Prediction of Heart Failure using Machine Learning Methods

Eric Lundholm Wiik
KTH Royal Institute of Technology
ericlw@kth.se

David Haapanen
KTH Royal Institute of Technology
davidhaa@kth.se

Noah Eriksson Tewolde Berhane
KTH Royal Institute of Technology
noahetb@kth.se

Code and data: github.com/Ericlwiik/SF2935

1 Introduction

As of today the number one leading cause of death is Cardiovascular diseases, CVDs. These are disorders that mainly affect the heart and blood vessels leading to diseases like coronary heart disease, cerebrovascular disease, rheumatic heart disease etc. It was estimated in 2022 that 19.8 million people died form CVDs, making up 32% of all global deaths that year. More so 85% of these deaths was the result from heart attack and stroke. The cause of CVDs are linked to both genetic but also social-economic factor affecting daily habits. Thus most CVDs can be prevented by healty diet, physical activity and abstaining from tobacco or alcohol use. It is however very important to detect CVDs as early as possible to receive proper treatment and counselling before the disease becomes more severe.

It is therefore of high interest to be able to detect weather a patient is of risk of having heart failure since this is often caused by CVDs. Thus developing a model that can classify this using values from medical tests would be very benefitual for identifying if a patient needs treatment.

The purpose of this project was thus to answer the research question of which Machine Learning, ML, methods are suitable and provide accurate and reliable predictions of heart failure among patients. For the scope of the project the methods that were chosen to be developed and applied were

- k-Nearest Neighbors, k-NN
- Decision trees
- Neural Networks
- Support Vector Machine, SVM
- Gaussian Process Classifier

2 Methodology

2.1 Data Preparation

The dataset used in this project contained 918 observations stemming from independent clinics from various countries. Each data point has values for 11 different features representing things like age, gender, heart rate, blood pressure etc. Each observation has a target variable with a binary label $\{0, 1\}$ indicating whether the patient has suffered from a heart failure or not. The dataset is provided by Kaggle and is denoted as: *Heart Failure Prediction Dataset* [1].

2.1.1 Preprocessing

Upon inspection, no outliers or missing values were found in the dataset. To prepare the dataset for modeling, the features $\mathbf{x}^{(i)} = (x_1^{(i)}, x_2^{(i)}, ..., x_{11}^{(i)})$ were separated from the target variable $y^{(i)}$ (HeartDisease), for each observation $i = 1, ..., 918$. Since the dataset combined both numerical and categorical variables, a `ColumnTransformer` was used as a preprocessing step. This made it possible to apply the appropriate transformations to each type of feature separately.

The numerical variables were standardized using `StandardScaler`, which transforms each feature variable according to

$$x' = \frac{x - \mu_j}{\sigma_j},$$

where μ_j is the mean and σ_j the standard deviation for a given feature j. This ensures that all numerical features have mean 0 and standard deviation 1, making them comparable in scale and improving the performance of distance-based algorithms such as k-NN.

The categorical variables were encoded using `OneHotEncoder`, which creates a binary indicator for each category. For example, if the variable `Sex` has two levels (`Male`, `Female`), two new columns are generated:

$$\texttt{Sex\_Male} \in \{0, 1\}, \quad \texttt{Sex\_Female} \in \{0, 1\}.$$

This is needed so that the models can interpret them numerically without imposing an artificial order, but also increases the dimensionality of the feature space.

The two steps were combined into a single `Pipeline`, ensuring that the same preprocessing was consistently applied during training and evaluation. The training/test split was set to 0.8/0.2, using stratified sampling to ensure that both the training and test sets ($X_{\text{train}}, y_{\text{train}}$ and $X_{\text{test}}, y_{\text{test}}$) contain the same proportion of each class. No validation set was used in this project, as model hyperparameters were tuned using cross-validation on the training set.

2.1.2 Dimensionality reduction

The use of `OneHotEncoder` for categorical variables increases the dimensionality of the feature space. This can negatively affect the performance of distance-based classifiers such as k-NN, as high-dimensional spaces amplify the so-called *curse of dimensionality*. To address this issue, Principal Component Analysis, PCA, was applied as a dimensionality reduction technique when developing a k-NN classifier.

PCA is a statistical method that transforms the original correlated features into a new set of uncorrelated variables, called principal components. These components are ordered such that the first captures the largest possible variance in the data, the second captures the next largest variance whilst being orthogonal to the first, and so on. By selecting only the first few principal components, PCA reduces the dimensionality of the dataset while retaining most of the relevant information. This can improve both computational efficiency and classification performance. [2]

2.2 Evaluation Strategy

The performance of each developed model was evaluated using several metrics:

- **Accuracy**: The proportion of correctly classified observations among all observations.

- **Precision**: The proportion of correctly predicted positive observations among all observations predicted as positive. High precision means few false positives.
- **Recall**: The proportion of correctly predicted positive observations among all actual positive observations. High recall means few false negatives.
- **F1-score**: The harmonic mean of *precision* and *recall*. It balances the trade-off between the two and can be useful when the class distribution is imbalanced.
- **Confusion matrix**: A tabular summary of predictions, showing the amount of true positives, false positives, true negatives, and false negatives.

Hyperparameters for the different models were tuned using cross-validation on the training set, before being exposed to the test-data.

2.3 Classification Methods

2.3.1 k-Nearest Neighbors, *k*-NN

The k-NN method a local averaging method which utilizes information of the k nearest points of a given input. It can be applied for classification and regression tasks and has previously been used for prediction of heart failure [3]. Given binary class labels the method classifies an input $\mathbf{x}$ as the majority class among its k nearest neighbors. Formally, we can write the prediction as

$$\hat{y}(\mathbf{x}) = \frac{1}{k} \sum_{\mathbf{x}_i \in N_k(\mathbf{x})} y_i,$$

where $y_i \in \{0, 1\}$ is the class label of a training point $\mathbf{x}_i$, and $N_k(\mathbf{x})$ denotes the set of the k training samples closest to $\mathbf{x}$. The class assigned to input $\mathbf{x}$ is then determined as class 1 if $\hat{y}(\mathbf{x}) > 0.5$ and as class 0 if $\hat{y}(\mathbf{x}) < 0.5$.

Since the method utilizes distances to choose the nearest neighboring data points, it is very important to preprocess and standardize data since some features may have larger ranges in values. Without standardization these may thus dominate the distance evaluation between points and introduce bias. Furthermore one can also choose different metrics for the distance between points. The most commonly used, and therefore chosen for the project, is the *euclidean distance* which between points $\mathbf{x} \in \mathbb{R}^p$ and $\mathbf{x}' \in \mathbb{R}^p$ is stated as

$$d(\mathbf{x}, \mathbf{x}') = \sqrt{\sum_{j=1}^{p} (x_j - x'_j)^2}$$

However it can be worth mentioning that other distances can also be implemented [4].

The value of k will affect the models bias-variance trade off and therefore needs to be chosen carefully. A very low value of k will lead to high variance and overfitting, whilst a larger k will introduce more bias. The tuning of k was made via 10-fold cross validation on the training set which led to the final decision of $k = 13$.

2.3.2 *k*-NN with PCA

A problem when applying k-NN in high-dimensional feature spaces is the *curse of dimensionality*. Formally, consider a dataset uniformly distributed in a p-dimensional unit hypercube. The volume of a hypersphere with radius r is given by

$$V_p(r) = \frac{\pi^{p/2}}{\Gamma\left(\frac{p}{2} + 1\right)} r^p,$$

where $\Gamma(\cdot)$ is the Gamma function. As p increases, $V_p(r)$ rapidly tends to zero for any fixed $r < 1$. This implies that in high dimensions, points become increasingly sparse and distances between them grow. As the number of dimensions increases, an increasingly larger proportion of the ball's

volume is concentrated near its surface. Consequently, the distinction between "nearest" and "farthest" neighbors diminishes, which amplifies noise and deteriorates the performance of a k-NN for increasing p. [5]

To mitigate these issues, PCA was applied before developing a k-NN model. By projecting the data onto a lower-dimensional subspace that preserves most of the variance, PCA reduces noise and redundancy while alleviating the negative effects of high dimensionality. This improves both the reliability and efficiency of the k-NN classifier, as described in section 2.3.1.

A gridsearch was performed to find the optimal dimension of the new feature space after PCA was applied to the input space, as well as the new value k for this classifier. Best results were found when the PCA was reduced until 98% of the variance was retained after the dimensionality reduction (which was a reduction from $\mathbb{R}^{21}$ to $\mathbb{R}^{14}$ in the feature space). The value of $k = 13$ was also used for this model.

2.3.3 Decision Trees

Decision Trees are supervised learning methods that recursively partition the feature space into regions where predictions can be made more homogeneous. At each split, the algorithm selects a feature and a threshold that maximizes the reduction of an impurity measure. Common impurity measures are the *Gini index* and *entropy*. For a node t with class proportions $p_{k|t}$ for class k, these are defined as

$$\text{Gini}(t) = \sum_{k=1}^{K} p_{k|t}(1 - p_{k|t}),$$

$$\text{Entropy}(t) = -\sum_{k=1}^{K} p_{k|t} \log p_{k|t}.$$

The recursive partitioning continues until a stopping criterion is met, such as a maximum tree depth or a minimum number of samples per leaf. A new sample is classified by traversing the tree from the root to a leaf, following the feature-based splits, and then assigning the class of the majority vote in that leaf. Since decision trees compare each feature separately in each branch, no normalization of the data is needed within this method. Decision trees also handles mixed data types naturally, so there is no need for encoding categorical features to make them comparable to numerical parameters.

2.3.4 Random Forest

A single decision tree often suffers from high variance, since small changes in the training data can lead to very different splits and predictions. This makes decision trees prone to overfitting and motivates the use of ensemble methods such as *bagging*, where multiple decision trees are trained on different bootstrap samples of the data and their predictions are averaged (or voted) to reduce variance.

Suppose each fitted tree can be expressed as

$$\hat{f}_b(x) = f^*(x) + \varepsilon_b(x),$$

where $f^*(x)$ is the true underlying function and $\varepsilon_b(x)$ is a random error with $\mathbb{E}[\varepsilon_b(x)] = 0$ and $\text{Var}(\varepsilon_b(x)) = \sigma^2(x)$. The bagged predictor for regression problems is the average over B trees:

$$\hat{f}_{\text{bag}}(x) = \frac{1}{B} \sum_{b=1}^{B} \hat{f}_b(x).$$

Since we are dealing with a classfication problem however, the bagged predictor will be the majority vote over the B different trees. If the errors are uncorrelated, the variance reduces as

$$\text{Var}\Big(\hat{f}_{\text{bag}}(x)\Big) = \frac{1}{B^2} \sum_{b=1}^{B} \text{Var}(\varepsilon_b(x)) = \frac{\sigma^2(x)}{B},$$

showing that averaging B trees can reduce variance by a factor of B when using bagging [6].

In practice however, the bootstrapped trees are not independent but correlated. If each tree has variance σ^2 and pairwise correlation $\rho > 0$, then the variance of the average becomes

$$\mathrm{Var}\left(\frac{1}{B}\sum_{b=1}^{B} Y_b\right) = \rho\sigma^2 + \frac{1-\rho}{B}\sigma^2,$$

where Y_b represents the prediction from tree b. As $B \to \infty$, the second term vanishes, but the variance is bounded below by $\rho\sigma^2$. This means bagging cannot eliminate variance completely when the base learners are correlated [6].

To further reduce variance, we must decrease the correlation (ρ) between individual trees. Random Forests achieve this by randomly selecting a subset of m predictors at each split, which helps to decorrelate the trees. As a result, Random Forests extend the idea of bagging by introducing an additional layer of randomness, yielding a more powerful and stable ensemble method that effectively reduces the variance of decision trees. This motivates the use of Random Forests in practice. See Appendix for full algorithm description section C.

Gini index was chosen as the impurity measure for the Random forest algorithm. Number of features of per tree, m, was set to 3 since a common rule of thumb for choosing the number of predictors m in a Random Forest is:

$$m = \begin{cases} \sqrt{p}, & \text{for classification problems,} \\ \frac{p}{3}, & \text{for regression problems,} \end{cases}$$

where p is the total number of available predictors [6]. A grid search with 10-fold cross-validation on the training set was then performed to identify the optimal number of trees, which was found to be n = 100.

2.3.5 Boosted decision trees (Adaboost)

Another approach to enhance the performance of decision trees is *boosting*, which builds weak learners (smaller decision trees in this case) to iteratively improve predictive accuracy. Unlike Random Forests, which reduce variance by averaging many independent trees, boosting reduces bias by fitting each new tree using information from the previous tree. Within this project, the boosting algorithm *Adaboost* was implemented, starting from equal weights on observations, the algorithm increases emphasis on misclassified or hard cases and combines all learners via a weighted vote that gives more influence to the more accurate learners [6]. The AdaBoost algorithm can be interpreted as performing gradient descent on the exponential loss function. For a more detailed explanation of the algorithm, see Appendix section D.

Boosting often results in a more overfitted model that might be sensitive to noise as opposed to RFC since this method focuses on reducing bias and not variance. Because of the sensitivity, tuning the parameters is crucial.

Cross validation was performed to decide the complexity of the weak learners (maximum tree depth), learning rate ν as well as the number of trees M. The optimal values were found to be a maximum tree depth of 1, a learning rate of 0.1, and 100 trees.

2.3.6 Neural Networks

Neural networks (NNs) are powerful models for capturing non-linear relationships with the idea based on biological neurons. It works by layering nodes that are interconnected and transform the inputs through weighted sums and non-linear activiations [7, 8]. Moreover, in healthcare they have also been shown to outperform simpler methods in predictions related to heart failure risk [10]. In this project we will use a feedforward Multi-Layer Perceptron (MLP) because it should provide a balance between the model complexity and its interpretability. We will not need convolutional or recurrent networks here because our data is just tabular with 11 features per patient. And because the features are not time-dependent or in any particular order there is no reason to model sequences. Therefor, an MLP is a good choice for tabular medical data, as it allows us to model non-linear feature interactions while keeping the structure relatively simple and computationally efficient [11].

Each patient's 11 features form the input vector $X \in \mathbb{R}^{11}$. A hidden unit is computed as

$$Z_m = \sigma(\alpha_{0m} + \alpha_m^T X), \quad m = 1, \ldots, M,$$

where α_{0m} is a bias term and $\sigma(\cdot)$ is the activation function. We use the Rectified Linear Unit (ReLU),

$$\sigma(z) = \max(0, z),$$

because it helps with gradient propagation and speeds up training compared to the sigmoid or $\tanh$ [12]. The derivative of the ReLU function is larger than the derivative of the $\tanh$ and sigmoid functions for most input values. This results in larger update steps during the gradient descent algorithm, allowing the network to locate a minimum of the cost function more quickly. Another advantage of using ReLU is that its derivative is very simple to compute, which makes the overall calculations more efficient.

The output layer applies the sigmoid function

$$\phi(z) = \frac{1}{1 + e^{-z}},$$

to produce a probability in $[0, 1]$ that can be directly interpreted as the likelihood of heart failure.
The neural network is trained using the binary cross-entropy loss:

$$R(\theta) = -\frac{1}{N} \sum_{i=1}^{N} \big[y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i) \big],$$

which is the standard loss for binary classification. We optimize this with Adam which is an adaptive gradient-based optimizer that works well for smaller datasets, which our dataset is, and requires little tuning [13]. Training is carried out with mini-batches, and gradients are computed through backpropagation [14].
Since our dataset is relatively small ($N = 918$), we risk overfitting if the network is too flexible. To counteract this, we include:

- **Early stopping**: training stops once validation loss starts to increase, which prevents the network from memorizing noise [9].
- **Weight decay (L_2 penalty)**: shrinking weights towards zero encourages simpler models and avoids extreme parameter values,

$$J(\theta) = \sum_{k,m} \beta_{km}^2 + \sum_{m,\ell} \alpha_{m\ell}^2.$$

- **Dropout**: randomly turning off neurons during training to aim for making the network more robust and to prevent neurons from relying too much on each other [15].

Cross-validation was performed through grid search to determine the architecture and training parameters of the MLP. The number of hidden layers and neurons, dropout rate, learning rate, weight decay, batch size, and early-stopping patience were varied. An initial broad grid search was done to find the best regions of the hyperparameter space, followed by a refined search around the best-performing configurations. The optimal setup was found to consist of two hidden layers with 48 and 24 neurons, a dropout rate of 0.26, a learning rate of 2.1×10^{-3}, weight decay of 1×10^{-5}, a batch size of 32, and an early-stopping patience of 12 epochs. These design decisions are in accordance with the bias–variance trade-off discussed in Geman, Bienenstock, and Doursat [16].

2.3.7 Support Vector Machines (SVM)

Support Vector Machines' goal is to find a hyperplane that maximizes the margin between classes, making them less sensitive to noise [17, 9]. This means that only the closest points, i.e., the support vectors, affect the decision boundary, which gives SVMs good generalization properties. Moreover, our dataset consists of 918 patients with 11 features each, which is small enough that NNs may overfit, while k–NN may not capture complex relationships, and SVMs could provide a good trade-off because they are robust and, in addition, can model non-linearities through the kernel trick [18].

The primal optimization problem to maximize the margins can be written as

$$\min_{\mathbf{w},\, b} \frac{1}{2}\|\mathbf{w}\|^2 \quad \text{subject to} \quad y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1,\ \forall i.$$

This problem is equivalent to its dual form

$$\max_{\boldsymbol{\alpha}} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^\top \mathbf{x}_j,$$

where the optimization depends only on scalar products $\mathbf{x}_i^\top \mathbf{x}_j$. This observation is crucial: even if we map the data to a higher-dimensional feature space, the optimization still only involves scalar products. Hence, by replacing these scalar products with a kernel function

$$K(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^\top \phi(\mathbf{x}'),$$

we can efficiently compute the solution in high-dimensional spaces without explicitly performing the mapping. We will compare two types of kernels. With the linear kernel

$$K(x, x') = x^\top x'$$

(so $\phi(x) = x$), we get the decision function

$$f(x) = \beta^\top x + \beta_0$$

and the classifier

$$\hat{y} = \text{sign}(f(x)).$$

The decision boundary is the hyperplane

$$H = \{x \in \mathbb{R}^{11} : f(x) = 0\}.$$

In the dual, we have

$$\beta = \sum_{i=1}^{N} \alpha_i y_i x_i.$$

This will act as our baseline, and if the data are close to linearly separable it will be good enough and less likely to overfit. But if this is not the case, the Radial Basis Function (RBF) kernel is more flexible. It is defined as

$$K(x, x') = \exp(-\gamma \|x - x'\|^2), \quad \gamma = \frac{1}{2\sigma^2}, \quad \gamma > 0,$$

where γ is a hyperparameter. The RBF kernel measures similarity between two points, so if x and x' are close, then $K(x, x')$ is near 1, and if they are far apart, the value goes towards 0. In practice this means that each point, for a small γ, influences a wide region and gives smoother decision boundaries, while a large γ makes the model focus locally, which in turn can lead to very complex and overfitted boundaries [19].

The other key parameter is C, which controls the trade-off between the margin size and misclassification. In the soft-margin formulation, the optimization problem is

$$\min_{\beta, \beta_0, \xi} \frac{1}{2}\|\beta\|^2 + C \sum_{i=1}^{N} \xi_i$$
$$\text{subject to } y_i\big(\beta^\top \phi(x_i) + \beta_0\big) \geq 1 - \xi_i, \quad \xi_i \geq 0,$$

Here, ϕ denotes the feature map, used implicitly via the kernel (i.e., $K(x, x') = \langle \phi(x), \phi(x') \rangle$), for the linear kernel we have $\phi(x) = x$. The regularization parameter C controls the trade-off between margin size and violations: a large C imposes a stronger penalty on violations (risk of overfitting), whereas a smaller C permits more violations and can improve generalization [9].

Cross-validation was performed through grid search to tune the SVM hyperparameters. A broad search over kernel type, class weights, and the parameters C and γ was followed by a refined search around the best-performing region. The optimal model was an RBF SVM with $C = 1.0$, $\gamma =$ `auto` (≈ 0.0476), and `class_weight = balanced`.

2.3.8 Gaussian Processes Classifier

Gaussian Processes, GPs, can be utilized to find probability distributions over functions. By definition a Gaussian process is a collection of random variables such that any finite subset has a joint Gaussian distribution. One usually writes

$$f(\mathbf{x}) \sim \mathcal{GP}\big(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}')\big).$$

where $m(\mathbf{x}) = \mathbb{E}[f(\mathbf{x})]$ is the mean function and $k(\mathbf{x}, \mathbf{x}') = \mathbb{E}[(f(\mathbf{x}) - m(\mathbf{x}))(f(\mathbf{x}') - m(\mathbf{x}'))]$ is the so-called covariance or kernel function. This means that for any finite set of inputs $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$,

$$\mathbf{f} \;=\; \big(f(\mathbf{x}_1), \ldots, f(\mathbf{x}_n)\big)^\top \sim \mathcal{N}\big(\mathbf{m},\, \mathbf{K}\big),$$

with mean vector $\mathbf{m} \;=\; \big(m(\mathbf{x}_1), \ldots, m(\mathbf{x}_n)\big)^\top$ and Gram matrix $\mathbf{K} \;\in\; \mathbb{R}^{n \times n}$ given by $\mathbf{K}_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$ [20].

GPs can be applied both for regression and classification tasks, although the implementation is simpler for the regression case. In regression, the goal is to find the distribution of function values $\mathbf{f}$ given inputs $\{\mathbf{x}\}_{i=1}^n$, which map to outputs $\{y\}_{i=1}^n$. This is done by assuming the prior for the function as $f(\mathbf{x}) \sim \mathcal{GP}\big(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}')\big)$ where it's common practice to set the mean function $m(\mathbf{x})$ to zero and the function $k(\mathbf{x}, \mathbf{x}')$ as a kernel function of choice. Then by conditioning on observed data using Bayesian inference, one can get a Gaussian posterior predictive distribution for the functions.

However for binary classification one wants to find class probabilities given observed data. Letting the target values be $y \in \{0, 1\}$ and by placing a GP prior for a latent function $f(\mathbf{x})$ one then applies the logistic function to connect the latent function to class probabilities

$$\pi(\mathbf{x}) := p(y = 1 \mid f(\mathbf{x})) = \sigma(f(\mathbf{x})) = \frac{1}{1 + \exp\big(-f(\mathbf{x})\big)}$$

[20]. This means that the likelihood is a Bernoulli distribution such that.

$$y \mid f(\mathbf{x}) \sim \text{Ber}(\sigma(f(\mathbf{x}))) \Rightarrow p(y \mid f(\mathbf{x})) = \pi(\mathbf{x})^y (1 - \pi(\mathbf{x}))^{1-y}$$

Further more assuming each data point is independently drawn it can be concluded that the full likelihood is

$$p(\mathbf{y} \mid \mathbf{f}) = \prod_{i=1}^{n} p(y_i \mid f(\mathbf{x}_i)).$$

Denoting $\tilde{\mathbf{x}} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$, Bayes theorem gives the posterior over latent function values $\mathbf{f}$ as

$$p(\mathbf{f} \mid \tilde{\mathbf{x}}, \mathbf{y}) \;=\; \frac{p(\mathbf{y} \mid \mathbf{f})\, p(\mathbf{f} \mid \tilde{\mathbf{x}})}{p(\mathbf{y} \mid \tilde{\mathbf{x}})}.$$

Unlike the regression case this likelihood is non-Gaussian, which causes this posterior to be analytically intractable and requires approximate inference methods such as Laplace approximation, Expectation Propagation, or other methods. [20].

For the scope of the project the Laplace approximation was deemed fit, due to it's simple implementation. The idea is to approximate the true posterior with a Gaussian denoted $q(\mathbf{f} \mid \tilde{\mathbf{x}}, \mathbf{y})$, centered at the maximum a posteriori estimate, MAP

$$\hat{\mathbf{f}} = \arg\max_{\mathbf{f}} \log p(\mathbf{f} \mid \tilde{\mathbf{x}}, \mathbf{y})$$

This is often found with iterative methods e.g. using Newton–Raphsons. Around this mode, the log posterior is approximated by a second-order Taylor expansion, yielding a Gaussian approximation

$$q(\mathbf{f} \mid \tilde{\mathbf{x}}, \mathbf{y}) = \mathcal{N}(\hat{\mathbf{f}}, (\mathbf{K}^{-1} + \mathbf{W})^{-1}),$$

where $\mathbf{W} = -\nabla\nabla p(\mathbf{y} \mid \mathbf{f})$ is a diagonal matrix with entries from the second derivatives of the log likelihood. This Gaussian serves as a tractable approximation to the true posterior, and can be used for approximate predictive inference.

Predictions at a new test point $\mathbf{x}^*$ can then be obtained in two steps. First the posterior from the Laplace approximation at input $\mathbf{x}^*$ is a Gaussian with closed form mean and variance s.t.

$$f^* \mid \mathbf{x}^*, \tilde{\mathbf{x}}, \mathbf{y} \ \sim \ \mathcal{N}(\mu^*, \sigma^{*2}),$$

the expression for μ^*, σ^{*2} can be seen in section A. This latent variable is then mapped to a class probability by the integral

$$p(y^* = 1 \mid \mathbf{x}^*, \tilde{\mathbf{x}}, \mathbf{y}) \ = \ \int \sigma(f^*)\, q(f^* \mid \mathbf{x}^*, \tilde{\mathbf{x}}, \mathbf{y})\, df^*.$$

However this integral has no closed form, but in practice it is approximated where the choice of method may vary[20].

The choice of kernel for implementing the method is quite crucial where one should consider previous knowledge about the possible functions that would be able to fit the data. There are many kernels that can be implemented but for the scope of the project only combinations of Radial Basis kernels, RBF, Matérn kernels, constant kernels and white noise kernels were considered. Their closed form expression including some explanations can be seen in section B.

The kernels that were considered consisted of a RBF kernel combined with a constant and a noise kernels. As well as two Matérn kernels ($\nu = 3/2$ and $\nu = 5/2$) both also combined with a constant and a noise kernel. Using 10-fold cross validation the best kernel combination was determined to be

$$k_{CV} = 4.23^2 \cdot k_{\text{RBF}}(\sigma_f^2 = 4.97) + k_{\text{white}}(\sigma_n^2 = 1.8 \cdot 10^{-7})$$

Since the white noise kernel had negligible variance the kernel for the final model was chosen as

$$k_{final} = 4.23^2 \cdot k_{\text{RBF}}(\sigma_f^2 = 4.97)$$

3 Results

In this chapter we present the evaluation results for all implemented methods. For each model, we report Accuracy, as well as the *weighted average* of F1-score, Precision, and Recall across the two classes on the test set.

Method	Acc	F1	Prec	Rec
k-NN	0.918	0.92	0.92	0.92

Table 1: k-NN

Method	Acc	F1	Prec	Rec
k-NN+PCA	**0.929**	0.93	0.93	0.93

Table 2: k-NN with PCA

Method	Acc	F1	Prec	Rec
RF	0.902	0.90	0.90	0.90

Table 3: Random Forest

Method	Acc	F1	Prec	Rec
AdaB	0.897	0.90	0.90	0.89

Table 4: AdaBoost

Method	Acc	F1	Prec	Rec
SVM	0.902	0.902	0.902	0.902

Table 5: Support Vector Machine

Method	Acc	F1	Prec	Rec
NN	0.880	0.880	0.882	0.880

Table 6: Neural Network

Method	Acc	F1	Prec	Rec
GP	0.891	0.89	0.89	0.89

Table 7: Gaussian Process Classifier

For confusion matrices of all methods, see Appendix section E.

4 Conclusion

In this project we evaluated several different machine learning methods for predicting heart failure using the *Heart Failure Prediction Dataset* [1]. Among the tested models, k-NN with PCA achieved the highest accuracy (92.9 %), followed by k-NN without PCA (91.8 %). Whereas the esamble methods such as Random Forests and AdaBoost, and SVMs had around 90 % accuracy. The more flexible methods, Neural Network and Gaussian Processes, performed slighly worse at around 88–90 %. These results highlights that in the context of relativley small tabular datasets, simple classifier combined with effective preprocessing can outperform the more flexible and complex models. In addition, we could observe that applying PCA to k-NN further highlights the importance of adressing the problem with dimensionality and reducing noise in the feature space.

Moreover, while the results for all methods have and high accuracy, there are several limitations. The dataset consisted of only 918 patients, which limits the generalization of the results and increase the risk of overfitting. The more complex models, such as Neural Networks, are particularly sensitive to this constraint which may be a reason for its lower performance rather than an inherent weakness of the approach to use Neural Networks to detect heart failure. This outcome is in contrast with some previous studies suggesting that deep Neural Networks can outperform simpler classifiers on heart failure risk [10]. These results underscores that model performance is dependent on the given context. Given limited data, methods with fewer parameters or those leveraging bias may excel, whereas more data heavy approaches may not reach their full potential. This highlights the significance of bias-variance trade-off where simpler models with higher bias but lower variance can often generalize better in data-scarce environments while high-capacity models risk overfitting noise rather than learning important patterns. The results therefore demonstrate that model choice should be guided not only by theoretical expressiveness but also by data size and structure.

In addition model interpretability is an important aspect, since it impacts how models can be used in reality. The best performing methods for the project such as k-NN and SVMs are relatively non-transparent in terms of how predictions are made. This can limit their relevance and acceptance in healthcare contexts. Tree-based methods like Random Forests and AdaBoost offer more insight on feature importance, altough they too do not have an intuitive clarity of the conclusion and choices within the model which would be beneficial for medical decision making. To make these models more useful in practice, one should strive to couple strong models with tools that can explain their classification in simpler terms. This can include summaries of feature importance, decision plots etc. By having these it would be easier for medical staff to evaluate weather a model prediction is reasonable or not.

In conclusion, the models developed in this project are capable of capturing important patterns in the data to identify patients at risk of heart failure. This shows the potential of machine learning as a vaild tool to support early diagnosis which give a chance for preventive healthcare. But to make these predictions appliacable in real settings, future work should focus on improving scalability, interpretability and validation. Scalability is essential to be able to handle larger and more diverse datasets while interpretability ensures that researchers and doctors can draw trustworthy, well-supported conclusions. Lastly, external validation on independent data would be necessary to be sure that the models remain accurate and reliable when applied in real world medical environments.

References

[1] fedesoriano. (September 2021). *Heart Failure Prediction Dataset*. Retrieved September 2025 from `https://www.kaggle.com/fedesoriano/heart-failure-prediction`

[2] I. T. Jolliffe and J. Cadima. Principal component analysis: a review and recent developments. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 374(2065):20150202, 2016. doi:10.1098/rsta.2015.0202.

[3] D. Rahmat, A. A. Putra, Hamrin, and A. W. Setiawan. Heart Disease Prediction Using K-Nearest Neighbor. In *2021 International Conference on Electrical Engineering and Informatics (ICEEI)*, pages 1–6. IEEE, 2021. doi:10.1109/ICEEI52609.2021.9611110. `https://ieeexplore.ieee.org/document/9611110`.

[4] L.-Y. Hu, M.-W. Huang, S.-W. Ke, and C.-W. Tsai. The distance function effect on k-nearest neighbor classification for medical datasets. *SpringerPlus*, 5:1304, 2016. doi:10.1186/s40064-016-2941-7. `https://doi.org/10.1186/s40064-016-2941-7`.

[5] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006. Available online.

[6] S. H. Lim. Lecture 9: Tree-Based Methods and Ensembling Methods. Department of Mathematics, KTH Royal Institute of Technology, Stockholm, September 23, 2025. Lecture slides.

[7] Y. LeCun, Y. Bengio, and G. Hinton. Deep Learning. *Nature*, 521:436–444, 2015.

[8] IBM Think. Neural Networks: What They Are and How They Work, 2024. `https://www.ibm.com/think/topics/neural-networks`

[9] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2nd ed., corrected 12th printing, 2017.

[10] S. Hajishah et al. Evaluation of machine learning methods for prediction of mortality in heart failure patients. *PLoS One*, 2025. `https://pmc.ncbi.nlm.nih.gov/articles/PMC11974104/`

[11] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. Chapter 6.

[12] V. Nair and G. E. Hinton. Rectified Linear Units Improve Restricted Boltzmann Machines. In *ICML*, 2010.

[13] D. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. In *ICLR*, 2015.

[14] D. Rumelhart, G. Hinton, and R. Williams. Learning Representations by Back-Propagating Errors. *Nature*, 323:533–536, 1986.

[15] N. Srivastava et al. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *JMLR*, 15:1929–1958, 2014.

[16] S. Geman, E. Bienenstock, and R. Doursat. Neural Networks and the Bias/Variance Dilemma. *Neural Computation*, 4(1):1–58, 1992.

[17] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20:273–297, 1995.

[18] B. Schölkopf and A. Smola. *Learning with Kernels*. MIT Press, 2002.

[19] C. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[20] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. MIT Press, Cambridge, MA, 2006. `http://www.gaussianprocess.org/gpml/chapters/RW2.pdf`

Appendix

A Predictive mean and variance under the Laplace approximation

Here $\tilde{\mathbf{x}} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ denotes all observations of inputs $\mathbf{x}_i$, $i = 1, ..., n$. For a new test input $\mathbf{x}^*$ with latent value $f^* = f(\mathbf{x}^*)$, the predictive distribution is Gaussian.

$$f^* \mid \mathbf{x}^*, \tilde{\mathbf{x}}, \mathbf{y} \;\sim\; \mathcal{N}(\mu^*, \sigma^{*2}),$$

with mean

$$\mu^* = \mathbf{k}_*{}^\top \mathbf{K}^{-1} \hat{\mathbf{f}},$$

and variance

$$\sigma^{*2} = \mathbf{k}_{**} - \mathbf{k}_*{}^\top (\mathbf{K} + W^{-1})^{-1} \mathbf{k}_*.$$

Here:

- $\mathbf{K} \in \mathbb{R}^{n \times n}$ is the kernel matrix on the training inputs $\tilde{\mathbf{x}}$,
- $\mathbf{k}_* = k(\mathbf{x}^*, \tilde{\mathbf{x}})$ is the n-dimensional vector of covariances between $\mathbf{x}^*$ and each training input
- $\mathbf{k}_{**} = k(\mathbf{x}^*, \mathbf{x}^*)$ is the prior variance at the test input
- $\hat{\mathbf{f}} = \arg\max_{\mathbf{f}} \log p(\mathbf{f} \mid \tilde{\mathbf{x}}, \mathbf{y})$ is the MAP estimate of the latent function values.

[20]

B Kernel definitions and notes

Throughout, let $r = \|\mathbf{x} - \mathbf{x}'\|$ denote the Euclidean distance, σ_f^2 the signal variance, and ℓ the characteristic length–scale.

- **Radial Basis Function (RBF)**:

$$k_{\mathrm{RBF}}(\mathbf{x}, \mathbf{x}') = \sigma_f^2 \exp\!\Big(-\tfrac{r^2}{2\ell^2}\Big).$$

- **Matérn family** With smoothness parameter $\nu > 0$ and modified Bessel function K_ν,

$$k_{\text{Matérn},\nu}(\mathbf{x}, \mathbf{x}') = \sigma_f^2 \, \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\frac{\sqrt{2\nu}\, r}{\ell} \right)^{\nu} K_\nu \left(\frac{\sqrt{2\nu}\, r}{\ell} \right).$$

  Common half-integer special cases (simpler closed forms):

$$k_{\nu=\frac{3}{2}}(\mathbf{x}, \mathbf{x}') = \sigma_f^2 \Big(1 + \tfrac{\sqrt{3}\, r}{\ell}\Big) \exp\!\Big(-\tfrac{\sqrt{3}\, r}{\ell}\Big),$$

$$k_{\nu=\frac{5}{2}}(\mathbf{x}, \mathbf{x}') = \sigma_f^2 \Big(1 + \tfrac{\sqrt{5}\, r}{\ell} + \tfrac{5r^2}{3\ell^2}\Big) \exp\!\Big(-\tfrac{\sqrt{5}\, r}{\ell}\Big).$$

- **Constant kernel** (assumes constant covariance between all points):

$$k_{\mathrm{const}}(\mathbf{x}, \mathbf{x}') = \sigma_c^2$$

- **White noise kernel** Models independent noise on each observation:

$$k_{\mathrm{white}}(\mathbf{x}, \mathbf{x}') = \sigma_n^2 \delta_{\mathbf{x}\mathbf{x}'},$$

  where $\delta_{\mathbf{x}\mathbf{x}'}$ is the Kronecker delta function

$$\delta_{\mathbf{x}\mathbf{x}'} = \begin{cases} 1, & \mathbf{x} = \mathbf{x}', \\ 0, & \mathbf{x} \neq \mathbf{x}'. \end{cases}$$

Practical notes on kernel choice

- **Smoothness:** The RBF kernel implies functions that are infinitely differentiable (very smooth). The Matérn family however allows control over smoothness with the parameter ν. Smaller values leads to more roughness, while as $\nu \to \infty$ the kernel will be more like the RBF kernel. Choices of ν=$3/2$ or ν=$5/2$ are often good practical choices [20].

- **Length–scale:** Larger ℓ corresponds to smoother, slowly varying functions while smaller values allows more rapid variation.

- **Signal variance:** σ_f^2 and σ_c^2 controls the overall vertical scale of variation for the different kernels.

- **Noise:** The white noise kernel adds independent variance σ_n^2 to account for measurement noise.

- **Kernel combinations.** In practice, sums and products of the above kernels are used to model richer covariance structures. For example, an RBF kernel plus white noise models smooth functions with noisy observations, while adding a constant kernel allows for a global bias term.

C Random Forest algorithm

Algorithm 1 Random Forest

0: **for** $b = 1, \ldots, B$ **do**
0: Draw bootstrap sample Z^* of size N from training data.
0: Grow a random-forest tree T_b on Z^* by recursively repeating until minimum node size is reached:
- Select m variables at random from the p variables.
- Pick the best variable and split among the m.
- Split the node into two daughter nodes.

0: **end for**
0: Output the ensemble of trees $\{T_b\}_{b=1}^{B}$. =0

D Adaboost algorithm

Algorithm 2 AdaBoost

Input: Training data $\{(x_i, y_i)\}_{i=1}^{N}$, where $y_i \in \{-1, +1\}$.
Initialize observation weights $w_i^{(1)} = \frac{1}{N},\ i = 1, \ldots, N$.
for $m = 1$ to M **do**
Fit a weak classifier $G_m(x)$ using weights $w_i^{(m)}$.
Compute the weighted error:

$$\text{err}_m = \frac{\sum_{i=1}^{N} w_i^{(m)} I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i^{(m)}}.$$

Compute the classifier weight:

$$\alpha_m = \log\left(\frac{1 - \text{err}_m}{\text{err}_m}\right).$$

Update observation weights:

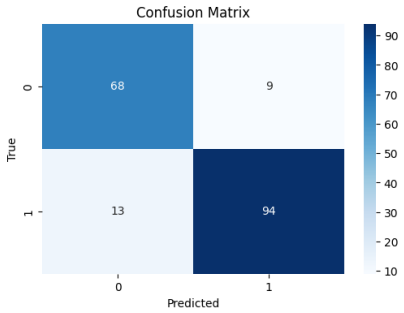
$$w_i^{(m+1)} = w_i^{(m)} \exp\big(\alpha_m I(y_i \neq G_m(x_i))\big), \quad i = 1, \ldots, N.$$

Normalize weights so that $\sum_{i=1}^{N} w_i^{(m+1)} = 1$.
end for
Output: Final classifier

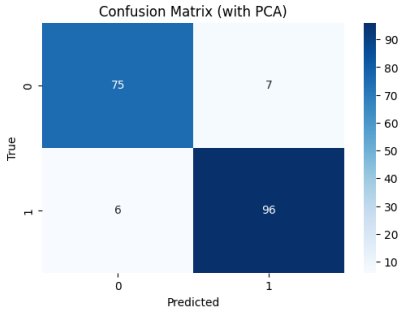
$$G(x) = \text{sign}\left(\sum_{m=1}^{M} \alpha_m G_m(x)\right).$$

=0

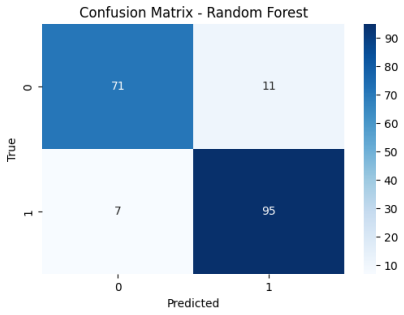
E Confusion matrices of applied models



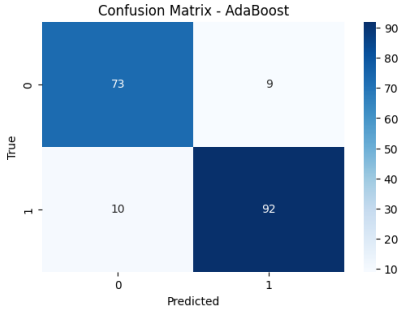
(a) k-NN



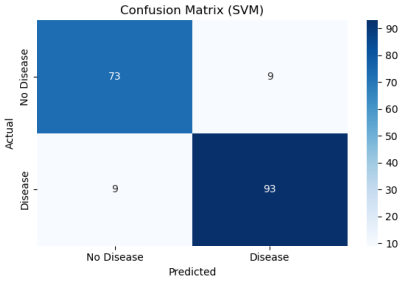
(b) k-NN + PCA



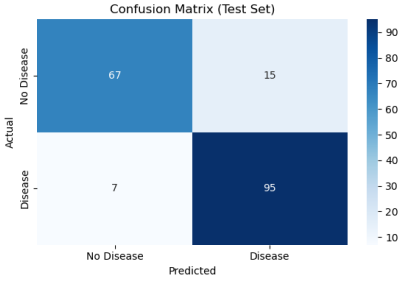
(c) Random Forest



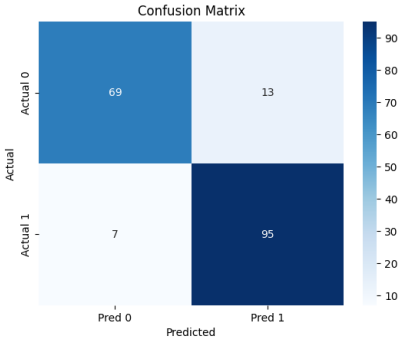
(d) AdaBoost



(e) Support Vector Machine



(f) Neural Network



(g) Gaussian Process Classifier

Figure 1: Confusion matrices for the evaluated classifiers.