UiO : Department of Physics
University of Oslo

FYS-STK4155: Applied Data Analysis and Machine Learning

Project 3

Machine Learning Classification Methods for Diagnosis of Breast Cancer

Oda Hovet Ilse Kuperus Erik Alexander Sandvik

December 17, 2020

Abstract

In this article we investigate the possibility of utilizing machine learning classification algorithms to train models that can be used to detect malignant tumors in breast cancer patients. Trained predictive models can potentially help giving cancer patients an earlier diagnosis and thus earlier and more effective treatment with less severe side effects. We use the Wisconsin Breast Cancer dataset to train models using support vector machines, logistic regression, decision trees and feedforward neural networks. We find that models trained using logistic regression and gradient boosting of decision trees can accurately classify 95.8% and 97.9% of the test data respectively. A model trained using support vector machines can accurately classify 98% of the test data with a standard deviation of 1% by using a linear kernel function and tuning a regularization parameter. After employing a feature selection we find that a neural network can correctly classify 97.6% of the test data with a standard deviation of 0.03%. We argue from these results that trained predictive models can supplement the process of diagnosing cancer patients.

Contents

1 Introduction **1**

2 Theory and Method **1**
2.1 Support Vector Machines . . . 1
2.1.1 Linearly Separable Case . . . 2
2.1.2 Non-separable case . . . 5
2.1.3 The Kernel Method . . . 5
2.2 Logistic Regression . . . 6
2.2.1 Stochastic Gradient Descent . . . 7
2.3 Decision Trees . . . 8
2.4 Gradient Boosting and XGBoost . . . 10
2.5 Feedforward Neural Networks . . . 11
2.5.1 Grid Search and Feature Selection . . . 12

3 Results and Discussion **13**
3.1 Logistic regression . . . 13
3.2 Decision Trees . . . 14
3.3 XGBoost . . . 16
3.4 Support Vector Machines . . . 17
3.5 Feedforward Neural Networks . . . 19
3.5.1 Grid Search . . . 20
3.5.2 Feature Selection . . . 21

4 Conclusion **25**

References **26**

1 Introduction

Machine learning is a powerful tool for tuning models using old data to make predictions of new data. Classification algorithms have been used to train models that can detect spam mail at a mean accuracy as high as 99.7% [1], or that can recognize digits from images of handwritten digits at a mean accuracy of 96% [2]. In this article we investigate the ability of various classification algorithms to make predictive models that can be put to use in the medical field, specifically to detect cancerous breast tumors based on a number of tumor features. To train the models we use the Wisconsin Breast Cancer data set, which is available in scikit-learn [3]. It contains 569 samples of 30 features each, like the mean tumor radius and symmetry of the tumor, where 212 of the samples belong to the class malignant (M) and 357 belong to the class benign (B). The classification methods we use are support vector machines, logistic regression, decision trees and feed-forward neural networks, which we implement using the Python module scikit-learn [4].

If successful, applying machine learning methods for diagnostics in the medical field can have a wide range of benefits. Most importantly, it can save lives by giving an early diagnosis, which is essential for treating diseases like breast cancer where treatment must start as early as possible. Early treatment can also reduce the total amount of treatment needed, limiting the patients exposure to radiotherapy and chemotherapy which can have severe and life-long side effects. It can also potentially limit the number of misdiagnoses so that patients don't get unnecessary treatment while also freeing up resources.

In order to avoid overfitting and misdiagnoses, we stress the importance of a model's ability to generalize to all potential data from just a small subset of training data. Therefore we use K-fold cross validation to see how the respective models perform on several subsets of the available data and we'll also see how scaling the input data affects the performance. The metric we use to evaluate a model's performance is the mean accuracy, i.e the fraction of correctly classified tumors, and we perform scaling of the input data as described by the documentation of the StandardScaler class in scikit-learn [5].

This project is structured by first presenting an overview of the aforementioned machine learning algorithms as well as optimization methods and improved performance schemes in Section 2. This is followed by a presentation and discussion of the results in Section 3. Lastly, in Section 4 the results are concluded upon and an outline of possible continuations are presented.

2 Theory and Method

2.1 Support Vector Machines

In this section we briefly explain the Support Vector Machine (SVM) classification algorithm for binary classification. We consider a set of training data $\{(\mathbf{x}_i, y_i)\}_{i=1}^m$ where each vector $\mathbf{x}_i \in \mathbb{R}^N$ is a collection of N features and is labelled $y_i \in \{-1, +1\}$. The basic idea of SVMs is that we can find some boundary in $\mathbb{R}^N$which "best" separates the vectors labelled -1 from the vectors labelled $+1$. We first consider boundaries that are hyperplanes, before we go on to more complicated non-linear boundaries.

2.1.1 Linearly Separable Case

To start out simple we will first consider the case where the vectors $\mathbf{x}_i$ are *linearly seperable*. This means that we can find a hyperplane $\mathbf{w} \cdot \mathbf{x} + b = 0,\ \mathbf{w} \neq \mathbf{0}$ such that all the vectors labelled -1 fall on one side and all the vectors labelled $+1$ fall on the other. More specifically, we want to find $\mathbf{w}$ and b such that all the vectors labelled -1 satisfy $\mathbf{w} \cdot \mathbf{x}_i + b < 0$ and all the vectors labelled $+1$ satisfy $\mathbf{w} \cdot \mathbf{x}_i + b \geq 0$. We can write this condition compactly as

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 0,\ \forall i \in [m]. \tag{2.1}$$

Once we've found $\mathbf{w}$ and b that satisfies this condition for all the training data, we can label *new* data using a linear classifier

$$h(\mathbf{x}) = \operatorname{sgn}(\mathbf{w} \cdot \mathbf{x} + b).$$

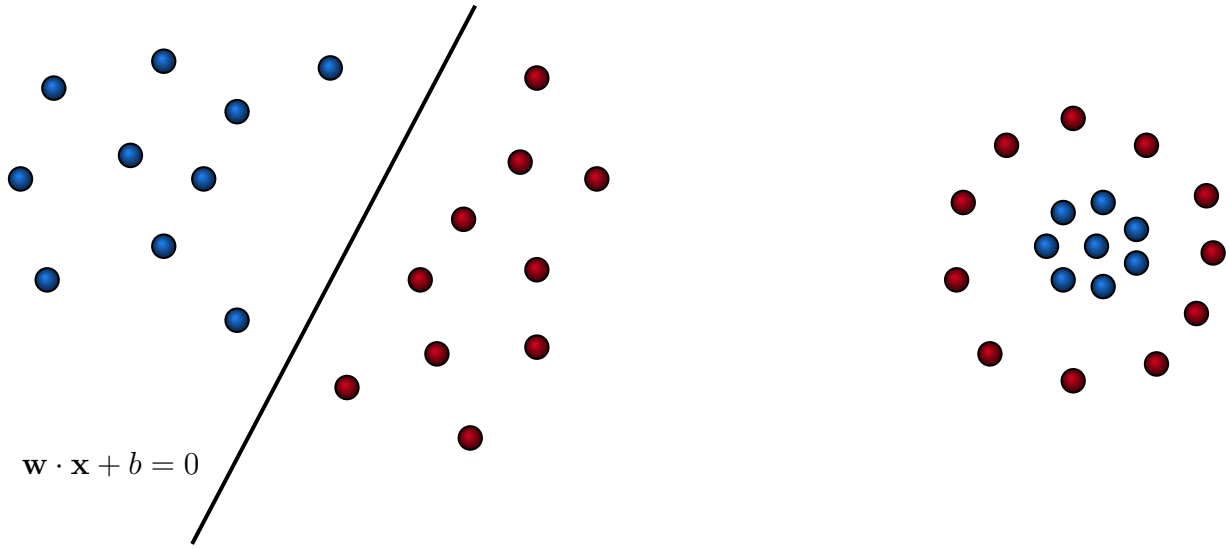


Figure 2.1: ***Left****: A linearly separable training set.* ***Right****: A non-separable training set.*

However, from Figure 2.1 we realize that there are infinitely many hyperplanes which satisfies the condition, so which one do we choose? It might be tempting to use *any* hyperplane that satisfies the condition, but misclassification of new data is always a possibility. And intuitively, it would make sense that in order to avoid misclassification, we must not pick a hyperplane too close to either the vectors labelled -1 or $+1$, but square in the middle. This extra condition restricts us to a single hyperplane. An useful quantity for us in order to solve this problem is the so-called geometric margin.

Definition of the Geometric margin: The geometric margin $\rho_h(\mathbf{x})$ of a linear classifier $h : \mathbf{x} \mapsto \operatorname{sgn}(\mathbf{w} \cdot \mathbf{x} + b)$ is the shortest Euclidean distance between $\mathbf{x}$ and the hyperplane $\mathbf{w} \cdot \mathbf{x} + b = 0$.

To find an explicit expression for $\rho_h(\mathbf{x})$, consider the orthogonal projection $\hat{\mathbf{x}}$ of the point $\mathbf{x}$ onto the hyperplane. The distance between $\hat{\mathbf{x}}$ and $\mathbf{x}$ is $\rho_h(\mathbf{x})$ by definition. The direction of the vector $\mathbf{x} - \hat{\mathbf{x}}$ is normal to the hyperplane, so it is given by the normal vector $\mathbf{w}$. Thus we have

$$\rho_h(\mathbf{x})\frac{\mathbf{w}}{|\mathbf{w}|} = \mathbf{x} - \hat{\mathbf{x}}.$$

Since the point $\hat{\mathbf{x}}$ lies in the hyperplane, it satisfies the equation $\mathbf{w}\cdot\hat{\mathbf{x}} + b = 0$, or $\mathbf{w}\cdot\hat{\mathbf{x}} = -b$. Taking the dot product of $\mathbf{w}$ and both sides of the equation we get

$$\rho_h(\mathbf{x})\frac{\mathbf{w}\cdot\mathbf{w}}{|\mathbf{w}|} = \mathbf{w}\cdot\mathbf{x} - \mathbf{w}\cdot\hat{\mathbf{x}},$$

$$\rho_h(\mathbf{x})|\mathbf{w}| = \mathbf{w}\cdot\mathbf{x} + b,$$

$$\rho_h(\mathbf{x}) = \frac{\mathbf{w}\cdot\mathbf{x} + b}{|\mathbf{w}|}.$$

Since $\rho_h(\mathbf{x})$ represents a distance it should be a positive function. Since $\mathbf{w}\cdot\mathbf{x} + b$ may be negative we can take the absolute value and redefine

$$\rho_h(\mathbf{x}) \equiv \frac{|\mathbf{w}\cdot\mathbf{x} + b|}{|\mathbf{w}|} = \frac{y_i(\mathbf{w}\cdot\mathbf{x} + b)}{|\mathbf{w}|}.$$

Specifically, we'll use the smallest geometric margin over the training set $\rho_h \equiv \min_{i\in[m]} \rho_h(\mathbf{x}_i)$. The SVM solution is the hyperplane $\mathbf{w}\cdot\mathbf{x} + b = 0$ which maximizes ρ_h under the constraint in Eq. (2.1). The distance between this hyperplane and the closest training vector is given by

$$\rho = \max_{\mathbf{w},b} \min_{i\in[m]} \frac{y_i(\mathbf{w}\cdot\mathbf{x} + b)}{|\mathbf{w}|} \ : \ y_i(\mathbf{w}\cdot\mathbf{x}_i + b) \geq 0, \ \forall i \in [m].$$

An observation that will simplify the optimization problem substantially is that the geometric margin is invariant under multiplication of $(\mathbf{w}, b)$ by a positive scalar. This freedom allows us to choose $(\mathbf{w}, b)$ such that we can set $\min_{i\in[m]} y_i(\mathbf{w}\cdot\mathbf{x} + b) = 1$. Under this constraint the optimization problem is

$$\rho = \max_{\mathbf{w},b} \frac{1}{|\mathbf{w}|} \ : \ y_i(\mathbf{w}\cdot\mathbf{x}_i + b) \geq 1, \ \forall i \in [m].$$

Since $|\mathbf{w}|^{-1}$ is maximized when $|\mathbf{w}|^2$ is minimized, which in contrast is infinitely differentiable everywhere and thus easier to deal with, we can reformulate the optimization problem as

$$\min_{\mathbf{w},b} \frac{1}{2}|\mathbf{w}|^2 \ : \ y_i(\mathbf{w}\cdot\mathbf{x}_i + b) \geq 1, \ \forall i \in [m].$$

We can solve the optimization problem using the Lagrange multiplier method. The Lagrangian to be minimized is given by

$$\mathscr{L}(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}|\mathbf{w}|^2 - \sum_i^m \alpha_i[y_i(\mathbf{w}\cdot\mathbf{x}_i + b) - 1],$$

where the Lagrange multipliers are $\alpha_i \geq 0 \ \forall \ i \in [m]$. Taking the gradient of $\mathscr{L}$ with respect to $\mathbf{w}$ and the derivative with respect to b and setting both to zero we obtain

$$\nabla_{\mathbf{w}}\mathscr{L} = \mathbf{w} - \sum_i^m \alpha_i y_i \mathbf{x}_i = \mathbf{0} \rightarrow \mathbf{w} = \sum_i^m \alpha_i y_i \mathbf{x}_i, \tag{2.2}$$

$$\frac{\partial \mathscr{L}}{\partial b} = -\sum_i^m \alpha_i y_i = 0 \rightarrow \sum_i^m \alpha_i y_i = 0. \tag{2.3}$$

Furthermore, according to Karush-Kuhn-Tucker's theorem (see Theorem B.30 in [6]), the following equation is satisfied at the minimum of $\mathscr{L}$.

$$\alpha_i[y_i(\mathbf{w}\cdot\mathbf{x}_i + b) - 1] = 0 \ \forall \ i \in [m] \rightarrow \alpha_i = 0 \vee y_i(\mathbf{w}\cdot\mathbf{x}_i + b) = 1 \ \forall \ i \in [m]$$

This means that all the α_i's corresponding to all *but* the vectors $\mathbf{x}_i$ closest to the hyperplane are zero. These vectors are called support vectors and they satisfy $\mathbf{w}\cdot\mathbf{x}_i + b = \pm 1$.

The *dual optimization problem* can be obtained by putting Eq. (2.2) and (2.3) back into the Lagrangian and *maximizing* with respect to $\boldsymbol{\alpha}$. I.e

$$\max_{\boldsymbol{\alpha}} \mathscr{L}(\boldsymbol{\alpha}) = \max_{\boldsymbol{\alpha}} \sum_i^m \alpha_i - \frac{1}{2}\sum_i^m\sum_j^m \alpha_i\alpha_j y_i y_j \mathbf{x}_i\cdot\mathbf{x}_j \ : \ \alpha_i \geq 0 \ \forall \ i \in [m], \ \sum_i^m \alpha_i y_i = 0. \tag{2.4}$$

Once the optimal $\boldsymbol{\alpha}$ has been found[1], we can calculate $\mathbf{w}$ using Eq. (2.2). If $\mathbf{x}_j$ is a support vector we can calculate b using the equation

$$y_j(\mathbf{w}\cdot\mathbf{x}_j + b) = 1 \ \rightarrow b = y_j - \mathbf{w}\cdot\mathbf{x}_j,$$

where we've used that $(y_j)^{-1} = y_j$. Once the optimal $\mathbf{w}$ and b has been found, we classify new data according to the equation

$$h(\mathbf{x}) = \text{sgn}\left(\sum_i^m \alpha_i y_i \mathbf{x}_i\cdot\mathbf{x} + b\right). \tag{2.5}$$

At this point we can make two important observations: The first observation is that since $\alpha_i = 0$ except for all the support vectors, the solution in Eq. (2.5) as well as the solution

[1] Eq. (2.4) is a convex optimization problem which we are treating as a black box due to our limited knowledge on the subject. We leave this as something to be handled by the magic of sklearn.svm.SVC. For a crash course in convex optimization involving lots of lovely theorems and mathematical jargon, see Appendix B in the absolutely wonderful *Foundations of Machine Learning* [6].

of the optimization problem in Eq.(2.4) depends only on the support vectors. The second observation is that these depend only on inner products between vectors. This observation leads to a major simplification when we'll consider non-linear boundaries which, after all these mathematics for just the simplest form of boundary, would otherwise seem outright frightening.

2.1.2 Non-separable case

In the linearly separable case we relied on the constraint $y_i(\mathbf{w}\cdot\mathbf{x}_i+b)\geq 1,\ \forall\ i\in[m]$, however in the non-separable case the constraint $y_i(\mathbf{w}\cdot\mathbf{x}_i+b)\geq 0,\ \forall\ i\in[m]$ is violated by definition. What we can do in the non-separable case is to introduce so-called *slack variables* $\xi_i\geq 0\ \forall\ i\in[m]$ which measures the amount by which this constraint is violated. More specifically, we introduce a new constraint

$$y_i(\mathbf{w}\cdot\mathbf{x}_i+b)\geq 1-\xi_i,\ \xi_i\geq 0\ \forall\ i\in[m].$$

The goal now is to maximize the geometric margin $\rho=\max_{\mathbf{w},b}1/|\mathbf{w}|$ while at the same time minimizing the total amount of slack. I.e, the optimization problem in the non-separable case is

$$\min_{\mathbf{w},b,\boldsymbol{\xi}}\frac{1}{2}|\mathbf{w}|^2+C\sum_i^m\xi_i^p\ :\ y_i(\mathbf{w}\cdot\mathbf{x}_i+b)\geq 1-\xi_i,\ \xi_i\geq 0\ \forall\ i\in[m],$$

where p is a positive integer and $C\geq 0$ is a regularization parameter which determines the trade-off between maximizing the geometric margin and minimizing the total amount of slack. A small C leads to the algorithm prioritizing the maximization of the geometric margin, while a large C leads to the algorithm prioritizing the minimization of the total amount of slack. scikit-learn's SVC class uses $p=1$ [7].

2.1.3 The Kernel Method

In order to handle non-linear boundaries we can use so-called kernels. The basic idea behind the kernel method is that, for some non-separable set of training data, we can make a non-linear transformation $\mathbf{x}_i\mapsto\boldsymbol{\Phi}(\mathbf{x}_i)$ to some inner product space where the training data *are* linearly separable. Then we can use the exact same approach as in the linearly separable case to find the best separating hyperplane in the new inner product space. As we've seen, the solution in the linearly separable case depends only on inner products between vectors. Therefore, all we have to do is to make the substitution $\mathbf{x}_i\cdot\mathbf{x}_j\to K(\mathbf{x}_i,\mathbf{x}_j)$ where

$$K(\mathbf{x}_i,\mathbf{x}_j)=\langle\boldsymbol{\Phi}(\mathbf{x}_i),\boldsymbol{\Phi}(\mathbf{x}_j)\rangle$$

is called a kernel which calculates the inner product in the new space given input vectors from the old space. Actually, we don't need to know what $\boldsymbol{\Phi}$ even is. We just need to choose some inner product that works well which, as far as we're concerned, is a matter of trial and error. However, in order for the optimization problem to be convex (and thus

guarantee a unique solution) we must choose a kernel that is positive definite for $\mathbf{x}_i, \mathbf{x}_j \neq \mathbf{0}$ and symmetric in the sense that $K(\mathbf{x}_i, \mathbf{x}_j) = K(\mathbf{x}_j, \mathbf{x}_i)$ [6].

Apart from the ordinary dot-product kernel (also called a "linear" kernel), some popular choices for $K(\mathbf{x}_i, \mathbf{x}_j)$ are the polynomial kernel

$$K(\mathbf{x}_i, \mathbf{x}_j) = (\gamma \mathbf{x}_i \cdot \mathbf{x}_j + c)^d \ : \ \gamma > 0, \ c \geq 0, \ d \geq 1, \ \gamma, c \in \mathbb{R}, \ d \in \mathbb{N},$$

the radial basis function (RBF) kernel

$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{|\mathbf{x}_i - \mathbf{x}_j|^2}{2\sigma^2}\right) \ : \ \sigma^2 \in \mathbb{R},$$

and the sigmoid kernel

$$K(\mathbf{x}_i, \mathbf{x}_j) = \tanh(a\mathbf{x}_i \cdot \mathbf{x}_j + b) \ : \ a > 0, \ b \geq 0, \ a, b \in \mathbb{R}.$$

2.2 Logistic Regression

As with Support Vector Machines we consider a set of training data $\mathscr{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$ where each vector $\mathbf{x}_i \in \mathbb{R}^N$ is a collection of N features. However, each vector is labelled $y_i \in \{0, 1\}$ for mathematical convenience that will soon become apparent. In contrast to Support Vector Machines, which is a so-called hard classifier where a new feature vector is classified based on what side of some boundary it falls on, logistic regression is a soft classifier where a new feature vector is classified based on probability.

The star of logistic regression is the logistic function

$$p(t) = \frac{1}{1 + \exp(-t)},$$

which can be interpreted as a probability distribution of a continuous variable t, since it is bounded between 0 and 1. Given a *new* feature vector $\mathbf{x}_j$ and a set of corresponding weights $\mathbf{w} \in \mathbb{R}^N$ and a bias $b \in \mathbb{R}$ we define the probability that $\mathbf{x}_j$ belongs to the class $y_j = 1$

$$\mathbb{P}[y_j = 1 | \mathbf{x}_j, \mathbf{w}, b] \equiv p(\mathbf{w} \cdot \mathbf{x}_j + b).$$

If $\mathbb{P}[y_j = 1] \geq 0.5$ we label the feature vector $y_j = 1$, else we label it $y_j = 0$. The probability distribution of y_j is

$$\mathbb{P}[y_j | \mathbf{x}_j, \mathbf{w}, b] = p(\mathbf{w} \cdot \mathbf{x}_j + b)^{y_j} [1 - p(\mathbf{w} \cdot \mathbf{x}_j + b)]^{1 - y_j}.$$

One can check that this expression reduces to $\mathbb{P}[y_j = 1] = p(\mathbf{w} \cdot \mathbf{x}_j + b)$ and $\mathbb{P}[y_j = 0] = 1 - p(\mathbf{w} \cdot \mathbf{x}_j + b)$. For a set $\mathscr{S} = \{(\mathbf{x}_k, y_k)\}_{k=1}^m$ of m feature vectors, we assume that their labels are i.i.d., thus the probability distribution for a set of feature vectors is given by

$$\mathbb{P}[\mathscr{S}|\mathbf{w}, b] = \prod_k^m p(\mathbf{w} \cdot \mathbf{x}_k + b)^{y_k}[1 - p(\mathbf{w} \cdot \mathbf{x}_k + b)]^{1-y_k}.$$

Since $\mathscr{D}$ is the training set we've got, we assume that this set is the one that is most likely. We're looking for the weights $\mathbf{w}$ and the bias b such that $\mathbb{P}[\mathscr{D}|\mathbf{w}, b]$ is maximized. Equivalently, we can minimize the negative natural logarithm of $\mathbb{P}[\mathscr{D}|\mathbf{w}, b]$, i.e

$$C(\mathbf{w}, b) \equiv -\sum_i^m [y_i \ln\{p(\mathbf{w} \cdot \mathbf{x}_i + b)\} + (1 - y_i) \ln\{1 - p(\mathbf{w} \cdot \mathbf{x}_i + b)\}], \tag{2.6}$$

which we interpret as the cost function of the classification problem. Since the cost function is non-linear in $\mathbf{w}$ and b, we can find the global minimum of $C(\mathbf{w}, b)$ numerically using some variant of gradient descent, like stochastic gradient descent.

2.2.1 Stochastic Gradient Descent

For a general cost function $C = C(\boldsymbol{\beta})$ of some parameter vector $\boldsymbol{\beta}$, the basic gradient descent algorithm goes like this:

$$\boldsymbol{\beta}_{k+1} = \boldsymbol{\beta}_k - \eta_k \nabla_{\boldsymbol{\beta}} C(\boldsymbol{\beta}_k)$$

for $k = 1, 2, \ldots$ up to some $K \in \mathbb{N}$. I.e, we start out with a guess $\boldsymbol{\beta}_1$, calculate the gradient $\nabla_{\boldsymbol{\beta}} C$ of the cost function at $\boldsymbol{\beta}_1$, then move in the opposite direction of the gradient with a stepsize given by the "learning rate" η_k. Then we've arrived at some $\boldsymbol{\beta}_2 \ : \ C(\boldsymbol{\beta}_2) \leq C(\boldsymbol{\beta}_1)$ and we repeat the process until we've found what is hopefully a global minimum of C.

The first and foremost problem with this approach is that we might very well get stuck at a local minimum or a saddle point. Another problem is that when the data set is large, calculating the gradient of the cost function at every $\boldsymbol{\beta}_k$ can be computationally expensive and wasteful when we're not even guaranteed to reach a global minimum!

Stochastic gradient descent solves the problem of high computational cost when the data set is large, and makes it less likely to get stuck at local minima. The idea is to use an approximation of the gradient instead of the exact gradient itself. For example, if the cost function can be written as an average over the data set

$$C(\boldsymbol{\beta}) = \frac{1}{m} \sum_i^m c_i(\mathbf{x}_i, y_i, \boldsymbol{\beta}),$$

we can sample the points $(\mathbf{x}_i, y_i)$ in the data set uniformly and use the sample average

$$\frac{1}{n} \sum_i^n c_i(\mathbf{x}_i, y_i, \boldsymbol{\beta}) \approx C(\boldsymbol{\beta}) \rightarrow \frac{1}{n} \sum_i^n \nabla_{\boldsymbol{\beta}} c_i(\mathbf{x}_i, y_i, \boldsymbol{\beta}) \approx \nabla_{\boldsymbol{\beta}} C(\boldsymbol{\beta})$$

where $n < m$. More specifically, we divide the data set at random into M so-called minibatches. Then at each step k we select a minibatch and use the sample average over that minibatch of the cost function gradient, before moving on to another minibatch at step $k + 1$. When each minibatch has been used exactly once, we have completed a so-called epoch, at which point we can just repeat the process for a given number of epochs.

This solves the problem of computational cost when having large data sets right away, because now the computational cost depends on the size of each minibatch, and not the size of the data set as a whole. Getting stuck at local minima and saddle points becomes more unlikely because, even if we reach a point where the exact gradient is zero, we're using an approximation of the gradient which is probably not quite zero. And even if the approximation is also zero, another minibatch will be selected the next step anyway. This increases the likelihood of escaping saddle points and local minima, but is in no way a guarantee and also requires a sufficiently large learning rate in order to escape.

Our cost function in Eq. (2.6) is a sum over the data set, but not an average. However, simply multiplying the cost function by $1/m$ leaves the minimization problem unchanged. Alternatively, we can absorb the factor of $1/m$ into the learning rate η and not worry about it.

However, plain stochastic gradient descent is not available in scikit-learn's LogisticRegression class, but a similar method called Stochastic Average Gradient (SAG) is available, which is the one we've settled on. SAG, like stochastic gradient descent, is independent of the number of points in the data set, but converges faster to the global minimum [8].

2.3 Decision Trees

Decision trees are supervised learning algorithms that can be used both for regression and classification cases. The main idea of a decision tree is to take a feature which describes the data and split the data along a value for this feature. This way we reduce the amount of the data that is relevant for the target. The decision tree is divided into a root node, interior nodes and the leaf nodes. The leaf nodes are the predictions the method will make based on the splits that happen in the interior nodes.

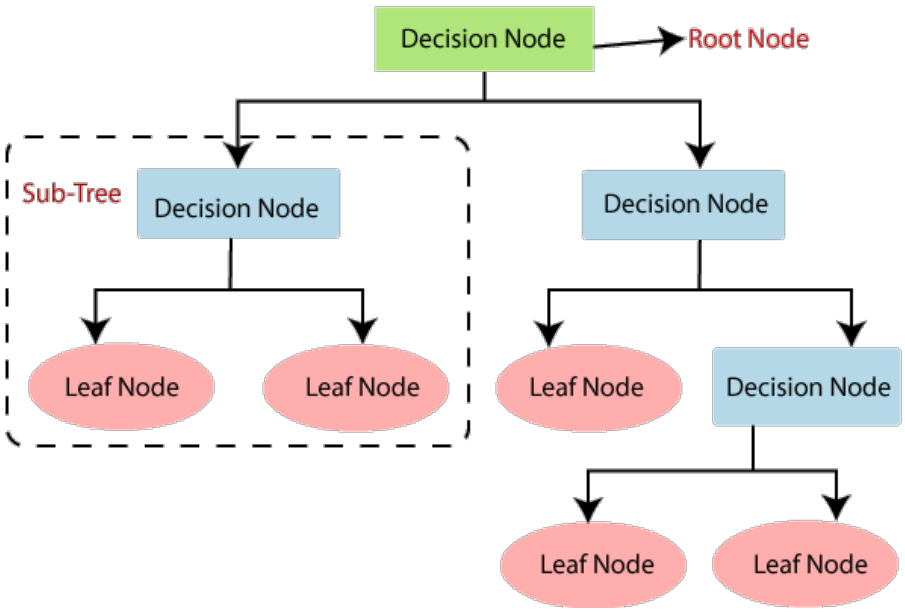


Figure 2.2: *Illustration of a decision tree.*

When we set up a decision tree we use a data set that contains a number of descriptive features and a number of target features. Then we make the tree by continuously splitting the data along the values of the descriptive features of the data set. During the training process we use a measure of information gain to see how much we can learn from each split. We continue with the decision nodes until we reach a point where the leaf node is created, this leaf node is the prediction we want to make. When then testing the data the instance is run through the tree and all the different decision nodes until it reaches one leaf node, this will then be the prediction given. This is essentially the method used for both regression and classification.

When building a decision tree for regression we split the prediction space into J distinct non-overlapping regions, $R_1, ..., R_J$. For every observation that falls within a region we make the same prediction, this prediction is generally the mean of all the values in this region. We use rectangular regions when splitting for simplicity. The goal is then to find regions that minimize the mean squared error, $\sum_{j=1}^{J} \sum_{i \in R_J} (y_i - \bar{y}_{R_j})^2$. This method is rather easy, however it can also very easily lead to overfitting if the regions are too small. To reduce the overfitting we perform what is called pruning on the tree. This is a method to reduce the complexity of the tree, the amount of splits and in the regression case reduce the amount of different regions we have. When pruning, we can for example use cost complexity, instead of considering every possible sub-tree, we consider a sequence of trees indexed by a non-negative tuning parameter α. This α value can be found by using k-fold cross validation on the whole tree. For each k-value we find the α, finally we find the value that gives the lowest MSE and will use this. Then we will choose the sub-tree that corresponds to this α value.

When building a classification tree the procedure is very similar to the regression tree, however in this case it is used to predict a specific outcome. Instead of using the mean value in a certain region to be the leaf the classification tree will use the class that is most common in this region as the leaf node. In the classification case we cannot use the MSE as the criterion for making the split, therefore we instead use the classification error rate. The classification error rate is the fraction of training observations in that region that do not belong to the

most common class. When setting up a classification tree, the Gini index or the entropy are usually used to evaluate the quality of a particular split, since these approaches are more sensitive to node purity than the classification error rate is. The Gini index is given by

$$g = \sum_{k=1}^{K} p_{mk}(1 - p_{mk}),$$

where the sum runs over the K classes and $p_{m,k}$ is the proportion of samples labeled k in the m'th region. The entropy is given by

$$s = -\sum_{k=1}^{K} p_{mk} \log p_{mk}.$$

In the Wisconsin breast cancer data set we will be using, we use a classification decision tree where the decision nodes are based on the values of the different features of the tumor. The leaf nodes at the end are 1, which means cancerous, and 0 which means not cancerous. We will be making the tree using the library that exists in scikit-learn for decision trees. With this library we can fit the train data, making a model, and predict the outcome of the test data by using the model. The library also includes a function for finding the accuracy of the model which we will use to determine how good the model is. The prediction we will use for seeing how many the model will predict right and how many it will predict wrong, we will also see if there are more false positives or false negatives in the prediction.

2.4 Gradient Boosting and XGBoost

Gradient boosting is a machine learning prediction method that can be used for both regression and classification problems. It produces a prediction model by using a collection of weaker prediction models, in this context a weak model is a model that predicts slightly better than random guessing would. This is done by applying a weak model iteratively to modify the data. In each iteration we emphasize the observations that are misclassified by weighting them with a factor. For a gradient boosting method we have to look at a a version of the steepest decent algorithm. If we have a cost function $C(f) = \sum_{i=0}^{n-1} L(y_i, f(x_i))$ where y_i is our target and $f(x_i)$ the function models the target. The cost function could be our standard squared-error function,

$$C(y, f) = \sum_{i=0}^{n-1} (y_i - f(x_i))^2.$$

We can then perform the iteration by using this cost function, we start by initializing our estimate for $f_0(x)$. Then for all the predictors, m, in the data we compute the negative gradient vector $u_m = -\partial C(y, f)/\partial f(x)$ at $f(x) = f_{m-1}(x)$, then we fit the base-learner to the negative gradient $h_m(u_m, x)$. Then for each predictor we update the estimate by using this base-learner $f_m(x) = f_{m-1}(x) + h_m(u_m, x)$. When having looped over all the predictors we end up with the final estimate being,

$$f_M(x) = \sum_{m=1}^{M} h_m(u_m, x).$$

XGBoost, or extreme gradient boost, is an optimized gradient boosting library made to be highly efficient, flexible and portable. This library provides a parallel tree boosting that

can solve many data science problems, this method is in many cases very fast and accurate. We want to implement this method when predicting for our data set because this method has been shown to be highly accurate in the analysis of many other data sets. Because XGBoost has outperformed other prediction methods for other data sets we want to see the accuracy this method can achieve in the Wisconsin breast cancer data set. We will use the preexisting XGBoost library for this prediction, we will fit the training data and targets with this method and then predict the outcomes for the test targets by performing the prediction on the design matrix for the test data. We will then see how many it predicted correctly and which targets it predicted wrong.

2.5 Feedforward Neural Networks

For a brief explanation on the basics of feedforward neural networks, as well as the back-propagation algorithm, we suggest section 2.3 in [2]. The e-book by Michael Nielsen on feedforward neural networks is also highly recommended [9].

Multilayer perceptron (MLP) networks can be described as networks containing three or more layers with neurons that have non-linear activation functions. The multilayer network architecture consists of an input layer, one or more hidden layers and an output layer [10]. In MLP, each neuron receives a weighted sum of the outputs of all neurons in the previous layer, it is therefore said to be fully-connected. For our neural network employed here, we use the multi-layer perceptron classifier (MLPClassifier) from scikit learn [11]. In the MLPClassifier we can select a training method (solver), activation function, L2 regularization parameter (alpha) and learning rate schedule for the weight optimization. As with logistic regression we use the cost-function given in Eq. (2.6), but we substitute $p(t)$ with the output of the activation function in the output layer.

The activation function for the hidden layers are either logistic, tanh and ReLU. For the logistic/sigmoid activation function, see Section 2.2. For tanh and ReLU, the reader is referred to another paper by authors [2]. The solver for weight optimization are either LBFGS, Adam or SGD. For details about SGD, see Section 2.2.1. LBFGS and Adam will be briefly explained here.

Limited memory Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm (LBFGS) is an optimizer in the family of quasi-Newton methods. This family of optimization methods can be used for finding extrema. They are typically used when the Jacobian or Hessian is unavailable or too expensive to compute at every iteration. LBFGS approximates the Hessian matrix using updates specified by gradient evaluations. LBFGS usually performs better and can converge faster for smaller datasets than the other methods [11].

Adaptive Moment Estimation (Adam) is a stochastic gradient-based optimizer. Adam combines the advantages of two other popular extensions of SGD; namely Adaptive Gradient Algorithm (AdaGrad) and Root Mean Square Propagation (RMSProp). The former adapts the learning rate to the parameters and the latter adapts the learning rate based on the change in gradient with respect to the weights. Adam is popular in deep learning as it achieves good results fast and works well on large datasets [11].

2.5.1 Grid Search and Feature Selection

In order to select the most suitable parameters for the MLPClassifier, a grid search can be utilized. In our case we have used scikit learn's GridSearchCV [12] which search the combinations of the parameters specified to find the the most optimized by cross-validation over a parameter grid.

Feature selection can be performed on a data set to find the features that have the most impact on the response. Having irrelevant features can decrease the accuracy as the model is trained on misleading data. The overall aim of feature selection is thus to improve the model by only training on features strongly correlated with the response. As we then are training on less data, feature selection also reduces the training time. One way of performing feature selection is by applying the filter method. Here, we first generate a correlation matrix, where the correlation between variables (both features and responses) are mapped to values, called correlation coefficients, between ± 1. If the correlation coefficient between two variables is near ± 1, the correlation is said to be (near) perfect: as one variable increases, the other variable tends to also increase (if positive) or decrease (if negative). There is no correlation if the correlation coefficient is zero. In the filter method, we filter out the features with weaker correlation to the response of interest by setting a threshold [13].

3 Results and Discussion

3.1 Logistic regression

When using the scikit-learn logistic regression method for prediction there are are few parameters one can change. For instance the type of solver or the regularization parameter. Here, we have the results when using the 'svg' solver and an 'l2' regularization parameter.

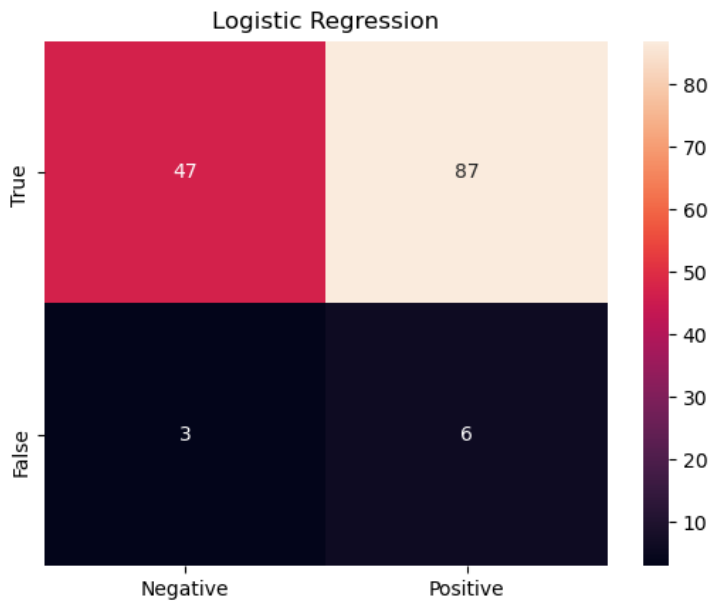


Figure 3.1: *Confusion matrix calculated using logistic regression.*

In Figure 3.1 we see the heatmap for true positive, true negative, false negative and false positive for logistic regression. From this heatmap we see that the logistic regression method predicts 7 cases wrong of which 3 are false negatives and 6 is a false positive. Since we have more positive cases in the test set it is unexpected to see more false positives than false negatives. Therefore, this means that the logistic regression method on this data set is more likely to predict a false positive than a false negative. Since this data set is about a breast tumor being cancerous or not a false positive might be better than a false negative. If the prediction would give a false positive the tumor will be further investigated, a false negative however will be more dangerous as the patient will believe that it is not cancerous even when it is. So, since we know that early detection is important with breast cancer a false negative is more harmful than a false positive and the method predicting more false positives than false negatives it therefore beneficial.

Logistic regression gives an accuracy of 93.7% when we use the data from the data set without scaling it. When we use the scikit-learn scaling feature we increase the accuracy to 95.8%. This is quite a good accuracy for the data set, however since we have seen in earlier projects [2] that we can reach higher accuracies on other data sets, we want to aim for a higher accuracy score. Especially for a prediction method that would be used within the medical field and for healthcare an accuracy that is as high as possible is of importance.

3.2 Decision Trees

We use the scikit-learn functionality to build the decision tree needed for this data. We have fit the tree to the data by using the training data and training targets. The decision tree for the breast cancer data is illustrated in the image below.

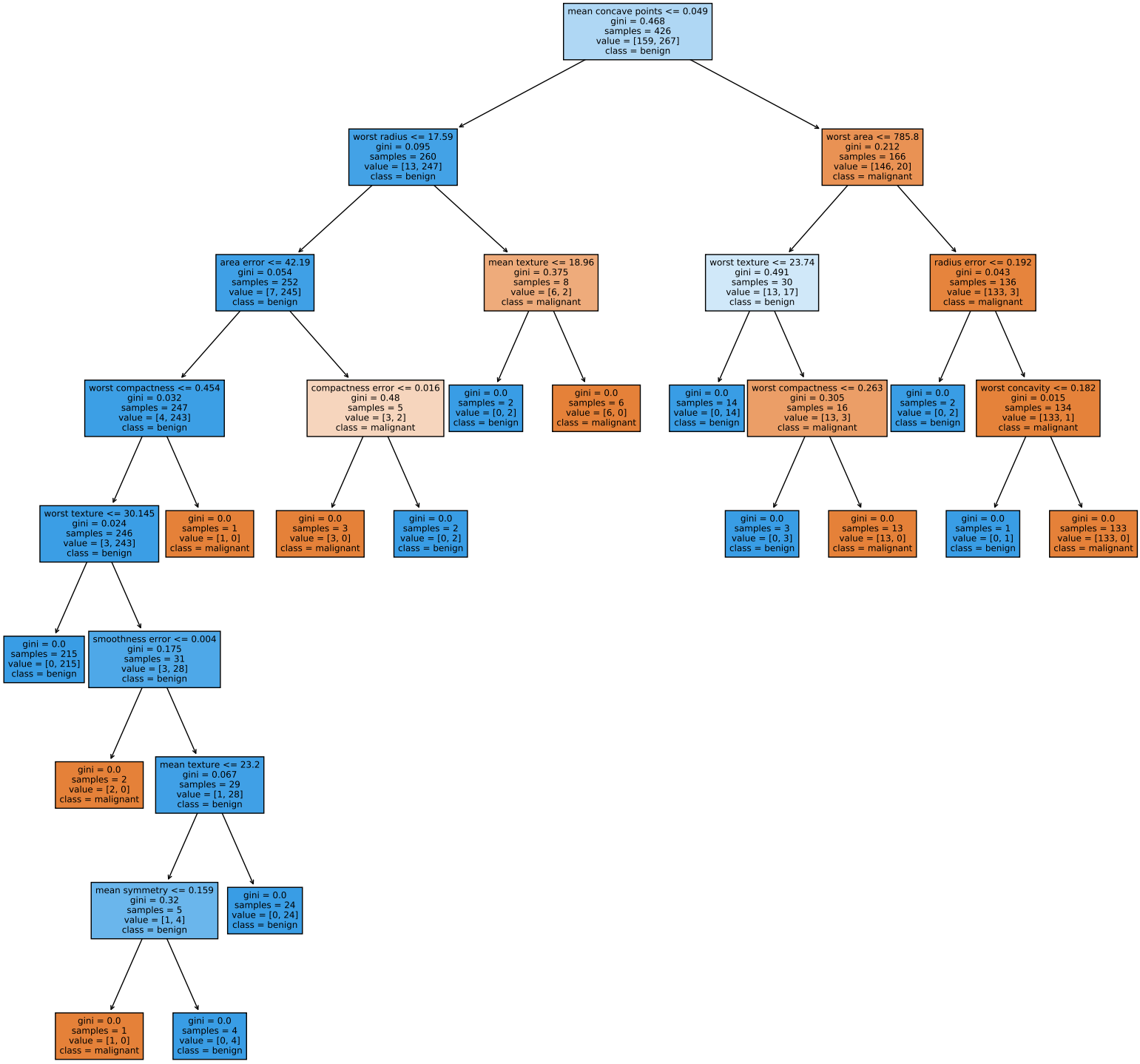


Figure 3.2: *Decision tree for the cancer data*

From Figure 3.2 we see how the computer will make choices on the class based on the values for the different features. We also see the corresponding Gini value for each split. We see that the color blue represents the splits where the value will be benign and the orange is for splits where the condition means the class is malignant.

Figure 3.3: *Confusion matrix calculated using a decision tree.*

Figure 3.3 shows that there are in this prediction of the test data 17 instances that have been predicted wrong. In contrast to the logistic regression prediction the decision tree predicts more false negatives than false positives. As we mentioned in the logistic regression case this is not ideal when dealing with this data.

The decision trees, like any other prediction method, have pros and cons. One of the big pros with decision trees is that they are easy and intuitive for people to understand. The method can handle both classification and regression problems. It can be easily visualised and can handle non-linear relationships. However, the downside of the decision trees is that they are generally not very accurate in comparison with other methods. Trees are prone to overfitting of the data, especially if the stopping or improving methods like pruning are not applied. Another con of the decision trees is that even a small change in data will completely change the tree. To combat these problems we can use methods like bagging, boosting or random forests which use decision trees, but improve their accuracy by combining them. For the breast cancer data set we have with decision trees a 88.1 % accuracy when we have non-scaled data and a 91.6% accuracy with scaled data. An accuracy of around 90% is not bad for a prediction model, but as we have already seen other prediction methods yield higher accuracy. So, we know that decision trees is not the best method to use for the Wisconsin breast cancer data.

3.3 XGBoost

The XGBoost method has a prediction accuracy on the test set of 97.9% both with and without scaling. This is a very high accuracy for a prediction method, especially considering this data set is not very big. We see that even though XGBoost uses decision trees in its prediction boosting the trees and optimizing this boosting improves the accuracy significantly, we go from 91.6% accuracy in a singular decision tree to 97.9% when using this optimized tree boosting. When we look to see how the prediction compares to the targets we get the results heatmap below,

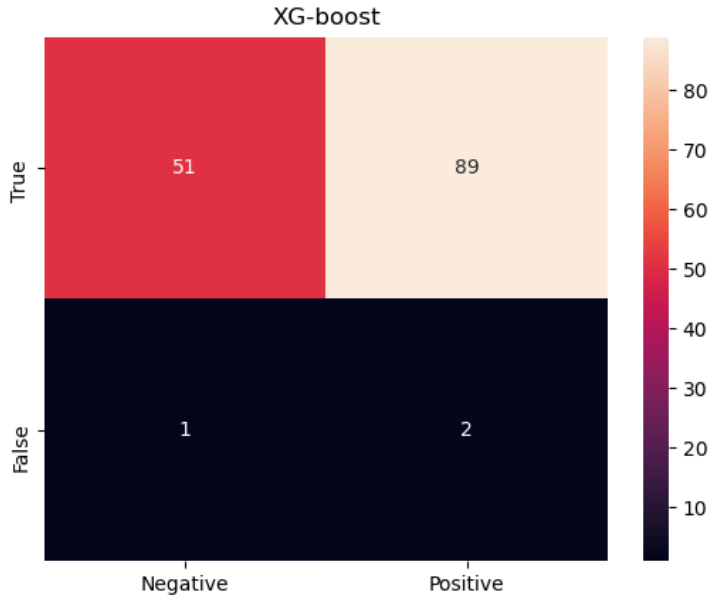
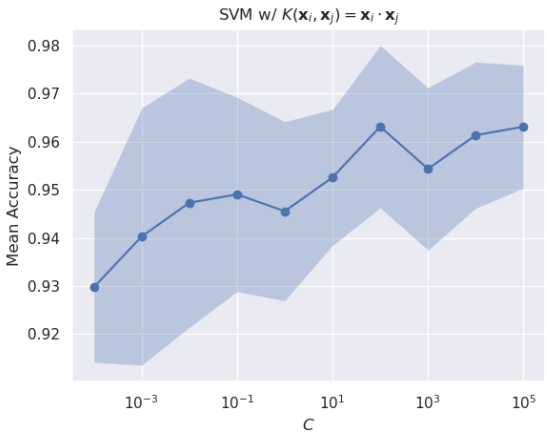


Figure 3.4: *Confusion matrix computed using XGBoost*

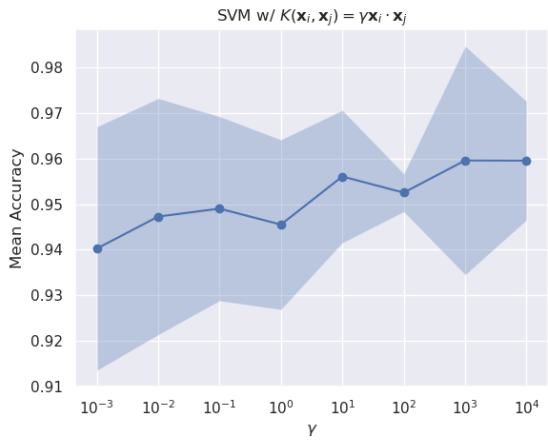
In Figure 3.4 we see the results from the prediction on the test data. In this test data only three of the instance have been predicted wrong. Just like with logistic regression we here have more false positives than false negatives, which is what we want in data of this kind.

So, from this we see that since XGBoost has a very high accuracy, is fast and gives more false positives than false negatives. We see that using this boosting method can reduce some of the major problems in decision trees, like overfitting. When using more trees we reduce the chance for overfitting.

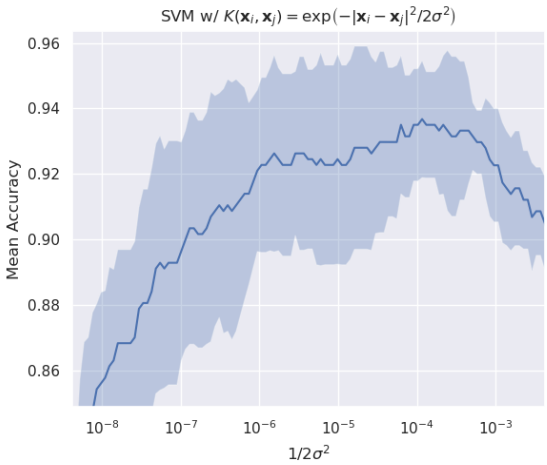
3.4 Support Vector Machines



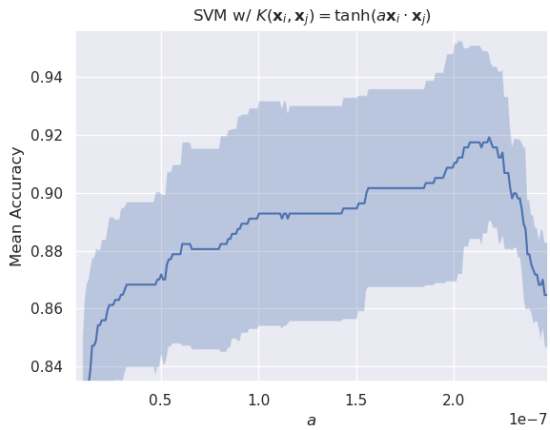
(a) *Mean accuracy as a function of the regularization parameter C when using a linear kernel.*



(b) *Mean accuracy as a function of the parameter γ when using a polynomial kernel.*



(c) *Mean accuracy as a function of $1/2\sigma^2$ when using a RBF kernel.*



(d) *Mean accuracy as a function of a when using a sigmoid kernel.*

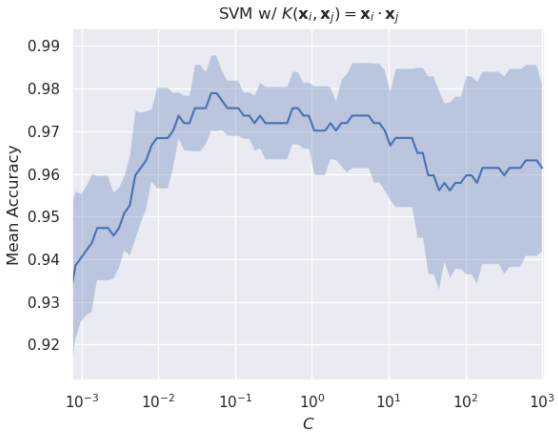
Figure 3.5: *Mean accuracies computed on unscaled data using 5-fold cross validation with various kernel functions. The blue shaded areas highlight one standard deviation away from the average mean accuracy.*

Figure 3.5 shows the mean accuracy computed on unscaled data using Support Vector Machines with a linear, polynomial, RBF and sigmoid kernel respectively. The blue lines show the average mean accuracy computed using 5-fold cross validation, and the blue shaded areas highlight one standard deviation from the average mean accuracy.

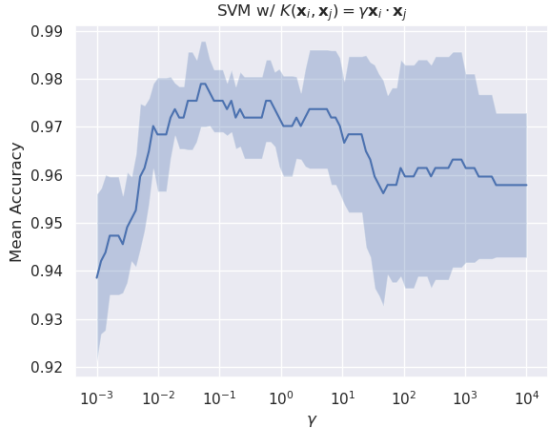
To generate Figure 3.5a we used the linear (dot-product) kernel and varied the regularization parameter C. We see that the mean accuracy increases with C in the range $\log C \in [-4,\ 5]$, but we have not calculated the mean accuracy for higher C-values for two reasons. Firstly, increasing C led to the estimator learn more and more slowly and considering that we achieved a higher mean accuracy on scaled data (see Figure 3.6a) we thought it'd be wasteful to extend Figure 3.5a further. This would make sense, given that a larger C leads to the estimator "focusing more" during training on minimizing the total amount of slack which

after a while becomes more and more difficult and ultimately utterly impossible if the input data is not linearly separable. This leads to the second reason, which is overfitting. If the estimator is "too focused" on minimizing the total slack during training for this particular data set, we are not sure that it would generalize well to inputs outside of this data set. This also makes us skeptical of the fact that, of all the kernels, the dot product kernel led to the highest average mean accuracy of ~ 0.96 on unscaled data, which is the only kernel we used while varying C up to the ridiculous value of $C = 10^5$ (for comparison, the scikit-learn documentation recommends $C \sim 1$). Meanwhile, for the other kernels, we used $C = 0$.

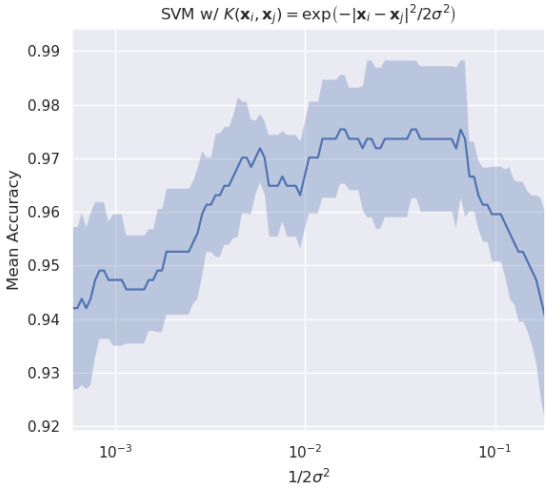
In any case, the main point to draw from Figure 3.5 is that, for all the kernels, we are able to tune some parameter in order to get a respectable accuracy above 0.90 on unscaled data, with at most a standard deviation of 0.03 for the sigmoid kernel. But Figure 3.6 shows that we can do better by simply scaling the data before training the respective estimators. We see that the highest mean accuracy obtained is at least 0.96 across the board. For the linear and polynomial kernels we obtain an accuracy of 0.98 with a standard deviation of 0.01. The estimator using the sigmoid kernel performs slightly worse at a mean accuracy of 0.97 with a standard devitation of 0.01. Then there is the estimator using the RBF kernel which obtains an accuracy of 0.975 with a standard deviation of 0.015.



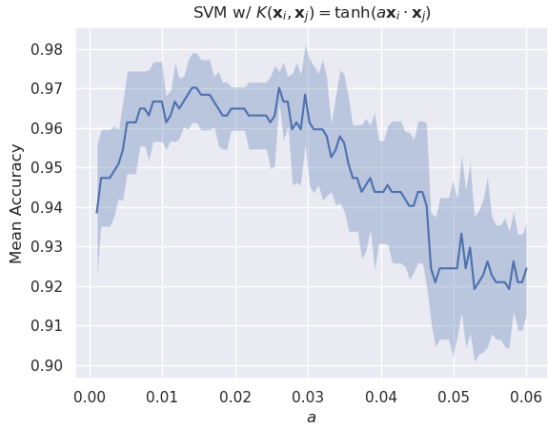
(a) *Mean accuracy as a function of the regularization parameter C when using a linear kernel.*



(b) *Mean accuracy as a function of the parameter γ when using a polynomial kernel.*



(c) *Mean accuracy as a function of $1/2\sigma^2$ when using a RBF kernel.*



(d) *Mean accuracy as a function of a when using a sigmoid kernel.*

Figure 3.6: *Mean accuracies computed on scaled data using 5-fold cross validation with various kernel functions. The blue shaded areas highlight one standard deviation away from the average mean accuracy.*

3.5 Feedforward Neural Networks

We have used scikit-learn functionality to build a FFNN for the breast cancer dataset. We implement multi-layer perceptron classifier with chosen parameters shown in Table 3.1.

Table 3.1: *Chosen parameters for MLPClassifier. Run for 1000 iterations.*

	Activation	Alpha	Hidden layers	Learning rate	Solver
Parameters	Relu	0.0001	(50,)	Constant	sgd

The train and test data were split into 85% and 25%, respectively. The data were scaled. We fitted the model and used the model to predict unseen test data. A confusion matrix was generated from the benign and malignant cases in the dataset in Figure 3.7.

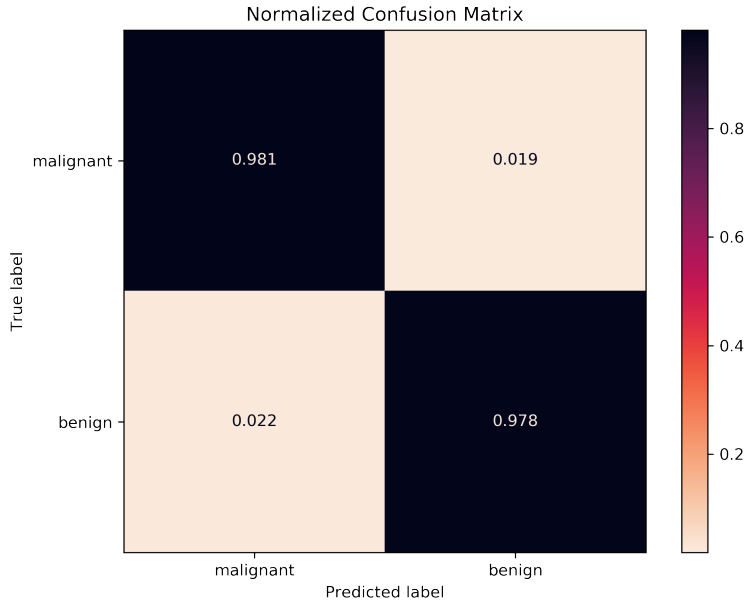


Figure 3.7: *Normalized confusion matrix on the breast cancer test set (25% of the whole data set). The diagonal elements represent the number of points for where the predicted label is equal to true label. The off-diagonal elements are those that are mislabeled by the classifier. The higher the values in the diagonal elements are, the better the predictions are from the classifier.*

The confusion matrix in Figure 3.7 evaluates the quality of the model. It can be observed that the classifier predicts the predicted labels of malignant or benign to true label correctly in almost all cases. Specifically, in 97.8% of all cases for benign and in 98.1% for malignant.

The accuracy score was computed from sklearn's accuracy score functionality to be 97.9%. Using this approach, the accuracy score is extraordinary high. This could be due to training on only a simple part of the data set. We need to assess the model's ability to learn and predict accurately, since this accuracy score could be imprecise. We need to evaluate our model on the whole data set with for example cross-validation. In addition, we can also search for better parameters by performing a grid search.

3.5.1 Grid Search

Employing a grid search from the GridSearchCV functionality from scikit-learn, the following parameters in Table 3.2 were found:

Table 3.2: *Best parameters found from grid search. Run for 1000 iterations.*

	Activation	Alpha	Hidden layers	Learning rate	Solver
Parameters	Logistic	0.01	(50, 20)	Adaptive	lbfgs

These parameters were implemented in the MLPClassifier and plotted in a confusion matrix shown in Figure 3.8.

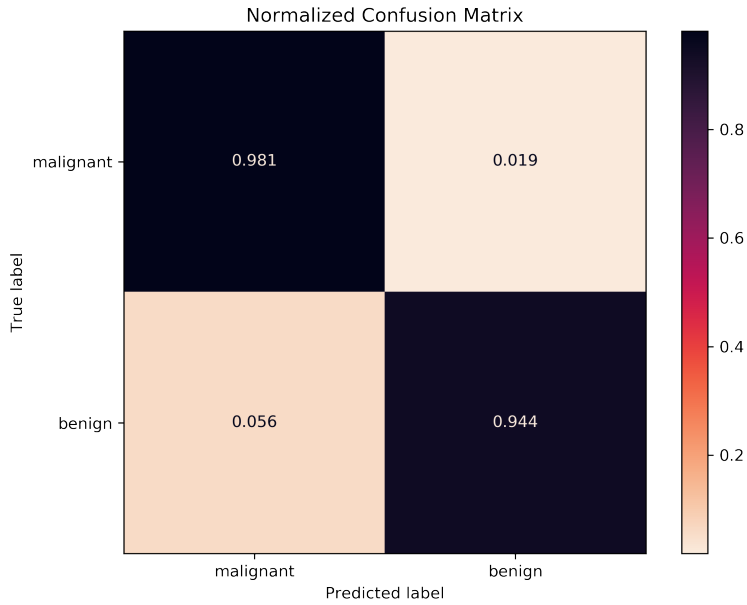


Figure 3.8: *Normalized confusion matrix on the breast cancer test set (25% of the whole data set) with grid search. The diagonal elements represent the number of points for where the predicted label is equal to true label. The off-diagonal elements are those that are mislabeled by the classifier. The higher the values in the diagonal elements are, the better the predictions are from the classifier.*

Figure 3.8 has 94.4% correct classification of the benign cases. The accuracy was calculated from a standard k-fold cross-validation partitioned into 5 subsets. For each of the validation fold, we record the corresponding performance score. The average of these scores was 96.58% ±0.03%. This is also high as the accuracy without implemented CV was, but is in return more accurate.

3.5.2 Feature Selection

The accuracy of the model can still be optimized in order to achieve a higher score. In order to do this, we employ a feature selection. As mentioned above, the dataset consists of 30 features (columns) characterizing the tumors. We want to find the features correlating the most to the response of the dataset and remove features that are less correlated, meaning that they are less significant.

Before selecting the features correlating the most to the response, all 30 features were plotted in correlation with target column in Figure 3.9.

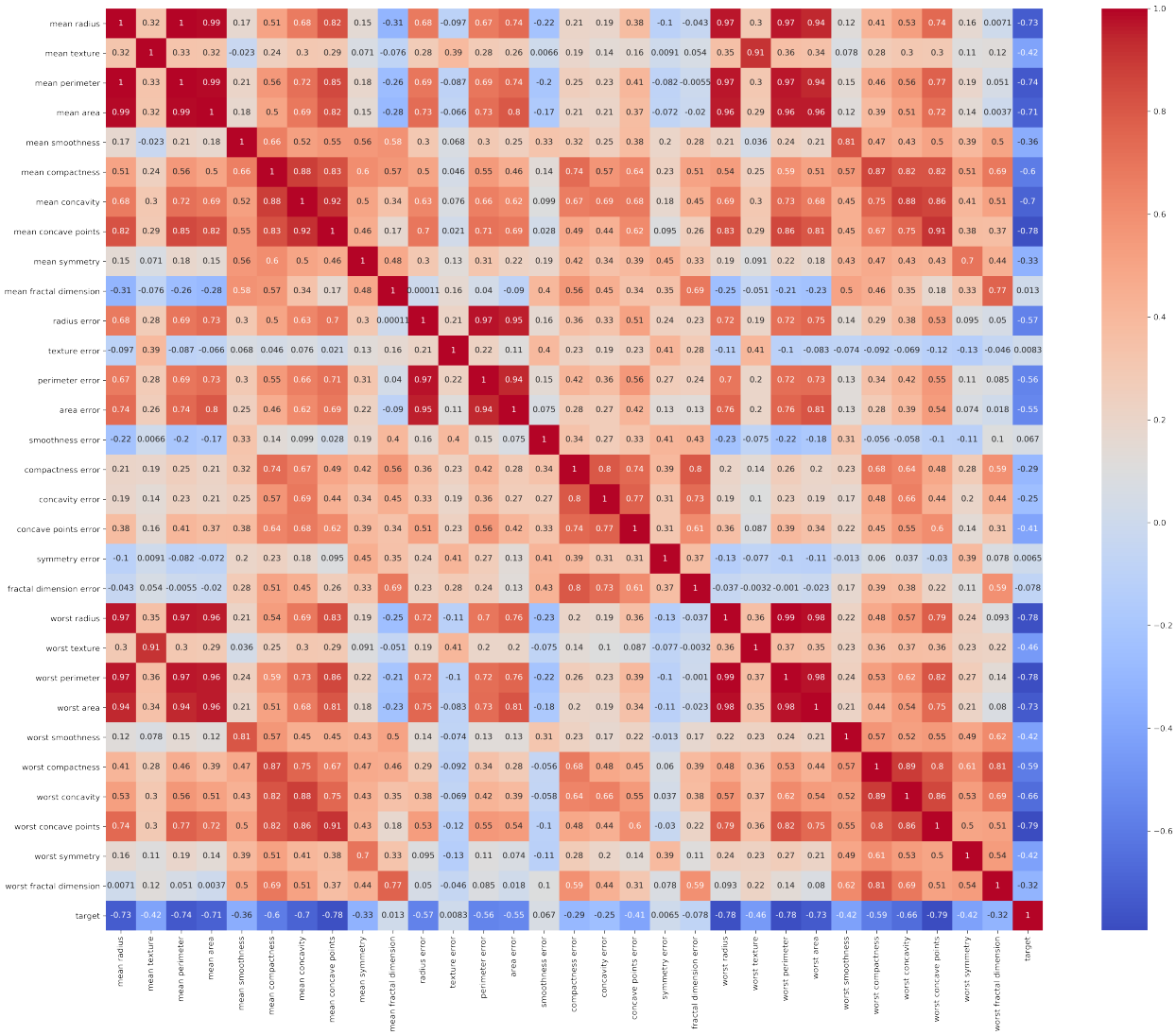


Figure 3.9: *Correlation matrix showing how the features correlate to target. If the value is near* ± 1*, then the value is in high correlation. If the value is near 0, then it is a low correlation.*

As observed in Figure 3.9, some features such as *worst concave points*, *mean concave points* and *mean radius* correlate highly to the response. To select the most relevant ones, we set a threshold to 0.3 for the absolute values of the features in Figure 3.9. This means that all features with value above 0.3 are considered the most relevant. This resulting 23 features are plotted in the correlation matrix in Figure 3.10.

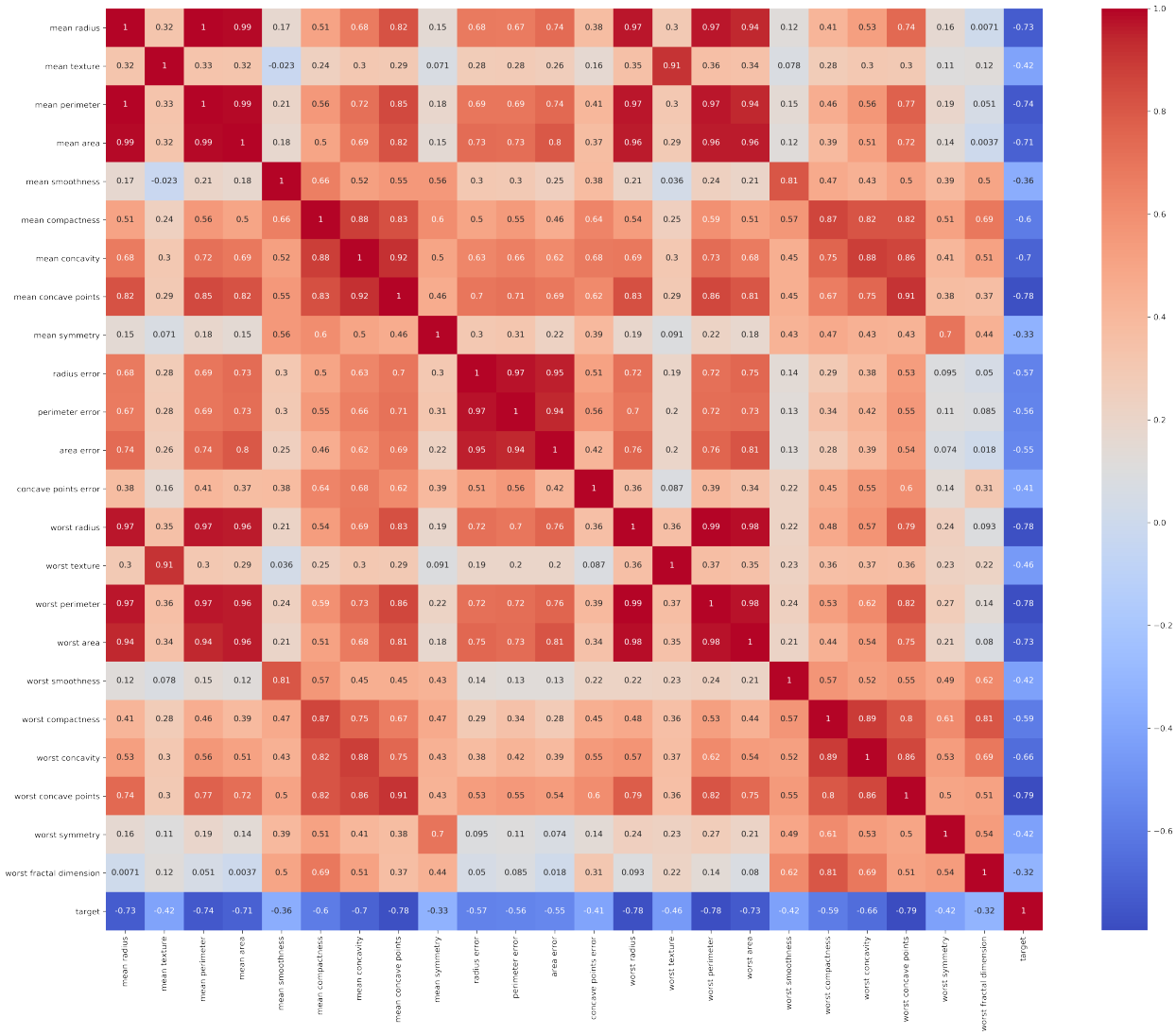


Figure 3.10: *Correlation matrix showing the most relevant features correlated to target for absolute value higher than threshold 0.3. If the value is near 1, then the value is in high correlation. If the value is near 0, then it is a low correlation.*

In Figure 3.10 the 23 most correlated features to the response are plotted and removed the most irrelevant ones. By performing a grid search on this "new", smaller data set, the resulting parameters were computed and are presented in Table 3.3.

Table 3.3: *Best parameters found from grid search for most relevant features. Run for 1000 iterations.*

	Activation	Alpha	Hidden layers	Learning rate	Solver
Parameters	Relu	0.001	(50,)	Adaptive	Adam

After a feature selection, the grid search changed the activation function from logistic (without feature selection, Table 3.2) to relu and the training method from lbfgs (without feature selection, Table 3.2) to adam. The parameters in Table 3.3 were implemented in MLPClassifier. The corresponding confusion matrix is shown in Figure 3.11.

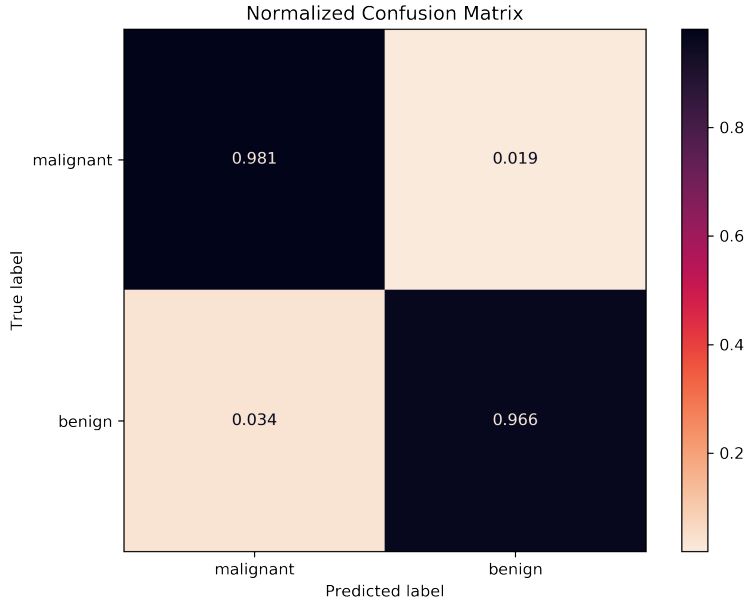


Figure 3.11: *Normalized confusion matrix on the breast cancer test set (25% of the whole data set) after grid search and feature selection. The diagonal elements represent the number of points for where the predicted label is equal to true label. The off-diagonal elements are those that are mislabeled by the classifier. The higher the values in the diagonal elements are, the better the predictions are from the classifier.*

At last, by utilizing feature selection and grid search on the model, it correctly predicts the predicted labels to true label in 96.6% of all benign cases and 98.1% of all malignant cases. Comparing to the confusion matrix without feature selection in Figure 3.8, the percentage is the same for malignant cases, but higher for benign cases. The breast cancer dataset used in this model is small (569 samples of 30 features each) and is also biased towards benign tumors since they are the most prevalent in the data set. Thus, this affects the training which becomes biased and can overpredict for benign samples. This means that the value for classifying malignant cases in Figure 3.11 might be inaccurately high and give more false negatives.

The accuracy score after the feature selection has increased by approximately 1% to 97.57% $\pm$ 0.03% compared to 96.58% $\pm$ 0.03% for without feature selection. It is likely that this accuracy could have been higher if the threshold for absolute values in feature selection was set higher than 0.3.

4 Conclusion

In this study, we have investigated the possibility of utilizing machine learning classification algorithms to train models that can be used to detect malignant tumors in breast cancer patients.

We've seen that predictive models trained using classification algorithms can correctly classify a respectable share of the test data, and are thus suitable for supplementing the process of diagnosing cancer in patients. With logistic regression we obtained an accuracy of 95.5%. Although a single decision tree performed relatively poorly at 91.6%, the mean accuracy was greatly increased to 97.9% by implementing gradient boosting using XGBoost. With support vector machines we used a variety of kernel functions, but obtained the greatest accuracy with a linear kernel function at 98% with a standard deviation of 1%. Not far behind was the model trained using feed forward neural networks which accurately predicted 97.57% of the test data with a standard deviation of 0.03%. Here, we performed a grid search of the model parameters and a feature selection for finding the features with strong correlation to the response.

However, as we see it there are some problems with the available data that was used for training and testing. The first is that the Wisconsin Breast Cancer dataset is quite small with just 569 samples, which are all from a single state, which puts into doubt whether our models have generalized to predict from a wider population. The second problem is that the data set is somewhat skewed, with 357 samples belonging to the benign class and 212 samples belonging to the malignant class. This could potentially lead to overprediction of benign tumors and thus more false negatives. We may have observed this effect in Figure 3.3 of the confusion matrix when we used a decision tree classifier, although we see no signs of this effect in Figure 3.4 and Figure 3.11.

For future work we suggest using a much larger dataset which is more representative of the wider population. We also suggest that machine learning can be used to train predictive models for other types of cancer, like lung cancer, prostate cancer and melanoma. Finally, we also suggest that predictive models can be trained to detect other diseases, in particular heart disease which is the world's leading cause of death.

References

[1] Megha Rathi and Vikas Pareek. "Spam Mail Detection through Data Mining – A Comparative Performance Analysis". In: *International Journal of Modern Education and Computer Science* 5 (Dec. 2013), pp. 31–39. DOI: 10.5815/ijmecs.2013.12.05 (cit. on p. 1).

[2] Erik Alexander Sandvik Oda Hovet Ilse Kuperus. *Classification and Regression: From Linear and Logistic Regression to Neural Networks*. Nov. 2020. URL: https://github.com/erikasan/fys-stk-project2/blob/main/report/FYS_STK_Project_2.pdf (cit. on pp. 1, 11, 13).

[3] URL: https://scikit-learn.org/stable/datasets/index.html#breast-cancer-dataset (cit. on p. 1).

[4] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830 (cit. on p. 1).

[5] URL: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html (cit. on p. 1).

[6] Ameet Talwalkar Mehryar Mohri Afshin Rostamizadeh. *Foundations of Machine Learning*. URL: https://github.com/erikasan/textbooks/blob/master/Machine%20Learning/Foundations%20of%20Machine%20Learning.pdf (cit. on pp. 4, 6).

[7] URL: https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html (cit. on p. 5).

[8] Francis Bach Mark Schmidt Nicolas Le Roux. "Minimizing Finite Sums with the Stochastic Average Gradient". In: *Mathematical Programming B 62 (1-2)* (2017), pp. 83–112. URL: https://hal.inria.fr/hal-00860051/document (cit. on p. 8).

[9] Michael Nielsen. *Neural Networks and Deep Learning*. URL: http://neuralnetworksanddeeplearning.com/ (cit. on p. 11).

[10] Morten Hjorth-Jensen. *Data Analysis and Machine Learning lectures: Optimization and Gradient Methods*. URL: https://compphysics.github.io/MachineLearning/doc/pub/week40/html/week40.html (visited on 11/03/2020) (cit. on p. 11).

[11] ScikitLearn. $sklearn.neural_network.MLPClassifier$. URL: https://scikit-learn.org/stable/modules/generated/sklearn.neural%5C_network.MLPClassifier.html (visited on 12/17/2020) (cit. on p. 11).

[12] ScikitLearn. $sklearn.model_selection.GridSearchCV$. URL: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html (visited on 12/17/2020) (cit. on p. 12).

[13] StatisticsSolution. *Pearson's Correlation Coefficient*. URL: https://www.statisticssolutions.com/pearsons-correlation-coefficient/?fbclid=IwAR1fMFuO2oy3SvUWTS8Gu9Pl0dyyFVAUtuF9G22yYSD YdY6qlayU (visited on 12/17/2020) (cit. on p. 12).