected using randomised search 5-fold cross validation on the train set (Table E.1). The performance of each predictive model in terms of the F1 and F5 score is shown in Figure 8.3 and Table 8.2, where the latter also contains the average rank of each model. Notice that all predictive models obtained high F1 and F5 scores on the NBS dataset and the differences between the models were relatively small. Balancing the training data with TVAE or SMOTE-NC increased the F1 score and F5 score of the classifiers compared to no oversampling strategy. The average rank of the classifiers combined with TVAE (4.7) was higher than the rank of SMOTE-NC (6.5). McNemar's test was performed for each pair of predictive models in order to determine whether one model has a higher predictive performance than the other. The exact p-values were used instead of the approximated p-values, because the high F1 score suggests a low number of false-positive and false-negative results for each model. The McNemar's tests showed no significant differences in performance between any of the predictive models. This may be related to the high predictive performance on the NBS dataset in general.

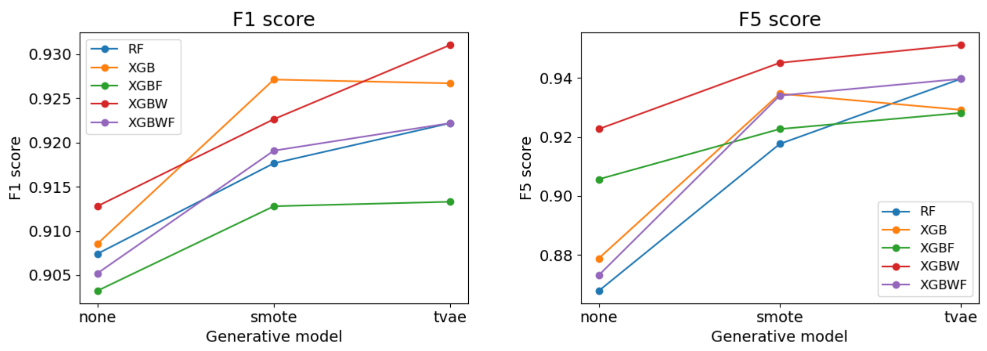


Figure 8.3.: The F1 score and F5 score of the 15 predictive models for the NBS dataset.

Table 8.2.: The F1 score, F5 score and average rank based on the two scores of the 15 predictive models on the NBS dataset.

Classifier	**Method**	**F1 score**	**F5 score**	**Rank**
RF	None	0.907	0.868	14.0
	SMOTE-NC	0.918	0.918	9.5
	TVAE	0.922	0.940	4.5
XGB-CE	None	0.909	0.879	12.5
	SMOTE-NC	0.927	0.935	3.5
	TVAE	0.927	0.929	5.0
XGB-F	None	0.903	0.906	13.5
	SMOTE-NC	0.913	0.923	10.0
	TVAE	0.913	0.928	8.5
XGB-W	None	0.913	0.923	10.0
	SMOTE-NC	0.923	0.945	3.0
	TVAE	0.931	0.951	**1.0**
XGB-WF	None	0.905	0.873	14.0
	SMOTE-NC	0.919	0.934	6.5
	TVAE	0.922	0.940	4.5

The XGBoost model with weighted cross-entropy loss in combination with TVAE had the best rank, with the highest F1 score (0.931) and highest F5 score (0.951). The predictions of this model on the test set were investigated in more detail. The confusion matrix on the test set is shown in Figure 8.4. The model had a positive predictive value (PPV) of 0.91, in which case 91% of the newborns that were positively predicted were true CH patients. The true positive rate (TPR) and false positive rate (FPR) were equal to 0.95 and 0.98, respectively. These metrics suggest that the model performed well at predicting the positive and negative class, with little false-positive or false-negative results.The feature importance on the test set is shown in Figure 8.5. The most important feature for the prediction of CH was TSH, followed by the T4/TBG ratio and T4 SD.

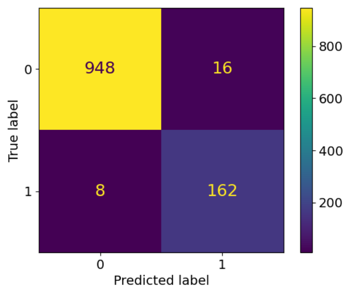


Figure 8.4.: Confusion matrix.

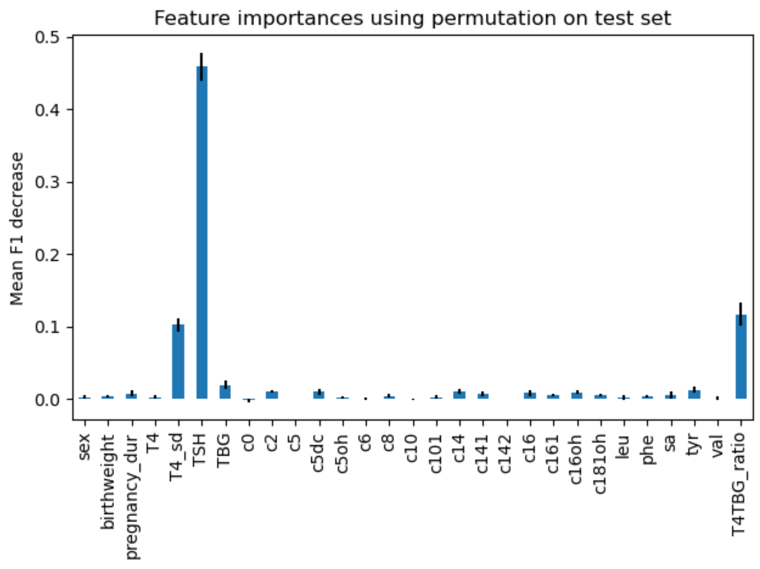


Figure 8.5.: Permutation feature importance on the test set of the XGB-W model trained on the TVAE sample.

8.2. Diabetes Health Indicators

The Diabetes Health Indicators (DHI) dataset was split in a stratified train and test set which included $157,294$ and $83,715$ subjects, respectively. Since the data did not contain missing values, imputation was not necessary.

8.2.1. Data generation

Two samples of synthetic diabetes patient data ($n = 122,601$) were generated, one with SMOTE-NC and the other with TVAE. The optimal hyperparameters of the generative models were selected beforehand (Appendix C). For SMOTE-NC, the number of neighbours was set to 10, and for TVAE, the network structure was $(96, 96)$, latent dimension 12 and batch size 2000. Table 8.3 shows the quality of the two synthetic samples in terms of the inverted KS D-statistic, CS test p-value, and log-cluster metric. Again, the Mann-Whitney U test was computed to compare the CS test p-value and log-cluster metric in the two synthetic samples. Since there is only one continuous variable, it is not possible to calculate the pairwise correlation difference for the DHI dataset. The inverted KS D-statistic was approximately 0.99 for SMOTE-NC and 0.92 for TVAE, which indicates that the average maximum difference between the cumulative distribution functions of the real and synthetic data was equal to 0.01 and 0.08, respectively. According to the CS test p-value, the probability that the categorical variables of the real and synthetic data were sampled from the same distribution was 0.96 for SMOTE-NC and 0.94 for TVAE. The CS test p-value and log-cluster metric did not differ significantly between the two synthetic samples ($p = 0.552$; $p = 0.229$). The quality metrics suggest that both samples well at estimating the marginal distributions of the continuous variable and the categorical variables.

Table 8.3.: The inverted KS D-statistic, CS test p-value, L2 norm of the pairwise correlation differences and mean log-cluster metric of the SMOTE-NC and TVAE generated patient data in the NBS dataset. The standard deviation of the metrics are given between the brackets.

Method	**invKS D-statistic**	**CS p-value**	**Log cluster**
SMOTE-NC	0.986 (−)	0.956 (0.051)	−4.675 (0.349)
TVAE	0.924 (−)	0.942 (0.072)	−5.015 (0.295)

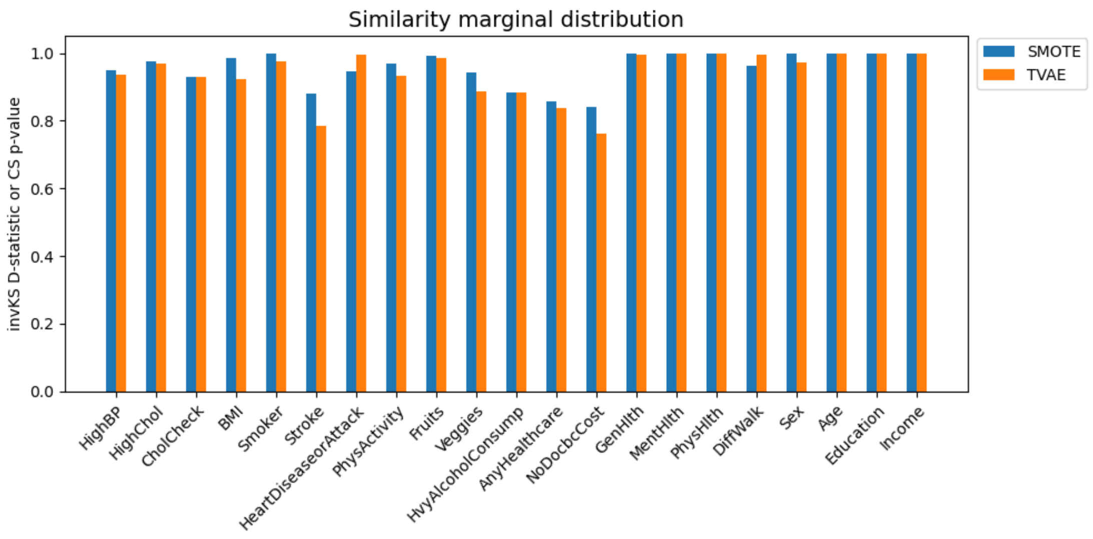


Figure 8.6.: Inverted KS D-statistic or CS test p-value of the two synthetic samples for each of the individual continuous respectively categorical features in the DHI dataset.

The inverted KS D-statistic or CS test p-value for each of the individual explanatory features in the synthetic samples is shown in Figure 8.6. The synthetic data, and especially TVAE, showed the lowest CS p-value on the binary variables with an highly imbalanced class distribution, such as high blood pressure, stroke and health care coverage. As an example, the distribution of health care coverage of the real data and two synthetic samples is shown in Figure 8.7. The generated samples consist almost exclusively of majority class cases, while this is not the case in the real dataset. Furthermore, note that both samples obtained a very high CS p-value for the categorical variables with more than two classes: age, education, income, general health, mental health and physical health. However, further inspection of the distribution of these variables shows that there were substantial differences between the real data and synthetic samples. The generated sample, in particular TVAE, shifted the density of the distribution towards the frequently occurring classes. One example is the variable age (Figure 8.7). In the TVAE sample, more than 95% of the subjects belong to the three most frequent classes (8, 9, 10) and none of the cases were in class 5, 6, 12 or 13.

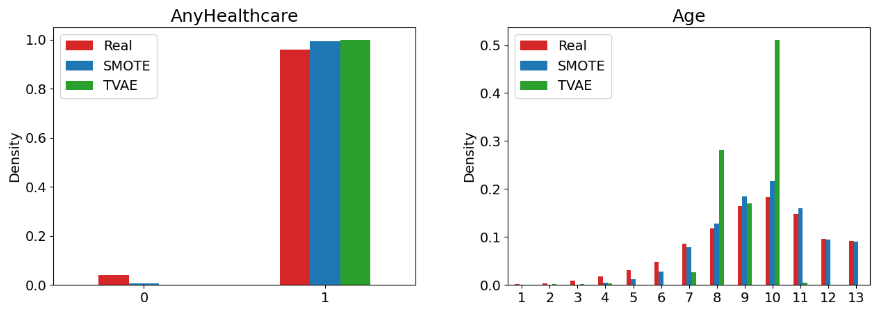


Figure 8.7.: Empirical distribution of health care coverage (left) and age (right) in the real patient data, SMOTE-NC sample and TVAE sample in the DHI dataset.

8.2.2. Diabetes prediction

Five classification models were trained on three different datasets, aiming to predict the binary variable diabetes using the remaining 20 features. The optimal hyper-parameters were selected using randomised search 5-fold cross validation, and are shown in Table E.2. The performance of each predictive model is shown in Figure 8.8 and Table 8.4. The three cost-sensitive models trained on the original data obtained the highest scores, and the XGBoost model with weighted cross-entropy loss had the best rank. The generative models improved the F1 and F5 score of the two standard classifiers (RF, XGB). Next, McNemar's test was performed to compare the performance of each pair of predictive models. The test results indicate that the performance of almost all predictive models is different from each other, except the following. XGBoost with weighted cross-entropy loss without oversampling did not significantly differ from XGB-F without oversampling and XGBW with TVAE. The performance of XGB-WF on the original dataset was not different from XGB with TVAE and RF with TVAE. Finally, XGB-F without oversampling did not have a different performance from XGB-W with TVAE.

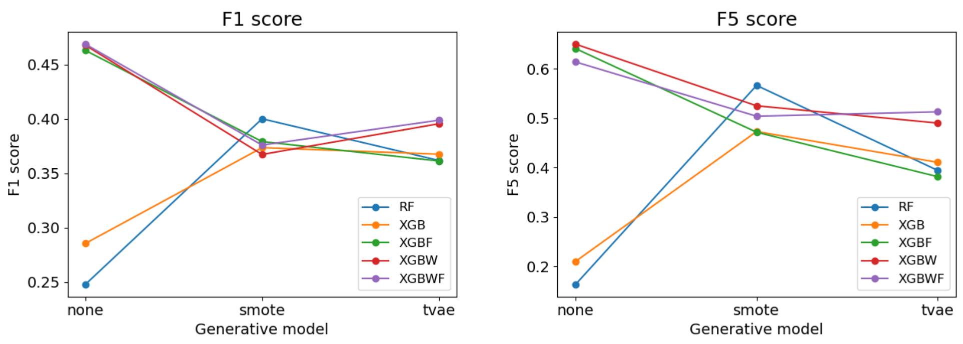


Figure 8.8.: The F1 score and F5 score of the 15 predictive models on the DHI dataset.

Table 8.4.: The F1 score, F5 score and average rank based on the two scores of the 15 predictive models on the DHI dataset.

Classifier	**Method**	**F1 score**	**F5 score**	**Rank**
RF	None	0.248	0.164	15.0
	SMOTE-NC	0.400	0.567	4.0
	TVAE	0.362	0.395	12.0
XGB	None	0.285	0.210	14.0
	SMOTE-NC	0.373	0.473	9.0
	TVAE	0.367	0.411	10.5
XGB-F	None	0.463	0.641	2.5
	SMOTE-NC	0.379	0.472	8.5
	TVAE	0.361	0.382	13.0
XGB-W	None	0.468	0.650	**1.5**
	SMOTE-NC	0.367	0.525	8.0
	TVAE	0.395	0.490	7.0
XGB-WF	None	0.469	0.614	2.0
	SMOTE-NC	0.376	0.504	7.5
	TVAE	0.399	0.513	5.5

The XGBoost model with weighted cross-entropy loss fitted on the original train set obtained the best rank, with the second highest F1 score (0.468) and highest F5 score (0.650). The predictions made by this model on the test set are discussed in more detail. The confusion matrix is shown in Figure 8.9. The model had an accuracy of 0.79 and a positive predictive value of 0.36. The true positive rate and false positive rate was equal to 0.67 and 0.19, respectively. The model obtains many false-positives on the test set, since only 36% of the samples that were predicted as positive were true diabetes patients. Furthermore, the model also makes a substantial number of false-negatives, since only two third of the true positives were predicted correctly. The feature importance on the test set of the XGB-W model without oversampling is shown in Figure 8.10. The most important features for the prediction of diabetes were the general health score and BMI, followed by age, high blood pressure and high cholesterol.

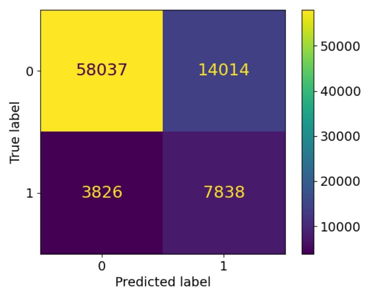


Figure 8.9.: Confusion matrix.

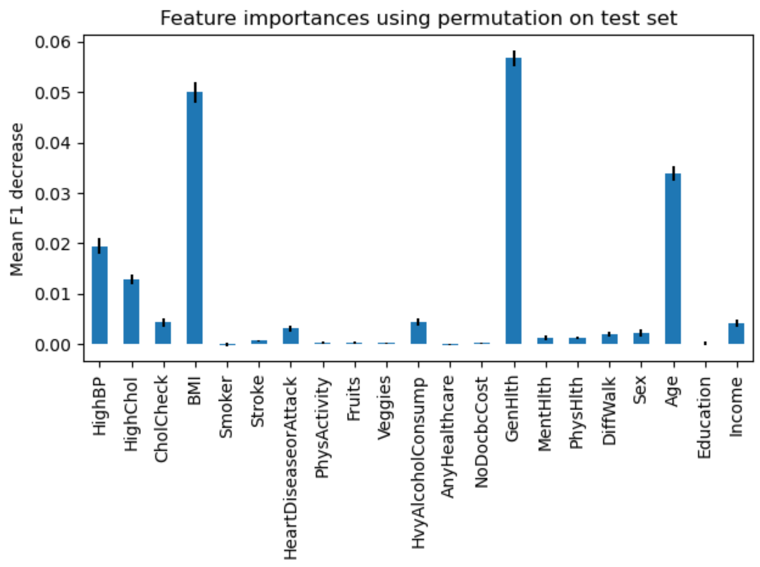


Figure 8.10.: Permutation feature importance on the test set of the XGB-W model trained on the original data.

9. Discussion

The goal of this thesis was to answer the following research question: which imbalanced learning method or combination of methods yields the best performance on binary disease classification problems? In order to answer this question, five classifiers combined with three oversampling strategies were evaluated on two binary disease classification problems. The experiment showed that imbalanced learning methods improved the predictive quality for both problems. In the NBS dataset, the generative models – in particular TVAE – obtained the best results, while the cost-sensitive models only improved in combination with a generative model. On the other hand, the cost-sensitive models trained on the original data had the best performance on the DHI dataset. The generative models only improved the performance of the standard classifiers. For both datasets, the classifier with the highest rank was the XGBoost model with weighted cross-entropy loss.

The performance of the standard classification models (RF, XGB) improved in combination with a generative model in both datasets, as expected from previous research [9, 10]. While both the F1 and F5 score improved, the increase was most pronounced for the F5 score. This indicates a major improvement in the classification of the patient class. In the NBS dataset, the generative models also improved the predictive performance of the cost-sensitive models. In particular, the model with the highest rank was a cost-sensitive model combined with TVAE. However, the cost-sensitive models performed worse in combination with any generative model on the DHI dataset. Further investigation showed differences between the scores obtained on the train and test set for classifiers that were trained on synthetic data, while these scores were similar for classifiers trained on the original data (Table E.1; Table E.2). Although this overfitting was observed in both datasets, the differences were more substantial for DHI. Moreover, the difference in feature importance on the train versus test set showed that the classifiers tend to overfit mostly on the high cardinality ordinal features (Table E.3; Table E.4). A possible explanation for this overfitting is a difference in the properties of the train and test set caused by the synthetic data. The results on the quality of the synthetic data for the DHI dataset showed that the generative models – in particular TVAE – did not accurately approximate the distribution of a number of variables (Section 8.2). Most of these features were high cardinality ordinal features or binary features with extremely imbalanced class distributions.

In the synthetic samples, the number of cases in the most frequent classes were overestimated, at the cost of the rare classes. It is likely that these shifts caused overfitting in the classifiers that were trained on the synthetic data.

It should be noted that the metric that was used to compare the similarity of the marginal distribution, the Chi-Squared (CS) test p-value, did not indicate that there were large shifts in the distributions of the synthetic data. The high cardinality ordinal features even obtained a perfect CS p-value, suggesting that the probability that the synthetic samples were from the same distribution as the real data is equal to 1. However, a visual comparison of the marginal distributions makes clear that these distributions are not alike. This behaviour is problematic and questions whether this is an accurate measure to evaluate the quality of the marginal distribution of ordinal features with more than two classes. The CS test p-value was computed using the `CSTest` function from the `SDV` python package. Further investigation of the corresponding source code of this function revealed that the input of the CS test are the observed and expected ratio of each category. However, it is a known assumption of the Chi-Squared test that the input is measured in frequencies [47]. Moreover, small values of the observed and expected frequencies may lead to unreliable test results. Therefore, the `CSTest` function should be used with caution, especially for high cardinality features where the observed and expected ratios can become very small. More research is necessary to determine whether this is an appropriate measure for the quality of synthetic categorical variables.

The predictive performance on the NBS dataset was much larger than the DHI dataset. For NBS, the F1 score ranged between 0.903 and 0.931 while for DHI it was between 0.248 and 0.468. This difference can be explained by the nature of the datasets. NBS contains biological measurements, some of which are directly linked to the pathophysiology of congenital hypothyroidism. As such, these models were able to accurately predict the presence of CH. The DHI dataset consists of various features regarding sociodemographic information, lifestyle and medical history, which are not directly associated with the pathophysiology of diabetes. By design, the models predicting diabetes in this dataset have much lower explanatory value because the relationship with the outcome is weaker. It is possible that this difference in design may have contributed to the varying results of the experiment.

The outcome of the experiment suggests that the answer to the research question is not straightforward, but depends on the properties of the data. The NBS dataset, which had a moderate sample size and consisted almost exclusively of continuous variables, obtained the highest F1 and F5 score with a hybrid approach of a generative model and cost-sensitive learner. The DHI dataset, which had a large sample size and consisted of many binary and ordinal variables, yielded the best performance with a cost-sensitive classifier without oversampling. Among the two generative models, there was not one superior approach. On the NBS dataset, both synthetic samples increased the F1 and F5 score of the classifiers. On the DHI dataset, the generative models (especially TVAE) did not accurately estimate the categorical variables with imbalanced class distributions, which resulted in poorer performance when combined with the cost-sensitive classifiers.

A strength of this research is that this is – to the best of our knowledge – the first study that evaluates multiple combinations of generative models and cost-sensitive classifiers on binary disease classification problems. Furthermore, this research has investigated the performance in terms of the F1 score and F5 score, thereby taking into consideration the possible unequal costs of misclassification. Limitations include the fact that only two datasets were considered in the experiment and that only cost-sensitive variants of the XGBoost algorithm were considered. The following recommendations for future research are proposed. First of all, repetition of this research on a wider range of medical datasets could determine the most suitable approach given the properties of the data. Second, the inclusion of other cost-sensitive classifiers could be considered. Third, more research is necessary to determine whether TVAE is appropriate to generate data that contains high cardinality ordinal variables.

In conclusion, this research shows that imbalanced learning methods improve the performance of disease classification models. Furthermore, hybrid procedures that combine two imbalanced learning methods have also shown promising results, but more research is necessary to determine for which datasets hybrid approaches are suitable.

Bibliography

[1] Tom M. Mitchell. *Machine Learning.* 1997. ISBN: 978-0-07-042807-2.

[2] Eric J. Topol. “High-performance medicine: the convergence of human and artificial intelligence”. In: *Nature Medicine* (2019), pp. 44–56. DOI: `10.1038/s41591-018-0300-7`.

[3] Andre Esteva et al. “A guide to deep learning in healthcare”. In: *Nature Medicine* (2019), pp. 24–29. DOI: `10.1038/s41591-018-0316-z`.

[4] Christopher J. Kelly et al. “Key challenges for delivering clinical impact with artificial intelligence”. In: *BMC Medicine* (2019), p. 195. DOI: `10.1186/s12916-019-1426-2`.

[5] Luis J. Mena and Jesus A. Gonzalez. “Machine Learning for Imbalanced Datasets: Application in Medical Diagnostic”. In: *Proceedings of the Nineteenth International Florida Artificial Intelligence Research Society Conference.* 2006, pp. 574–579.

[6] Nathalie Japkowicz. “The Class Imbalance Problem: Significance and Strategies”. In: *Proceedings of the 2000 International Conference on Artificial Intelligence ICAI* (2000), pp. 111–117.

[7] Yanmin Sun, Andrew K.C. Wong, and Mohamed S. Kamel. “Classification of imbalanced data: a review”. In: *International Journal of Pattern Recognition and Artificial Intelligence* 23 (2011). DOI: `10.1142/S0218001409007326`.

[8] Haibo He and Edwardo A. Garcia. “Learning from Imbalanced Data”. In: *IEEE Transactions on Knowledge and Data Engineering* (2009), pp. 1263–1284. DOI: `10.1109/TKDE.2008.239`.

[9] Sara Fotouhi, Shahrokh Asadi, and Michael W. Kattan. “A comprehensive data level analysis for cancer diagnosis on imbalanced data”. In: *Journal of Biomedical Informatics* (2019), p. 103089. DOI: `https://doi.org/10.1016/j.jbi.2018.12.003`.

[10] Fatihah Mohd et al. “Improving Accuracy of Imbalanced Clinical Data Classification Using Synthetic Minority Over-Sampling Technique”. In: (2019), pp. 99–110.

[11] Nitesh V. Chawla et al. “SMOTE: synthetic minority over-sampling technique”. In: *Journal of artificial intelligence research* (2002), pp. 321–357. DOI: 10.1613/jair.953.

[12] Diederik P. Kingma and Max Welling. “Auto-Encoding Variational Bayes”. In: 2014.

[13] Ian J. Goodfellow et al. “Generative Adversarial Nets”. In: *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*. 2014, pp. 2672–2680.

[14] Edward Choi et al. “Generating Multi-label Discrete Patient Records using Generative Adversarial Networks”. In: *Proceedings of the 2nd Machine Learning for Healthcare Conference* (2017), pp. 286–305.

[15] Justin Engelmann and Stefan Lessmann. “Conditional Wasserstein GAN-based oversampling of tabular data for imbalanced learning”. In: *Expert Systems with Applications* (2021), p. 114582. DOI: 10.1016/j.eswa.2021.114582.

[16] Zhongxian Xu and Zhiliang Wang. “A Risk Prediction Model for Type 2 Diabetes Based on Weighted Feature Selection of Random Forest and XGBoost Ensemble Classifier”. In: *2019 Eleventh International Conference on Advanced Computational Intelligence (ICACI)* (2019), pp. 278–283. DOI: 10.1109/ICACI.2019.8778622.

[17] Vivek Harsha Vardhan and Stanley Kok. “Synthetic Tabular Data Generation with Oblivious Variational Autoencoders: Alleviating the Paucity of Personal Tabular Data for Open Research”. In: (2020).

[18] Alberto Fernández et al. “Cost-Sensitive Learning”. In: *Learning from Imbalanced Data Sets*. 2018, pp. 63–78. DOI: 10.1007/978-3-319-98074-4_4.

[19] Charles X. Ling et al. “Decision Trees with Minimal Costs”. In: (2004), p. 69. DOI: 10.1145/1015330.1015369.

[20] Chen Wang, Chengyuan Deng, and Suzhen Wang. “Imbalance-XGBoost: leveraging weighted and focal losses for binary label-imbalanced classification with XGBoost”. In: *Pattern Recognition Letters* (2020), pp. 190–197. DOI: 10.1016/j.patrec.2020.05.035.

[21] Cristiano L. Castro and Antônio P. Braga. “Novel Cost-Sensitive Approach to Improve the Multilayer Perceptron Performance on Imbalanced Data”. In: *IEEE Transactions on Neural Networks and Learning Systems* 24.6 (2013), pp. 888–899. DOI: 10.1109/TNNLS.2013.2246188.

[22] Ibomoiye Domor Mienye and Yanxia Sun. “Performance analysis of cost-sensitive learning methods with application to imbalanced medical data”. In: *Informatics in Medicine Unlocked* (2021), p. 100690. DOI: `https://doi.org/10.1016/j.imu.2021.100690`.

[23] Badiuzzaman Pranto et al. “Prediction of diabetes using cost sensitive learning and oversampling techniques on Bangladeshi and Indian female patients”. In: 2020, pp. 1–6. DOI: `10.1109/ICITR51448.2020.9310892`.

[24] Candice Bentéjac, Anna Csörgo, and Gonzalo Martinez-Muñoz. “A Comparative Analysis of XGBoost”. In: *CoRR* (2019). DOI: `10.1007/s10462-020-09896-5`.

[25] Ravid Shwartz-Ziv and Amitai Armon. “Tabular data: Deep learning is not all you need”. In: *Information Fusion* (2022), pp. 84–90. DOI: `https://doi.org/10.1016/j.inffus.2021.11.011`.

[26] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning: data mining, inference and prediction.* 2nd ed. Springer, 2009.

[27] Leo Breiman. “Random Forests”. In: *Machine Learning* (2001), pp. 5–32. DOI: `10.1023/A:1010933404324`.

[28] Leo Breiman. “Bagging predictors”. In: *Machine Learning* (1996), pp. 123–140. DOI: `10.1007/BF00058655`.

[29] Jerome H. Friedman. “Greedy function approximation: A gradient boosting machine.” In: *The Annals of Statistics* (2001), pp. 1189–1232. DOI: `10.1214/aos/1013203451`.

[30] Tianqi Chen and Carlos Guestrin. “XGBoost: A Scalable Tree Boosting System”. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2016), pp. 785–794. DOI: `10.1145/2939672.2939785`.

[31] Sotiris Kotsiantis, D. Kanellopoulos, and P. Pintelas. “Handling imbalanced datasets: A review”. In: *GESTS International Transactions on Computer Science and Engineering* (2005), pp. 25–36.

[32] Haibo He et al. “ADASYN: Adaptive synthetic sampling approach for imbalanced learning”. In: (2008), pp. 1322–1328. DOI: `10.1109/IJCNN.2008.4633969`.

[33] Ofer Mendelevitch and Michael D. Lesh. “Fidelity and Privacy of Synthetic Medical Data”. In: *CoRR* (2021).

[34] Andre Goncalves et al. “Generation and evaluation of synthetic patient data”. In: *BMC Medical Research Methodology* (2020), p. 108. DOI: 10.1186/s12874-020-00977-1.

[35] Tsung-Yi Lin et al. “Focal Loss for Dense Object Detection”. In: *2017 IEEE International Conference on Computer Vision (ICCV)* (2017), pp. 2999–3007. DOI: 10.1109/TPAMI.2018.2858826.

[36] J.M.C. van Haeringen. *imbalanced-classification.* https://github.com/marijevhaeringen/imbalanced-classification.

[37] Fabian Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* (2011), pp. 2825–2830.

[38] *missingpy.*

[39] Daniel J. Stekhoven and Peter Bühlmann. “MissForest—non-parametric missing value imputation for mixed-type data”. In: *Bioinformatics* (2011), pp. 112–118. DOI: 10.1093/bioinformatics/btr597.

[40] Burim Ramosaj and Markus Pauly. “Predicting missing values: a comparative study on non-parametric approaches for imputation”. In: *Computational Statistics* (2019), pp. 1741–1764. DOI: 10.1007/s00180-019-00900-3.

[41] Fei Tang and Hemant Ishwaran. “Random Forest Missing Data Algorithms”. In: *Statistical analysis and data mining* (2017), pp. 363–377. DOI: 10.1002/sam.11348.

[42] Guillaume Lemaître, Fernando Nogueira, and Christos K. Aridas. “Imbalanced-learn: A Python Toolbox to Tackle the Curse of Imbalanced Datasets in Machine Learning”. In: *Journal of Machine Learning Research* (2017), pp. 1–5.

[43] Sebastian Raschka. “Model Evaluation, Model Selection, and Algorithm Selection in Machine Learning”. In: (2018). DOI: 10.48550/arxiv.1811.12808.

[44] Yoav Benjamini and Daniel Yekutieli. “The control of the false discovery rate in multiple testing under dependency”. In: *The Annals of Statistics* (2001), pp. 1165–1188. DOI: 10.1214/aos/1013699998.

[45] Prashant Banerjee. *A Guide on XGBoost hyperparameters tuning.* https://www.kaggle.com/code/prashant111/a-guide-on-xgboost-hyperparameters-tuning/notebook.

[46] Alex Teboul. *Diabetes Health Indicators Dataset.* 2021. URL: https://www.kaggle.com/alexteboul/diabetes-health-indicators-dataset.

[47] Mary L McHugh. “The chi-square test of independence”. In: *Biochemia medica* (2013), pp. 143–149. DOI: 10.11613/bm.2013.018.

[48] Caren I. Lanting et al. “Clinical effectiveness and cost-effectiveness of the use of the thyroxine/thyroxine-binding globulin ratio to detect congenital hypothyroidism of thyroidal and central origin in a neonatal screening program”. In: *Pediatrics* (2005), pp. 168–173. DOI: 10.1542/peds.2004-2162.

[49] Maynika V. Rastogi and Stephen H. Lafranchi. “Congenital hypothyroidism”. In: *Orphanet Journal of Rare Diseases* (2010), pp. 1–22. DOI: 10.1186/1750-1172-5-17.

[50] George Ford and Stephen H. LaFranchi. “Screening for congenital hypothyroidism: a worldwide view of strategies”. In: *Best Practise & Research Clinical Endocrinology & Metabolism* (2014), pp. 175–187. DOI: 10.1016/j.beem.2013.05.008.

[51] Kevin Stroek et al. “Critical evaluation of the newborn screening for congenital hypothyroidism in the Netherlands”. In: *European Journal of Endocrinology* (2020), pp. 265–273. DOI: 10.1530/EJE-19-1048.

[52] Mahmoud Al-Majdoub, Mikael Lantz, and Peter Spégel. “Treatment of Swedish Patients with Graves’ Hyperthyroidism Is Associated with Changes in Acylcarnitine Levels.” In: *Thyroid* (2017), pp. 1109–1117. DOI: 10.1089/thy.2017.0218.

[53] Chiaw-Ling Chng et al. “Physiological and Metabolic Changes During the Transition from Hyperthyroidism to Euthyroidism in Graves’ Disease”. In: *Thyroid* (2016), pp. 1422–1430. DOI: 10.1089/thy.2015.0602.

[54] Thomas Lange et al. “Comprehensive Metabolic Profiling Reveals a Lipid-Rich Fingerprint of Free Thyroxine Far Beyond Classic Parameters”. In: *The Journal of Clinical Endocrinology and Metabolism* (2018), pp. 2050–2060. DOI: 10.1210/jc.2018-00183.

[55] Carolin Jourdan et al. “Associations between thyroid hormones and serum metabolite profiles in an euthyroid population”. In: *Metabolomics* (2014), pp. 152–164. DOI: 10.1007/s11306-013-0563-4.

[56] Maik Pietzner et al. “Plasma proteome and metabolome characterization of an experimental human thyrotoxicosis model”. In: *BMC Medicine* (2017), pp. 1–18. DOI: 10.1186/s12916-016-0770-8.

Part III.

Appendix

A. Newborn screening dataset

For the purpose of this study, two cohorts (control and referred) were combined to create a sample that represents the newborn population. The control cohort ($n = 2262$) consists of T4, TSH, and TBG measurements performed in the neonatal screening laboratory in Amsterdam in February and March 2019, combined with the acylcarnitine and amino acid concentrations that are routinely measured in the newborn screening (NBS). The data contains variables regarding the sex, gestational age, gestational weight and measurements of thyroid hormones (T4, TSH, TBG) of the first and – if performed – second heelprick, various acylcarnitines (C0, C2, C5, C5DC, C5OH, C6, C8, C10, C10:1, C14, C14:1, C14:2, C16, C16:1, C16OH, C18:1OH) and amino-acid markers (leucine, phenylaline, succinylacetone, tyrosine, valine). Patient status was measured as a five level factor: no CH, CH-T, CH-C, CH with unknown cause, inconclusive. In addition to the thyroid hormone concentrations, the data included the coefficient of variation (CV) for T4, TSH, TBG, and the standard deviation (SD) of T4. The referred cohort consists of the CH NBS referrals between 2007 and 2017, and was obtained from the Netherlands Organization for Applied Scientific Research (TNO). The data was delivered in two separate files. The first file (referred TH; $n = 3308$) contained the same variables as the control cohort, excluding the acylcarnitines and amino-acid markers. The second file (referred AC; $n = 3784$) contained the acylcarnitines and amino-acid concentrations, year of birth, sex, gestational age, gestational weight, research lab and T4, T4 sd, TSH, and TBG concentrations either the (repeated) first or second heelprick. The NBS dataset that was used in this research was created as follows. First, subjects were excluded, then the three data files were combined and finally the redundant variables were removed. The details are discussed below.

In the control cohort, subjects were excluded if T4, TSH, and TBG concentration were all missing, the acylcarnitine and amino acids measurements were all missing, or the CV for T4 or TBG was larger than 20%. Furthermore, four cases were excluded manually based on the concentration and CV of TSH. After exclusion, 1842 subjects from the control cohort remained. The variable patient status was added, where all subjects were assigned the value 'No CH'. All variables regarding the second heelprick were removed. From the referred TH data, subjects with patient status 'inconclusive' or 'CH with unknown cause' were removed, as well as subjects with missing values for all thyroid hormones (T4, TSH and TBG). From the referred AC data, subjects with missing values for all thyroid hormones or all acylcarnitines and amino acids were excluded. Subjects that had an identical gestational age, gestational weight, sex, research lab and year of birth were excluded, under the assumption that these subjects appear in the data twice for two different heelprick measurements (e.g. first and second heelprick). After exclusion, 3111 and 1879 subjects remained in the referred TH and referred AC data, respectively. Due to the absence of an ID variable that links the subjects from referred TH and referred AC, a multi-step matching process was performed. Subjects with identical values for a number of variables were matched, and all one-to-one matches were extracted from the datasets. Matches that were not one-to-one (i.e. a row in one dataset corresponds to multiple rows in the other dataset) were investigated and manually adjusted. Finally, the data without a match were included in the next matching step. This was repeated five times for different sets of variables, as shown in Table A.1. The data that were not matched after these five steps were inspected manually in order to find additional matches that were not found before due to missing values or discrepancies. After all possible matches were established, a dataset (referred; $n = 1594$) was constructed that contained the sex, gestational age, gestational weight, patient status and first heelprick measurements of T4, TSH and TBG from the referred TH dataset, and the acylcarnitine and amino acid concentrations from the referred AC dataset. At last, the control and referred datasets were combined, after which the dataset included 3436 subjects with the following 30 variables: sex, gestational age, gestational weight, patient status, concentrations of T4, TSH, TBG and T4 sd of the first heelprick, and acylcarnitine and amino acid concentrations. In the final dataset, the 5 level categorical factor patient status was replaced by the binary variable CH that was equal to 1 if the patient status was CH-C or CH-T, and 0 otherwise.

Table A.1.: Variables included in each round of the matching process.

Round	**Variables**
1	sex, gestational age, gestational weight, birthyear, T4 (1st HP)
2	sex, gestational age, gestational weight, birthyear, T4 (2nd HP)
3	gestational age, gestational weight, birthyear, T4 (1st HP)
4	gestational age, gestational weight, birthyear, T4 (2nd HP)
5	gestational age, gestational weight, birthyear, T4 sd (1st HP)

B. Exploratory Data Analysis

This appendix contains the distribution plots of the features grouped by the target variable, which are discussed in detail in Chapter 7.

B.1. Newborn Screening

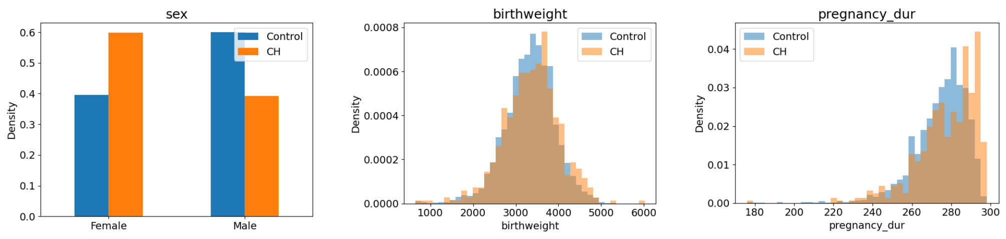


Figure B.1.: Distribution of sex, birth weight and pregnancy duration in the NBS dataset grouped by CH. The plots display the density of each variable relative to the target class.

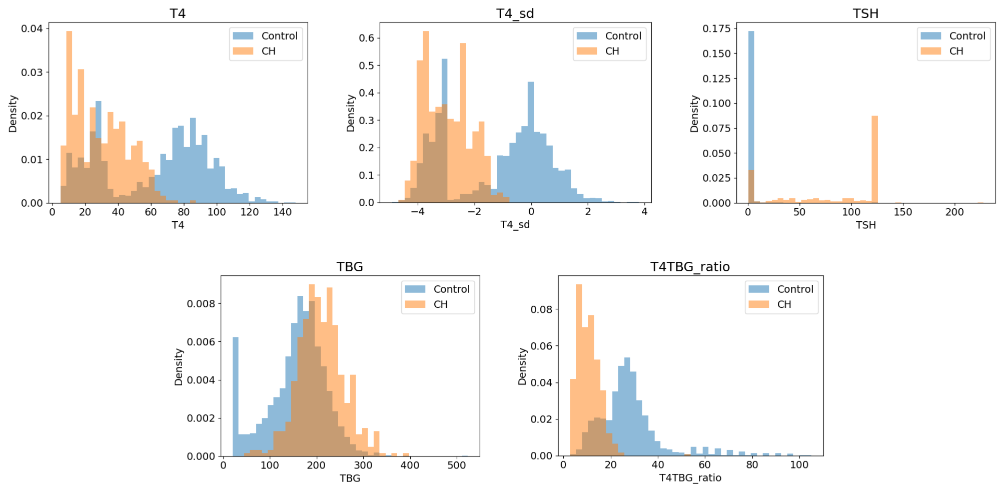


Figure B.2.: Distribution of thyroid hormones T4, T4 standard deviation, TSH, TBG and T4/TBG ratio in the NBS dataset grouped by CH. The plots display the density of each variable relative to the target class.

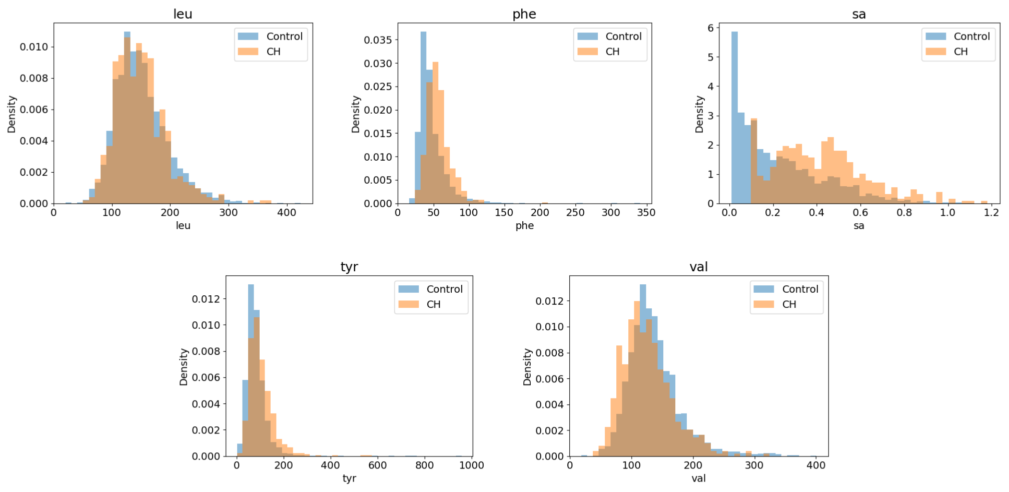


Figure B.3.: Distribution of the amino-acid markers leucine, phenylaline, succinylacetone, tyrosine and valine in the NBS dataset grouped by CH. The plots display the density of each variable relative to the target class.

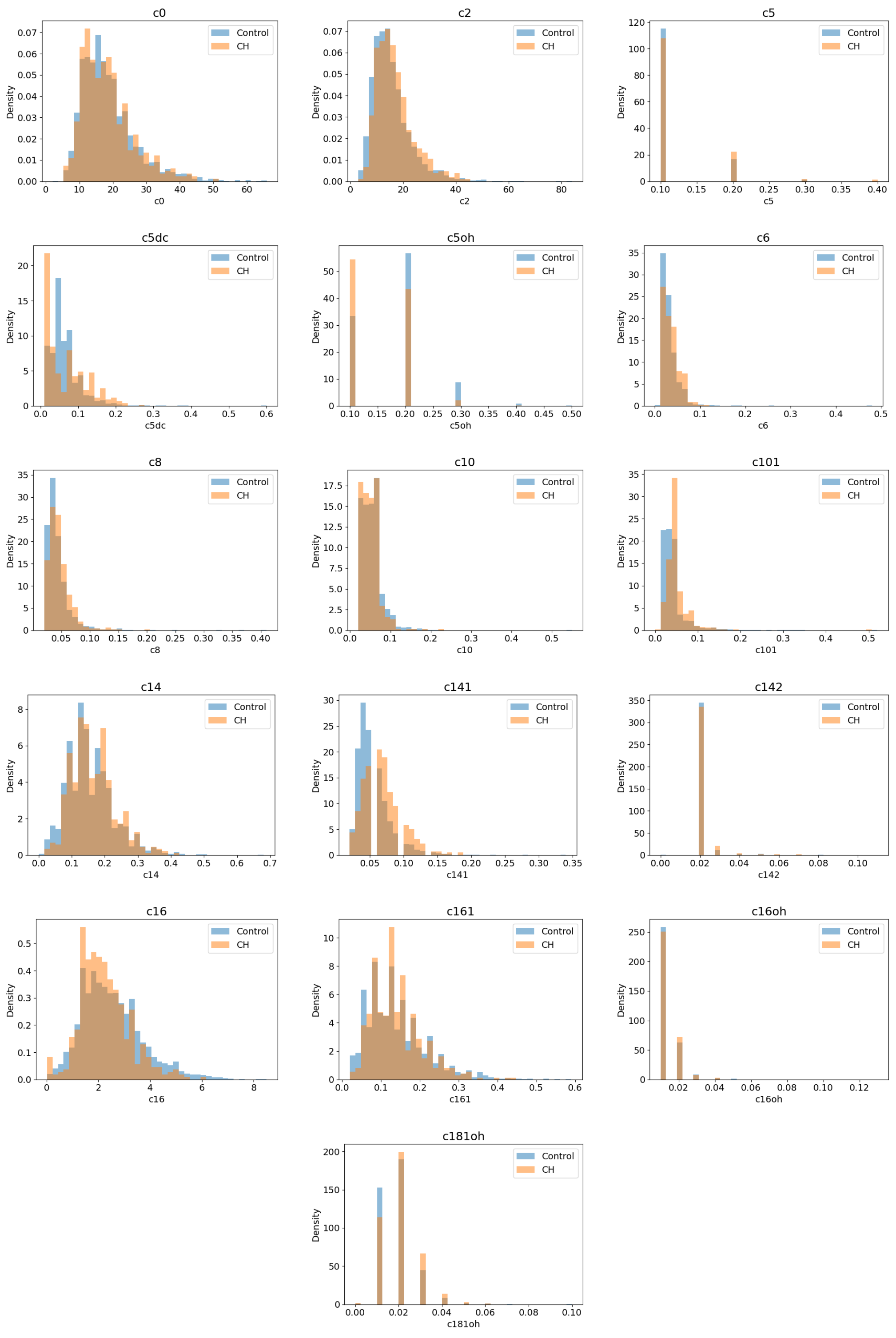


Figure B.4.: Distribution of the acylcarnitines C0, C2, C5, C5DC, C5OH, C6, C8, C10, C10:1, C14, C14:1, C14:2, C16, C16:1, C16OH and C18:1OH in the NBS dataset grouped by CH. The plots display the density of each variable relative to the target class.

B.2. Diabetes Health Indicators

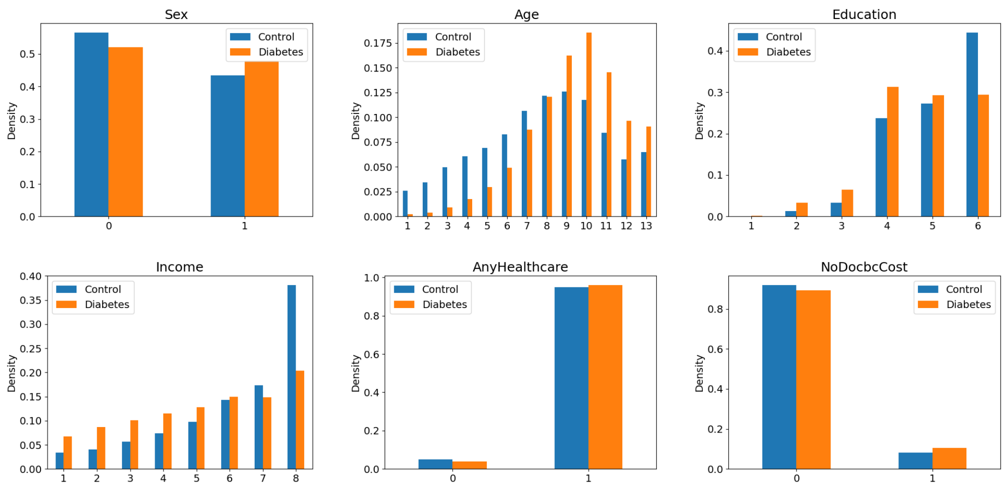


Figure B.5.: Distribution of sex, age, education, income, health care coverage and not able to afford a doctors visit in the DHI dataset grouped by diabetes. The plots display the density of each variable relative to the target class.

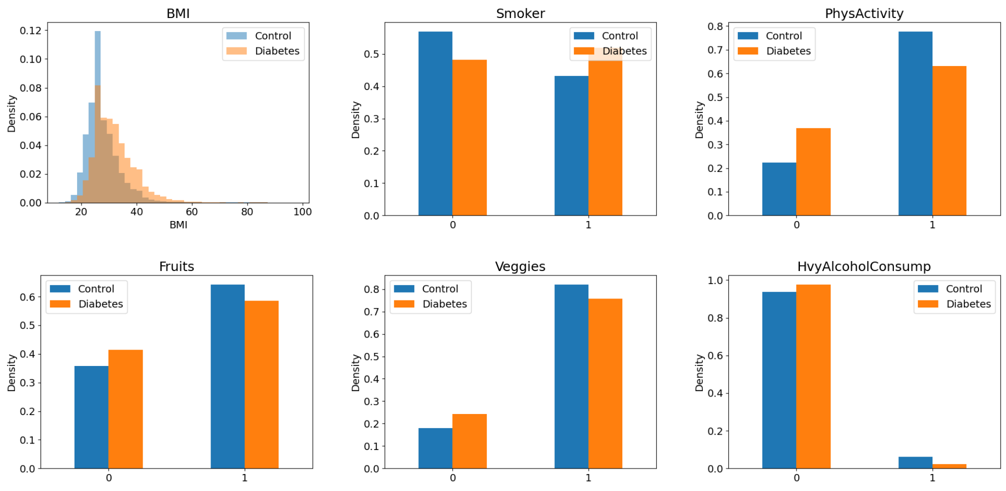


Figure B.6.: Distribution of body mass index (BMI), smoking status, physical activity, fruit consumption, vegetable consumption and heavy alcohol consumption in the DHI dataset grouped by diabetes. The plots display the density of each variable relative to the target class.

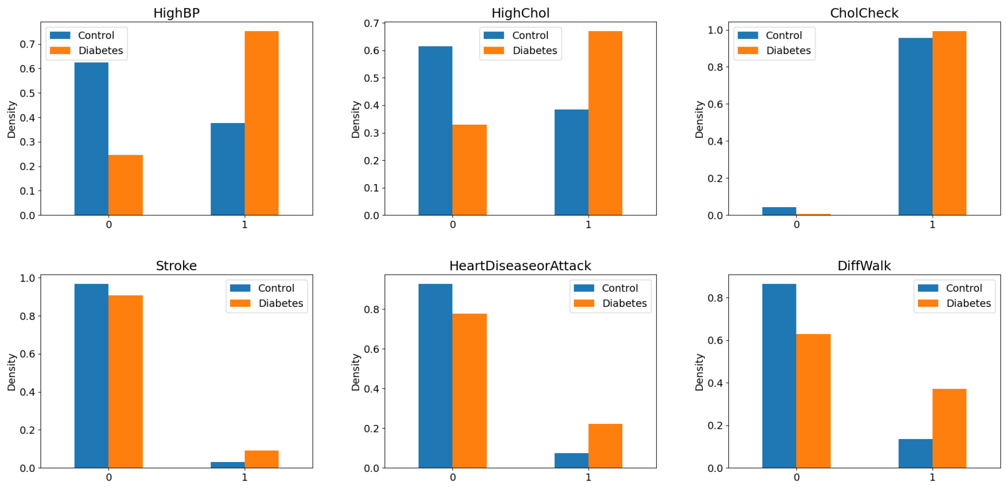


Figure B.7.: Distribution of high blood pressure, high cholesterol, cholesterol check, stroke, heart disease or heart attack and difficulty walking in the DHI dataset grouped by diabetes. The plots display the density of each variable relative to the target class.

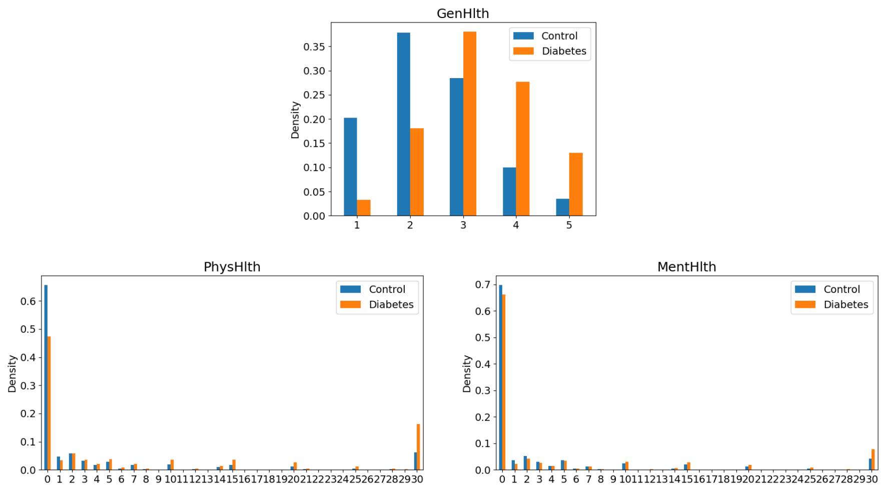


Figure B.8.: Distribution of general health score, number of days with poor physical or poor mental health in the last month in the DHI dataset grouped by diabetes. The plots display the density of each variable relative to the target class.

C. Hyperparameter tuning

This section describes the hyperparameter tuning for the two generative models (SMOTE-NC, TVAE). The tuning process was as follows. Each model was fitted on the parameter train set for each combination of hyperparameter values (Table 6.1). The performance of each model was quantified by the similarity between the data it generates and the minority cases from the validation set. The following quality measures were used: the mean inverted Kolmogorov-Smirnov Distance (KS D) statistic of the continuous features, mean p-value of the Chi-Squared test of the categorical features, L2 norm of the pairwise correlation differences (PCD-L2), and log-cluster metric. For the TVAE models, the mean loss of the model on the parameter train set was also calculated. Finally, the optimal set of hyperparameters was selected based on the values of these metrics. The results are discussed per dataset.

C.1. Newborn Screening

For the SMOTE-NC models, the number of nearest neighbours k was considered for tuning. The results for different values of k is shown in Figure C.1. The average inverted KS D-statistic was approximately equal for all number of neighbours considered, suggesting that the average maximum difference between the real and synthetic cumulative distribution function over all continuous variables was not influenced by the number of neighbours. The average p-value of the CS test suggests that around $k = 20$, the model was best able to capture the distributions of the categorical variable. The PCD-L2 and log-cluster metric do not show a clear pattern, indicating that the number of neighbours does not influence the similarity of the underlying structure of the real and synthetic data in terms of pairwise correlations or clustering. For the NBS dataset, the number of neighbours for the SMOTE-NC algorithm was set to 20.

For the TVAE model, the optimal network structure of the encoder and decoder, latent dimension and batch size were determined in this step. The encoder and decoder network had the same structure and only networks with two hidden layers with an equal number of neurons were considered. The performance of the models with different encoder/decoder network structures is shown in Figure C.2. The

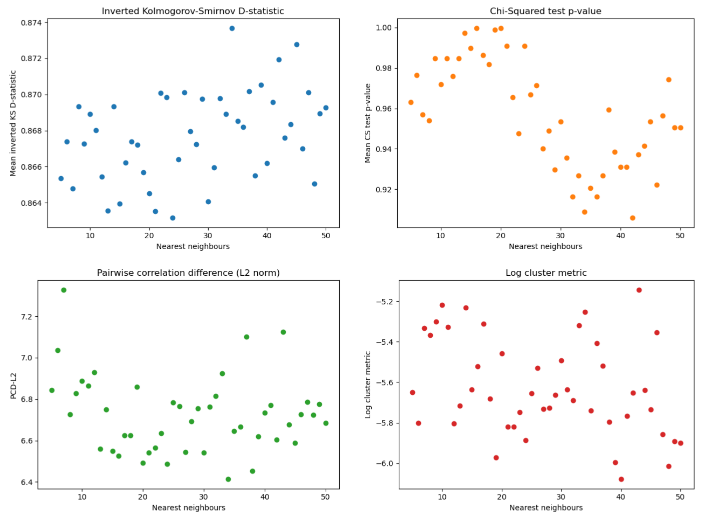


Figure C.1.: The mean inverted KS-D statistic, mean CS test p-value, L2 norm of the PCD and log-cluster metric as a function of the number of neighbours in the NBS dataset.

mean and variability of the train loss grows as the number of neurons of the two hidden layers increases. The average inverted KS D-statistic and CS test p-value increase for larger network structures, while the pairwise correlation differences decrease. This suggests that the synthetic sample is more similar to the validation set when the encoder and decoder network have larger hidden layers. The average log-cluster metric decreases for growing network structures, but increases for more than 96 neurons per layer. The mean loss on the train set decreases as the latent dimension increases (Figure C.3). The other four metrics do not show evidence of an relationship between the quality of the generated sample and the latent dimension. The batch size does not show a clear relation to any of the quality measures (Figure C.4).

It should be noted that the mean train loss shows an opposite relationship with the properties of the TVAE model than the KS D-statistic, CS test p-value and pairwise correlation difference. For example, these latter three metrics indicate higher quality for larger network, while the train loss suggests a less stable model which obtains a higher training error. Recall from Chapter 4 that the loss of the variational auto-encoder consists of two parts: the reconstruction loss and the Kullback-Leibler divergence. Further inspection of the train loss has shown that for the NBS dataset both the reconstruction loss and KL divergence increase for larger network structures. For the latter, this is not surprising. However, the reconstruction loss is expected to decrease for larger networks because the generated

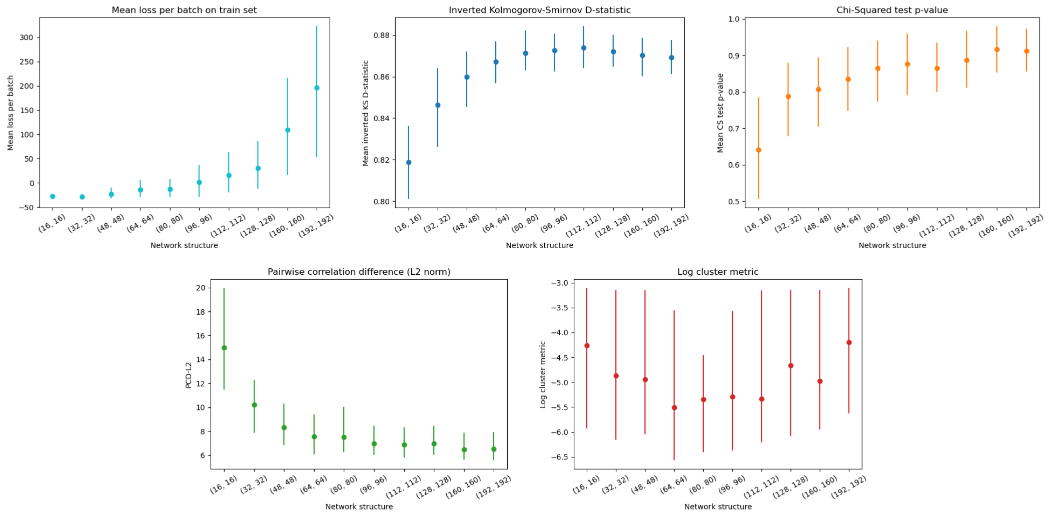


Figure C.2.: The mean loss on the parameter train set, mean inverted KS-D statistic, mean CS test p-value, L2 norm of the PCD and log-cluster metric for different encoder/decoder network structures in the NBS dataset.

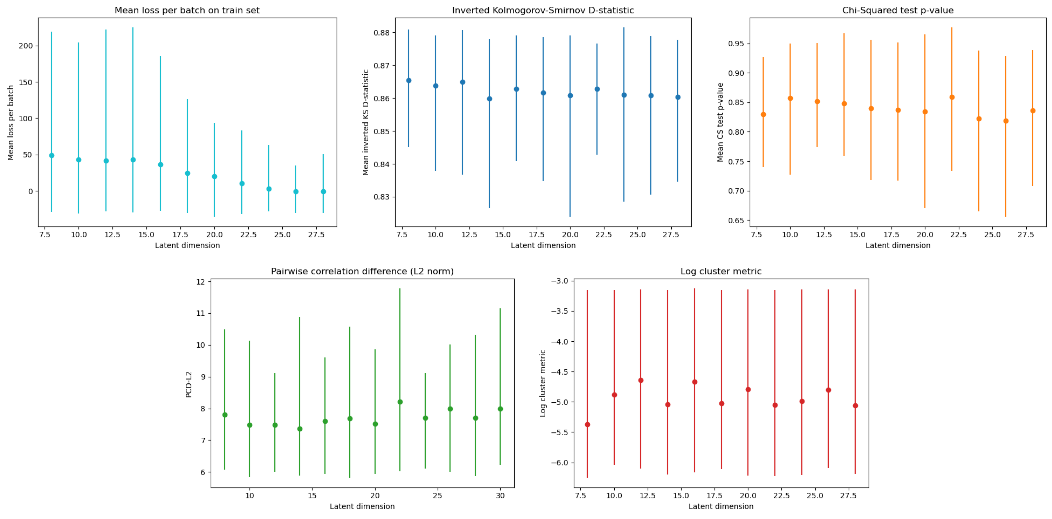


Figure C.3.: The mean loss on the parameter train set, mean inverted KS-D statistic, mean CS test p-value, L2 norm of the PCD and log-cluster metric for different latent dimensions in the NBS dataset.

data becomes more similar to the real data. In order to investigate whether one or more features caused this behaviour, the TVAE models were trained on each of the individual variables instead of all 28. This analysis suggested that the following low cardinality continuous variables were responsible: C5, C5OH, C6, C8,

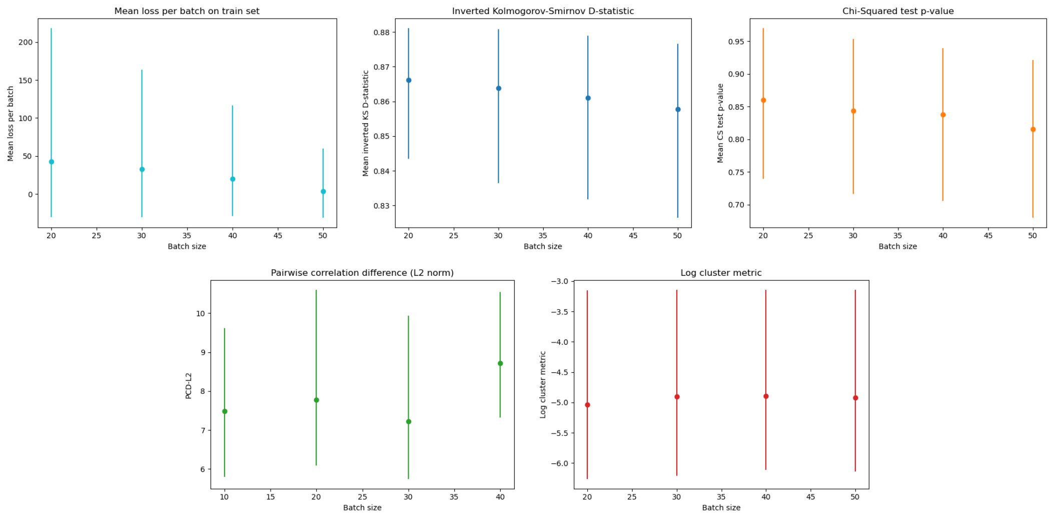


Figure C.4.: The mean loss on the parameter train set, mean inverted KS-D statistic, mean CS test p-value, L2 norm of the PCD and log-cluster metric for different batch sizes in the NBS dataset.

C16OH and C18:1OH. However, without these 6 variables, the train loss still increased in size and variability for larger network structures, contrasting to the KS D-statistic and PCD-L2 that indicate higher quality of the generated data. Next, the toy dataset **diabetes** from `sklearn` [37] was used to investigate whether this behaviour was specifically related to the NBS dataset or also occurred in other datasets with many continuous variables. The data ($n = 442$) consisted of 10 features, of which 9 were continuous. Only the 9 continuous variables were used in this analysis, and the TVAE model was trained for the same network structures as the NBS data, only with another set of latent dimensions because the number of features is smaller: $[2, 3, ..., 7, 8]$. For this dataset, the train loss, and in particular the reconstruction loss, also increased in size and variance for larger network structures, while the KS D-statistic and PCD-L2 indicate a better performance.

We conclude that the unexpected behaviour of the train set is not specific for the NBS dataset. Therefore, the hyperparameters were chosen based on the other four metrics, and the train loss was ignored. The following hyperparameter values were selected: network structure $(80, 80)$, latent dimension 14 and batch size 30.

C.2. Diabetes Health Indicators

The performance of the SMOTE-NC algorithm for various number of nearest neighbours was evaluated in order to select the optimal value for generating diabetes patient training data. The SMOTE-NC models were compared based on the quality of a synthetic sample generated by the respective model, using the mean inverted KS D-statistic, mean CS test p-value and log-cluster metric. It was not possible to calculate the pairwise correlation differences because there was only one continuous variable.

The results for SMOTE-NC are shown in Figure C.5. The inverted KS D-statistic does not appear to be influenced by the number of neighbours. The CS p-value decreased as the number of nearest neighbours increased, suggesting that a model with a lower number of neighbours is better able to reconstruct the marginal distributions of the categorical columns. However, the difference between the lowest and highest score was relatively small. The log-cluster metric indicates that the synthetic data better captures the underlying clustering structure of the real data for a smaller number of neighbours. According to these results, the number of neighbours in the SMOTE-NC algorithm was set to 10 for the DHI dataset.

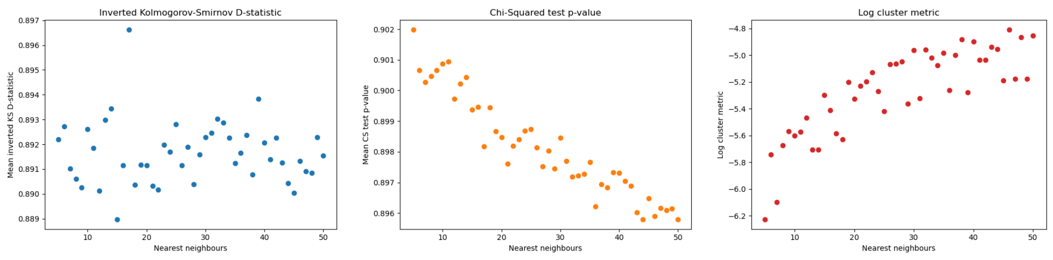


Figure C.5.: The mean inverted KS-D statistic, mean CS test p-value, and log-cluster metric as a function of the number of neighbours in the DHI dataset.

For TVAE, the optimal network structure of the encoder and decoder network, latent dimension and batch size were determined. Again, the encoder and decoder network were assumed to have the same structure and consist of two hidden layers with an equal number of neurons per layer. Figure C.6 shows the performance of the model for different network structures. The average loss on the train set was decreasing for increasing network structures. The inverted KS D-statistic, CS test p-value and log-cluster metric also indicated higher performance for larger network structures. The increase in performance was most prominent in the smallest network structures, and slowed down as the network structures increase in size. TVAE models with a latent dimension of 12 or higher showed equivalent results, while this performance decreased for smaller latent dimensions (Figure C.7). The overall performance of the models does not appear related to the batch size, and the large confidence intervals of the metrics for each batch size suggest that the performance depended more on the value of the other hyperparameters (Figure C.8). According to these findings, the following hyperparameter values were selected: network structure $(96, 96)$, latent dimension 12 and batch size 2000.

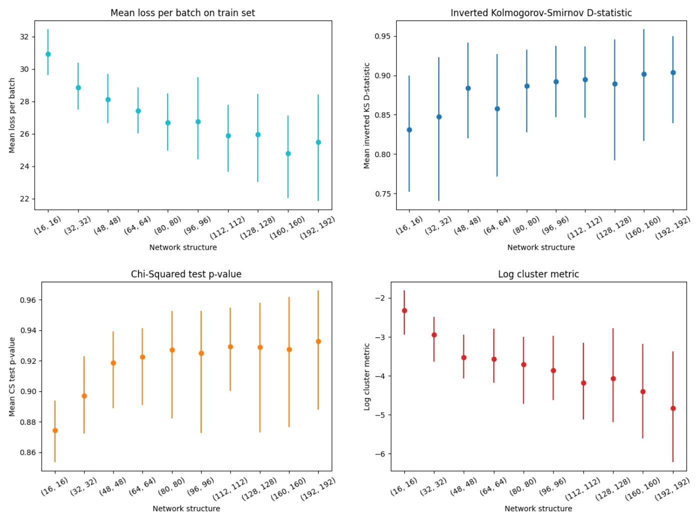


Figure C.6.: The mean loss on the parameter train set, mean inverted KS-D statistic, mean CS test p-value, and log-cluster metric for different encoder/decoder network structures in the DHI dataset.

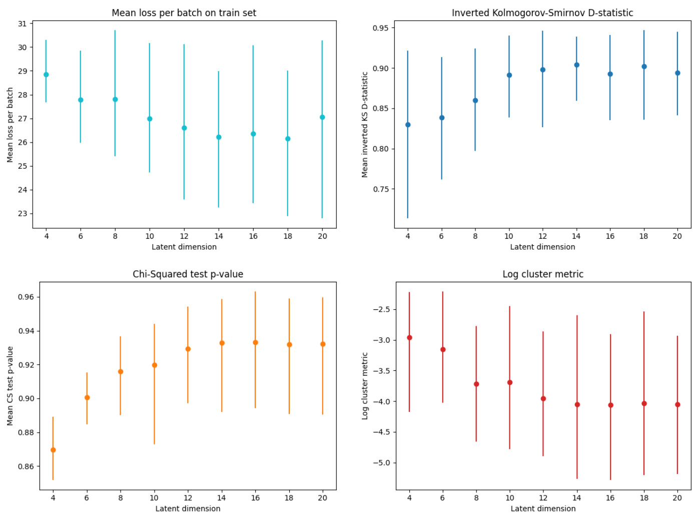


Figure C.7.: The mean loss on the parameter train set, mean inverted KS-D statistic, mean CS test p-value, and log-cluster metric for different latent dimensions in the DHI dataset.

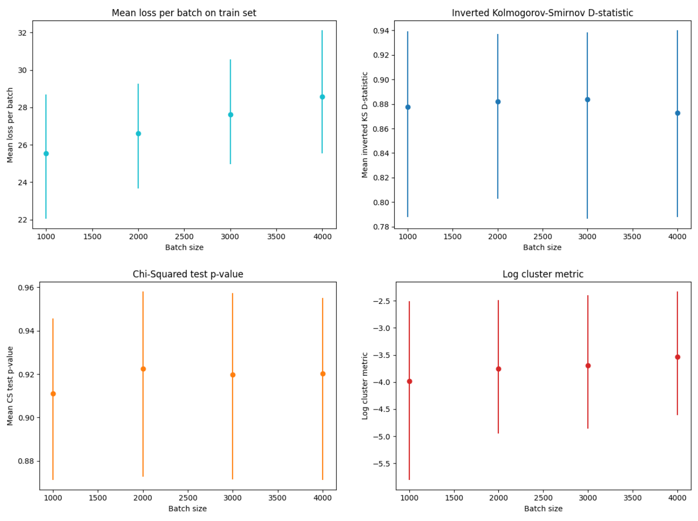


Figure C.8.: The mean loss on the parameter train set, mean inverted KS-D statistic, mean CS test p-value, and log-cluster metric for different batch sizes in the DHI dataset.

D. NBS use case

The newborn screening dataset was provided by the Amsterdam University Medical Centra (UMC) with the aim of building a machine learning model to accurately predict congenital hypothyroidism in the Dutch newborn screening. Although the main objective of this thesis is to investigate the performance of various generative models and classifiers on disease classification in a more general setting, this section translates this research to a real-world disease classification problem.

D.1. Background

Congenital hypothyroidism (CH) is a thyroid hormone deficiency present at birth. In most cases, hypothyroidism results from an underdeveloped thyroid gland or a disorder of thyroid hormone biosynthesis, called primary congenital hypothyroidism (CH-T). Secondary congenital hypothyroidism (CH-C) is caused by a deficiency of thyroid stimulating hormone (TSH). The prevalence of CH-T and CH-C in the Netherlands has been estimated at 1:3017 respectively 1:16,404 [48]. CH in newborns is characterised by a low thyroxine (T4) level in combination with increased TSH in case of CH-T or normal/decreased TSH in case of CH-C. When left untreated, CH can lead to mental retardation and motor impairment. Early detection and treatment of congenital hypothyroidism is very important in order to limit the neurological damage. However, patients are often left undiagnosed at birth because of the subtle or absent clinical symptoms [49]. To aid the early detection of CH, many countries have adopted a newborn screening programme (NBS) for serious congenital disorders, including congenital hypothyroidism [50]. The Dutch NBS detects CH by measuring the serum level of three markers: T4, TSH and thyroxine-binding globulin (TBG). TBG is a transport protein that carries thyroid hormones, such as T4, through the bloodstream. The Dutch T4-TSH-TBG based algorithm is as follows: T4 is measured in all newborns, TSH is measured in the lowest 20% of the daily T4 concentrations and TBG in the lowest 5% of the daily T4 concentrations. Possible CH-T is determined by the T4 and TSH levels, while the T4/TBG ratio, an indirect measure of free T4 (fT4) concentration, is used to detect CH-C. The TBG concentration is used to reduce the number of false-positive referrals caused by TBG deficiency [48]. The algorithm effectively detects CH-T as well as CH-C, at the cost of a low positive predictive value (PPV) of 21% [51]. Previous research aimed to improve the Dutch congenital hypothyroidism screening algorithm by implementing a random forest model including the following variables: gestational age, gestational weight, age at NBS sampling, T4 level, TSH level, TBG level and the T4/TBG ratio. The random forest model reduced the number of false-positive results, by increasing the positive predictive value (PVV) from 21% to 26%.

Recent studies have shown that the fT4 blood concentration is associated with the acylcarnitine profile and a number of amino acids (including tyrosine, phenylalanine and serine) [52, 53, 54, 55, 56]. Since a measure of fT4 is used to detect CH-C in the newborn screening, the acylcarnitines and amino acids may be associated with the presence of CH. It is hypothesised that the levels of acylcarnitines and amino acids differ between CH patients and healthy newborns, and that adding these to a machine learning model that predicts CH can improve the positive predictive value by reducing the number of false-positive referrals.

The objective of this use case is to build a machine learning model that predicts all cases of CH with less false-positive referrals than the current working algorithm or random forest model from Stroek et al. (under revision), using data from the Dutch newborn screening. We expect to improve the previous model by adding acylcarnitine and amino-acid measurements that are possibly associated with CH.

D.2. Methods

The sample was constructed from two cohorts of healthy newborns and referrals, as described in Appendix A. The data consists of 3436 newborns, including 431 CH-T, 84 CH-C, 1079 false-positive referrals and 1842 healthy controls. The prevalence of congenital hypothyroidism in the study population was 15.0%, of which 12.5% CH-T and 2.4% CH-C. The data contains the following 30 variables: sex, gestational age, gestational weight, presence of CH, and measurements of thyroid hormones (T4, T4 sd, TSH, TBG, T4/TBG ratio), various acylcarnitines (C0, C2, C5, C5DC, C5OH, C6, C8, C10, C10:1, C14, C14:1, C14:2, C16, C16:1, C16OH, C18:1OH) and amino-acid markers (leucine, phenylaline, succinylacetone, tyrosine, valine). The analysis consisted of two steps. First, the relation between the thyroid hormones and the acylcarnitines and amino-acids was calculated using Pearson correlations. The subjects with a TBG level of 105 or lower were excluded from the correlation analysis due to changes in the regulations of the CH screening process. Second, the fifteen models investigated in the main objective of this thesis were evaluated based on the specificity and positive predictive value (PPV) at 100% sensitivity. The random forest model with SMOTE-NC oversampling obtained the best results, and was therefore investigated in more detail for the purpose of this use case. The data generation process and model training are described in Chapter 6. The following hyperparameters were selected using randomised search cross validation: the number of trees was 500, maximum depth of the trees 15, 5 features were considered at each split and 0.77 of the data was considered at each tree (Table E.1). An receiver operating characteristic (ROC) analysis was performed to evaluate the performance of the model at different classification thresholds. In order to investigate which features contribute most to the predictions, the permutation feature importance was computed on the whole test set as well as the test set without the CH-T cases. The latter was used to determine which features contribute specifically to the prediction of CH-C.

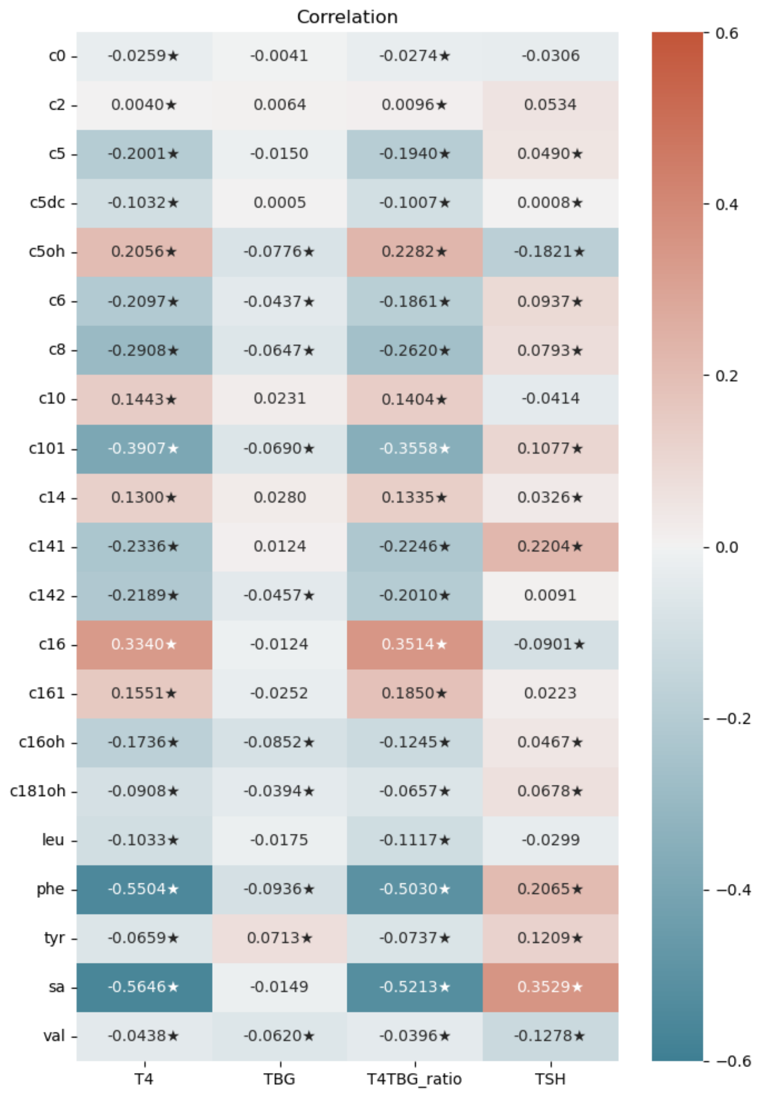
D.3. Results

The Pearson correlations between the thyroid hormones and the acylcarnitines and amino-acids are shown in Figure D.1. First of all, TBG was weakly correlated with the acylcarnitines or amino-acids. Of the 16 acylcarnitines, T4 and the T4/TBG ratio were most strongly correlated with C8, C10:1, C16, while TSH had the largest correlations with C5OH and C14:1. All three thyroid hormones were strongly correlated with phenylaline and succinylacetone. As expected, the sign of the correlations is opposite between T4, T4/TBG ratio and TSH.

Table D.1.: The specificity, accuracy and positive predictive value reached by the model for different values of the sensitivity (1.00, 0.99, 0.98), as well as the corresponding number of true negative, true positive, false negative and false positive results.

Spec	**Sens**	**Acc**	**PPV**	**TN**	**TP**	**FN**	**FP**
0.83	1.00	0.86	0.51	800	164	0	164
0.92	0.99	0.93	0.69	890	168	2	74
0.94	0.98	0.94	0.73	903	166	4	61

The model specificity, accuracy and positive predictive value obtained for different sensitivities are displayed in Table D.1. The Random Forest model was able to yield a sensitivity of 100%, while obtaining a specificity of 83% and PPV of 51%. Decreasing the sensitivity to 99% led to a substantial increase in specificity (92%) and positive predictive value (69%). Figure D.2 shows the permutation feature importance of the predictive model on the test set. As expected from previous research, TSH plays the most important role in predicting the presence of CH. Additionally, the other thyroid hormone markers (T4, T4 sd, TBG, T4/TBG



	T4	TBG	T4TBG_ratio	TSH
c0	-0.0259★	-0.0041	-0.0274★	-0.0306
c2	0.0040★	0.0064	0.0096★	0.0534
c5	-0.2001★	-0.0150	-0.1940★	0.0490★
c5dc	-0.1032★	0.0005	-0.1007★	0.0008★
c5oh	0.2056★	-0.0776★	0.2282★	-0.1821★
c6	-0.2097★	-0.0437★	-0.1861★	0.0937★
c8	-0.2908★	-0.0647★	-0.2620★	0.0793★
c10	0.1443★	0.0231	0.1404★	-0.0414
c101	-0.3907★	-0.0690★	-0.3558★	0.1077★
c14	0.1300★	0.0280	0.1335★	0.0326★
c141	-0.2336★	0.0124	-0.2246★	0.2204★
c142	-0.2189★	-0.0457★	-0.2010★	0.0091
c16	0.3340★	-0.0124	0.3514★	-0.0901★
c161	0.1551★	-0.0252	0.1850★	0.0223
c16oh	-0.1736★	-0.0852★	-0.1245★	0.0467★
c181oh	-0.0908★	-0.0394★	-0.0657★	0.0678★
leu	-0.1033★	-0.0175	-0.1117★	-0.0299
phe	-0.5504★	-0.0936★	-0.5030★	0.2065★
tyr	-0.0659★	0.0713★	-0.0737★	0.1209★
sa	-0.5646★	-0.0149	-0.5213★	0.3529★
val	-0.0438★	-0.0620★	-0.0396★	-0.1278★

Figure D.1.: Pearson correlations between the thyroid hormones and the acylcarnitines and amino-acids. Correlations with a p-value smaller than 0.05 are marked with a star.

ratio) and tyrosine contribute to the predictions. In order to gain more insight in which features are important in predicting CH-C, the feature importance was also computed for the test set without the CH-T cases (Figure D.3). Remark that for these cases, TSH has no explanatory value. The most important features are the other four thyroid hormones, followed by tyrosine, succinylacetone, phenylaline and C2, C10:1 and C5DC.

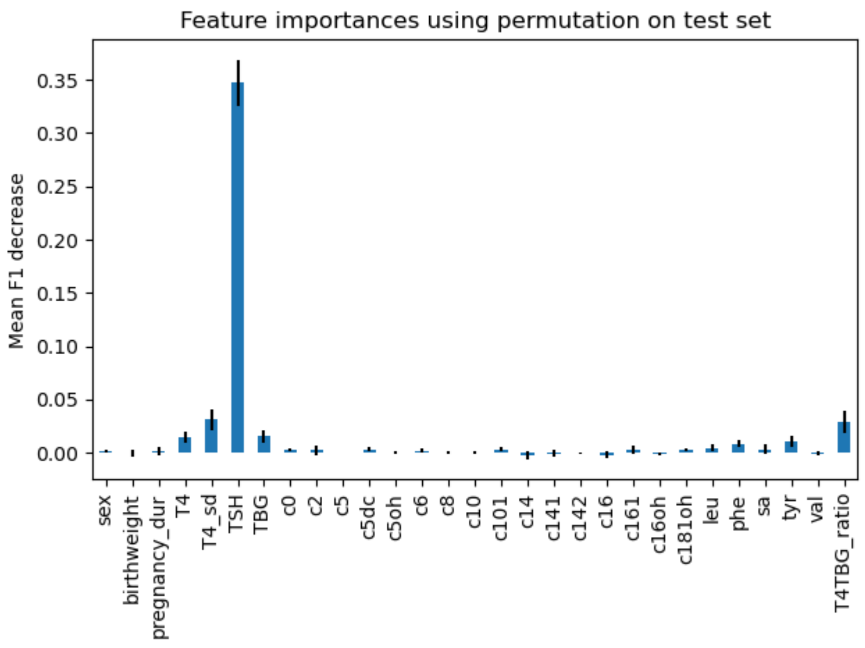


Figure D.2.: Permutation feature importance on the test set.

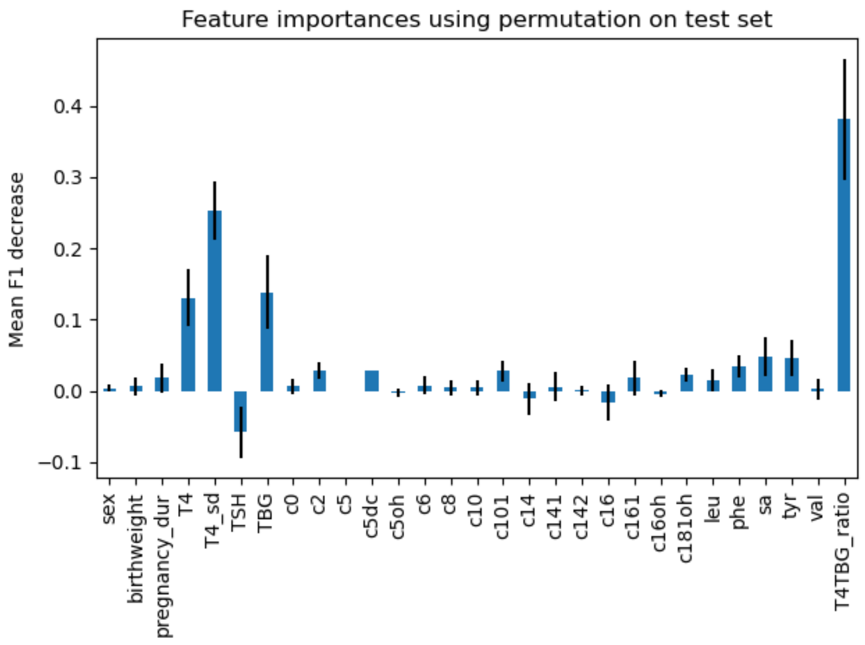


Figure D.3.: Permutation feature importance on the controls and CH-C cases in the test set.

D.4. Conclusion

As hypothesised, the current model decreased the number of false-positive referrals compared to the previous model from Stroek et al. (under revision). The model obtained a positive predictive value of 51% and specificity of 83%, while maintaining a sensitivity of 100%. The feature importance on the test set without CH-T showed that various amino-acids and acylcarnitines contribute in particular to the classification of CH-C cases.

E. Supplementary Tables

Table E.1.: Optimal hyperparameters of the 15 predictive models on the NBS dataset.

Classifier	**Method**	**num trees**	**max features**	**max depth**	**sub sample**	**learning rate**	**imb alpha**	**focal gamma**	**F1 mean**	**F1 s.d.**
RF	None	500	25	15	0.57	-	-	-	0.901	0.026
	SMOTE-NC	500	5	15	0.77	-	-	-	0.983	0.004
	TVAE	200	9	9	0.83	-	-	-	0.974	0.019
XGB-CE	None	30	-	5	0.64	0.17	-	-	0.902	0.024
	SMOTE-NC	35	-	15	0.77	0.38	-	-	0.984	0.006
	TVAE	35	-	7	0.70	0.21	-	-	0.977	0.019
XGB-F	None	45	-	9	0.87	0.04	-	2.5	0.913	0.020
	SMOTE-NC	45	-	12	0.79	0.37	-	2.0	0.985	0.007
	TVAE	15	-	8	0.54	0.37	-	2.0	0.977	0.016
XGB-W	None	50	-	14	0.55	0.03	2.5	-	0.907	0.024
	SMOTE-NC	45	-	7	0.59	0.37	2.0	-	0.985	0.006
	TVAE	35	-	14	0.82	0.25	3.5	-	0.977	0.013
XGB-WF	None	35	-	14	0.57	0.17	1.5	2.5	0.907	0.028
	SMOTE-NC	50	-	11	0.80	0.25	3.0	3.0	0.985	0.005
	TVAE	45	-	13	0.71	0.12	2.0	1.5	0.977	0.015

Table E.2.: Optimal hyperparameters of the 15 predictive models on the DHI dataset.

Classifier	**Method**	**num trees**	**max features**	**max depth**	**sub sample**	**learning rate**	**imb alpha**	**focal gamma**	**F1 mean**	**F1 s.d.**
RF	None	400	19	15	0.54	-	-	-	0.249	0.007
	SMOTE-NC	400	19	15	0.54	-	-	-	0.828	0.073
	TVAE	400	19	15	0.54	-	-	-	0.845	0.141
XGB-CE	None	50	-	12	0.91	0.34	-	-	0.293	0.009
	SMOTE-NC	50	-	13	0.94	0.38	-	-	0.844	0.086
	TVAE	15	-	14	0.55	0.05	-	-	0.843	0.139
XGB-F	None	30	-	9	0.96	0.02	-	1.0	0.461	0.006
	SMOTE-NC	50	-	15	0.87	0.25	-	1.5	0.848	0.086
	TVAE	30	-	11	0.99	0.18	-	2.0	0.840	0.162
XGB-W	None	25	-	7	0.81	0.20	4.0	-	0.467	0.005
	SMOTE-NC	45	-	15	0.98	0.37	3.0	-	0.842	0.077
	TVAE	45	-	12	0.77	0.06	1.5	-	0.834	0.140
XGB-WF	None	25	-	7	0.93	0.17	3.5	3.0	0.469	0.007
	SMOTE-NC	50	-	15	0.87	0.35	2.0	3.0	0.848	0.079
	TVAE	30	-	8	0.59	0.13	1.5	2.5	0.835	0.126

Table E.3.: Permutation feature importance on the test set of the 15 predictive models for the DHI dataset.

Variable	RF			XGB			XGB-F			XGB-W			XGB-WF		
	none	smote	tvae	none	smote	tvae	none	smote	tvae	none	smote	tvae	none	smote	tvae
HighBP	0.077	0.034	0.102	0.056	0.038	0.098	0.025	0.041	0.092	0.019	0.031	0.085	0.028	0.037	0.087
HighChol	0.056	0.018	0.013	0.037	0.022	0.011	0.015	0.024	0.014	0.013	0.018	0.013	0.014	0.019	0.010
CholCheck	0.002	0.004	0.002	0.004	0.006	0.001	0.000	0.005	0.006	0.004	0.005	0.003	0.004	0.006	0.004
BMI	0.106	0.045	0.034	0.075	0.045	0.032	0.056	0.050	0.037	0.050	0.040	0.035	0.055	0.044	0.028
Smoker	-0.003	-0.000	-0.001	-0.004	-0.001	-0.001	0.000	0.001	-0.002	-0.000	-0.000	-0.001	0.000	-0.001	0.000
Stroke	0.006	0.000	0.001	0.005	-0.002	0.000	0.001	-0.001	-0.001	0.001	-0.001	-0.000	-0.000	-0.001	-0.001
HeartDisease orAttack	0.015	0.001	0.003	0.011	0.003	0.002	0.004	0.005	0.005	0.003	0.002	0.004	0.002	0.002	0.004
PhysActivity	0.004	-0.000	-0.002	0.000	-0.001	-0.002	0.000	0.001	0.001	0.000	-0.003	-0.002	0.000	-0.001	-0.001
Fruits	0.002	0.000	-0.000	-0.002	0.001	-0.001	0.000	0.001	0.003	0.000	-0.001	-0.000	0.000	0.000	-0.001
Veggies	-0.001	0.000	0.000	0.000	-0.001	-0.001	0.000	0.001	0.000	0.000	-0.002	-0.000	0.000	-0.002	-0.000
HvyAlcohol Consump	0.006	0.009	0.006	0.004	0.008	0.005	0.002	0.008	0.007	0.004	0.007	0.005	0.005	0.008	0.005
AnyHealthcare	-0.001	0.000	0.001	-0.002	0.001	0.000	0.000	0.002	0.000	-0.000	0.000	-0.000	-0.000	0.000	-0.001
NoDocbcCost	-0.000	-0.002	-0.002	-0.000	0.001	-0.005	0.000	0.001	-0.001	0.000	-0.002	-0.000	-0.000	-0.001	-0.001
GenHlth	0.068	0.105	0.057	0.058	0.090	0.052	0.059	0.095	0.054	0.057	0.079	0.055	0.060	0.082	0.049
MentHlth	-0.000	0.003	0.007	-0.002	-0.001	0.000	0.001	0.001	0.006	0.001	-0.002	0.002	0.000	-0.002	-0.001
PhysHlth	0.002	-0.000	0.029	-0.003	0.001	0.008	0.002	0.002	0.013	0.001	-0.004	0.005	0.000	0.000	0.005
DiffWalk	0.005	0.006	0.002	0.009	0.008	0.002	0.001	0.009	0.013	0.002	0.001	0.003	0.001	0.004	0.005
Sex	0.002	-0.001	0.001	0.000	0.002	0.001	0.000	0.004	0.001	0.002	0.002	0.003	0.002	0.004	0.002
Age	0.029	0.044	0.037	0.027	0.046	0.039	0.035	0.047	0.039	0.034	0.040	0.036	0.032	0.040	0.035
Education	0.002	-0.002	0.001	-0.003	-0.001	0.000	0.000	0.001	0.002	0.000	-0.003	0.001	-0.000	-0.002	-0.000
Income	0.008	0.002	0.023	0.001	0.008	0.023	0.001	0.009	0.026	0.004	0.003	0.022	0.002	0.005	0.025

Table E.4.: The difference between the permutation feature importance on the train and test set of the 15 predictive models for the DHI dataset. Large values indicate that the model tends to overfit using that variable.

Variable	RF			XGB			XGB-F			XGB-W			XGB-WF		
	none	smote	tvae	none	smote	tvae	none	smote	tvae	none	smote	tvae	none	smote	tvae
HighBP	0.137	0.015	0.034	0.129	0.021	0.031	0.011	0.031	0.008	0.007	0.037	0.020	0.007	0.034	-0.018
HighChol	0.129	0.011	-0.004	0.123	0.023	-0.005	0.008	0.030	-0.007	0.003	0.035	-0.007	0.004	0.036	-0.008
CholCheck	0.002	0.002	0.001	0.010	0.003	0.000	0.000	0.003	0.002	0.001	0.003	-0.001	0.001	0.003	-0.001
BMI	0.243	0.028	0.001	0.317	0.090	-0.006	0.016	0.099	-0.014	0.011	0.095	-0.011	0.010	0.106	-0.016
Smoker	0.040	0.008	0.006	0.070	0.029	0.004	0.001	0.031	0.005	0.001	0.029	0.003	0.000	0.036	0.000
Stroke	0.022	0.003	0.001	0.021	0.012	0.000	0.001	0.010	0.004	0.001	0.008	0.001	0.002	0.009	0.002
HeartDisease orAttack	0.055	0.002	0.000	0.044	0.018	-0.000	0.004	0.011	-0.002	0.003	0.017	-0.001	0.003	0.015	-0.003
PhysActivity	0.034	0.007	0.012	0.071	0.024	0.009	0.001	0.027	0.009	0.001	0.025	0.008	0.001	0.027	0.005
Fruits	0.036	0.008	0.007	0.063	0.025	0.005	0.000	0.032	0.004	0.000	0.026	0.004	0.000	0.032	0.001
Veggies	0.033	0.005	0.004	0.049	0.022	0.003	0.001	0.020	0.004	0.000	0.020	0.003	0.000	0.023	0.001
HvyAlcohol Consump	0.007	0.007	0.006	0.021	0.007	0.005	0.001	0.008	0.004	0.002	0.007	0.002	0.002	0.007	0.001
AnyHealthcare	0.009	0.005	0.008	0.018	0.006	0.005	0.000	0.006	0.003	0.001	0.004	0.002	0.001	0.005	0.002
NoDocbcCost	0.018	0.007	0.023	0.033	0.012	0.023	0.000	0.011	0.010	0.001	0.011	0.007	0.001	0.011	0.003
GenHlth	0.216	0.037	-0.001	0.223	0.063	-0.010	0.012	0.067	-0.014	0.008	0.083	-0.019	0.008	0.077	-0.026
MentHlth	0.076	0.075	0.068	0.144	0.073	0.035	0.002	0.076	0.034	0.001	0.068	0.032	0.001	0.070	0.015
PhysHlth	0.106	0.079	0.065	0.186	0.095	0.036	0.003	0.100	0.033	0.001	0.091	0.026	0.002	0.089	0.007
DiffWalk	0.050	0.006	0.005	0.058	0.021	0.002	0.001	0.019	-0.007	0.001	0.022	0.001	0.002	0.020	-0.004
Sex	0.053	0.010	0.008	0.072	0.030	0.005	0.002	0.031	0.006	0.003	0.029	0.003	0.003	0.036	0.001
Age	0.166	0.054	0.119	0.248	0.103	0.110	0.011	0.116	0.096	0.007	0.105	0.098	0.008	0.122	0.072
Education	0.075	0.018	0.013	0.133	0.060	0.009	0.002	0.063	0.017	0.002	0.055	0.008	0.002	0.069	0.004
Income	0.126	0.021	0.053	0.201	0.089	0.046	0.003	0.096	0.034	0.003	0.084	0.032	0.003	0.099	0.014