

|  |
| --- |
| 2023/24 |
| Classification Task |
| 5CS037 – Concepts and Technologies of AI |

|  |
| --- |
| Leigh Clarke  2245151 |

Contents

**1.Introduction3**

1.1 Approach3

**2. Dataset4**

2.1 Data preparation4

2.1.1 Head of the Dataset5

2.1.1 Checking for NULL Values5

2.2 Visualisation of Data6

2.2.1 Correlation Heatmap6

2.2.2 Diagnosis Num Data9

2.2.3 Scatterplot Matrix10

**3. Support Vector Machines11**

3.1 Overview11

3.2 Model Predictions12

3.3 Review and Improvements14

3.3.1 Adjusting Kernel Type14

3.3.2 Adjusting Random State15

3.3.3 Adjusting Test Size16

3.3.4 Adding Features17

**4. Random Forest Classifier18**

4.1 Overview18

4.2 Model Predictions19

4.3 Review and Improvements20

4.3.1 Adjusting Random State20

4.3.2 Adjusting Test Size21

4.3.4 Adjusting Max Depth22

**5. Neural Network Classification25**

5.1 Overview25

5.2 Model Predictions25

5.3 Review and Improvements27

5.3.1 Adjusting Layers27

5.3.2 Adjusting Neurons29

5.3.3 Adjusting Optimizer30

**6. References31**

# Introduction

This report aims to predict the diagnosis of patients with breast cancer and whether the diagnosis is malignant or benign from a set of data with 32 features and 570 instances by using two classification models using Machine Learning (ML) and one neural network (NN) model. The primary objective is to analyse the dataset and create a model from a set of variables that have correlation to a specific diagnosis.

Discussing the distinction between benign and malignant cancer is crucial in medical diagnosis. Benign tumours are non-cancerous growths that do not invade nearby tissues or spread to other parts of the body, whereas malignant tumours are cancerous and have the potential to spread to other areas, making them more dangerous. Accurately determining whether a tumour is benign or malignant is vital for treatment planning and prognosis. According to the American Cancer Society (Cancer.org, 2021), early detection of malignant tumours significantly improves the chances of successful treatment and patient survival rates. Therefore, accurately classifying breast cancer diagnoses as benign or malignant is essential for timely intervention and improved patient outcomes.

## Approach

We'll develop three classification models: Support Vector Machines (SVM), Random Forest (RF), and a neural network model. By leveraging SVM and RF, we'll harness the powerful capabilities of robust Python libraries such as: Sklearn, Pandas and TensorFlow, while aiming for high accuracy in predicting whether a breast tumour is benign or malignant.

Simultaneously, with the neural network, we'll delve into deeper layers of data understanding. Our goal is not only to achieve high predictive accuracy but also to unveil underlying variables that correlate with tumour classification in breast cancer cases from the dataset being used.

# Dataset

The dataset being used for the classification models is: “Breast Cancer Wisconsin (Diagnostic) Data Set”.

Source: [https://www.kaggle.com/datasets/uciml/breast-cancer-wisconsin-data/data

The dataset compromises of 33 keys and 570 instances.

#### Keys:

Id, diagnosis, radius\_mean, texture\_mean, perimeter\_mean, area\_mean, smoothness\_mean, compactness\_mean, concavity\_mean, concave points\_mean, symmetry\_mean, fractal\_dimension\_mean, radius\_se, texture\_se, perimeter\_se, area\_se, smoothness\_se, compactness\_se, concavity\_se, concave points\_se, symmetry\_se, fractal\_dimension\_se, radius\_worst, texture\_worst, perimeter\_worst, area\_worst, smoothness\_worst, compactness\_worst, concavity\_worst, concave points\_worst, symmetry\_worst, fractal\_dimension\_worst

## Data Preparation

I've excluded both the ID column and the diagnosis column from the dataset. The ID column merely serves as a unique identifier for the patients, while the diagnosis column indicates 'M' for malignant or 'B' for benign. However, the diagnosis\_Num column already displays these values as 1 or 0, which is ideal for the classification model and eliminates the need for any conversion from string to integer in classification.

Further to the above I will also drop diagnosis\_Num from my X dataset and as this s the target variable (the feature to predict), this will be assigned to the Y dataset.

### Head of the dataset

A screenshot of a graph

Description automatically generated

Figure 1: Head of X dataset.

A screenshot of a graph

Description automatically generated

Figure 2: Head of X dataset after id, diagnosis and diagnosis\_Num removal.

### Checking for NULL values

The dataset that is being used has already been checked for any NULL values, however I did a check using a pandas function.

A screenshot of a computer

Description automatically generated

Figure 3 Check for NULL values.

## Visualisation of Data

### The assigned dataset contains 30 columns. Given the model and task at hand, this amount of columns might hinder the training process. However, this can be revisited during the model improvement phase. To address this initial concern, we employed heatmaps to visualise the correlation between features. This analysis guided the selection of the most relevant features, resulting in a reduced dataset of 15 columns.

### Correlation Heatmap

A blue and white grid with black text

Description automatically generated

Figure 4 Original Correlation heatmap

A graph of a triangle graph

Description automatically generated with medium confidence

Figure 5 Triangular correlation heatmap

A screenshot of a graph

Description automatically generated

Figure 6 Correlation between Diagnosis.

Due to the inherent complexity of visually analysing a correlation heatmap with 30 features, an alternative approach was employed. Here, we utilized a different type of heatmap that facilitated a more efficient sorting process based on numerical order. This streamlined visualization (Figure 7) was derived from the initial data presented in Figure 6.

A graph with different colored bars

Description automatically generated

Figure 7 Top 15 Features Correlating with diagnosis.

A graph of heatmap

Description automatically generated

Figure 8 Updated heatmap with 15 features.

#### Diagnosis Num Data

A graph of a patient with red and blue squares

Description automatically generated

Figure 9 Ration between benign and malignant within dataset.

When training a classification model, it learns from the patterns in the data to make predictions. If one class significantly outweighs the other in terms of the number of records (in this case, if either malignant or benign records dominate the dataset), the model may become biased towards the majority class.

For example, if there are far more benign records than malignant ones, the model might become biased towards predicting benign for most cases, as it's more likely to encounter benign examples during training. This could lead to lower accuracy, especially for predicting the minority class (in this case, malignant), because the model hasn't seen enough examples of it during training.

### Scatterplot Matrix

A grid of blue and orange dots

Description automatically generated

Figure 10 Scatterplot Matrix

Figure 10 depicts a scatterplot matrix, also known as a pairs plot, which aids in visualising relationships between the dataset's multiple features (variables). Analysis reveals a consistent diagonal trend (bottom left to top right) across the features, aligning with our initial hypothesis suggested by the heatmaps (see previous figure).

# Support Vector Machines

### Overview

Support Vector Machines (SVMs) are a powerful tool within machine learning, particularly for binary classification tasks. Originating nearly five decades ago, SVMs underwent a significant breakthrough in the early nineties, revolutionising computer science's approach to classification models. The core concept of SVMs revolves around defining an optimal hyperplane that separates two groups by maximizing the number of points closest to each group. This hyperplane, known as the separating hyperplane, is determined based on support vectors—data points that are crucial for defining the boundary between the groups. The SVM algorithm accomplishes this by nonlinearly mapping input vectors to a high-dimensional feature space, where a decision surface is constructed, aiming to maximize the margin between the observations and the separating hyperplane (Fávero. 2022).

Moreover, SVMs excel in scenarios where the boundary between two groups is nonlinear. While traditional methods like linear or sigmoid classifiers may fall short, SVMs offer a solution by utilizing kernels to expand the vector space of predictor variables, enabling the establishment of a nonlinear boundary between observation groups. This approach allows SVMs to handle complex data distributions effectively, leading to accurate classification outcomes. However, it's worth noting that employing nonlinear specifications, such as radial terms, may increase computational requirements for algorithm convergence. Nevertheless, the ability of SVMs to handle nonlinear boundaries makes them a versatile and valuable tool in various classification tasks, ensuring robust performance even in challenging scenarios (Fávero. 2022).

### Model Predictions

We implemented a support vector machine (SVM) classification model using the scikit-learn library in Python. The train\_test\_split function from scikit-learn was used to divide the data into training and testing sets, with default parameters except for the data (X, y) arguments. Initially, the SVM model utilises a linear kernel. Both the model type (SVM) and the hyperparameter settings (e.g., kernel) can be explored for potential accuracy improvements in future iterations.A screenshot of a computer

Description automatically generated

Figure 11 sklearn function: train\_test\_split and SVM model.

Prediction Accuracy = **94%** (0.94)

A green squares with white text

Description automatically generated

Figure 11 Confusion Matrix- Results.

Our initial SVM model, trained with default parameters, achieved an accuracy of 94%. As shown in the confusion matrix (Figure 11), the model correctly classified 84 benign and 50 malignant cases, with only 6 and 3 misclassifications, respectively.

To improve accuracy further, we will explore various model tuning techniques. Within the SVM model, we can experiment with different kernel types (e.g., linear vs. non-linear) to better capture the underlying data relationships. Additionally, we can adjust the train\_test\_split parameters, such as random\_state and test\_size, to potentially reduce variance and optimize the data split for training and testing. Finally, if model performance remains unsatisfactory after these optimizations, we will revisit the dataset and consider re-introducing previously removed features that might hold valuable information for classification..

### Review and Improvements

#### Adjusting Kernel Type

SVMs leverage kernel functions to handle data in high-dimensional spaces efficiently. The kernel essentially allows working with the original feature space without explicit high-dimensional calculations. In our classification task, the poly and rbf kernel yielded the best results.

One of the other kernels is the precomputed kernel, this requires a square matrix as input (e.g., a 10x10 matrix). This highlights the importance of considering both performance and computational efficiency when selecting a kernel.

From our results, we can see that changing the kernel to a different type didn’t have a significant impact on results so for future model testing we will keep kernel type to linear.

A graph with green line

Description automatically generated

Figure 12 SVM Kernel results

#### Adjusting Random State

As mentioned earlier, the train\_test\_split function from scikit-learn was used with default parameters for test\_size and random\_state in our initial model. We will first explore the impact of modifying random\_state. This parameter controls the randomness used to split the data into training and testing sets. To assess the effect of different random seeds, we will evaluate the model's performance using random\_state values ranging from 0 to 9.

|  |  |
| --- | --- |
| Random State | Accuracy(%): |
| 0 | 94 |
| 1 | 95 |
| 2 | 93 |
| 3 | 94 |
| 4 | 89 |
| 5 | 95 |
| 6 | 95 |
| 7 | 94 |
| 8 | 92 |
| 9 | 96 |

A graph with green line and green line

Description automatically generated

Figure 13 Results of the Random State improvements.

Changing the random\_state parameter to a ‘9’ has resulted in an improvement in accuracy, increasing it from 94% to 96%. This is a good increase, however, our goal is to achieve as close to 100% prediction accuracy as possible. We will look to changing test\_size and review it’s impact.

#### Adjusting Test Size

We will now explore how modifying the test\_size parameter within the train\_test\_split function affects the model's performance. This parameter controls the proportion of the data allocated to the testing set, ranging from 0 (using the entire dataset for training) to 1 (using no data for training). To analyse this effect, we will conduct eight experiments, incrementing the test\_size value by 0.1 in each iteration, starting from 0.1 and concluding at 0.8.

Current test\_size is 0.25 with an accuracy of 96%.

|  |  |
| --- | --- |
| Test Size | Accuracy(%) |
| 0.10 | 93 |
| 0.20 | 95 |
| 0.30 | 94 |
| 0.40 | 94 |
| 0.50 | 95 |
| 0.60 | 95 |
| 0.70 | 94 |
| 0.80 | 92 |

A graph with a green line

Description automatically generated

Figure 14 Results of changing test\_size.

Adjusting the test\_size parameter within train\_test\_split did not yield any improvements in model performance. The initial setting of 0.25 resulted in an accuracy of 96%, and the maximum accuracy achieved during the variations was 95%. These results suggest that the model's performance might not be highly sensitive to the specific test size within this range.

Given the limited impact of test\_size adjustments, we will now explore the effect of adding features, Previously, we identified features with high correlation to the target variable ("diagnosis\_num"). We will incorporate the features we removed back into the dataset and evaluate the model's accuracy with this new expansion (excluding id, diagnosis and the target variable diagnosis\_Num).

#### Adding Features

Adding in the features we previously dropped due to bad correlation into the dataset to see if this will impact the accuracy of our SVM model. Below are the results and displayed in Figure 16.

Dropped features: id, diagnosis, diagnosis\_Num(target variable).

Prediction Accuracy = **96%**

A green squares with white text

Description automatically generated

Figure 15 Results after adding features.

Reintroducing the previously removed features did not significantly alter the model's prediction accuracy, which remained at 96%. This suggests that the initial feature selection process effectively identified the most relevant features for classification.

In conclusion, the optimal model configuration for this task achieved a 96% prediction accuracy and employed the following parameters:

**train\_test\_split:** test\_size = 0.25, random\_state = 9 (for consistent random split)

**Kernel:** Linear

**Features:** Top 15 features based on correlation with the target variable

# Random Forest Classifier

### Overview

A random forest classifier model combines multiple decision trees, each trained on a random subset of data and features, to improve accuracy and reduce overfitting. It uses voting or averaging to make final predictions for new data points. Both Scikit-Learn and PySpark offer tools for building and using random forests Furthermore, the DTR model operates by analysing the dataset information and will create appropriate splits within the data itself. Meaning we can actually create the model with a predefined number of inquiries, such as stipulating a minimum of 3 or 5 questions.

Each decision tree is trained on a different subset of the data, created through a process called bootstrap aggregating or bagging. During training, each decision tree is exposed to a random subset of the features and a random subset of the training samples (Testas, 2023).

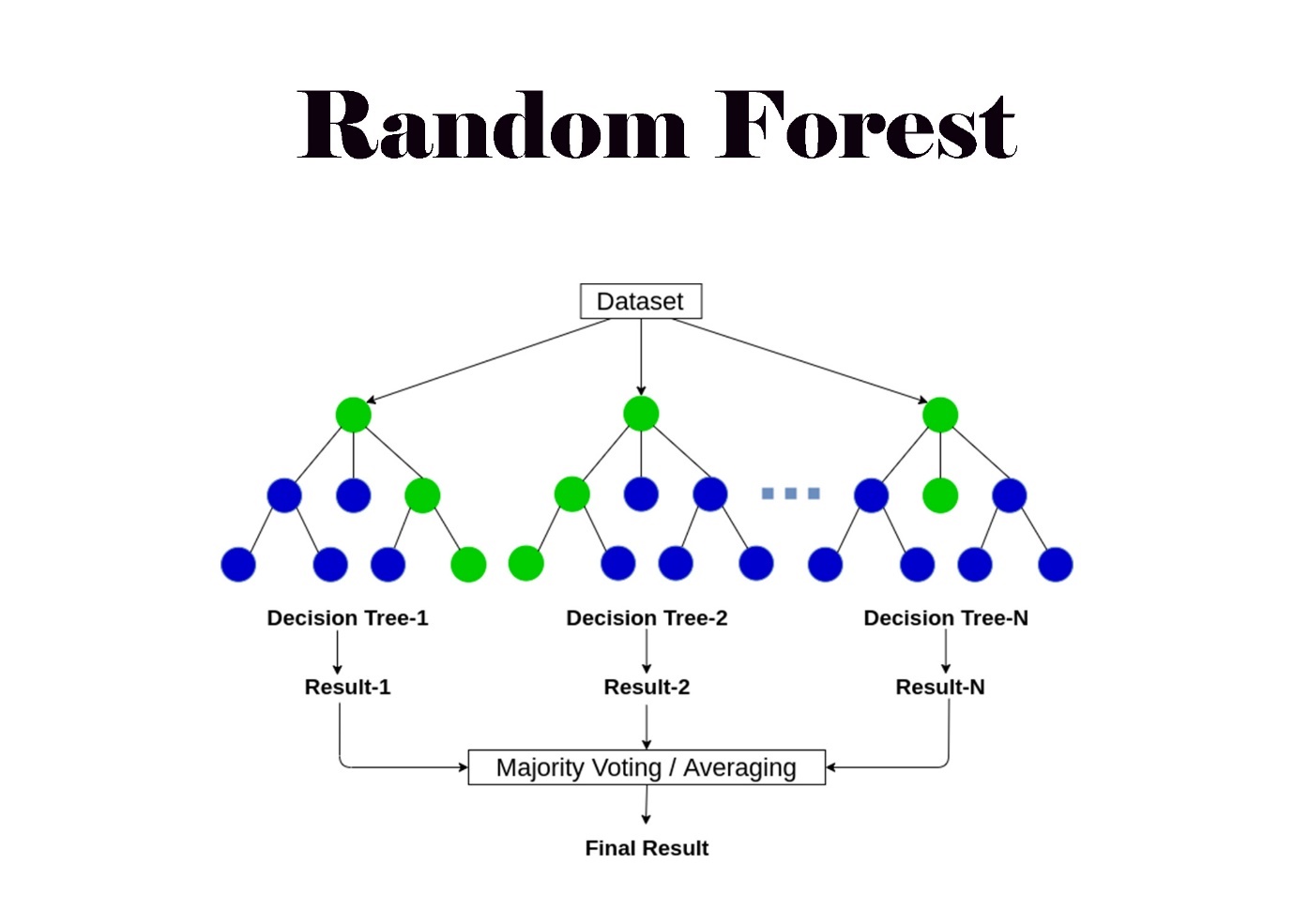


Figure 16 Random Forest (RF).

We will be using the same dataset that we previously used in the SVM model. With the top 15 features for correlation being used.

### Model Predictions

This time, we'll create a Random Forest Classifier (RFC) using scikit-learn, similar to the SVM model. We'll start by using default parameters and analyse the performance. Then, we'll focus on tuning parameters to improve accuracy.

A screenshot of a computer

Description automatically generated

Figure 17 train\_test\_split and RFC model parameters.

Prediction Accuracy = **94%** (0.94)

A green squares with white text

Description automatically generated

Figure 18 RFC results represented in confusion matrix.

### Our initial Random Forest Classifier (RFC) model with default settings achieved a 94% prediction accuracy, matching the SVM model's initial results. Examining the confusion matrix (Figure 18), we see the model correctly classified 84 samples with a value of 0 and 50 with a value of 1, with only 4 and 5 misclassifications respectively. To push beyond this initial success, we'll delve into parameter tuning techniques to enhance the model's performance.

### Review and Improvements

#### Adjusting Random State

In this section, we'll investigate the effect of the random\_state parameter in the train\_test\_split function from scikit-learn. We'll run multiple tests with random\_state values ranging from 0 to 9 (inclusive) to analyse how it influences the model's performance. This approach mirrors the one used for the SVM model previously.

Below are the results from this test:

|  |  |
| --- | --- |
| Random State | Accuracy(%): |
| 0 | 96 |
| 1 | 94 |
| 2 | 94 |
| 3 | 93 |
| 4 | 90 |
| 5 | 97 |
| 6 | 96 |
| 7 | 95 |
| 8 | 94 |
| 9 | 97 |

A graph with green line and numbers

Description automatically generated

Figure 19 Results of random\_state test.

Using a random\_state value of 5 or 9 has yielded the best prediction accuracy result with **97%** . An increase of 3% from the original prediction accuracy.

**Previous prediction accuracy result:** 94%.

**Result after random\_state changes:** 97%.

#### Adjusting Test Size

We will now look into testing the effectiveness of changing the value of ‘test\_size’. We will conduct 8 tests going up in increments of 0.10 starting with 0.10 and concluding at 0.80.

|  |  |
| --- | --- |
| Test Size | Accuracy(%) |
| 0.10 | 89 |
| 0.20 | 95 |
| 0.30 | 95 |
| 0.40 | 94 |
| 0.50 | 95 |
| 0.60 | 95 |
| 0.70 | 95 |
| 0.80 | 95 |

A graph with a green line

Description automatically generated

Figure 20 Results of test\_size test.

While we experimented with different test\_size values, none surpassed the accuracy achieved with the default value of 0.25. Meaning we are using the most optimal value or very close to.

In the next phase, we will delve into the analysis of the RandomForestClassifier() function within our model to identify potential modifications.

**Previous prediction accuracy result:** 97%.

**Result after test\_size changes:** 95%.

#### Adjusting Max Depth

This section explores how tuning the max\_depth parameters within RandomForestClassifier() can increase the RFC model's accuracy.

Max Depth: This parameter defines the maximum number of splits allowed in each decision tree. Deeper trees (higher max\_depth) can potentially fit the training data better, but also risk overfitting. We'll analyse how varying max\_depth affects accuracy.

The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min\_samples\_split samples (scikit-learn, 2024).

By carefully adjusting this parameters, we aim to achieve a better balance between model complexity leading to improved prediction accuracy. We will conduct 10 tests, incrementing the 'max\_depth' value by 1 starting from 0 and continuing until we reach 10.

|  |  |
| --- | --- |
| Max Depth | Accuracy(%) |
| 1 | 92 |
| 2 | 96 |
| 3 | 97 |
| 4 | 97 |
| 5 | 96 |
| 6 | 97 |
| 7 | 96 |
| 8 | 96 |
| 9 | 97 |
| 10 | 97 |
| 11 | 97 |
| 12 | 95 |
| 13 | 95 |
| 14 | 96 |
| 15 | 97 |

A graph with green line and numbers

Description automatically generated

Figure 21 Result of Max Depth test.

Using a 'max\_depth' value of 3, 4, 6, 9, 10, 11 and 15 yielded a prediction accuracy of **97%**, adjusting this parameter had no significant impact on our models prediction accuracy.

**Previous prediction accuracy result:** 97%.

**Result after test\_size changes:** 97%.

With an impressive 97% prediction accuracy, our Random Forest model has achieved a very good result, approaching the limitations inherent to this model type. While further hyperparameter tuning might yield minor improvements, the most significant gains in accuracy are likely to come from exploring a richer or more informative dataset.

A green squares with white text

Description automatically generated

Figure 22 Final confusion matrix of RFC model.

# Neural Network Classification

### Overview

This section focuses on employing a Neural Network Classification (NNC) model to predict breast cancer diagnoses. We'll assume you're familiar with the general structure and workings of neural networks.

Unlike regression neural networks (NNRs) used for continuous values, NNCs excel at predicting categorical outcomes like disease presence/absence. This choice aligns perfectly with our goal of classifying breast cancer diagnoses.

Four key attributes significantly influence the NNC's performance:

* Hidden Layers: The number of hidden layers determines the model's complexity and ability to learn intricate patterns.
* Neurons per Layer: This value defines the computational power within each layer.
* Activation Function: This function introduces non-linearity, crucial for complex relationships between features and outputs.
* Optimizer: This algorithm guides the model's learning process, minimizing errors and improving accuracy.

By carefully tuning these parameters, we can attempt to optimise the NNC for accurate breast cancer diagnosis prediction.

A screenshot of a computer program

Description automatically generated

Figure 26 Excerpt of NNC model code.

### Model Predictions

Initially our NNC model is with 4 layers (including input and output), with input and hidden layers each having 30 neurons and “relu” activation type. The output layer is set to 2 neurons due to there being 2 hidden layers and using an activation type of “tf.nn.softmax”.

Using the Adam optimizer and a loss value of “sparse\_categorical\_crossentropy” we were able to achieve the following results:

Prediction accuracy of: **98%**

A green squares with white text

Description automatically generated

Figure 23 Confusion Matrix displaying results of NNC model.

The above confusion matrix shows that our NNC model was able to predict 35 correct and 0 incorrect for the ) value and 21 correct and 1 incorrect for the 1 value.

Although, 98% is a very good result and far better than the initial SVM(94%) and RFC (94%) model results, we will look into whether the prediction accuracy of the NNC model can be increased with some improvements.

### Review and Improvements

#### Adjusting Layers

This section explores how the number of layers in a neural network (NNR) affects its performance and prediction accuracy. We'll investigate the impact of increasing layers from the current configuration of four (one input, two hidden, and one output) to two alternative options:

**Test 1:** Five total layers with five hidden layers.

**Test 2:** Six total layers with seven hidden layers.

**Constraints:** Due to limitations in computational power and time, we can only evaluate these two variations.

**Test 1 Results:**

A green squares with white text

Description automatically generated

Figure 24 NNC Five layer test 1 Confusion Matrix.

Prediction Accuracy: 100%

0 output prediction: 35/35

1 output prediction: 22/22

**Test 2 Results:**

A green squares with white text

Description automatically generated

Figure 25 NNR layer test 2.

Prediction Accuracy: 98%

0 output prediction: 35/35

1 output prediction: 21/22

Our test provided slightly better results, with Test 1 achieving an accuracy of 100%. This represents an improvement of 2% compared to both the baseline model and Test 2. Notably, Test 2 exhibited no change in accuracy compared to the baseline.

However, it is crucial to acknowledge the potential impact of dataset size on these results. While the observed accuracy improvement is encouraging, smaller datasets can be susceptible to overfitting. In overfitting, the model excels at predicting the training data but may not generalise well to unseen data.

To strengthen the validity of these findings and mitigate the potential for overfitting, it is recommended to evaluate the model's performance using a larger dataset. This will provide a more robust assessment of the model's generalizability and real-world effectiveness.

#### Adjusting Neurons

The number of neurons within a Dense layer directly influences the model's capacity to learn intricate relationships within the data. While a greater number of neurons allows the model to potentially capture more complex patterns, it also introduces an elevated risk of overfitting. Overfitting occurs when the model memorizes noise or irrelevant details within the training data, hindering its ability to generalize effectively to unseen data.

I will again create two tests, each one with less neurons than the other.

**Test 1 Results:**

20 neurons per layer

Prediction Accuracy: **98%**

**Test 2 Results:**

15 neurons per layer

Prediction Accuracy: **99%**

Initial observations suggest a minimal influence from variations in hidden layer neuron count. Reducing the number of neurons from 30 to 20 resulted in a decrease in prediction accuracy, while a further reduction to 15 neurons yielded a slight increase. These relatively minor changes in accuracy could be attributed to the size of the dataset.

With a limited dataset, the model might not have the capacity to fully utilise the increased complexity introduced by a larger number of neurons. Conversely, the initial model with 30 neurons might have been susceptible to overfitting the training data.

Over-fitted models tend to memorize all the data, including unavoidable noise on the training set, instead of learning the discipline hidden behind the data (Ying, 2019). The observed decrease in accuracy when reducing neuron count could be indicative of overfitting in the original model. This suggests that reducing model complexity through decreased neuron count potentially mitigated the overfitting issue.

Adjusting Optimizer  
We have seen by using the ‘Adam’ optimizer within our model we were able to achieve a prediction accuracy of 100%.



According to Kingma et al., 2014, the method is "computationally efficient, has little memory requirement, invariant to diagonal rescaling of gradients, and is well suited for problems that are large in terms of data/parameters".

However, we will look at some other optimizers that we could potentially use and see how they affect the models behaviour.

We will look at the following optimizer’s and analyse their results: SGD, RMSProp, AdaGrad

SGD: 100%

RMSProp: 96%

AdaGrad: 95%

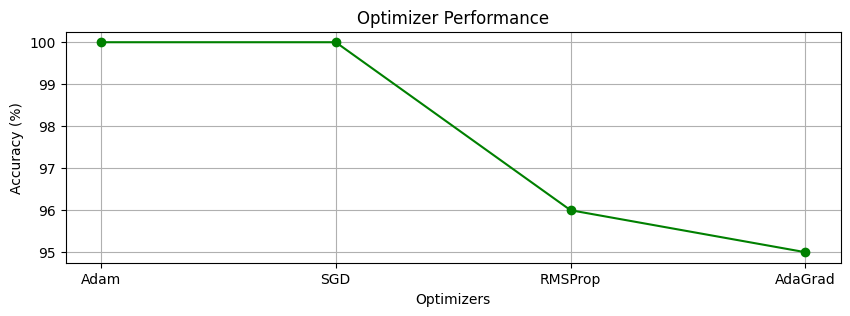


Figure 27 NNC Optimizer Test Results

The evaluation of various optimizers revealed significant differences in their impact on the neural network's performance. Notably, both Adam and SGD achieved a 100% accuracy, suggesting their effectiveness in minimizing the loss function and converging to an optimal solution for this particular task.

RMSProp exhibited a slightly lower accuracy of 96%, indicating a potentially greater difficulty in locating the optimal solution compared to Adam and SGD. AdaGrad displayed the lowest accuracy of 95%, suggesting it might not have been as efficient in optimizing the model parameters for this specific task.

In conclusion, the selection of an optimizer significantly influences the neural network's performance. This investigation demonstrates better performance using Adam and SGD optimizers in achieving the highest accuracy. While RMSProp showed a moderate decrease in performance, AdaGrad resulted in the lowest accuracy among the evaluated optimizers.

# References

Non-cancerous Breast Conditions. Available at: <https://www.cancer.org/cancer/types/breast-cancer/non-cancerous-breast-conditions.html> (Accessed: 04 January 2024).

Fávero, Luiz Paulo et al. “Classification Performance Evaluation from Multilevel Logistic and Support Vector Machine Algorithms through Simulated Data in Python.” *Procedia computer science* 214 (2022): 511–519. Web.

Testas, A. (2023). Distributed Machine Learning with PySpark Migrating Effortlessly from Pandas and Scikit-Learn. 1st ed. Berkeley, CA: Apress. Available at: [https://learning.oreilly.com/library/view/distributed-machine-learning/9781484297513/html/604220\_1\_En\_9\_Chapter.xhtml] (Accessed: 05 January 2024).

Scikit-Learn. (2024). Random Forest Classifier. Available at: <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html> (Accessed: 06 January 2024).

Ying, X. (2019) "An Overview of Overfitting and its Solutions." Journal of Physics: Conference Series, 1168(2), p. 022022. DOI: 10.1088/1742-6596/1168/2/022022