LSE ST 449 Final Project
Candidate Numbers: 26089 and 37674

# Comparing different predictive models applied to a heart disease dataset

## 1. Introduction

This report is written to document the results of a final project in LSE's course ST449: Artificial Intelligence. In total, we compare six binary classifiers and 6 ANN models regarding their accuracy, precision and recall when predicting whether a person has a heart disease or not.

### 1.1. Our objectives

The objective of this report is to compare the accuracy of different predictive models on a heart disease dataset to predict the likelihood of having a heart disease. We will be training different binary classifiers with sklearn and a neural network with Tensorflow and compare their results. To improve the different models' performances we shall be experimenting with the (hyper)parameters, reporting on the effect of changing the default parameters on the accuracy of the models and finally concluding which model performed best overall and is the most suitable.

### 1.2. Our dataset

We wanted to make a health-related application and ultimately chose a dataset regarding heart disease. We chose this dataset because cardiovascular diseases (CVDs) are the number 1 cause of death globally. People with cardiovascular disease or who are at high cardiovascular risk (due to the presence of one or more risk factors) need early detection and management and so a machine learning model can be of great help. Heart failure is a common event caused by CVDs and this dataset contains 11 features that can be used to predict a possible heart disease.

The 11 different features are (fedesoriano, 2021):

1. Age: age of the patient [years]
2. Sex: sex of the patient [M: Male, F: Female]
3. ChestPainType: chest pain type [TA: Typical Angina, ATA: Atypical Angina, NAP: Non-Anginal Pain, ASY: Asymptomatic]
4. RestingBP: resting blood pressure [mm Hg]
5. Cholesterol: serum cholesterol [mm/dl]
6. FastingBS: fasting blood sugar [1: if FastingBS > 120 mg/dl, 0: otherwise]
7. RestingECG: resting electrocardiogram results [Normal: Normal, ST: having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of > 0.05 mV), LVH: showing probable or definite left ventricular hypertrophy by Estes' criteria]

8. MaxHR: maximum heart rate achieved [Numeric value between 60 and 202]
9. ExerciseAngina: exercise-induced angina [Y: Yes, N: No]
10. Oldpeak: oldpeak = ST [Numeric value measured in depression]
11. ST_Slope: the slope of the peak exercise ST segment [Up: upsloping, Flat: flat, Down: downsloping]

### 1.3. Preparing the dataset

The dataset itself is nearly complete, only in the column "Cholesterol" there are a number of "0" values, which don't make sense for a living person. As these were only of a limited quantity, we proceeded with all 918 rows of the dataset. To enable our models, all columns have to contain numerical data. This made it necessary to convert a few columns: Sex, ChestPainType, RestingECG, ExerciseAngina and ST_Slope contained categorical data. For Sex and ExerciseAngina the conversion was simple as these were binary values. We converted the Sex column to a column "is_male" with value 1 for males and 0 for females and converted the values of the ExerciseAngina column in place, 1 for Y (Yes) and 0 for N (No).
For the other columns, there were more than two different values, so we used one-hot-encoding for them. This resulted in creating an additional 7 columns for a total of 19 columns. The code for preparing the dataset is in the file dataset_preparation.py. After preparation, the dataset looks like this:

| Age | RestingBP | Cholesterol | FastingBS | MaxHR | ExerciseAngina | Oldpeak | HeartDisease | is_male | CP_TA | CP_ATA | CP_ASY | CP_NAP | ECG_normal | ECG_ST | ECG_LVH | ST_Up | ST_Down | ST_Flat |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 40 | 140 | 289 | 0 | 172 | 0 | 0.0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| 49 | 160 | 180 | 0 | 156 | 0 | 1.0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 |
| 37 | 130 | 283 | 0 | 98 | 0 | 0.0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 |
| 48 | 138 | 214 | 0 | 108 | 1 | 1.5 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| 54 | 150 | 195 | 0 | 122 | 0 | 0.0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 |
| 39 | 120 | 339 | 0 | 170 | 0 | 0.0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 |
| 45 | 130 | 237 | 0 | 170 | 0 | 0.0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| 54 | 110 | 208 | 0 | 142 | 0 | 0.0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| 37 | 140 | 207 | 0 | 130 | 1 | 1.5 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| 48 | 120 | 284 | 0 | 120 | 0 | 0.0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| 37 | 130 | 211 | 0 | 142 | 0 | 0.0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 |
| 58 | 136 | 164 | 0 | 99 | 1 | 2.0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| 39 | 120 | 204 | 0 | 145 | 0 | 0.0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| 49 | 140 | 234 | 0 | 140 | 1 | 1.0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| 42 | 115 | 211 | 0 | 137 | 0 | 0.0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| 54 | 120 | 273 | 0 | 150 | 0 | 1.5 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| 38 | 110 | 196 | 0 | 166 | 0 | 0.0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| 43 | 120 | 201 | 0 | 165 | 0 | 0.0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |

Before we could run our models, we also needed to standard-scale the data and create a training and a test set of data. We did this using the functions StandardScaler from the sklearn.preprocessor module and train_test_split from sklearn.model_selection.

## 2. Binary classifiers with sklearn

The sklearn-module makes a range of binary classifiers available for supervised learning. We were interested to see which of these is the most successful in classifying our dataset and thus chose a broad range of classifiers: Support Vector Machines (sklearn.svm.SVC), Logistic Regression (sklearn.linear_model.LogisticRegression), Decision Trees (sklearn.tree.DecisionTreeClassifier), Random Forest (sklearn.ensemble.RandomForestClassifier), (Gaussian) Naive Bayes (sklearn.naive_bayes.GaussianNB) and K-Nearest Neighbors (sklearn.neighbors.KNeighborsClassifier). We first ran them with their default settings and then experimented with hyperparameter-tuning using GridSearchCV.

### 2.1. Initial results

There are no "official" published scores for our dataset, so it is not entirely clear what a good score is. However, one Kaggle-User reports accuracy-scores between 82% and 84% in his experiments.[1] Our initial scores were as follows:

| | Accuracy | Precision | Recall | Confusion Matrix |
|---|---|---|---|---|
| Logistic Regression | 0.853 | 0.841 | 0.9 | 67 17<br>10 90 |
| Support Vector Machines | 0.880 | 0.879 | 0.913 | 68 13<br>9 94 |
| Decision Trees | 0.788 | 0.748 | 0.87 | 65 27<br>12 80 |
| Random Forest | 0.87 | 0.879 | 0.895 | 66 13<br>11 94 |
| Naive Bayes | 0.859 | 0.841 | 0.909 | 68 17<br>9 90 |
| K-Nearest Neighbor | 0.853 | 0.832 | 0.908 | 68 18<br>9 89 |

For clarification: Accuracy measures the overall rate of correct predictions, while precision measures the ratio of true positives (people with heart disease) to total predicted positives (some of which are false positives) and recall the ratio of true predicted positives to total actual positives (some of which are predicted to be healthy) (Géron 2019).
With the exception of Decision Trees (accuracy 78,8%), the accuracy varies slightly between 85,3% (Logistic Regression and KNN) and 88% (SVC). Our results are hence somewhat better than those reported on Kaggle. SVC also shows the highest precision (87,8%) and recall (91,3%) while Decision Trees score worst on these metrics, too. At first sight, therefore, SVC seems to be the best binary classifier for our dataset and Decision Trees the worst.

### 2.2. Experimenting with hyperparameter-tuning

Except for the Gaussian Naive Bayes classifier, each classifier supports choosing a variety of hyper-parameters, for example the maximum depth for Decision Trees or the kernel for SVC. For a full list of possible parameters, see the official documentation or see the attached Python file (sklearn_binaryclassifiers.py) for those used in our search.
As our dataset is relatively small, we used GridSearchCV to find the best available hyper-parameters in our search space rather than RandomSearchCV which would only try out a fraction of the possible combinations of hyper-parameters. We also used k-fold cross-validation to better evaluate our models.

[1] https://www.kaggle.com/fedesoriano/heart-failure-prediction/discussion/279486

### 2.3. Results

The first thing to note about our results is that the scores varied slightly between different runs of the script. However, with the exception of Random Forest and Decision Trees, this did not have an impact on the reported best set of hyper-parameters. This variation can probably be explained by factors determined by randomness in the models - especially in Random Forest - and the relatively small size of our dataset where these factors can make a visible difference. Our optimized results were as follows (improvements over the unoptimized models in green, deteriorations in red):

| | Accuracy | Precision | Recall | Confusion Matrix | Reported Best Parameters |
|---|---|---|---|---|---|
| Optimized Logistic Regression | 0.842<br>-0.011 | 0.813<br>-0.028 | 0.906<br>0.006 | 68 9<br>20 87 | C: 0.1,<br>Penalty: l1,<br>Solver: liblinear |
| Optimized Support Vector Machines | 0.859<br>-0.022 | 0.850<br>-0.028 | 0.901<br>-0.012 | 67 10<br>16 91 | C: 0.001,<br>Degree: 1,<br>Gamma: 100,<br>Kernel: poly |
| Optimized Decision Trees | 0.859<br>+0.082 | 0.841<br>+0.093 | 0.909<br>+0.058 | 68 9<br>17 90 | Criterion: entropy,<br>Max_depth: 6,<br>Splitter: random |
| Optimized Random Forest | 0.891<br>+0.005 | 0.897<br>+0.019 | 0.914<br>-0.007 | 68 9<br>11 96 | Criterion: gini,<br>Max_depth: 15,<br>Max_features: 3,<br>Min_samples_split: 5,<br>N_estimators: 300 |
| Naive Bayes (for comparison) | 0.859 | 0.841 | 0.909 | 68 17<br>9 90 | No hyperparameters to tune |
| Optimized K-Nearest Neighbors | 0.880<br>+0.027 | 0.879<br>+0.047 | 0.913<br>+0.004 | 68 9<br>13 94 | Leaf_size: 10,<br>Metric: minkowski,<br>N_neighbors: 20,<br>P: 1,<br>Weights: distance |

What is surprising about these results is that SVM and Logistic Regression performed worse with the supposedly optimized parameters. This effect persisted over multiple runs of the script. It is unclear to us how this might be explained, given that the default parameters were also part of the search space and GridSearchCV should have explored those as well (had we used RandomSearchCV instead, this deviation could have been explained by randomness of the search). Other than that, we see a significant improvement for the Decision Trees, which is no longer the worst model, and marginal improvements for the other models.

The optimized Random Forest beats all other models (including the default SVM) in terms of accuracy, precision and recall despite a slight deterioration in terms of recall compared to the default settings.

## 3. ANNs and Sequential Modelling with Tensorflow

Artificial Neural Networks (Multi-Layer Perceptron) are used to solve Sequence Modelling problems. Sequence Modelling is the ability of a computer program to model, interpret, and make predictions about any type of sequential data (Paparaju, 2018). The Sequential model API is a way of creating deep learning models where an instance of the sequential class is created, and model layers are created and added to it. The Sequential model is a linear stack of layers.

### 3.1. Initial Model

For the initial Sequential model, we started off with only one layer and compiled the loss function, the optimizer, and the metrics with the model (Radečić, 2021). The model had a single layer with 128 dense neurons, the binary crossentropy loss function, an SGD optimizer (Stochastic Gradient Descent) and 3 different metrics: Accuracy, Precision (measure of people that we correctly identify having a heart disease out of all the people) and Recall (measure of our model correctly identifying people as having a heart disease out of all the people who have a heart disease) (Géron, 2019; Huilgol, 2020).

The results for our initial model were:

| Initial model | Accuracy | Precision | Recall | Confusion Matrix |
|---|---|---|---|---|
| Fitted against training data | 0.914 | 0.91 | 0.94 | - |
| Evaluated against testing data | 0.84 | 0.96 | 0.75 | 74 3<br>27 80 |

From our initial model's results, the model looks to have a high precision meaning that this model is good at correctly identifying people who have a heart disease. The model does not have a particularly high recall compared to the precision meaning it's not as accurate at identifying people who have a heart disease out of all the people who have a heart disease. Our initial model's accuracy is relatively high, so it is good at correctly predicting people who have a heart disease and people who do not.

### 3.2. Experimenting with hyperparameter-tuning

To improve our model we decided to experiment by changing the hyperparameters of our initial model and subsequent models, each time testing the model against the training and testing data.

There are many ways to improve a model by changing the hyperparameters, for example adding additional layers to the model, increasing the number of neurons in each layer, choosing different activation functions for different layers, changing the optimizer, and adding dropout layers. In our subsequent models we will be experimenting with a few of these ways.

For our **second model**, the only change we made was that we added another layer of 128 dense neurons to the model to see if it would improve the initial model.

The results for our second model were:

| Second model | Accuracy | Precision | Recall | Confusion Matrix |
|---|---|---|---|---|
| Fitted against training data | 0.958 | 0.953 | 0.97 | - |
| Evaluated against testing data | 0.83 | 0.96 | 0.73 | 74 3<br>29 78 |

The second model's results show a slightly lower accuracy compared to the first model but the same precision rate meaning that this model is not as good at identifying people having a heart disease. This model also has a lower recall rate meaning it's less accurate at identifying people who have a heart disease.

In the **third model**, we added a third layer of 256 dense neurons to the second model.

The results for our third model were:

| Third model | Accuracy | Precision | Recall | Confusion Matrix |
|---|---|---|---|---|
| Fitted against training data | 0.992 | 0.99 | 0.995 | - |
| Evaluated against testing data | 0.83 | 0.94 | 0.76 | 72 5<br>26 81 |

The third model's results show that it has the same accuracy as the second model as well as a higher recall rate which tells us that this model is more accurate at identifying people who have a heart disease out of all the people who have heart disease compared to the second model. However this model has a lower precision rate than the previous two models, so it's worse at correctly identifying people who have heart disease.

For the **fourth model**, we changed the optimizer from SGD to Adam from our third model. Adam optimization is a stochastic gradient descent method that is based on adaptive estimation of first-order and second-order moments.

The results for our fourth model were:

| Fourth model | Accuracy | Precision | Recall | Confusion Matrix |
|---|---|---|---|---|
| Fitted against training data | 0.9605 | 0.9973 | 0.9302 | - |

| | | | | |
|---|---|---|---|---|
| Evaluated against testing data | 0.84 | 0.91 | 0.81 | 68 9<br>20 87 |

The fourth model's results indicate that this model has the same accuracy as the initial model but has lower precision meaning that this model is not as good at correctly identifying people who have a heart disease. This model does have a better recall rate than the initial model meaning it's better at accurately identifying people who have a heart disease out of all the people who have heart disease.

In the **fifth model**, we added a dropout layer and changed the optimizer back to SGD from the fourth model. The dropout hyperparameter is the probability of training a given node in a layer, where 1.0 means no dropout, and 0.0 means no outputs from the layer.

The results for our fifth model were:

| Fifth model | Accuracy | Precision | Recall | Confusion Matrix |
|---|---|---|---|---|
| Fitted against training data | 0.9755 | 0.9752 | 0.98 | - |
| Evaluated against testing data | 0.84 | 0.95 | 0.76 | 73 4<br>26 81 |

The fifth model's results tell us that this model has the same accuracy, slightly lower precision rate and slightly higher recall rate than the initial model meaning that this model performs very similarly to the initial model.

For our **sixth and final model**, we increased the momentum of the SGD from the fifth model. By increasing the momentum of gradient descent it causes gradient vectors to converge faster.

The results for our sixth model were:

| Sixth model | Accuracy | Precision | Recall | Confusion Matrix |
|---|---|---|---|---|
| Fitted against training data | 0.997 | 0.998 | 0.998 | - |
| Evaluated against testing data | 0.85 | 0.94 | 0.79 | 72 5<br>22 85 |

The sixth model's results show that it has a higher accuracy and recall rate which tells us that this model is more accurate at identifying people who have a heart disease out of all the people who have heart disease compared to the initial model. However this model has a slightly lower precision rate than the initial model, so it's not as good at correctly identifying people who have heart disease.

### 3.3. Conclusion on hyperparameter-tuning on Sequential Modelling

From the models created by experimenting with the hyperparameters, we can come to some conclusions about how changing certain hyperparameters can affect the quality of a sequential model.

In the second and third models, we added more layers and neurons to the original model, which led to those models having less accuracy and precision than the first model as the model was potentially overfitting the data. By adding more neurons we added more non-linearity to our model which can cause it to overfit the data (Bhandari, 2020). By adding more layers it makes the model more flexible and stronger, which could be why the recall value was higher in both those models (Padaki, 2020).

In the fourth model, we changed the optimizer of the third model, which improved the accuracy and precision values and had a similar recall value. This could be because the Adam optimizer converges faster than the SGD optimizer thus resulting in an improved model (Park, 2021).

In the fifth model, we reverted the optimizer back to SGD and added a dropout layer to the fourth model, leading to the model being very similar in performance to our initial model. The dropout layer is usually added to help prevent overfitting of the data and this can be seen in our results since adding a dropout layer to the third model improved the quality of the model (cf. Tensorflow documentation).

In the sixth model, we increased the momentum of the SGD optimizer of the fifth model, which as result did improve the initial model as it had better accuracy and recall rate. The momentum optimizer is meant to increase the speed at which the model is optimised and overall improve the model which can be seen by the results of the model fitted against the training data (Wang, 2019).

Looking at all the models that we created using Sequential Modelling, we would say that the **sixth model** is the most suitable for predicting heart disease as it had higher accuracy and recall rate than the first model and a slightly higher precision rate.

## 4. Comparison/Discussion of results

When we directly compare the best binary classifier from the sklearn model and our most successful ANN model, we see the following values:

| Model | Accuracy | Precision | Recall |
|---|---|---|---|
| Optimized Random Forest | 0.891 | 0.897 | 0.914 |
| 6th ANN Model | 0.85 | 0.94 | 0.79 |

The Optimized Random Forest Model scores a little higher on accuracy and a lot higher on recall than the sixth ANN model, but it has somewhat lower precision. With regards to our dataset, a false positive is better than a false negative: It is better to wrongly classify someone as having a heart disease (and therefore maybe have a doctor conduct additional examinations) than classifying someone who actually

has a heart disease as healthy - such a person might miss out on treatment and therefore die. Hence, high recall is more important than high precision. This being said, the Optimized Random Forest model is our most successful model overall.

## 5. Conclusion

We have compared and optimized six binary classifiers from the scikit-learn module and compared the most successful model with our most successful ANN model. Because the Optimized Random Forest Model scored much higher on recall, which is more important than precision for our dataset, than the best-performing ANN model, we can come to the conclusion that it is the best model of those surveyed. With a recall value of 91.4 %, this model detects just more than 9 out of 10 patients with heart disease. As heart diseases are among the most important causes of death, this prediction rate can possibly save many lives when used in an early warning system.

## References


Bhandari, Aniruddha. Everything you Should Know about Confusion Matrix for Machine Learning. 2020. Retrieved 21.01.2022 from https://www.analyticsvidhya.com/blog/2020/04/confusion-matrix-machine-learning/.

Brownlee, Jason. Scikit-Optimize for Hyperparameter Tuning in Machine Learning. 2020. Retrieved 21.01.2022 from https://machinelearningmastery.com/scikit-optimize-for-hyperparameter-tuning-in-machine-learning/.

Fedesoriano. Heart Failure Prediction Dataset. 2021. Retrieved 21.01.2022 from https://www.kaggle.com/fedesoriano/heart-failure-prediction.

Géron, Aurélien. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow. 2019.

Huilgol, Purva. Precision vs. Recall – An Intuitive Guide for Every Machine Learning Person. 2020. Retrieved 21.01.2022 from https://www.analyticsvidhya.com/blog/2020/09/precision-recall-machine-learning/.

Karabiber, Faith. Binary Classification. N.A. Retrieved 21.01.2022 from https://www.learndatasci.com/glossary/binary-classification/.

N. N. tf.keras.layers.Dropout in TensorFlow Core v2.7.0 Information. N. A. Retrieved 21.01.2022 from https://www.tensorflow.org/api_docs/python/tf/keras/layers/Dropout#:~:text=The%20Dropout%20layer%20randomly%20sets,time%2C%20which%20helps%20prevent%20overfitting.

Padaki, Anup. What is a difference between adding one more layer and increasing neurons in one layer? 2020. Retrieved 22.01.2022 from https://www.quora.com/What-is-a-difference-between-adding-one-more-layer-and-increasing-neurons-in-one-layer.

Paparaju, Tarun. Sequence Modelling. 2018. Retrieved 21.01.2022 from https://medium.com/machine-learning-basics/sequence-modelling-b2cdf244c233.

Park, Sieun. A 2021 Guide to improving CNNs-Optimizers: Adam vs SGD. 2021. Retrieved 21.01.2022 from https://medium.com/geekculture/a-2021-guide-to-improving-cnns-optimizers-adam-vs-sgd-495848ac6008 .

Radečić, Dario. How to Train a Classification Model with TensorFlow in 10 Minutes. 2021. Retrieved 21.01.2022 from https://towardsdatascience.com/how-to-train-a-classification-model-with-tensorflow-in-10-minutes-fd2b7cfba86.

Scikit-learn developers. “Supervised learning” in scikit-learn 1.02 Official Documentation. N. A. Retrieved 21.01.2022 from https://scikit-learn.org/stable/supervised_learning.html#supervised-learning.

Wang, Mike. Why to Optimize with Momentum. 2019. Retrieved 21.01.2022 from https://medium.com/analytics-vidhya/why-use-the-momentum-optimizer-with-minimal-code-example-8f5d93c33a53