Feature Selection Methods with Classification Algorithms

Felicia Jacobsen
(Dated: December 15, 2020)

ABSTRACT

The research presented is a study of feature selection with the combination of two different classification methods, AdaBoost and an artificial Neural Network (NN). The well-known Wisconsin Breast cancer data consists of 30 features for a total of 569 patients, and has been used in order to predict between benign and malignant tumors. This study emphasises the importance of studying feature reduced models in machine learning algorithms, as well as serving as a motivational study of including classification algorithms into medical clinical trials. Two feature selection methods were used, correlation filtering on the training data, and the Recursive Feature Elimination (RFE) method. The latter was only used together with boosting. Without filtering, NN obtained the accuracy of 92% and AdaBoost gave 98%. The results with filtering of a correlation threshold of 0.95 gave an accuracy score of 97% with the NN and 98& with AdaBoost. With RFE, AdaBoost gave a perfect accuracy of 100% with 25 features and an ensemble of 300 decision trees of depth 1.

I. INTRODUCTION

In medicine today, doctors undergo many years of training in order to analyze patient data among many other important tasks. Such data involves X-ray images, CT- or MRI-images, tissue samples and so on. Some diagnoses requires the medical experts to analyze up to thousands of images per patient. Machine learning is one of many tools which will have a bright future in many fields, including medicine among them. Introducing ML into medical clinics can potentially help to diagnose patients, and also speed up the time of diagnosis. These results would evidently result in saving many lives.

Malignant tumors, also known as cancer, is a common diagnosis and many patients will undergo long and painful tests in order to be diagnosed. About 9.6 million of the world population die of cancer each year [6]. Because of the enormous incidents of cancer patients in the world, there is a particularly large interest in incorporating artificial intelligence in medicine today. With this said, this process follows many challenges, one of them is having access to large variations of training data in order for the model to recognise the cancer tumor.

In the case of classification problems, there are many characteristics of cancer to consider. They need to be extracted and fed into a model in order to classify the cancer. In the context of ML, we call these characteristics for features. The process of selecting which features to use is a common problem in Machine Learning today. Having a successful model is strongly dependent on a good set of features to train on. There is no go-to method for finding the correct features for a model and there exists several feature selection methods to choose from. Having a model that lacks enough relevant features or too many irrelevant ones will affect the model badly.

In this project, we're interested in exploring some feature selection methods and combining them with classification methods in order to classify between benign or malignant tumor of cell samples of breast tumors. This particular data set is known as the Wisconsin Breast Cancer data, and is vastly used in the context of machine learning. The two classification methods which will be applied to this data is an artificial Neural Network (NN), and a Boosting method. The two methods are highly popular for classification, especially the latter. Boosting algorithms has obtained a lot of attention after winning many ML-competitions, and they are known for representing a different perspective of ML algorithms because of their ability to transform weak models into strong models.

The strategy for solving this classification problem is to utilize two feature selection methods and to use NN and AdaBoost as classifiers. With this, we will study the relationship between the importance of feature selection methods as well as comparing the two different classification methods mentioned. This numerical study will be presented through five main sections. The first being the theory where the ground principles of these methods will be presented, the second being the method section explaining how these methods were implemented and explaining the analyzing process of the methods used. The results will be presented in a separate section and will be discussed in the following section. Lastly, we sum up the main results in the conclusion section.

For further engagement through this project, all of the results including the report, figures and codes will be available at **this GitHub repository.**

II. THEORY

Wisconsin Breast Cancer Data

The data set which will be used for application in the feature reduction and classification methods is the Wisconsin Breast Cancer data set. As mentioned earlier, this set of data has been used many times in the context of exploring ML algorithms. The data consists of a total of 30 features collected from 569 patients. In reality, there were 10 features which were extracted from the patients with a fine needle injected in the breast tumor. The cells from the needle were digitized as an image and the features were extracted from the images.

These are 10 main features, but the data set consists of a total of 30 features. This is because the other 20 features is constructed from the main features, including the mean of each feature, the standard error of each feature, and "worst" values which is the mean of the three maximum values of each feature.

The following list sums up the 10 main features extracted from the images:

- radius (mean of distance from center to the cell perimeter),
- texture (standard deviation of gray-scale values),
- perimeter,
- area,
- smoothness (variations of radius),
- compactness (perimeter^2 / area - 1.0),
- concave points (no. of concave points on the contour),
- concavity (severity of concave points),
- symmetry,
- fractal dimension.

There are a total of 357 benign and 212 malignant tumors in the data set.

The Importance of Feature Selection in Machine Learning

Feature selection is a method of reducing the number of input variables to include in a predictive model. Facing the problem of identifying the features from a data set and removing the irrelevant or less important features is a common task in the field of ML. In an earlier project, we have studied the importance of regularizing the weights in a linear regression model in order to obtain successful predictions [3]. These results indicated that fewer features is desirable because it reduces the complexity of the model and thus the case of overfitting.

The idea of introducing feature selection methods comes from the fact that having a model of reduced complexity is often desirable since it can reduce the variance. The consequence of high variance is the fact that the model overfits onto the training data. Having a model which is overfitted, means that it generalizes poorly onto unseen (test) data as explained in project 1 [3].

There exists many techniques of feature selection, but these methods is often categorized onto three, being

- **Filter methods**,
- **Wrapper methods**,
- **Embedded methods**.

The **filter method** is included in the preprocessing step of the data set. This means that it is included before introducing any predictive algorithm. In this process, features are selected on the basis of their scores in form of a statistical variable for their correlation with the target data. Correlation is a subjective term. In many cases, the Pearson correlation coefficient is most often used. The former is used for quantifying linear dependence between two variables X and Y, and its value can vary from -1 to 1. If for example X and Y has a correlation of -1, it means they have a strong negative linear dependence, and if it was 1, they would have a strong positive linear dependence. This correlation score is is given mathematically as

$$\mathrm{Corr}(X,Y) = \frac{\sigma_{XY}}{\sigma_X \sigma_Y}, \tag{1}$$

where σ_{XY} denotes the covariance between the variables X and Y, computed as

$$\sigma_{XY} = E[XY] - E[X]E[Y], \tag{2}$$

and σ_X and σ_Y denotes the standard deviation of X and Y respectively.

It is important to note that filter methods do not remove multicollinearity. It is possible to remove multicollinearity as well as highly correlated values based on the Pearson correlation score. Multicollinearity occurs when two features are highly intercorrelated, but can also exist when two features provide similar and repetitive results. This can lead to misleading results when attempting to determine how well each feature can be used most effectively to predict or understand the target value in a model. However, if the primary goal is to make predictions and not to understand the role of each feature, it is not important to reduce high factors of multicollinearity [2].

When calculating the correlation score, for example the Pearson correlation index between the features in a data set, this can be represented visually as a correlation matrix. This matrix is symmetric, with its diagonal elements being 1 since every feature will have perfect correlation to itself. The correlation matrix will for example show that the correlation index between features A and B to be 0.99 (almost perfect), and also show that correlation between B and A is 0.99. For the purpose of feature extraction, we would only be interested in only extracting either feature A or B, and thus only interested in studying the upper triangular or lower triangular elements in the matrix.

The **wrapper methods** consists of choosing a subset of features and train the model with the given features. We add or remove features in our subset and choose a combination subset of features based on the score of the model. The score could be accuracy in terms of classification and R^2 or MSE in terms of regression. This process is essentially a search problem for the features which give the highest score. These methods are computationally expensive because every given subset or combination of features involves training of the model, whereas the filter method requires no training in order to select features.

Some common techniques of such wrapper methods are

- Forward Selection,
- Backward Selection,
- Recursive Feature elimination.

Forward selection is an iterative wrapper method where the model includes start with one single feature at first, and keep adding features which best improves the model until an additional feature will not improve the performance of the model. Backward selection is similar, only that the model starts by including all the features and keeps removing one feature at a time until there is no more improvement of the model.

Recursive feature elimination (RFE) aims to find the subset of features which give the best performing model. It selects out and keeps either the best or the worst performing features aside at each iteration, and constructs the next model with the remaining features. It can rank the features based on the order of their elimination, but the importance of the features is based on the Gini importance, also called the Gini impurity index. Mathematically it is given as

$$G = \sum_{i=1}^{N} p_i(1 - p_i), \tag{3}$$

where N is the total number of classes of a given target variable, and p_i denotes the ratio of this class. This index ranges from 0 to 1. A Gini of 0 represent perfect equality, and 1 represent perfect inequality. If a node in a decision tree has a Gini of 0, it means that all training instances it applies to belong in the same class [1]. The higher Gini index of a feature, the more important it is considered as.

RFE method prune the least important features from the current set of features. This process is recursively repeated until the desired number of features is reached.

With ML models follow many important questions, such as: which variables are the most engaging in the model and the presence of correlation and so on. Tree-based models such as Boosting provides us with the benefit of answering these questions because they manage to provide with information about the importance of a feature (such as Gini) fairly easily. However with NNs, these benefits are considered unconventional since they are often considered as "black box" models, and it is therefore difficult to extract useful information for another purpose like feature explanations. For example, if a NN manages to classify a dog breed based on its picture, it is difficult to know what contributed to this prediction. Was it the dog's eyes, or its nose? It's color or its size of paws? Extracting this kind of information from the model is difficult. This is the main reason for RFE not being applicable to NNs.

Lastly, **Embedded methods** can select out features iteratively and be implemented in a model itself, i.e. the model has a built-in feature selection method. The most popular models with these qualities are the Lasso and Ridge regression which have built-in regularization. These methods were studied and explained further in an earlier project and can be found with **this link**.

AdaBoost

AdaBoost is one of many popular Boosting methods. These Boosting methods is based off Decision Trees and Random Forests. In order to fully comprehend the AdaBoost method, we must explain the concept of Decision Trees and Random Forests through two subsections.

Decision Trees

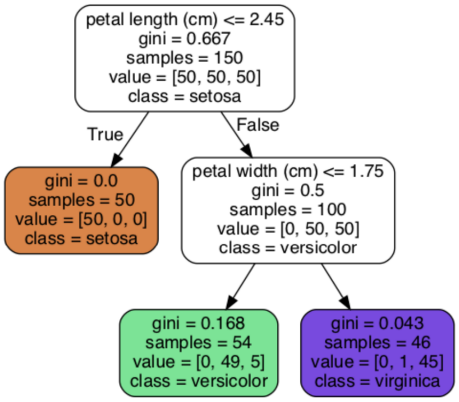

FIG. 1. Illustration of a type of Decision Trees. Image borrowed from Hands-On Machine Learning with Scikit-Learn and TensorFlow [1].

The figure 1 illustrates the main concept behind Decision trees. Suppose we have a total of 150 flowers of the type Iris. Iris flowers have different types. The tree in figure 1 illustrates that we at least have more than three classes of Iris flowers. Every tree is constructed from a root node, and is defined to be at a depth of zero. The root nodes divide the sample into two groups, based on if the statement: petal length is smaller or equal to 2.45 cm. If the statement is true, the sample is classified into being of the type "Iris Setosa" and is moved into the left orange node which is defined as a leaf node at depth 1. In this leaf node, all the samples belongs to one class, therefore the Gini index is zero, and the leaf node is considered as pure. The right internal node at depth 1 is however not pure (since Gini=0.5) and is thus considered as a node consisting of samples belonging to two or more classes, and is further split into two more leaf nodes. Note that the internal nodes have arrows pointing towards them, and arrows pointing away from them, whereas the leaf nodes have arrows pointing towards them, but no arrows pointing away from them.

More generally, the Decision Tree classifies the input data based on a series of true/false statements, the process starts with the root node and work your way down until you can't go further. In most often cases, the goal is to stop when all leaf nodes have a Gini index of zero. The choice of features in the internal nodes are determined in terms of accuracy. First, if the feature placed in the root node leads to the minimum Gini impurity in the nodes at depth 1, this will be chosen as the feature in the root node, and this process is repeated until all leaf nodes have Gini of zero. The features which contribute to a large decrease of the Gini index, will be considered as the most important since they contribute to more accurate classifications, e.g. the feature at the root node will be considered the most important and so on.

Random Forests

One main drawback with Decision Trees is that even though they work well with the data set which is used to create them, they generalize poorly onto new data. The Random Forests (RF) classifier combine the simple concept of Decision Trees with the ability to be flexible onto new data, resulting in a classifying method which can predict multi-class problems with great accuracy.

The algorithms starts off by making a Boostrap-sample from a data set. The idea of Boostrapping is explained further in an **earlier project**.

The next step is to create a Decision Tree using the Boostrap-sample, but only with a random subset of features at each iteration. Instead of considering all features to figure out how to split the root node, we now only consider for example two features as candidates for the root node. The feature which give the minimum Gini-index for the following two nodes as depth 1 will be chosen for the root node. Once a feature has been used, this will not be considered as a possible candidate for the next internal nodes. Proceeding with considering a subset as candidates for the next internal nodes. We build the tree as usual, but only considering a random subset of features for each step.

When a Decision Tree has been built, the procedure is repeated until we have a sample of trees. This process of building a RF can be summed up into three steps:

1. Create a Boostrap sample from original data set.
2. Consider a random subset of features as candidates for each internal node/split in the tree.
3. Repeat 1) and 2) until a desirable number of trees has been obtained.

The Random Forest is now created, but we need to run the training data through each tree in the RF. Let's consider the Iris flower classification again. We run a training sample through the RF and if the majority of trees classify the flower as e.g. an Iris Setosa, the RF model will consider the sample as a Iris Setosa. The most frequent result for a given sample will be the final predicted classification. Note: Boostrapping the data and using the aggregate to make a classification is called **bagging**.

The out-of-bag (OOB) samples not included in the training can be used to estimate the accuracy of the RF, and this score is used to evaluate its performance. If a OOB sample was to be run through the algorithm, we can use the number of trees which wrongly classified the sample and the number of trees which accurately

classified the sample to calculate the accuracy. We can use this accuracy in order to improve the RF by tweaking its hyperparameters. These consists of the number of estimators (number of trees), maximum no. of features to split a node, and the maximum depth for each tree.

As mentioned earlier, extracting important information from tree-based methods such as RF is not a considered a challenging process. Because of this reason, the algorithm is considered as a "white box." For instance, it is possible to extract the measure of the relative feature importance on a prediction.

The RF method has an advantage of being a so-called "white-box" algorithm, and can consist of many weak learners (trees with small depths are considered as such) and generalizes well onto new data, thus avoiding the case of overfitting. A process of obtaining a prediction with RF can be done parallelized, as each Decision Tree can be created independently of the other. Overall, RF is fast, flexible and conceptually simple.

AdaBoost

As mentioned earlier, this algorithm is based on RF and Decision Trees. In contrast to a RF, the trees built with AdaBoost normally consists of a root node and two leaf nodes. This is generally called a stump, meaning that the method consists generally consists of a forest of stumps instead of trees. Stumps only make a single split at the root, and thus only using a single feature to make a prediction. The stump will therefore make inaccurate classifications, and is referred to as "weak learners."

In a RF, each vote corresponding to a tree is equally much worth, the worth of a vote is defined as the "amount of say," but in a forest of stumps made with AdaBoost, the amount of say between the stumps are not equally distributed. Some stumps get more say in determining the final classification the other stumps.

Every tree in a RF are made independently of each other, while in the AdaBoost model each stump is made dependently of each other. It is a iterative process where the next stump is a correction to the previous stump. Including errors of which first stump will influence the making of the next stump and so on.

The process of building the AdaBoost model can be generally considered as seven main steps:

1. Build a weak learner, they are often stumps.
2. Make stumps for each feature. Choose best feature to split in the root node based on the lowest Gini index.
3. Initialize the stump, each sample has equal weight.
4. Determine the accuracy of the stump, i.e. the total error. Some stumps get more the amount say in the final classification the others.
5. Calculate the amount of say for the stump based on total error.
6. The next stump are a correction to the previous stump, where the new weights are calculated by the amount of say.
7. Repeat the process until a desirable amount of stumps are obtained.

Let us now consider the classification problem with the Iris flowers as described earlier. We consider three different features, the petal length, the petal width and the petal color. Each sample in the training set are assigned with a sample weight that indicates how important the sample is with regards to classification. All sample weights will be initialized as uniformly distributed, where every sample weight is equal to $\frac{1}{\text{number of samples}}$. This initialization makes all the samples equally important. After we have passed the training data thought the first stump, the sample weights will change in order to make the next stump.

Starting by making the first stump, one of the three features is assigned to the first node depending on which makes the best classification of the training samples. We start by how well petal color classifies the sample. The result is that 10 % of the samples were incorrectly classified. We proceed by constructing a stump for the two remaining features: petal width and petal length and we calculate the Gini index for all three stumps. Let's suppose the Gini index of the stump with petal color gives the lowest Gini, so this will be the first stump in the forest of stumps. The process proceeds by calculating how much the amount of say for this stump based on how well in classifies the samples. We already know that we have a total of 150 flowers in the training set and that the misclassification error from this stump var 10%, thus having misclassified 15 samples. The total error corresponding to this stump is the sum of the weights corresponding to the misclassified errors. Therefore, in this case the total error will be **Total error** $= \frac{15}{150}$. Since the sum of the sample weights will be 1, the total error will always range from 0 to 1.

The total error corresponding to the stump will be used to calculate the amount of say with the following formula

$$\text{Amount of say} = \frac{1}{2}\log_e\left(\frac{1-\text{Total Error}}{\text{Total Error}}\right), \qquad (4)$$

thus if the stump gives a large total error, then it will have a small amount of say. Stumps with small errors will have a higher amount of say and thus have larger impact on the final classification.

The weights from the misclassified samples is used to calculate the amount of say for each stump. How do we modify the weights in order make the next stumps? Going back to the Iris flower example, we know that 15 samples were misclassified. Since every sample got the same amount of weight, the importance of correctly classifying those 15 samples did not get emphasized. When making the next stump, we will emphasize the importance of correctly classifying these 15 samples in the next stump by increasing the sample weights corresponding to those 15 incorrectly classified samples. The new sample weights for the misclassified samples will calculated by

$$w_{\text{new}} = w_{\text{old}} \cdot e^{\text{amount of say}}, \tag{5}$$

where w_{new} denotes a sample weight from the new stump, and w_{old} denoting the value of a sample weight for the old. If the previous stump classified the sample accurately, then the amount of say is large and the new sample weight will be scaled with a large number. If the amount of say is a small number, the sample weight for the new stump will be scaled by a small number. Thus, the new sample weights will always be a little larger than the previous weights.

For the new sample weights for accurately classified samples will be calculated by the following formula

$$w_{\text{new}} = w_{\text{old}} \cdot e^{-\text{amount of say}}, \tag{6}$$

and thus the correctly classified samples will be scaled down. Note that if the amount of say for the previous stump were small, the new sample weight will be almost unchanged (scaled with a value ≈ 1).

We have now calculated the new sample weights for the next stump, and these weights need to be normalized in order for them to sum up to 1. These normalized new sample weights will be used to construct the next stump.

These new (normalized) sample weights will have impact on which feature to choose for the next stump in terms of Gini index. The Weighted Gini index will have more emphasis on correctly classifying the samples that were misclassified by the previous stump, because these specific sample weights are now increased.
Alternatively, instead of using a Weighted Gini index, we can randomly select from the old samples based on their new sample weights in order make a new collection of samples. By randomly picking out samples based on the sample weights as a distribution, the larger sample weight the sample has, the more duplicates of this specific sample will exist in the new collection of samples. The new collection of sample will have the same size as the previous one, but having more duplicates of samples that was previously misclassified samples. The new samples will be initialized with having equal sample weights, but since this new collection contains more duplicates, they will act as a large penalty when being misclassified and thus have larger impact of the choice of feature to create the split in the new stump.

This process of making stumps, and scaling the weights in order to make new stumps is repeated until we have a forest of stumps. The stumps which is more accurate will have a large amount of say, while the stumps with low accuracy will have a small amount of say. Since the sum of the amount of say from the accurate stumps will be larger (if we have a good model) the model will classify the sample according to the prediction of these stumps. This is essentially the process of AdaBoost.

The main drawback of using AdaBoost, is that the process can not be parallelized since every stump depends on the previous.

Neural Network

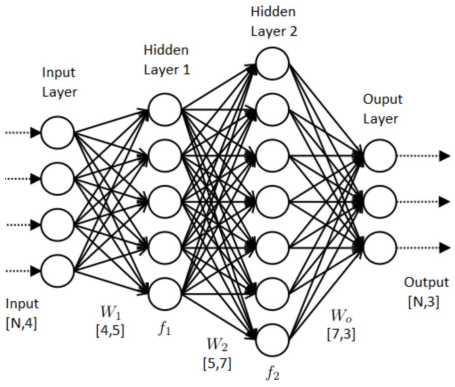

FIG. 2. Figure borrowed from Medium.com. This image shows the illustration of the architecture of an artificial neural network.

A connected NN consists of an input, one or several hidden, and an output layer. An illustration of an example of a network is shown in figure 2. In a Feed Forward Neural Network the information moves forwards from one layer to the next. The information which is passed through the network depends on the inputs, weights and biases, and activation functions. When feeding the input from the input layer to the first hidden layer, we matrix-multiply the input with the weights and thereafter sum with the bias vector. Between each layer, the signal is transformed with an activation function. These functions introduce non-linear behaviour to our model in order to predict non-linear relationship in the data set. The process signal transformed by an activation function is often referred to activating the signal, or activated output. Some of these activation

functions are listen in the Appendix B at the end of the report.

The mathematical expression of passing forward an input form one layer to the next layer is as follows

$$a^l = f(z^l) = f(y^{l-1}W^l + b^l), \tag{7}$$

where f denotes an activation function, y^{l-1} is an column vector of target data along with z^l. The W^l denotes the weight matrix and the b^l is a bias column vector. The shapes are $z^l, b^l \in \mathbb{R}^n$, $y^{l-1} \in \mathbb{R}^p$, and the weights matrix has the shape of $W \in \mathbb{R}^{p\times n}$. Note that the letter l tells us which layer we're currently in. This specific algorithm in equation 7 is called the *forward pass*. The need for training is crucial before making a prediction. The process of training in terms of NN is called the backpropagation or simply backpropagating.

Backpropagation is when sending the same input, which has earlier been passed forward as described by equation 7, backwards in the NN. As the signal moves backwards, the weights and biases corresponding to the signal will be modified with a Gradient Descent (GD) method between each layer in order to improve the model. This GD method is to move in the opposite direction of the gradient of the cost function, because this is the direction where a minimum is obtained. This method is further elaborated in project 2 [4].

The backpropagation algorithm can be described mathematically with the following four equations

$$\delta^L = \nabla_a E \odot f'(z^L), \tag{8}$$

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot f'(z^l), \tag{9}$$

$$\frac{\partial E}{\partial b^l_j} = \delta^l_j, \tag{10}$$

$$\frac{\partial E}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j, \tag{11}$$

where δ is either called the local gradient or simply delta. This is used to update the weights and biases between each hidden layer. The $\nabla_a E$ denotes the gradient cost function with respect to the activated signal a in layer the last layer L. The activated signal was computed as given i eq. 7. $f'(z^l)$ denotes the gradient of the activation function of the signal z^l in layer (current layer) l. w^{l+1} denotes the weights matrix in layer $l+1$ (next layer). Whereas $\frac{\partial E}{\partial b^l_j}$ and $\frac{\partial E}{\partial w^l_{jk}}$ denotes the partial derivative of the cost function with respect to biases and weights in layer l, respectively.

Using the expressions in eq. 10 and eq. 11 in order to update the weights and biases, the two following expressions are obtained

$$b^l_{j,new} = b^l_{j,old} - \eta \frac{\partial E}{\partial b^l_j}, \tag{12}$$

$$w^l_{jk,new} = w^l_{jk,old} - \eta \frac{\partial E}{\partial w^l_{jk}}, \tag{13}$$

where E represents the cost function, and η representing the learning rate. Observing from eq. 12 and 13, the bacpropagation algorithm utilizes the gradients to update each parameter with a GD step.

When the all of the training data has been passed once forward and backward through the NN, we say that we have performed one *epoch*. This is a measure of the number of times the network has been trained. If the total number of epochs is too small, the NN may not have converged to the optimal solution. However, if epochs are too large there will be a increased probability of obtaining a network which overfits onto the training data.

There are other important hyperparameters to take into account in addition to the number of epochs. These are the number of hidden layers, number of nodes in each hidden layer, the activation functions between each layer and the learning rate. There is no conventional way of determining these hyperparameters. But many consider this task as a matter of trial and error as the go-to method for obtaining the favourable hyperparameters for the problem at hand.

When dealing with regression problems, no activation function will be assigned between the last layer and the output layer, since we then want to obtain unbounded values. However, if dealing with a classification problem either Softmax (for multi-class problem) or Sigmoid (for two-class problem) is used between the last hidden and output layer since the main task is to obtain bounded values of probabilities for each node (class) which sum up to 1.

Summing up the process of building and using the NN into six main steps:

1. Initialize the weights from a Normal distribution and a column vector of zeros for the biases. Using Xavier or He initialization for the weights is commonly preferred.
2. Tune the hyperparameters.
3. Train the network by bacpropagation signal of training data.
4. Predict an output with the network on the test data as input.
5. Evaluate the prediction in terms of accuracy, F1 or recall.

6. Tweak hyperparameters and repeat the steps in an attempt to obtain higher accuracy score.

A. Confusion Matrix

The results from models when solving classification problems is conventionally represented with so-called confusion matrix along with the accuracy score. The confusion matrix is a table that is often used to describe the performance of a classification model on a set of test data for where we predicted and true values. Each matrix element C_{ij} counts the number of inputs of label i which was predicted as label j [4]. The elements on the diagonal show the "true positives", meaning the number of correctly predicted targets for each class. The values outside this diagonal provides information about what was incorrectly classified.

III. METHOD

Data Preprocessing

The Wisconsin breast cancer data is imported from Scikit-Learn's dataset package by using a simple line:

```
from sklearn.datasets import load_breast_cancer
```

By specifying that the data will be returned as a tuple with

```
load_breast_cancer(return_X_y=True)
```

Where the first element in the tuple will be transformed into an array called X, and have the shape of $\mathbb{R}^{n\times p}$ where n is the number of patients which is 569, and p is the number of features which is 30.

The second element from the tuple is the target array, y. It will be of shape $\mathbb{R}^n$ and will consist of 0's and 1's, where 1 being a benign tumor and 0 being a malignant tumor.

When using a data set from Sklearn, it is possible to print a detailed description of the data set by

```
load_breast_cancer().DESCR
```

The total data set is split into a training and test set, where the former is used for training and feature selection, and the latter is used for validation. The training set is 80% of the total data set, and 20% goes to the test set.

Feature Selection

The theory section presented a number of various feature selection methods, only two of these will be applied to the training data. One of the filter methods will be utilized, namely the method where a threshold can be predetermined of the correlation relationship between all features in order to remove those who don't satisfy this threshold. The second feature selection method will be the RFE method which is one of the Wrapper methods as explained in the theory section.

When the variable columns (or features) has been selected by the feature extraction method on the training data X_{train}, these same features will also be extracted from both training and test data X_{train}, X_{test}.

1. Removing Features by Filtering

The Pearson correlation factor will be used as a score to measure the correlation between the features. The correlation threshold will be set to 0.95, because this is closely to having perfect correlation. Having a lower threshold with e.g. 0.90 would reduce too many features and the model will not be able to make many correct classifications.

Note that the threshold is set to 0.95 because the Pearson correlation can vary from -1 to 1, and thus if there exists a correlation factor of 0.95 or -0.95 between two features, this would imply almost perfect correlation.

The correlation factor between each feature will be illustrated in a **correlation matrix**. This is a square matrix with dimensions equal to the number of features, and thus will be symmetric. Its diagonal elements will be 1 since every feature will have perfect correlation with respect to itself. If feature A has high correlation with feature B, only *one* of them will be extracted from the data set. It is not important whether selecting the upper or lower triangular part of the correlation matrix, but it is important to not take the diagonal elements into account (if so all the features would be extracted by the filtering). We select the upper triangular.

We take the absolute value of all the elements in the upper triangular, and select the indices of the columns which has a correlation higher than 0.95, and make a list of these indices. By

```
dropped_features = [
    column for column in upper_tri.columns if
    any(upper_tri[column] > 0.95)
]
```

where upper_tri is the upper triangular elements of the correlation matrix. When selecting the upper triangular elements, we ensure that we only select one out of two of the correlating features and thus disregard the corresponding correlation which is symmetrically located on the lower triangular matrix.

Proceed by printing the names of the dropped features by the filter and return the indices of the kept features to be used later in the NN and AdaBoost method.

2. Recursive Feature Elimination with AdaBoost

This method is mainly used for tree-based methods where the Gini-index is easily extracted. Therefore, this method will only be used with the AdaBoost, and not NN.

The RFE method is implemented from Scikit-learns package by

```
from sklearn.feature_selection import RFE
```

With this method, we specify the parameters **estimator** and **n_features** which is the estimator used for training between each feature extraction and the number of features to keep in the model respectively. This RFE method requires an estimator that assigns weights to features. The estimator needs to have a coef_ or feature_importances_ attribute in order to work [5]. The AdaBoostClassifier provides such an attribute, and the feature's importance is based on the Gini index from each feature.

RFE from Scikit-Learn uses the specified estimator to train on the whole dataset, then extracts the importance of each feature and the least important feature (with the lowest ranking) is removed from the initial set of features [5]. The process is repeated until a predetermined set of features is reached. This is specified in the n_features parameter in the RFE class.

AdaBoost

AdaBoost classifier is implemented from the Scikit-learn package trough importing

```
from sklearn.ensemble import AdaBoostClassifier
```

and provides a great tool for measuring the feature's importance by analysing how much the stump nodes that use that feature reduce impurity across all trees in the forest. The feature's importance can be extracted through

```
AdaBoostClassifier(base_estimator,
    n_estimators,
    learning_rate).feature_importances_
```

A great quality with the AdaBoost algorithm is that it's fairly easy to measure the relative importance of each feature in the model. By using AdaBoost from Sklearn, it provides us with the feature's importance by analyzing how much the tree nodes that uses a specific feature reduce impurity across all trees in the forest. It also analyzes how many times a feature has been used for all the root nodes in the forest, and the ones that appear most often will be considered as important. Sklearn computes this score automatically through the mentioned attribute for each feature after training, and scales the results so that the sum of all feature importances add up to one (normalization).

By looking at the feature importance it can be determined which features to select out because they don't contribute enough to the forest of stumps.

In order to use the AdaBoostClassifier from Sklearn, we need to specify three important parameters. The base_estimator parameters is used to specify the weak learner to use in the boosted ensemble. As mentioned in the theory section, stumps are often used as a weak learner a Boosting method. Therefore, we specify this base estimator to be Decision Tree with depth equal to 1, which is equivalent to being a stump. This is however the default parameter for this class. The n_estimators specify the total number of boosted weak learners. The learning_rate parameter shrinks the amount of say from each stump. There exists an trade-off between the choice of learning rate and the number of stumps in the model. The amount of say from each stump will not be scaled down, such that this parameter will be set to be 1 which is the default parameter. Therefore, we only experiment with the n_estimator parameter in order to optimize the accuracy score of from the AdaBoost method.

The AdaBoostClassifier gives us precise values for a given input instead of probabilities. E.g. for a given patient, the predicted class is either 0 (for malignant) or 1 (for benign).

Neural Network

A clear description of how this exact code was implemented was also provided in an earlier project [4]. The implemented FFNN is a class which takes a flexible number of layers and nodes for each hidden layer. The activation functions can be chosen between each layer when the user provides with a list of strings. The user needs to specify the number of epochs as well. The network utilizes stochastic GD with the training set split into a desirable number of mini-batches, thus the class takes learning rate and the number of mini-batches as parameters in addition to the other earlier mentioned.

3. Initialization

The weights are initialized using He initialization as mentioned in the theory section. The biases will be initialized as column vectors of zeros. The shapes of the weights and biases are specified according to the number

of nodes in the surrounding layers [4]. The lines of code where initialization is performed is as follows

```
weights = np.random.randn(layers[i],layers[i+1])
    * np.sqrt(2/layers[i])
biases = np.zeros((1, layers[i+1])).
```

where "np" specifies the Numpy framework for Python. We also need to make a series of if-statements, because a list of strings are specifying the activation functions for each layer. The network class contains list attributes which specifies the activation functions and their gradients for all layers.

4. Forward Pass

The NN class contains a forward pass method which is an implementation of the algorithm in eq. 7. This method takes an input signal z, and loop over the number of layers. Before a signal reaches each layer, the signal is activated with the activation function specified by the list of strings which has been provided by the user. The forward pass thus returns the predicted output from the last layer.

5. Backpropagation

The NN class contains another method for backpropagation.

We start by looping over the no. of epochs, which we refer to as the outermost loop. A second level loop, iterates over instances in every batch, and we will keep referring to these two loops as the main loop. In the beginning of the main loop, instances are randomly chosen from input and output-data in our mini-batch using SGD implementation. We call the input- and output-data as **x_batch** and **y_batch**. We specify our signal **z** and activated signal **a** as empty lists, except the fist column in list **a** is the first input **x_batch**. A third loop runs over every layer, the forward pass algorithm is applied to the signal. Two lists are stored containing output signal and activated output signal for every layer.

Inside the main loop, we define the gradient cost function with the cross entropy 30 from Appendix C. The local gradient, delta, is initialized using eq. 8 which needs the list of activation derivatives of the signal **z** and is multiplied with the gradient of the Cross-Entropy. The local gradient starting from the parameters obtained from the forward pass in last layer L to proceed.

In the main loop, the list of the cross entropy with respect to the weights, called **dW**, is initialized with the mean of the local gradient as in eq. 11 from layer L. The list of partial derivatives of cross entropy with respect to the biases, **dB**, in the current layer is updated with the activated signal from the second last layer and and the local gradient delta in layer L, as in eq. 10.

Another loop is added inside the main loop, this can be called this the delta loop, which iterate backwards through the hidden layers. Using the obtained local gradient, dW and dB for the last layer, i.e. delta[-1], dW[-1] and dB[-1], the loop is ranged from 2 to L. The index is specified as [-i], which means that the lists are updated from the second last layer to the first (remember that the last layer gradients were computed right before the delta loop due to δ^L having a slightly different expression).

A list of dW and dB has been obtained between all the layers. Proceeding by looping over all layers inside the main loop to update the weights and biases using the algorithm as described in the theory, eq. 12 and eq. 13.

Let's summarize the NN class:

1. init(self, layers, activation functions): initialize weights and biases and adds lists of activation functions and activation derivatives to be used later based input values given by the user. The input needs to be a list of integers specifying the number of layers. The activation functions is defined by the user with a list of strings.

2. forward_pass(self,x): a method for passing the data through NN and predicts an output. Returns predicted outputs.

3. back_prop(self, x, y, η, epochs, batch_size): method which will train the network a number of times specified by epochs parameter. Returns nothing, but updates the weights and biases iteratively based on derivatives of activation functions (SGD method), and thus trains the network.

The FFNN class gives probabilities for a given input belonging to a specific class. We transform the probabilities into an one-dimensional vector in order to compare them with the test target data.

Tuning hyperparameters for AdaBoost and NN

When tuning hyperparameters in the Adaboost function, we only tune the number of stumps we want in the forest in order to obtain an accuracy score which is closer to 1. There will sometimes be a upper limit for the possible accuracy score depending on the data set, problem and the model used. We try with a number of runs for different values of stumps in order to come closer to the highest possible accuracy score.

For the NN model, we tried with the same architecture as in the earlier project [4], but needed to increase the

number of mini-batches from 15 to 30 and decrease the learning rate from 0.1 to 0.01 and have 2000 epochs instead for 1000 in order to obtain higher accuracy score. Once the hyperparameters has been selected based on the highest accuracy, the filtering method will be applied for this fixed set of parameters. There has earlier been observed that some data sets or other classification problems favour some activation functions than others [4].

Accuracy score

It is common to score the predictive performance of a classification model by its obtained accuracy score. This score is calculated by the following formula

$$\textbf{Accuracy score} = \frac{TP+TN}{TP+TN+FP+FN}, \tag{14}$$

Where TP is an abbreviation of *true positives*, TN stands for true negatives, FP stands for false positives and FN stands for false negatives. With the specific problem of classifying Wisconsin Breast cancer data, the TP indicates the total number of correctly classified malignant cancer, and TN is the correctly classified benign tumors. FN is the number of patients which has been wrongly classified with having benign tumor, when they in fact had a malignant breast tumor, and FP is the number of patients which has been wrongly classified with malignant tumor when they in fact had a benign tumor.

Presenting the Results

To present the results, there will be made an correlation matrix, with the Pearson correlation index of all the features in the data set to indicate which of the features are highly correlated. We take the absolute values of all the elements in the correlation matrix.

The reduced data set which has been reduced with the filtering method will be applied with both AdaBoost and NN method and their corresponding accuracy scores will be calculated. The filter will have a correlation threshold of 0.95, 0.90 and 0.85.

The two classification methods will also be applied on the data set which has not had any feature reduction.

The number of stumps, which is one of the hyperparameters of the AdaBoostClassifier were varied from 100 to 500 with a step length of 100 in order to estimate the optimal number of stumps for this specific classification problem. No future selection method were applied when finding this hyperparameter.

With the AdaBoostClassifier, we will also apply the RFE method with a predetermined number of 15, 20, 15 and 10 features. A heatmap of the correlation matrix will be given for 20 features obtained with RFE and AdaBoost as classifier.

IV. RESULTS

A. Correlation Matrix

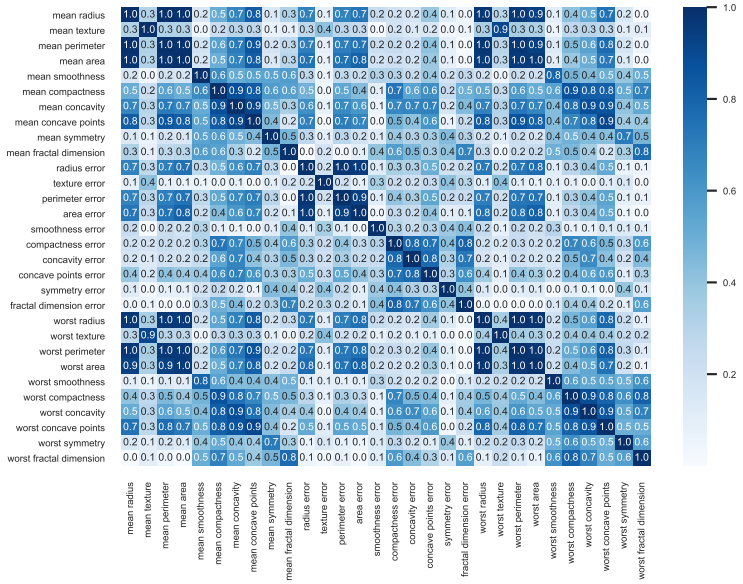

FIG. 3. This matrix contains of all Pearson correlation indexes between each pair of features in the Wisconsin Breast cancer data. Note that the absolute values of the correlation index has been calculated and thus having no negative elements.

Figure 3 shows a computed correlation matrix between each of the 30 features in the Wisconsin Breast Cancer data set. Each correlation index has been calculated using the Pearson correlation index given in equation 1. The absolute value has been taken of each element such that only positive correlation values is shown.

AdaBoost with Feature Selection

TABLE I. AdaBoost method without feature selection. The number of weak classifiers (stumps) in the boosted forest has been tweaked in order to determine which hyperparameter to use for the feature selection methods.

No. of stumps	Accuracy
500	0.96
400	0.96
300	0.98
200	0.94
100	0.93

Table I has the obtained accuracy scores for different values of weak classifiers in the boosted forest given the AdaBoost method. These weak classifiers were stumps, i.e. decision trees of depth equal to one. The model predicted on the Wisconsin Breast cancer test set containing 20 percent of the total data set of 569 patients.

TABLE II. The obtained results from having different values of correlation thresholds for the feature selection filter with the AdaBoost as classifier. The number of weak classifiers (stumps) in the boosted forest were held constant at 300. For the other hyperparameters, the default values were used.

Threshold	No. of stumps	No. of features	Accuracy
0.95	300	23	0.98
0.90	300	20	0.96
0.85	300	17	0.95

Table II is the obtained accuracy scores with feature thresholds of 0.95, 0.90 and 0.85 of the Pearson correlation index. These filters resulted in a remaining 23, 20 and 17 features respectively. The classification model used was the AdaBoost method with a fixed number of 300 stumps for each filter. These results were obtained using 80 percent of the total data set (Wisconsin breast cancer), and predicted on the test set which consisted of 20 percent of the total data.

TABLE III. Obtained accuracies when varying the predetermined number of features the RFE method will keep in the AdaBoost model. RFE method ranks the features according to the Gini index for every iteration, and removes the feature which has the lowest rank until the predetermined number of features are obtained.

No. of stumps	No. of features	Accuracy
300	25	1.00
300	20	0.98
300	15	0.97
300	10	0.94

Table III contains the obtained accuracy scores with the wrapper method RFE. A number of features were predetermined at 25, 20, 15 and 10 before the feature selection method was applied. The classification model used was the AdaBoost method with a fixed number of 300 stumps for every iteration. These results were obtained using 80 percent of the total data set (Wisconsin breast cancer), and predicted on the test set which consisted of 20 percent of the total data.

NN with Feature selection

TABLE IV. This table contains the obtained accuracies when having filtering feature selection with Pearson correlation threshold of 0.95, 0.9 and 0.85. The implemented NN was used as classifier predicted on the training set of Wisconsin Breast cancer data.

Threshold	No. of features	Accuracy
0.95	23	0.97
0.90	20	0.95
0.85	17	0.95

Table IV contains the obtained accuracy scores using our own implemented NN and with a filtering feature selection method using Pearson correlation index as threshold. The thresholds were set at 0.95, 0.90 and 0.85, the one of the features in the pair which had correlation above this threshold was picked out from the data set. The network was thus trained using the reduced data set.

The networks architecture had a total of 6 hidden layers and 300, 200, 150, 100, 70 and 25 nodes respectively in each layer. Between each layer, the hyperbolic tangent was used with Sigmoid before the output layer since this is a two-class classification problem. There was a total of 30 mini-batches, 2000 epochs and a learning rate of 0.01.

NN and AdaBoost without Feature Selection

TABLE V. This table has the obtained results from using the implemented Neural Network and AdaBoostClassifier from Sklearn when classifying Wisconsin Breast Cancer data. These accuracy scores were obtained without performing any feature selection.

Model	Accuracy
AdaBoost	0.98
NN	0.92

Table V shows the NN and AdaBoost method. The AdaBoost model had a total of 300 weak classifiers being stumps. The networks architecture had a total of 6 hidden layers and 300, 200, 150, 100, 70 and 25 nodes respectively in each layer. Between each layer, the hyperbolic tangent activation function listed in Appendix B was used. However the Sigmoid activation function was used before the output layer, because this is a two-class classification problem predicting patient breast tumors between Benign and Malignant. There was a total of 30 mini-batches, 2000 epochs and a learning rate of 0.01.

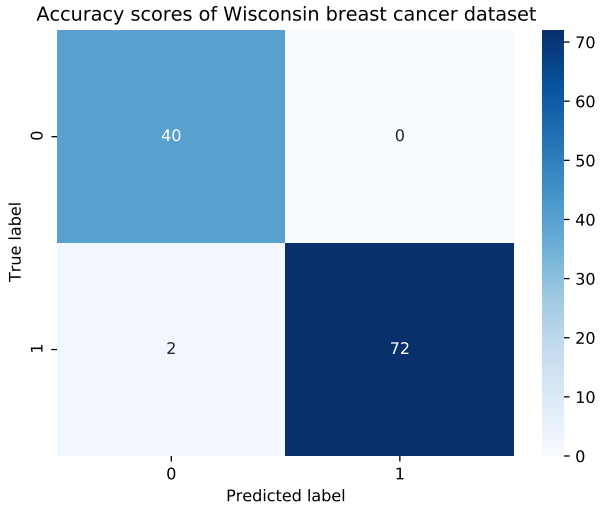

FIG. 4. Confusion matrix obtained from AdaBoostClassifier class by Sklearn for 300 weak classifiers (stumps). No feature selection method was performed. The data set used was the Wisconsin Breast cancer data, where output 1 is Benign breast tumor and 0 is Malignant breast tumor (i.e. cancer).

Figure 4 shows the computed confusion matrix obtained from AdaBoostClassifier class by Sklearn with no feature selection method applied. This shows an accuracy score of 98%.

V. DISCUSSION

Feature Selection

From the correlation matrix in figure 3, the upper triangular matrix indicates that there are 7 pairs of features that have perfect correlation. Note that they may have correlation between 0.95 and 1.0 since they will be rounded up if they are printed with only one decimal. As explained in the theory, these pairs of highly correlated features indicate that it's only necessary with *one* of them. Having pairs of features which essentially explains the same thing can give the case of overfitting the model when performing training. One of these seven pairs of highly correlated features were reduced by the lowest filter of 0.95 as indicated in table II and IV.

AdaBoost

Since the highest accuracy score was obtained with the AdaBoost model with a total of 300 weak classifiers as given in table I, this hyperparameter was also chosen along with RFE and filtering when studying feature selection. These values indicate that 300 weak classifiers is the optimal hyperparameter for this specific classification problem. Having too many classifiers in the boosted forest may have lead to overfitting, which may be the case when having 400 or 500 stumps in the model and the reduced accuracies when having 400 or 500 stumps instead of 300 may indicate that the model is too well tailored onto the training data and generalize poorly onto the test data. Having too few estimators in the boosted forest may lead to the case of obtaining a underfitted model. Meaning that the model is too sparse for making a highly accurate prediction on unseen data (test data). Reducing to 300 and 200 weak estimators also reduces the accuracy score. Because of this, table indicate that 300 stumps is an optimal hyperparameter for this specific classification problem.

Proceeding with 300 stumps in the AdaBoost model, a filter consisting of a correlation threshold of 0.95, 0.90 and 0.85 is applied on the training data before training the model on the data. An accuracy score of 0.98, 0.96 and 0.95 is obtained for these three filters. With no feature selection, the accuracy score was 0.98 according to table V. Thus the results of including a filter gives no improvement of the accuracy score and may indicate a poor method of feature selection when using AdaBoost as classifier. With *stricter* filters of 0.90 and 0.85, the results shows a decrease in terms of accuracy score. Increasing the filter and thus reducing from 23 to 20 or 17 features indicate that the model becomes too sparse when removing up to 10 features.

When including RFE for reducing features along with the AdaBoost method, the perfect score was obtained with reducing down to 25 features according to table III. This shows an increase of accuracy in terms of choosing 25 features in comparison to when including all 30 features which leads to accuracy of 98%. Thus leaving us with a perfect accuracy score of 100%. This iterative feature selection model indicates that this may have the strongest performance for reducing features and obtaining a higher accuracy for this specific classification problem in comparison with the filter method. However, when having 20 features as the predetermined value for the RFE method, it gives an accuracy of 98% which is not an increase when including no feature selection as observed in table V. But reducing from 25 to 20 features gives no increase or decrease in accuracy. When reducing even further to 15 and 10 features, the accuracy score decrease also. This may be an indication of obtaining a too sparse model, which lead to the decrease in accuracy. Even though the accuracy decreases, the accuracy is relatively high even for 10 remaining features (being 94%), which can also be an indication that this wrapper method is better at reducing features in comparison with the filter method. This can be because the Gini index gives a stronger estimate of feature importance than the correlation index. The AdaBoost method allows for weak classifiers giving incorrect predictions since the incorrectly classified stumps have lower impact when doing the final vote. In addition to this, when

reducing the features which are used the least amount of times for the stumps, this may not give a very different model even when reducing the features. Because some features were not often used, or may be not used at all. Reducing 5 of these features did result in a higher accuracy score, which gave more room for features which appeared more often in the boosted forest. The boosted forest now consists of features which appear the most, giving perfect predictions when performing on the test data.

The confusion matrix in figure 4 illustrates that no Malignant tumors (class 0) were wrongly classified as a Benign tumor (class 1). However, Benign tumors coming from two patients was wrongly predicted as having Malignant tumors. In terms of cancer research, these kind of "conservative" predictions are more favourable, since wrongly predicting benign as being malignant evidently leads to less harm since this will be uncovered afterwards, and the tumor can grow without being harmful, and can neither spread or invade other parts of the human body. Some benign tumors can progress into being malignant, but this is in terms months or years. However if predicting a malignant tumor with being benign and sending the patient home without no treatment can lead to serious consequences. The most serious outcome is the death of the patient depending on the severity on the tumor. Without treatment, malignant tumors can grow large or spreading to other areas in the body, making the cancer more aggressive. Having a model which is conservative in this sense is viewed as more favourable.

Neural Network

When combining the filter method before training the network, the highest resulting accuracy was when having a filter of 0.95 leading to an accuracy score of 97%. Stricter filters of 0.90 and 0.85 resulted in a decrease of accuracy, as observed for the AdaBoost model, which may also indicate obtaining a too sparse model. However, the highest accuracy of 98% was obtained having no feature selection at all. The results also thus show that the NN gives less accurate predictions without feature selection according to table V. This table gives an accuracy of 92% when having all 30 features, and increase to an accuracy of 98% when having a filter of 0.95. This indicates that this specific feature selection method reduces unnecessary features which reduce the case of overfitting and thus obtaining a high performing predictive network in terms of accuracy.

Comparison

Using the RFE method with 25 features with AdaBoost with 300 weak classifiers resulted in the highest obtained accuracy of 100% in comparison with feature threshold with AdaBoost and NN. This indicates that combining this filtering method along with AdaBoost is the best procedure at classifying the Wisconsin Breast cancer data compared to the other procedures used in this study. However, when combining AdaBoosting with the filtering resulted in giving only the same accuracy as having no filter at 98%, or decreasing the accuracy when increasing the strictness of the filter. The filter method only resulted in no effect or decreased accuracy score with the AdaBoost. This indicates that the RFE method which scores the features according to the Gini index is a better estimate for selecting out features compared to the correlation index, and this evidently leads to a more accurate model.

With no feature selection method, AdaBoost gives the highest accuracy of 98% while the NN gives the accuracy of 92% according to the values in table V. In terms of overall accuracy, the results with AdaBoost gave most often higher accuracies compared to the NN. This same network also had a long execution time in terms of minutes and needing a total of 2000 epochs for training, whereas the boosting method had a quick execution time in terms of only a few seconds. Even with filtering, the NN gave the highest accuracy score of 97%. Filtering gave a increased accuracy for the network in contrast to AdaBoost which gave no improvement.

The filtering method had an advantage of being computationally fast since it involved no training, while the RFE involves training for every iteration and include the ML algorithm in order to select features.

Even though filtering gave no improvement of the prediction in terms of accuracy with the boosting method, it can often be viewed as more favourable having a model with 23 features obtaining 98% accuracy instead of a model consisting of 30 features with 98% accuracy. The accuracy score is often used for determining the best model for a specific problem, and the number of features is rarely considered as part of a performance criterion in ML. However, reducing the number of features to analyze may have the possibility of simplifying the work of doctors when detecting malignant tumors.

AdaBoost gave the highest accuracy scores both with and without feature selection methods in contrast to using NNs. This indicates that the AdaBoost method is both faster and more accurate than NN for this specific problem.

Further Improvements

With applying the filtering method on the data set and predicting with AdaBoost or NN did not result in

the highest accuracy score. As mentioned in the theory, the NN has a disadvantage of being a so-called black box. This makes extracting a measure of the feature importance unconventional and difficult. What can be done instead is to perform forward or backwards selection and keep adding/removing features until the maximum accuracy score is obtained for the NN. The main challenge with this method is that the NN takes a long time to train, as well as being computationally expensive.

More advanced methods of tuning hyperparameters for both AdaBoost and NN could have been performed. The Wisconsin Breast Cancer data has been studied a large number of times, and in the literature, some NN architectures is considered favourable for this specific data set. Some may consider cross-validation as a possible method of tuning the number of weak learners for the AdaBoost method, and using random grid search can be used when determining the hyperparameters for the NN.

VI. CONCLUSION

The effect of applying future selection methods along with two different classification models has been studied. With this research, the methods has been applied on the well known Wisconsin Breast cancer data which contains 30 characteristic features coming from cells of benign and malignant tumors.

The filtering method with correlation filters of 0.95, 0.90 and 0.0.85 was applied on the data set before applying the classification methods AdaBoost and an artificial Neural Network. The first filter gave the highest accuracy for the NN of 97% which was an improvement compared to having no feature reduction which resulted in accuracy of 92%.

We continuously used a AdaBoost model with a forest of boosted decision trees of depth 1 called stumps. The first filter of 0.95 gave no increase of accuracy score with AdaBoost in contrast to having no filter which was 98%. The RFE method included training of the model between every time a feature is picked in order to observe how many times the feature appears in the boosted forest. This RFE method with a predetermined choice of 25 features resulted in obtaining a perfect accuracy of 100% with AdaBoosting which was the combined process which gave the overall accuracy.

Overall, the filtering method gave an improved accuracy for the NN with 97%, but no improvement with AdaBoost. Combining RFE of 25 features with AdaBoost gave a perfect accuracy of 100%. AdaBoost gave the highest accuracy both with and without feature selection methods in contrast to using NN for classification. These results indicate that the AdaBoost method is both faster and more accurate than NN for this specific problem.

A. Future work

For future work it will be interesting to study more comprehensive search of hyperparameters for both AdaBoosting and NN in order to observe their predictive performance. It will also be interesting to combine other feature selection methods such as forward or backward selection, but these algorithms would require more computational power.

REFERENCES

[1] Aurelie-Geron (2007). *Hands On Machine Learning with Scikit Learn Keras and Tensorflow*. Oreilly.
[2] Frost, J. (2017). Multicollinearity in regression analysis: Problems, detection, and solutions. `https://statisticsbyjim.com/regression/multicollinearity-in-regression-analysis/`.
[3] Jacobsen, F. (2020a). A numerical study of linear regression and resampling methods. `https://github.com/feliciajacobsen/FYS-STK4155/tree/main/Prosjekt%201`. Accessed: 23-11-2020.
[4] Jacobsen, F. (2020b). A numerical study of regression and classification with neural networks. `https://github.com/feliciajacobsen/FYS-STK4155/blob/main/Prosjekt%202/report.pdf`. Accessed: 23-11-2020.
[5] learn developers, S. (2007 - 2020). Recursive feature elimination by scikit-learn. `https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.RFE.html`.
[6] Roser, M. and Ritchie, H. (2015). Cancer. `https://ourworldindata.org/cancer`.

VII. APPENDIX A: GLOSSARY

A simple glossary list is presented. The intention of table VI is to serve as a encyclopedia for abbreviations for whenever needed.

Abbreviation	Full word
CV	Cross validation
ML	Machine learning
OLS	Ordinary least squares
MSE	Mean squared error
SD	Standard deviation
FFNN	Feed Forward Neural Network
NN	Neural Network
NN	Neural Network
DNN	Deep Neural Network
GD	Gradient Descent
SGD	Stochastic Gradient Descent
VIF	Variance Inflation Factor
RF	Random Forest
RFE	Recursive Feature Elimination

TABLE VI. A glossary list containing often used abbreviations.

VIII. APPENDIX B: ACTIVATION FUNCTIONS

This appendix contains a list of various activation functions and their derivatives applied in the analysis above.

Sigmoid

$$f(z) = \frac{1}{1+e^{-z}}, \tag{15}$$

$$f'((f(z))) = f(z)(1-f(z)). \tag{16}$$

Softmax

$$f(z) = \frac{e^z}{\sum e^z}, \tag{17}$$

$$f'((f(z))) = f(z)(1-f(z)). \tag{18}$$

Relu

$$f(z) = \max(z, 0), \tag{19}$$

$$f'((f(z) > 0)) = 1, \text{else } f'(f(z)) = 0. \tag{20}$$

Leaky Relu

$$f(z) = \max(0.01 \cdot z, z), \tag{21}$$

$$f'((f(z) > 0)) = 1,\ f'(f(z) < 0) = 0.01 \tag{22}$$

Relu6

$$f(z) = \min(\max(z, 0), 6) \tag{23}$$

$$f'(0 < f(z) < 6) = 1, \text{else } f'(f(z)) = 0. \tag{24}$$

Tanh

$$f(z) = \frac{2}{1+e^{2z}} - 1, \tag{25}$$

$$f'((f(z)) = 1 - f(z)^2. \tag{26}$$

Identity

$$f(z) = z, \tag{27}$$
$$f'((f(z))) = 1. \tag{28}$$

IX. APPENDIX C: CROSS-ENTROPY

The cost function for classification given Softmax or Sigmoid as activation function before the last output layer. This cost function is referred to as the Cross-Entropy, and is given mathematically as

$$E = -\frac{1}{n}\sum_{i=1}^{n} y_i \cdot \ln(a_i), \tag{29}$$

the sum runs over the number of classes n, where a_i denotes the predicted output by an activation function, which is the Softmax function from eq. **??** for the multi-class case, and Sigmoid **??** for two-class case. Both y_i and a_i are discrete probability distributions.

Since we're dealing with a SGD method, we need the gradient of the Cross-Entropy given that we have Softmax (Sigmoid multi-class case) as the activation function,

$$\nabla_z E = a - y, \tag{30}$$

this equation holds for both MSE and Softmax (or Sigmoid). Because the gradient of the cross entropy is equivalent in both regression (given MSE as cost function) and classification (given Softmax as activation function) when computing the partial derivative for E with respect to a in equation 29. By setting $a = f(x) = x$ as the identity activation function, and with MSE being $\mathrm{C(y,f(z))} = \frac{1}{2}(y - f(z))$, the derivative of the cost function with respect to f(z) is given as:

$$\nabla_z E = \frac{\partial C(y, f(z))}{z} = \frac{\partial C(y, f(z))}{\partial f(z)} \cdot \frac{\partial f(z)}{\partial z}, \tag{31}$$
$$= \frac{\partial}{\partial f(z)}(\frac{1}{2}(y - f(z))) \cdot \frac{\partial z}{\partial z}, \tag{32}$$
$$= -(y - f(z)) \cdot 1 = f(z) - y, \tag{33}$$
$$= a - y, \tag{34}$$

and will obtain the equivalent expression when combining $\nabla_a C$ with cross-entropy loss with Sigmoid or Softmax activation functions. Note that the derivative of MSE is in reality $2(a - y)$, but it's generally accepted to use eq. 34 since the factor 2 can be baked into the learning rate.