THE MACHINE LEARNING PROJECT: APPLYING MACHINE LEARNING TECHNIQUES FOR STROKE PREDICTION

University of Amsterdam
Minor in Artificial Intelligence

25 / 06 / 2021

Authors:
Jana Bersee
Koen Ceton
Jeroen Dijkmans
Dominique Weltevreden

Introduction

It is very useful to be able to predict if a person is at risk of having a stroke. Over the years much healthcare data has been collected, including if people have had a stroke. Now this data can be used to tell which people are likely to suffer a stroke in the future. We will be working with a dataset from Palacios (2021). This is a tabular dataset with eleven clinical features. Considering this is a tabular dataset, there are multiple algorithms that can be used on this data. Therefore, we will be creating the following classifiers: k-Nearest Neighbours, Decision Tree and a Neural Network. With every chapter of this report the classification models improve and their input data, data pipeline, training and evaluation will be discussed. Eventually the models will be coupled, to get a final, more accurate prediction.

Contents

Chapter 1. Baseline model

Section 1: Data Analysis

Data reading and pre-processing

First of all, we imported the data from kaggle.com as a Pandas dataframe. When we checked what the data looked like, using info(), we saw that there were some categorical and some numerical values. The outcome variable 'stroke' was already one-hot-encoded. We removed the value 'ID', because this does not influence the outcome variable and should thus not be trained on. For the 'gender' feature the category 'other' was removed since this category contained only one sample.
Subsequently, we replaced the variable columns that contained categorical values with one-hot-encoded columns. Not only multi-class features, but also binary features were one-hot encoded to ensure independent weighing of both gender types in the deep neural network.
There were a few missing values in the 'BMI' column. We chose to remove the cases where BMI was missing as this does not constitute a significant data loss. The total number of removed samples was 202, which was 3.95% of the total dataset.

Splitting

Finally, we split the data into features and labels and into training, testing and validation data; we used a 60% - 20% - 20% split. The validation data will be used for parameter optimization, while the testing data will be used for final model evaluation.

Data imbalance

One important thing we noted was that the stroke and non-stroke groups were highly unbalanced; there were much more subjects who did not have a stroke compared to the subjects who had a stroke. For our baseline models we have not yet implemented any methods to counteract this imbalance. However, we noticed that we should emphasize this imbalance in our model evaluation (see model evaluation).

Data visualization

In order to have a better grasp of our quantitative data, we visualized the age, average glucose level and BMI with respect to the chances of a stroke occurring (Figure 1). In total, we made six plots. Three histograms with age, average glucose level and BMI on the x-axis, and the count of strokes and non-strokes on the y-axis. These plots clarified the kinds of distributions. The other three plots are cumulative histograms, plotted against density. These plots better display the distribution of the volume, since the heavy imbalance in strokes and non-strokes makes it difficult to see the distribution of the strokes.
From the age plots, we can conclude that the higher the age, the bigger the chances are of having a stroke. The stroke distribution strongly rises around the age of 70, whilst being almost non-existent in around the 10 to 20 age range.
The average glucose level plots tell us that most strokes occur at an average glucose level of 70 mg/DL. This is also the most occurring level for people where a stroke did not occur. At 200 mg/DL, there is also a significant change of having a stroke. The amount of people having a level of 200 mg/DL is significantly lower than that of 70 mg/DL.
The plots on BMI tell us that the BMI of a person has very little to do with the probability of having a stroke. The density distribution of stroke vs non-stroke is more or less the same.

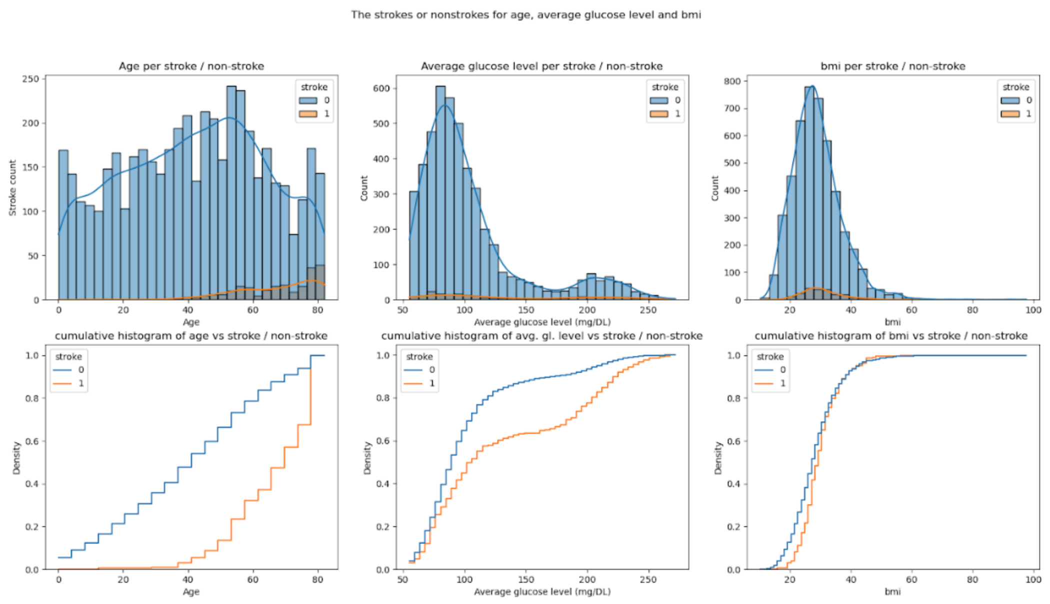


Figure 1: *Normal and cumulative histograms, displaying the age, average glucose level and bmi against the stroke occurrence.*

Figure 2 shows the categorial features of the dataset, for stroke and non-stroke. It contains mostly information about which features are more present in a category. It is difficult to tell if there is a feature that causes a higher chance for having a stroke. Therefore, we also calculated the ratio of people having a stroke in the sample with the total sample for all the categories by dividing the people with a stroke with a certain feature by the total number of samples with this feature. For the gender category and the residence category this led to very similar ratios for the features (Appendix, Table 1). A category where there were noteworthy differences is hypertension (Appendix, Table 1).

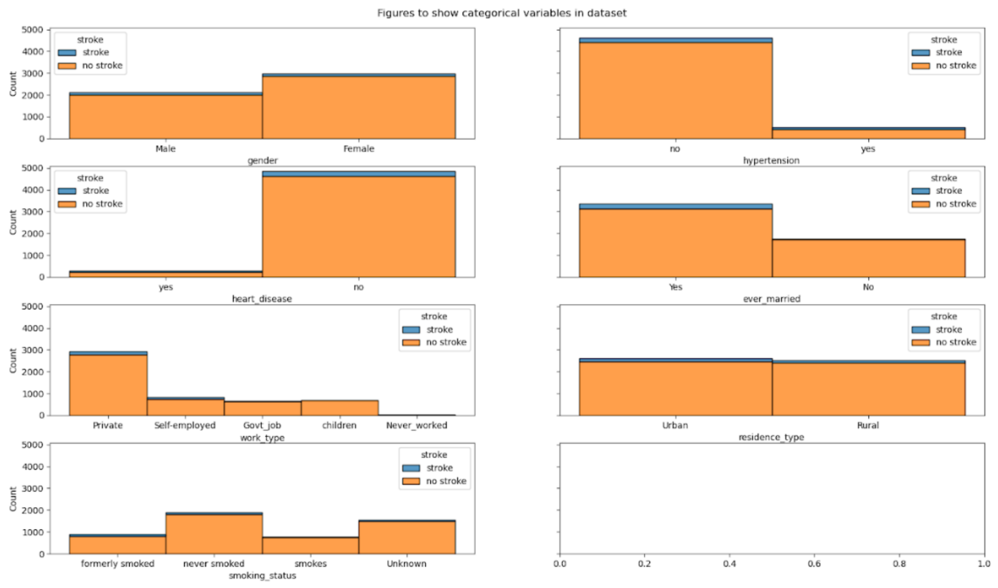


Figure 2: *Stroke and non-stroke samples per categorical data feature*

Section 2: Data Pipeline

First of all, we made 3 separate base models to see how well they would be able to classify the patients that were at risk of getting a stroke. We created a k-Nearest Neighbours classification model, a decision tree and a deep neural network. Our idea is that these models can first be optimized separately. If several models are predicting strokes significantly above chance level on validation data, these models can potentially be stacked to further boost the predictions.
The k-Nearest Neighbours algorithm classifies unknown data points based on similarity to other, known data points. We do not necessarily expect this model to perform very well, but it is a very simple model which might do a decent job at separating the stroke patients from the healthy patients.
The decision tree is highly interpretable and is well fit to deal with both categorical and numerical data. Besides, its implementation is relatively trivial, which makes it a good choice as a classification method. The model can be further extended to a random forest in later models and be tweaked to deal with overfitting which easily occurs in decision trees.
Finally, a deep neural network is used for classification which can learn more complex non-linearity from the input features. The number of parameters to add is virtually limitless, but especially with unbalanced data the challenge will be to overcome overfitting of the model to the training data.

k-Nearest Neighbours

For k-Nearest Neighbours (k-NN), we used the sklearn KNeighborsClassifier as our base model.
The main steps for this model are choosing the right hyperparameters, fitting the classifier to the training data and letting the model predict the classes of the testing data. As we have learned, k-NN classifiers typically use numerical data, for which a distance metric can be calculated such as the 'Euclidean distance' or 'Manhattan distance'. However, there are also several metrics to use categorical data in a kNN classifier, for example the Jaccard distance. For this initial exploration of k-NN, we chose to split our dataset in distinct dataframes, based on the type of feature (numerical data and categorical data). Two models are used separately with a different distance metric. Finally, one more model was created in which all features were used and categorical data were treated as if they were numerical.

Decision Tree

A basic decision model is built using the sklearn toolkit and fit using the training data. The decision tree from sklearn is an optimized version of the CART algorithm for the decision tree. This algorithm technically does not support categorical variables, but as we saw in an earlier module, the classifier does work decently for categorical data. For this basic model we used the defaults of sklearn's DecisionTreeClassifier. This includes the Gini impurity measure. This is a measure of the likelihood of an incorrect classification for a new instance of a random variable, if that new instance were randomly classified according to the distribution of the class labels from the dataset (Ambielli, 2017).

Neural Network

For our basic neural network, we used TensorFlow to build a sequential model. The model has an input layer with one node for each input feature in the model. The input layer is connected to a hidden layer with 25 nodes (slightly more than the number of features in the input node) and a ReLU activation function. The hidden layer is connected to a sigmoid output layer with 1 node. The output of the final sigmoid function is a float between 0 and 1 which can be set to a threshold at 0.5 for a prediction of 0 for no stroke and 1 for stroke.

Section 3: Model Training

K-Nearest Neighbours

The k-NN model is relatively simple; the only parameters that can be tweaked are the amount of neighbours that should be considered for classification, the metric used for distance and the weight function to determine how a point in a ‘neighbourhood’ should be classified. For the starting models, we used a k of 5, which was randomly chosen. The weight function was set to “distance”, which meant that a point was classified as one class or another by taking the distance to all k neighbours into account (instead of just picking the majority).

Decision Tree

No distinction was made between the categorical and continuous data, so the categorical data was actually treated as continuous in this basic model. This is something that might need to be improved during model optimization. For our first training of the model, no limitations in the depth of the tree or other optimization methods were added so far.

Neural Network

The loss function in this model is defined by the binary cross-entropy. We tried to train the model using 30 epochs, since more epochs did not change the loss and accuracy much more. There were no additional parameters set for this basic model.

Section 4: Model Evaluation

Our data labels were highly unbalanced. There are 4908 people in total: 209 people with a stroke and 4699 people without a stroke. This can lead to very high accuracies when the label with the highest prevalence is always predicted, namely an accuracy of 96%. In such a case there are no false positives, but many false negatives. For all three models we have calculated the confusion matrix to inspect this, and found that indeed after training each of these models almost no validation samples were predicted positive for stroke. Since accuracy is not a good metric for unbalanced data, we also added the sensitivity, specificity and balanced accuracy scores as a metric to evaluate the model.

Balanced accuracy

The balanced accuracy can be used for data with a binary class. It is based on specificity and sensitivity and uses the following formula: (specificity + sensitivity) / 2. Specificity is the true negative rate, calculated as the ratio of the true negatives over the actual negative cases (so true negatives + false positives). Sensitivity is the true positive rate, calculated as the ratio of true positives out of all actual positive cases (Tay, 2020).

K-Nearest Neighbours

The accuracies were measured for the testing data for all three models using the three different feature sets: categorical, numerical and total dataset. ‘k’ was kept constant at 5.

Categorical data accuracy: 95.21%
Numerical data accuracy: 94.91%
All data accuracy: 94.23%

From these data, it seems that using only the categorical data provides the best results for the k-NN model. However, as stated in the paragraph above, the model hardly exceeds chance level. Therefore, we also calculated the balanced accuracy:

Categorical data
balanced accuracy: 52.90%
sensitivity: 0.0600
specificity: 0.9979

TN: 970	FP: 2
FN: 47	TP: 3

Numerical data
balanced accuracy: 51.79%
sensitivity: 0.0400
specificity: 0.9959

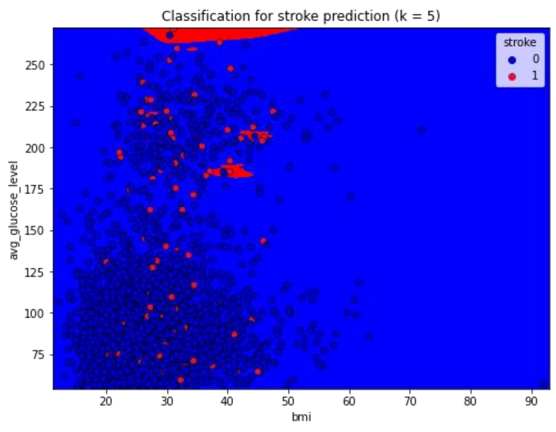
TN: 968	FP: 4
FN: 48	TP: 2

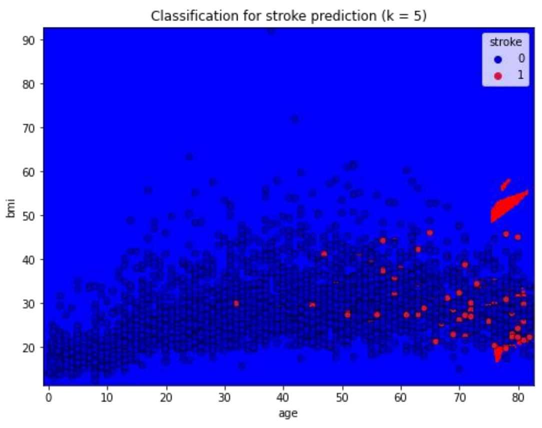
All data
balanced accuracy: 49.54%
sensitivity: 0.0000
specificity: 0.9907

TN: 963	FP: 9
FN: 50	TP: 0

This shows that the categorical data do a somewhat better job than the numerical data, but all predictions are around chance level (as suggested by the 50.00% for balanced accuracy) and sensitivity is extremely low for all models. Note that the accuracies are even lower than if the model would predict everything as non-stroke (around 96%). The model makes some false positive mistakes as well.

Decision Boundaries for numerical data at k=5.





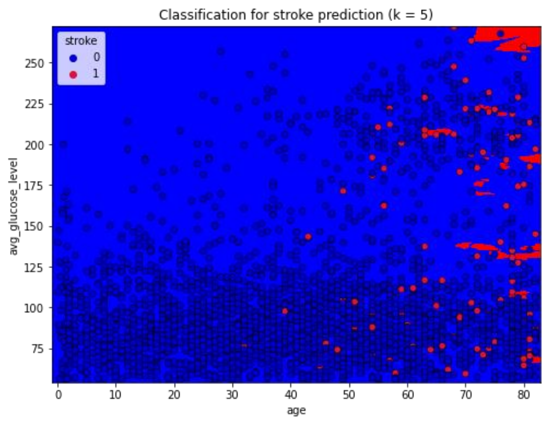


Figure 3: *The figures above show the decision boundaries for the different numerical data measures. The scatteredness of the stroke prediction areas seem to confirm the 'randomness' suggested by the low accuracies of the test predictions. Besides, the distributions of stroke and non-stroke samples largely overlap, showing the difficulty of separating these data using neighbour information, at least with limited amounts of features.*

Decision Tree

For the Decision Tree we implemented a tree with the default minimal split depth of 2. The tree uses both categorical and numerical data, but removing the categorical data does not influence the accuracy. This can be caused by the use of not normalized data, leading to a large difference in the range between the categorical and numerical data.
Looking at the figure 4, the visualized to depth 2 decision tree, we see that age is the first, and thus most important split, as it apparently has the largest gain of all features. The numbers seem a little off, but this is because we used normalized data.

Train metrics:
Accuracy: 100 %
Balanced accuracy: 100 %
Sensitivity: 1.00
Specificity: 1.00
Confusion matrix:

TN: 2822	FP: 0
FN: 0	TP: 122

Test metrics:
Accuracy: 91.2 %
Balanced accuracy: 57.3 %
Sensitivity: 0.20
Specificity: 0.95
Confusion matrix:

TN: 886	FP: 45
FN: 41	TP: 10

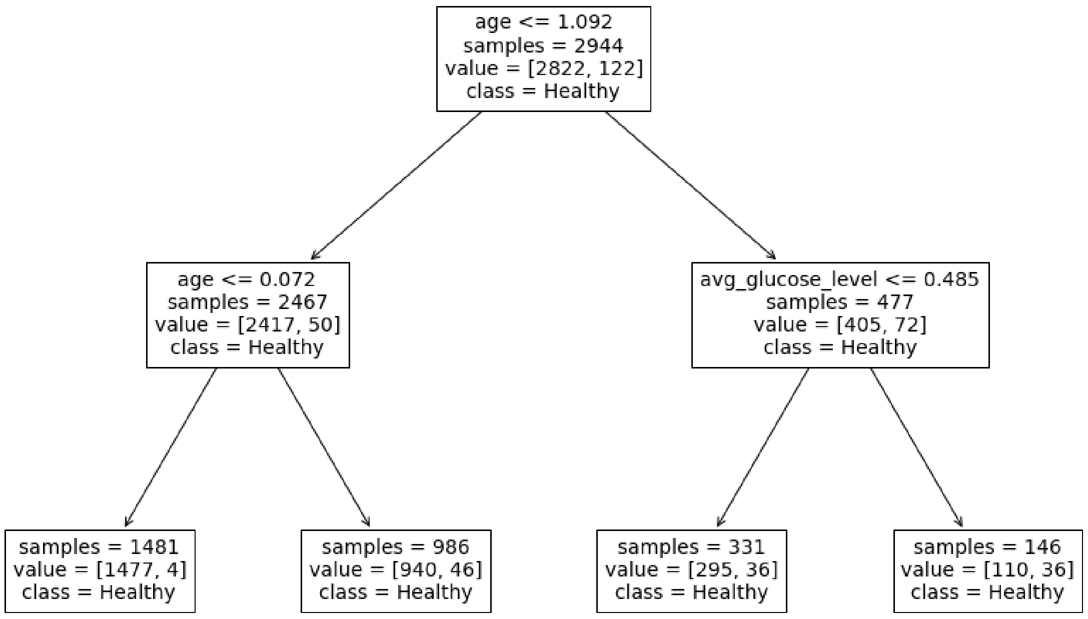


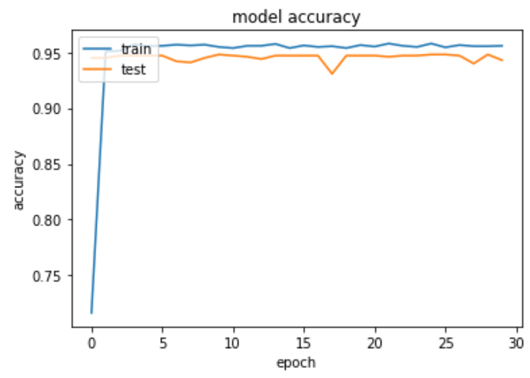
Figure 4: *Decision tree for a maximum depth of 2 and without minimum split restrictions*

Neural Network

The balanced accuracy of our basic neural network is about 50%, which is exactly at chance level. However, there are many parameters to add such as the addition of more layers to the model and hyperparameters to change such as the cost and activation functions. Furthermore, the accuracy on the testing data does not get much higher than 0.95, which is as it were a local minimum; the model just classifies every datapoint as not having a stroke. So, there is much room for improvement here.

The performance of the baseline model of the neural network can be seen in Figure 3. It achieved a training accuracy of 95.5%. This percentage gives the impression of an already properly trained neural network. However, when looking at the confusion matrices of the training data, there were only 5 'True positives'. From all 5000 samples, only 5 were correctly predicted to have a stroke. The loss in the final epoch was 0.1551.

When running the testing data, three samples correctly predicted a stroke. The accuracy was 1% lower than the accuracy of the training data. For further improvement of our baseline NN, it is a viable option to add weights to the losses. This may increase the sensitivity. Also, we could oversample the stroke data by applying data augmentations for better training of our model.



```
train metrics:

accuracy: 95.5163 %

balanced accuracy: 51.7834 %

confusion matrix:
[[2807   15]
 [ 117    5]]

[["True Negative", "False Positive"]
 ["False Negative", "True Positive"]]

test metrics:

accuracy: 94.2974 %

balanced accuracy: 52.5115 %

confusion matrix:
[[923   8]
 [ 48   3]]

[["True Negative", "False Positive"]
 ["False Negative", "True Positive"]]
```

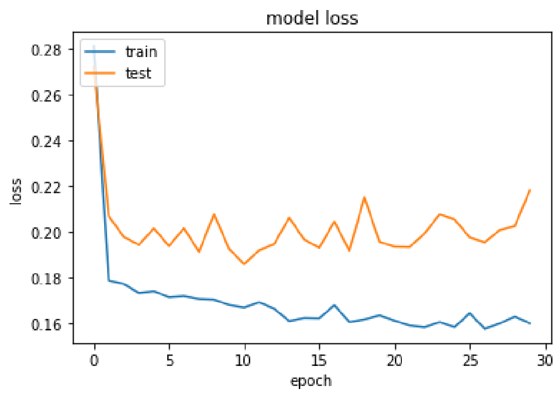


Figure 5: *Training and testing accuracy and loss of the baseline neural network*

Train metrics:
accuracy: 95.17 %
balanced accuracy: 50.33 %
sensitivity: 0.0067
specificity: 1.0000

confusion matrix:

TN: 2917	FP: 0

FN: 148	TP: 1

Test metrics:
accuracy: 95.1%
balanced accuracy: 50 %
sensitivity: 0
specificity: 1

confusion matrix

TN: 972	FP: 0
FN: 50	TP: 0

Chapter 2. Actual model

This model's pipeline contains all the options for different (hyper)parameter settings such that our final model can be optimally tweaked. Some preliminary investigations of the different hyperparameters are described here, to show what kind of effect such parameter alterations have.

Section 1: Data Analysis

For the second model, we redid some of our data processing.
For the first collection of models, we dropped the N/A values in the dataset, as this was not a very significant amount of the dataset; only 3.95%. However, on closer inspection, we found out that by dropping these N/A's, we deleted 16% of the data of stroke patients; we went from 249 to 209 samples. Considering we already have little data from people with strokes, it is better to replace the missing values with a mean value than to delete these samples. Therefore, we replaced the missing BMI values from people with strokes with the mean BMI of this group and did the same for the group without strokes. This left us with 5109 samples in total.
Furthermore, we normalized the input data for the numerical features by calculating the z-score over the values using scipy.stat z-score; this makes it so that the mean for the features is zero, and the standard deviation has a mean of 1. By doing this, the data points are roughly in the same league, which is better for the neural network and for the distance calculation for k-NN.
In addition, we edited the split from the entire dataset into training, testing and validation data; instead of just splitting the data randomly, we enabled stratification for the labels, which means that the original ratio of stroke to non-stroke patients is maintained. This ensured that there were stroke patients in the testing and validation data; this is better for calculating (balanced) accuracy for the testing data, as we want a model that does (correctly) predict strokes; if there are no strokes in the testing data, the model would be 'punished' for predicting samples as stroke, which is the opposite of what we want.

Data oversampling

Another method implemented to deal with our imbalanced dataset is oversampling of the minority stroke class (Yu & Radewagen, 2017). For the implementation of oversampling on our training dataset, the SMOTENC function from the imbalanced-learn library was used. Unlike SMOTE, SMOTE-NC can be used on a dataset containing numerical and categorical features.

For the generation of new samples, the k-nearest neighbours algorithm is used. Resampling the numerical features is pretty straight forward, the numerical value is the distance between the data samples. When creating new numerical features of samples with a stroke, values around numerical features where a stroke occurred are used. The k value is set to 5, so within areas of a stroke, new numerical values are created. As for categorical features, one cannot assign a distance metric. Hence, the distribution of the classes in the feature is used. Let us say we are looking at the smoking status feature. For instance, the percentage distribution of the smoking classes of people having a stroke are: non-smoking = 8 %. some smoking = 22 % and smoking = 70 %. Considering this data, the smoking people are more likely to have a stroke, therefore those samples should be closer together for the k-NN algorithm. So, in SMOTENC, the distance between the categorial features is measured by the probability distribution of the minority target label.

Section 2: Data Pipeline

K-Nearest Neighbours

Input data was normalized for this version of k-NN. There were no further alterations to model settings and input data for k-NN in this edition of the model.

Decision Tree

For the basic decision tree we used the same algorithm as before with the same input (so both categorical and numerical); however, we did not normalize the input data for this model, as this is not needed for a decision tree, and is even detrimental to the model, as the results are no longer interpretable.
Furthermore, we also implemented a random forest using RandomForestClassifier from sklearn. A random forest trains a number of decision trees, with each tree being trained on a subsection of the data. How large that subsection of data should be, we have defined in section 3. We trained every tree in the forest on the square root number of the amount of input features (which is the default setting), and used the default of 100 trees in the forest.
Third, we implemented a forest that is an ensemble of different resampled datasets; we will be referring to it as a resampled forest from this point onwards. Every tree in this forest is trained on a subset of the total data of patients without strokes, combined with **all** the data of the patients with strokes. This is a spin on undersampling, where parts of the data from the non-rare class are removed from the dataset, but in this case the forest has still seen the total training dataset, as every tree sees a different, small subset of the non-stroke dataset. This approach was based on the article by Yu & Radewagen (2017). This hopefully counteracts the imbalanced dataset, as the ratio of stroke to non-stroke data is now larger in the batches of training data that each tree in the forest is trained on. The actual prediction is made by taking the majority vote from these trees.

Neural Network

Architecture

The architecture of the model has not yet been adapted in this version of the model. So, as for the base model, the input layer has the same number as the input features, which is 22 now since all binary features are one-hot encoded as well. Then one hidden layer is added with 25 nodes and a ReLU activation function. These nodes are all connected to one sigmoid activated output node.

Weight initialization

In earlier versions of our deep neural network model, we found out that the random initialization of our weight parameters in the model caused much fluctuation in the resulting predictions. In order to better evaluate the model, we implemented an Initializer object (from Keras) in our model, which allows for specification of the distribution of weights and using a fixed seed for pseudo-randomization. These fixed seeds are only used when directly comparing the effect of different values of a certain hyperparameter such as class weights or oversampling rates, using these fixed initial weights can help choosing the ideal parameter for our model during model optimization. However, fixed seeds will not be used for the evaluation of different neural network architectures with predetermined hyperparameter settings while testing the model.

Class weight changes for loss function

For our actual model, the first alteration we did was to change the loss function in a way that false negatives are weighted heavier than false positives (Yu & Radewagen, 2017; Wolfram, 2021). In our code, this was done by defining the 'class_weight' parameter of the model.fit() function. This functionally changes the binary cross-entropy cost function slightly, from

$$-(y_{true} * log(p) + (1 - y_{true}) * log(1 - p))$$

to

$$-(y_{true} * log(p) * weight_{stroke-class} + (1 - y_{true}) * log(1 - p) * weight_{non-stroke-class})$$

Now, if a prediction is a false negative, this will increase the cost more than if the prediction is a false positive.

5-fold cross-validation

For the neural network, we have implemented a 5-fold cross-validation pipeline for better validation of our model outcomes. Instead of using one split of the training and validation data, this pipeline iteratively splits 80% of the data into 5 (stratified) folds, each time using 1 fold for validation and 4 folds for training. The average evaluation metrics (both final metrics and accuracies and cost over epochs) were then used for model evaluation.

Section 3: Model Training

K-Nearest Neighbours

For the model training step for K-Nearest Neighbours we tried to plot the accuracies and balanced accuracies for the various options for k. We tried every value for k between 1 and 33, incremented in steps of two so that the number of neighbours would remain uneven.

All data

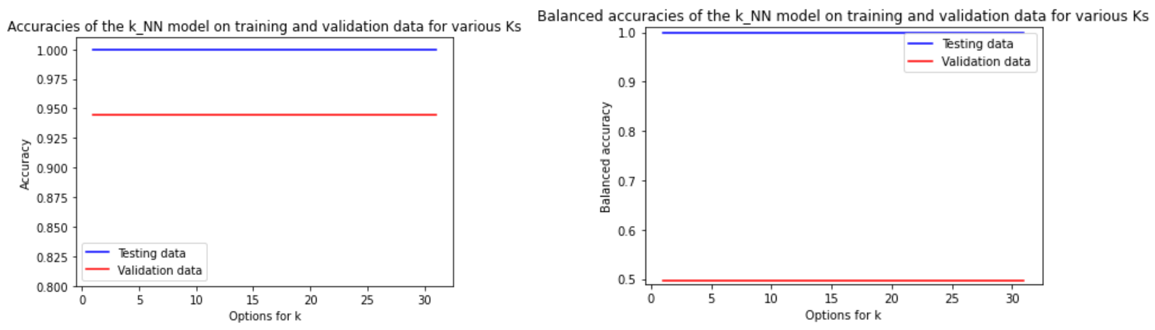


Figure 6.1, 6.2: *K-NN performance against different k values for all data*

Only the numeric features

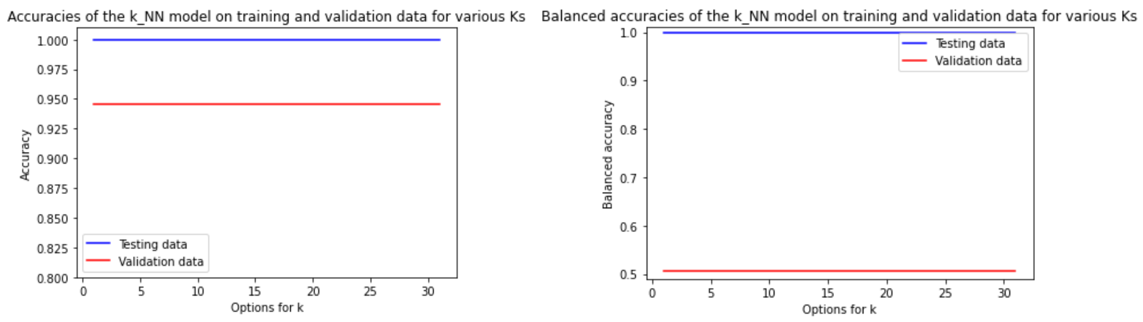


Figure 6.3, 6.4: *k-NN performance for different k values for the numerical data features*

Only the categorical features

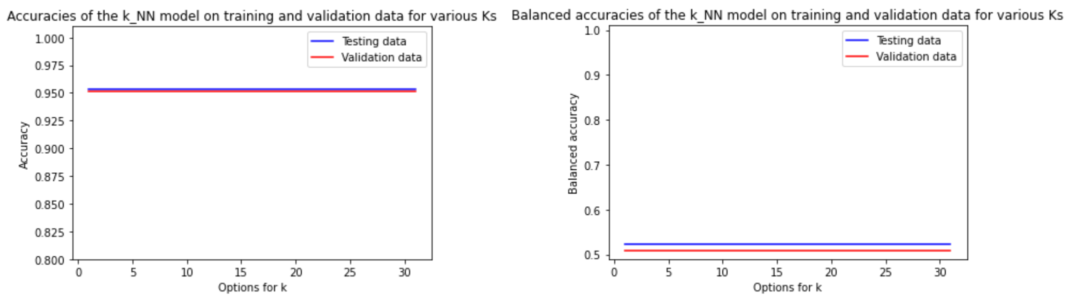


Figure 6.5, 6.6: *k-NN performance for different k values for the categorical data features.*

As visible in the figures above, the k did not influence the accuracies for any of the feature combinations. This is probably the result of having a lot more data points from people without stroke than with stroke as well as the distribution of stroke data largely overlapping with non-stroke data (see milestone 1). Increasing the number of points considered for the predicted label, does not influence the prediction since the majority vote will always be in favour of non-stroke.

Decision tree

Optimization of the decision tree

We tried several methods to optimize the decision tree by reducing the complexity of the tree. This can be done by reducing the depth of the tree, by either setting a maximum depth or altering the number of samples in a node before it is considered a leaf node, or by pruning the model after the model is completed.

Minimum samples in a node

First, we tried to alter the amount of samples that should be left over in a batch before it could be called a leaf node. We tried samples left between 0 and 800, as we have quite a large dataset. We plotted the (balanced) accuracy for the training and validation data for these various minimum sample sizes.

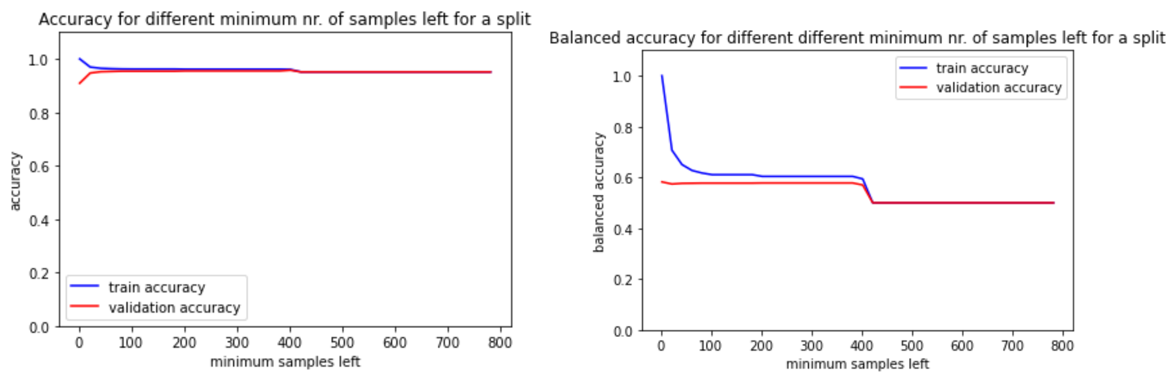


Figure 7.1, 7.2: *Decision tree performance with differing sample minimums*

The figure above shows that 2 is the best split size for the balanced accuracy (= 0.58) for the validation data, which is a bit odd as this is basically where the model is still overfitting quite a lot, when looking at the accuracy for the training data. The balanced accuracy is also still not particularly high; it just seems to decrease for a bigger minimum amount of samples left. Notably, there's also a dip in accuracy around 400 samples per split: after this point, the tree depth is probably too limited.

Tree depth

Second, we tried to limit the tree depth for the model, trying tree depths between 1 and 50.

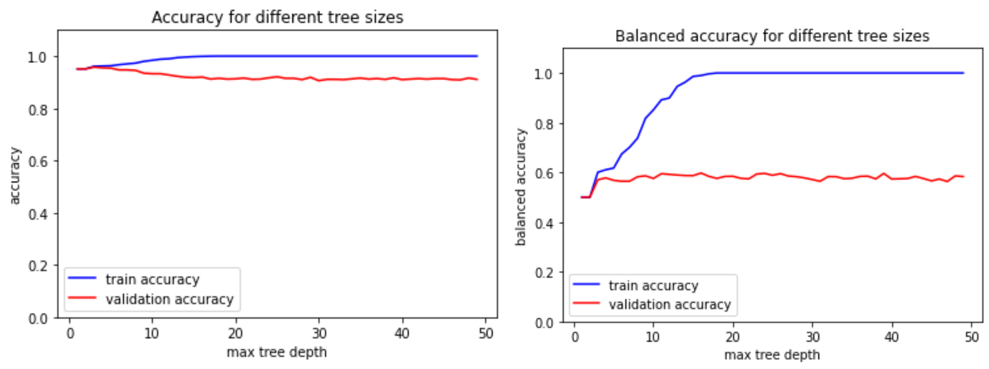
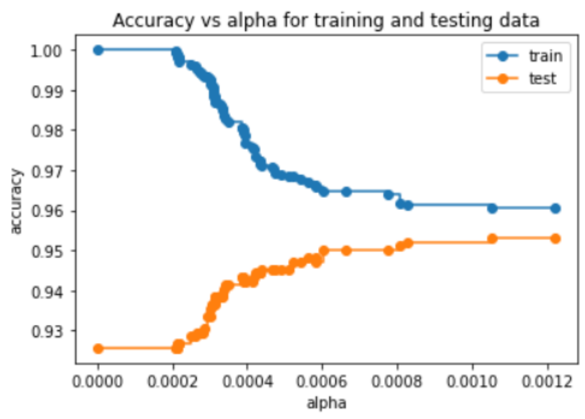


Figure 8.1, 8.2: *Decision tree performance for different depths*

The plot and calculations showed that 16 was the optimal tree depth for a maximum balanced validation accuracy (0.597); there is however not a clear peak, as the accuracy seems to oscillate somewhat.

Pruning

Furthermore, we tried to prune the tree after it was completed with different parameters. We used the cost complexity parameter (alpha) to measure the pruning. The higher alpha, the more leaves are pruned. An alpha of 0.007 left our tree with just one node. We plotted the accuracy and balanced accuracy (see below) and found the alpha with the highest balanced accuracy: 0.00034, with a balanced accuracy of 0.590 . This alpha results in a tree with 165 nodes.



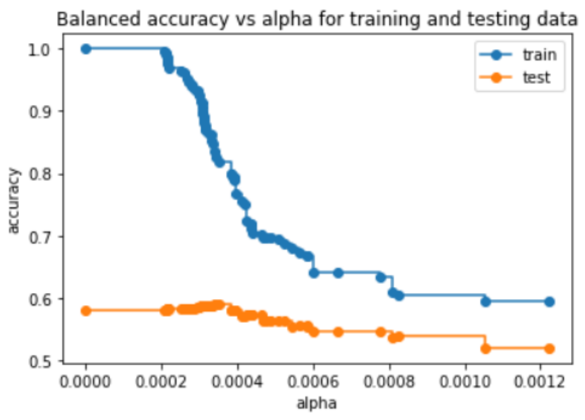
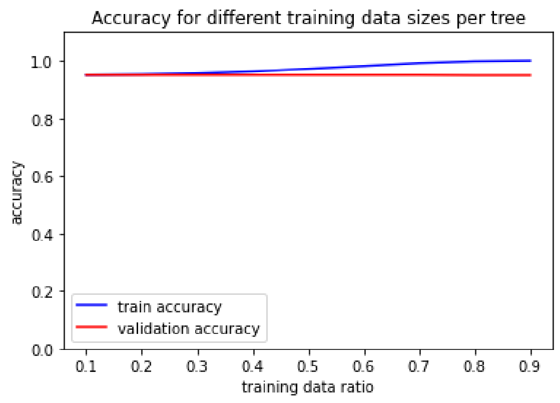


Figure 9.1, 9.2: *Decision trees for different alpha values*

Random forest

For the random forest, we plotted the accuracy of the forest on the validation data for various ratios of the training data that each tree was trained on. As seen, the training accuracy increased as the trees were trained on a larger part of the training data, but the validation accuracy did not. The optimal training data ratio was found to be 0.7, as this was where the balanced accuracy for the validation data is optimal (0.52); curiously, there does seem to be a bump in the balanced accuracy at this point, but it is likely that this is just a random variation.



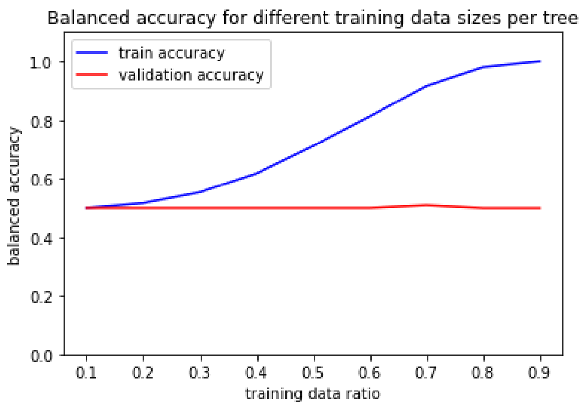
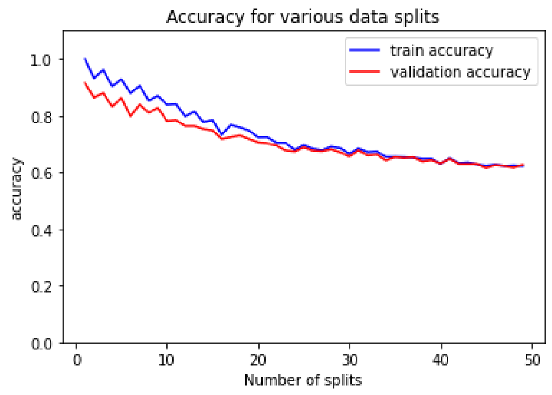


Figure 10.1, 10.2: *Performance for different training data ratios*

Resampled forest

For the resampled forest, the parameter we could change was the number of splits, which is also the number of trees in the forest. This split determines how many batches the non-stroke data was split, and thus also the ratio of stroke to non-stroke data. The larger the number of splits, the more equal the ratio. We plotted the validation and training accuracies for splits between 0 and 50.



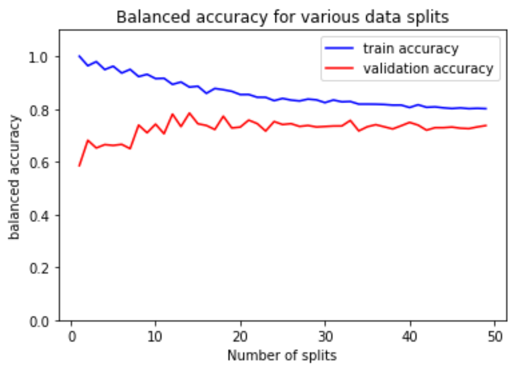


Figure 11.1, 11.2: *Performance for different numbers of data splits*

The optimal number of splits for the highest balanced accuracy on the validation data was 14, with a balanced accuracy of 0.597.

Neural networks

Regarding the training of the neural network, we have so far implemented standardized initial weight distributions, class weight imbalance and oversampling of the training data. Here the effects of different hyperparameters are evaluated individually.

Weight initialization

For initialization of the model's weight parameters, we used a set Gaussian parameter distribution with 0 mean and 0.05 standard deviation. For our optimized model, these measures can be tweaked if we suspect that the model has reached a local minimum. For validation of the model parameters we used a set seed for pseudo-random initialization in order to keep all variables (except the parameter to research) constant.

Class weight changes for loss function

For our actual model we have tried to find some parameters that can optimize training of the model. The main focus here is again to find ways to be more sensitive to the stroke class. We prefer having no false negatives to no false positives, since all strokes are at least found and all samples with positive classifications could be screened to further assess their risk of having a stroke.
In order to increase the data for validation of the model and to get a more robust measure of model performance, 5-fold cross-validation is used during model evaluation.

During training of our model, we have tried different class weight ratios. In the figures below, on the horizontal axis the different weight ratios of the stroke class relative to the non-stroke class are shown. For each weight ratio, the model was trained and validated to calculate the validation metrics. The first figure shows the trade-off between accuracy and balanced accuracy as an effect of class weight changes. With higher relative weights of the stroke class, the accuracy drops due to an increase of false positives. However, the balanced accuracy is improved due to the increase of sensitivity of the model, which outweighs the drop in specificity. It can be easily seen that balanced accuracy and sensitivity of the model are improved by increased stroke weights. This increase reaches a maximum at somewhere between 10 and 15, after which there is no further gain. The overall accuracy and specificity actually decrease, which is due to an increase in false positive results. However, overall the model has become a more useful tool for stroke detection.

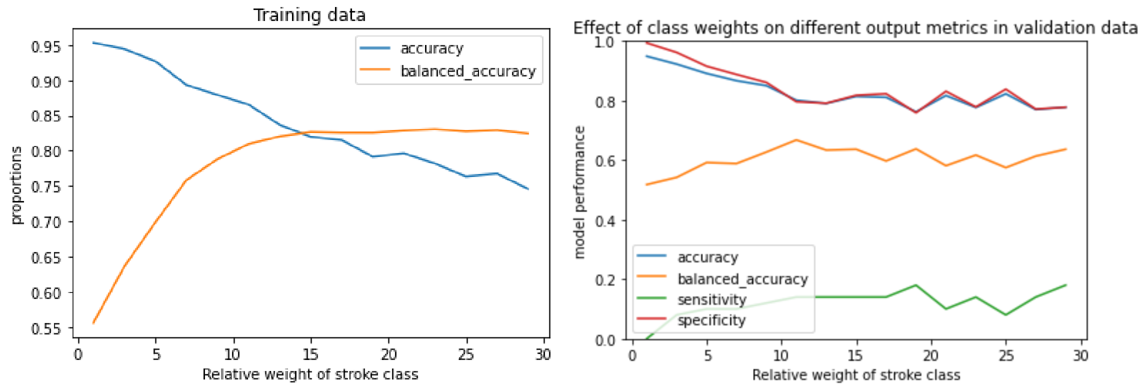


Figure 12.1, 12.2: *Performance of NN on training data (left) and validation data (right) for different relative weights of the stroke class, in different metrics.*

Effect of oversampling of stroke class data

In order to visualize what the effect of the data oversampling was on the performance of the model, we trained the network with different oversampling ratios. In figure 6, the ratio of oversampling (stroke samples / non-stroke samples) against the performance of the model on the validation data was measured. To best measure the effect of the oversampling, the class-weight was set to 1:1. We also observed a significant increase in sensitivity. The biggest increase is measured at ratios around 0.2. After a ratio of 0.3, there are no more significant fluctuations. In further improvements, it is likely that an oversampling ratio of 0.3 will be used.

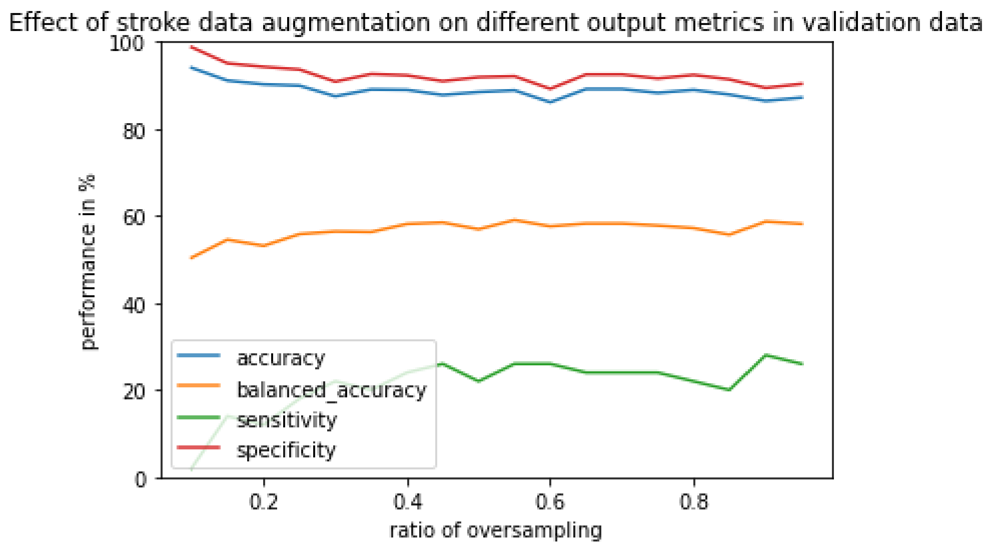


Figure 13: *Plot of the ratio of oversampling against the performance of the neural network*

Combining Class weights and Data oversampling

Both class weight changes and stroke data oversampling are methods that were implemented to counteract the imbalance of data that leads to overclassification of the abundant (non-stroke) class (Yu & Radewagen, 2017). Both of these methods have shown a positive effect on the sensitivity and therefore, the balanced accuracy of the model performance on validation data. However, if these imbalance methods are used together, this can potentially lead to an overshoot in false positives, which actually reduces the model performance measured by the balanced accuracy; the specificity of the model is reduced too much when too much oversampling and class weights are used.

Therefore, we plotted the performance in a 3D plot as a function of both class weight and oversampling ratio (see Figures below). This way, we are able to see the optimal combination of class weight and oversampling for our basic model. For this figure, the basic actual model was used (see pipeline in paragraph 2). It shows that either method has a large positive effect on the balanced accuracy. However, if the oversampling ratio is very low (0.1) and the class weight ratio is high (15), the model performs best. Besides, the plot below confirms that while the sensitivity increases with increased values for both hyperparameters, for the specificity the opposite is true.

Balanced accuracy

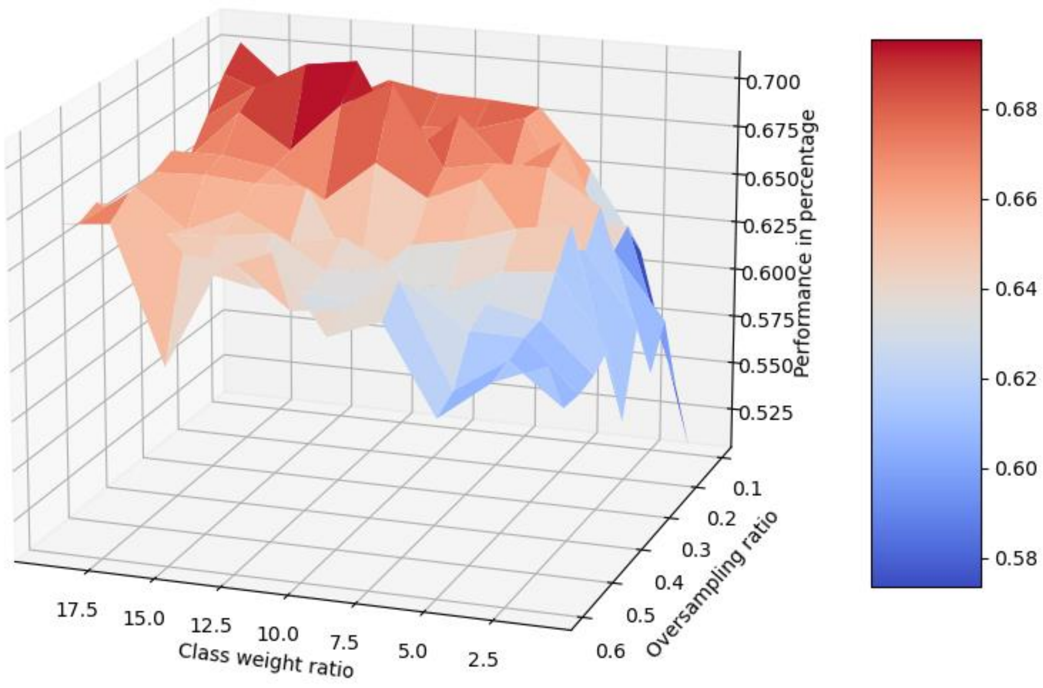
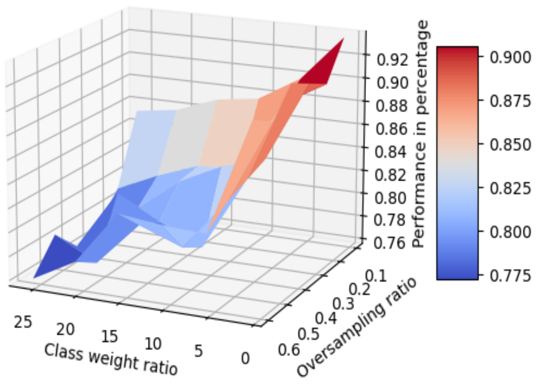


Figure 14: *3D plot of the performance (balanced accuracy) of the baseline neural network with different combinations of the class weight ratio and the oversampling ratio*

Specificity **Sensitivity**



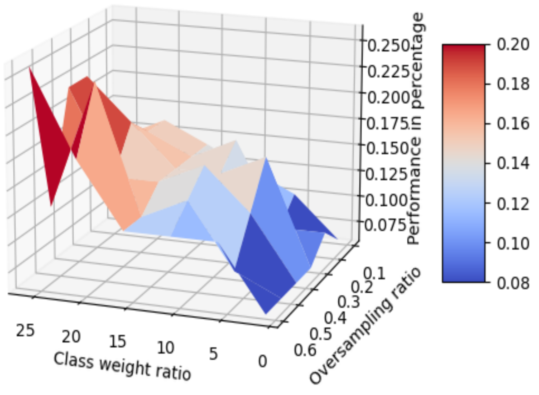


Figure 15.1, 15.2: *3D plot of the performance (left specificity, right sensitivity) of the baseline neural network with different combinations of the class weight ratio and the oversampling ratio*

Section 4: Model Evaluation

K-Nearest Neighbours

The k-Nearest Neighbours algorithm was not meaningfully changed in this version of the model, as the various values for k did not affect the predictions. Therefore, the accuracies for this version are the same as the accuracies in chapter 1, section 4. The only thing we will supply here are the updated graphs with the decision boundaries, as the input data we use is now normalized, which has altered the appearance of the plots.

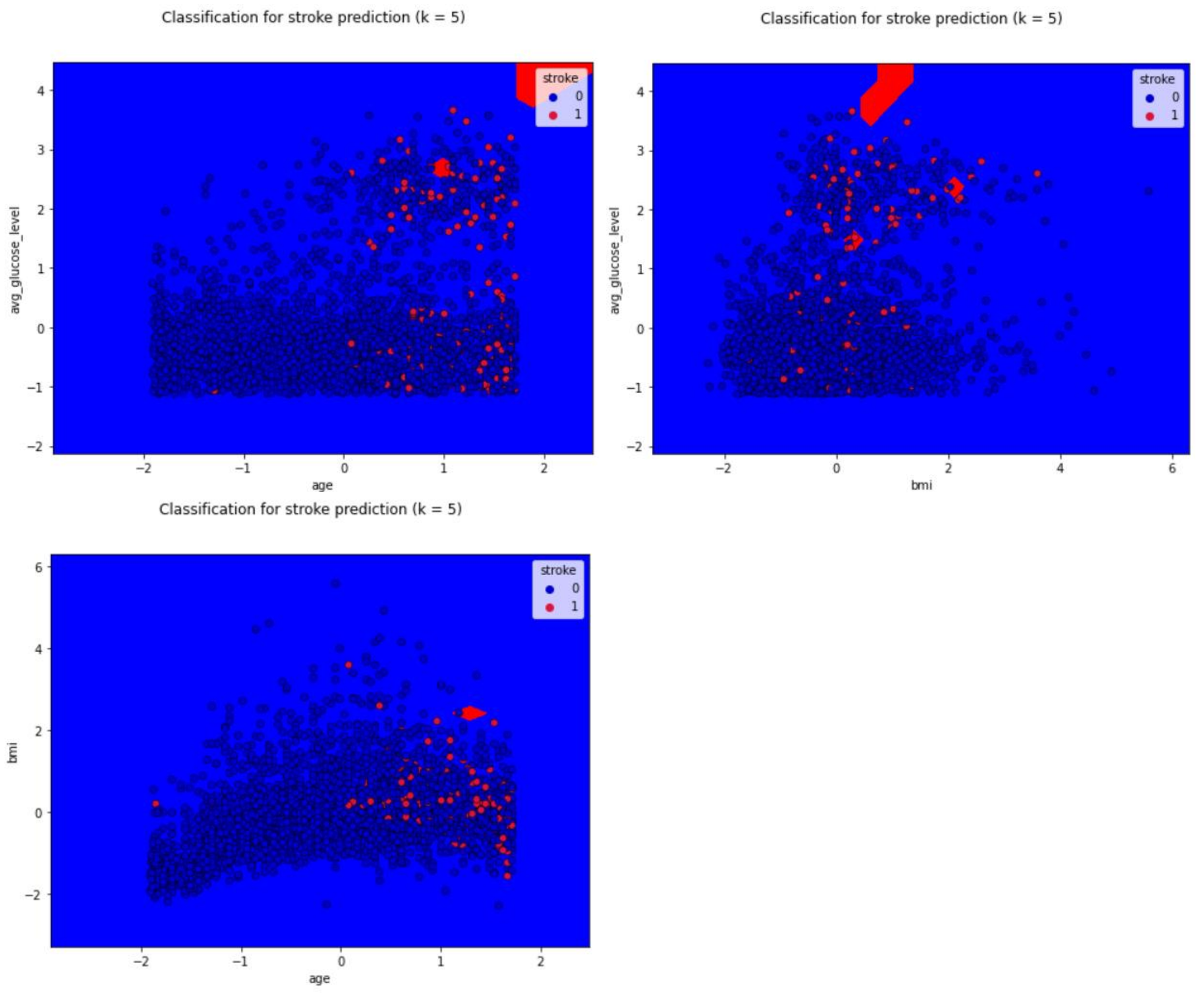


Figure 16: *The decision boundaries for the different normalized numerical data measures.*

Decision tree

We report the test metrics for four different instances of the model: the decision tree that was optimized by limiting complexity, the decision tree that was pruned, the random forest and the resampled forest.

Limited complexity

We analysed what the optimal depth of the tree and the optimal minimal number of splits was to reach the highest balanced accuracy on the validation data. Using both the optimal maximum tree depth and the optimal minimal number of samples present for a split in the decision tree the following accuracies were reached:

Train metrics:
Accuracy: 100 %
Balanced accuracy: 100 %
Sensitivity: 1.00
Specificity: 1.00
Confusion matrix:

TN: 2916	FP: 0
FN: 0	TP: 149

Test metrics:
Accuracy: 92.5 %
Balanced accuracy: 58.1 %
Sensitivity: 0.20
Specificity: 0.96
Confusion matrix:

TN: 935	FP: 37
FN: 40	TP: 10

Pruning

The tree that was optimally pruned using an alpha of 0.00034 leads to predictions with the following metrics:

Train metrics:
Accuracy: 98.3 %
Balanced accuracy: 82.5 %
Sensitivity: 0.65
Specificity: 1.0
Confusion matrix:

TN: 2915	FP: 1
FN: 52	TP: 97

Test metrics:
Accuracy: 94.1 %
Balanced accuracy: 59.0 %
Sensitivity: 0.20
Specificity: 0.98
Confusion matrix:

TN: 952	FP: 20
FN: 40	TP: 10

The basic model for the decision tree without adjustments had a balanced accuracy of 57.3%. So, restricting the depth of the tree does improve balanced accuracy of the model by about 1 percentage point and pruning the tree based on alpha by about 2 percentage point.

We also once again plotted our decision tree. In the figure below you can see that there is a split made on BMI = 30.436 and on BMI = 30.486. This is probably caused by replacing the N/A values from BMI with the mean value: 30.471. It is very likely that the people with the missing BMI actually had a higher or lower BMI, therefore, this split is not reliable.

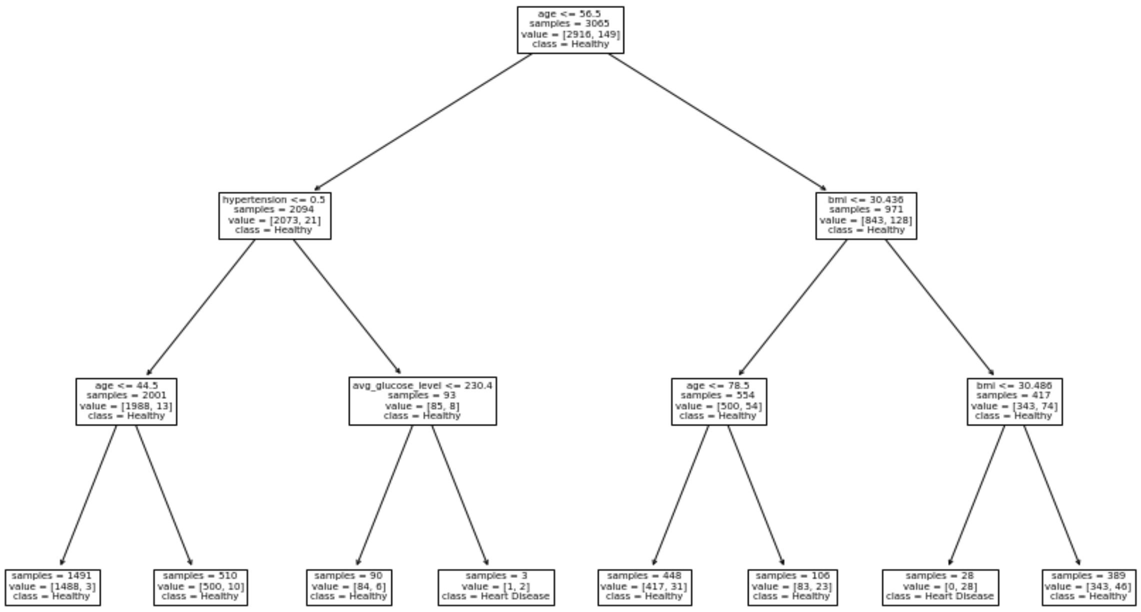


Figure 17: *Visualization of the actual decision tree*

Random forest

Using the standard random forest from sklearn, with 100 trees and a max_samples ratio of 0.7, led to the following accuracies:

Train metrics:
Accuracy: 99.2 %
Balanced accuracy: 92.3 %
Sensitivity: 0.85
Specificity: 1.00
Confusion matrix:

TN: 2916	FP: 0
FN: 23	TP: 126

Test metrics:
Accuracy: 95.1 %
Balanced accuracy: 50.0 %
Sensitivity: 0.00
Specificity: 1.00
Confusion matrix:

TN: 972	FP: 0

FN: 50	TP: 0

The balanced accuracy of this random forest is extremely low; it performed worse than a single tree, probably because the majority vote system effectively cancels out all cancellations of strokes, as this is the 'rare' class.

Resampled forest

The resampled forest, made by training a forest on all stroke data combined with different parts of the non-stroke data (split 14 ways) led to a great improvement in balanced accuracy:

Train metrics:
Accuracy: 82.9 %
Balanced accuracy: 91.0 %
Sensitivity: 1.00
Specificity: 0.82
Confusion matrix:

TN: 2392	FP: 524
FN: 0	TP: 149

Test metrics:
Accuracy: 79.6 %
Balanced accuracy: 77.0 %
Sensitivity: 0.74
Specificity: 0.80
Confusion matrix:

TN: 777	FP: 195
FN: 13	TP: 37

This model seems promising, and is something we should try to develop further.

Neural Network

Here we will evaluate the model that contains the basic model architecture (using 1 hidden layer with 25 nodes, and using the best hyperparameters as described in the previous section. We trained a model using these parameters (no data augmentations and class weights = 15).

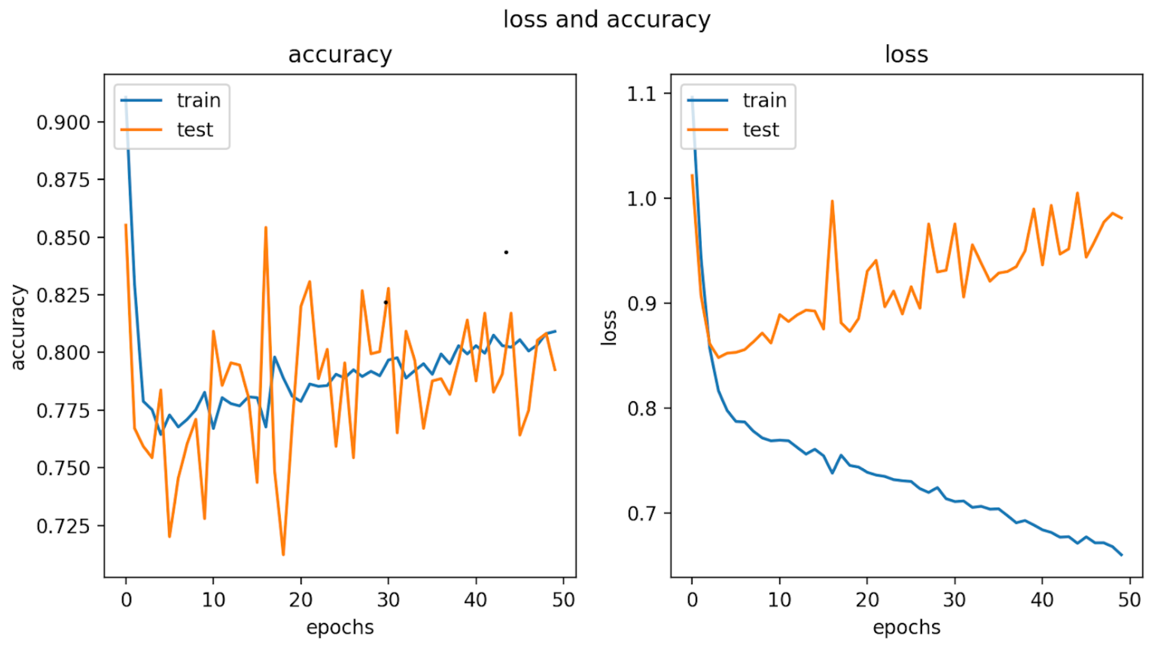


***Figure 18**. Model performance of actual model using class_weight = 15 and normalized initialized parameter weights.*

final average validation accuracy: 82,38%
final average validation accuracy balanced: 73.09%
final average sensitivity: 0.6800
final average specificity: 0.7819
Confusion matrix:

TN: 760	FP: 212
FN: 16	TP: 34

The model performance of 82.38% balanced accuracy reached by the model above is a large improvement to the balanced accuracy around chance level (50.00%) found by our baseline model. The sensitivity has significantly increased, on the cost of the specificity.

One point of concern is the largely increasing loss of the validation which is likely due to overfitting of the model. If the model already overfits using a relatively simple model architecture, this means that for more complex models, the overfitting should be counteracted. One way to do that is to add dropout during training of the model.

Another thing to note is that, it can be seen that there are large fluctuations in the accuracy over epochs. We discovered that Tensorflow's standard gradient descent method uses mini-batches of size 32. For future models it could help to train each model using all data, especially with a dataset this unbalanced where it is not sure if a batch contains stroke data.

Chapter 3. Improved model

Section 1: Data Analysis

After feedback, we found that our current way of one-hot encoding the data was not sufficient; we only transformed categorical features with more than 2 values to one-hot encoded features, leaving binary features like gender in one column. This meant that 'male' for example would always have the value of 0, which meant that it would not be propagated to the next layer, no matter how large the weights were. Therefore, we encoded every categorical feature into one-hot encoded columns, so gender became a female and a male column. This also meant that we didn't have to replace 0 with -1 for the neural networks; this is not a good solution for the issue, as this requires the weights to be negative if we wanted a positive activation.

Section 2: Data Pipeline

K-Nearest Neighbours

In this version we used the oversampled data on the same model as in chapter 2.

Decision Tree

The input for the models in the decision tree notebook remained similar as in chapter 2; data was not normalized, as this is not essential for decision trees and it has a negative effect on the interpretability of the model. Data *was* one-hot encoded in the new way described in section 1, but this had no significant consequences for the decision tree based models.
We updated the random forest to now train every tree on every feature in the dataset, instead of the square root of the number of features, which was the default setting.

Neural Network

Most of the neural network pipeline can be found in the previous chapter. In our model creation we added the option of placing dropout layers right before each hidden layer. The dropout rate can be passed as an argument for each dropout layer individually. Finally, for this model the batch_size was set to None, since mini-batch learning could make training actually worse with highly unbalanced data. (If mini-batches could be split in a stratified manner, this would work but was too hard to implement in keras). Since all the methods to tweak our hyperparameters were already implemented, most of our focus for model implementation went to finding these optimal hyperparameter values. Our parameter optimization will be further discussed in section 3 of this chapter.

Combined model

In this last chapter, we combined our three models into one; this model will use the predictions made by k-NN, the resampled forest and the neural network. Every model will work as it does individually, but the predictions from all models will be combined by a self-written or function; if one of three models predicted that a patient was at risk for a stroke, it was classified as a 'stroke' sample. We also tried a decision based on a majority vote (2 out of 3 algorithms predicted a sample as 'stroke'), to see if this worked better than the or function.
The parameters for the models were chosen based on model training for the individual models. The k-Nearest Neighbours model was trained on only numerical data, as this was found to have the highest validation accuracy. The resampled Forest used a split of 17 and the Neural Network was trained with

2 hidden layers of 25 and 15 nodes respectively, class weights of 15, 50 epochs and a dropout layer with a rate of 0.3 before both hidden layers.

Section 3: Model training

K-Nearest Neighbours

The model is trained on categorical data and numerical data separately and on a combination of both categorical and numerical data. All data has been oversampled in different ratios. A ratio of 0.05 means there was no oversampling and a ratio of 1 means there are as many stroke samples as non-stroke samples. The model was trained on oversampling ratios from 0.1 to 1 with steps of 0.2.
We trained the model on different oversampling ratios. In the figure you can see that a ratio of 0.9 got the highest balanced accuracy (0.67) on the numerical data; this is therefore the model we should continue to use.

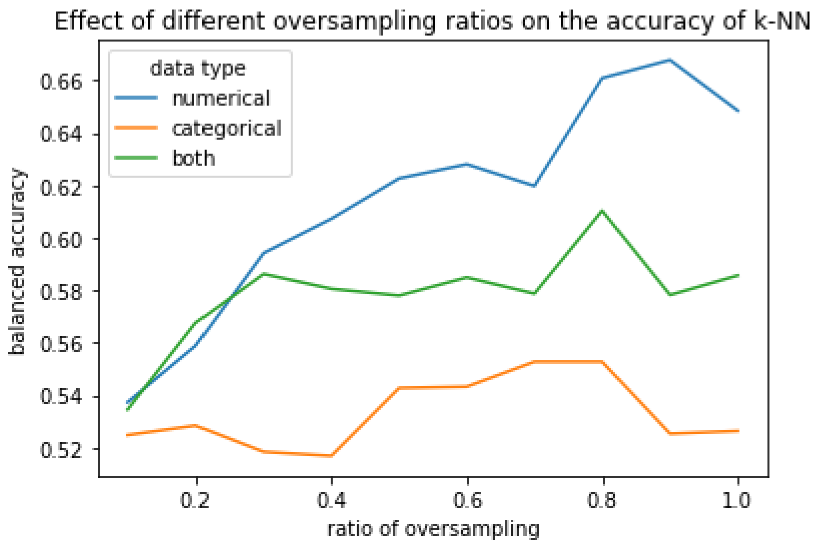


Figure 19: *Performance of k-NN for different oversampling ratios in different metrics*

To show the influence of oversampling data on the number of people classified with a stroke in k-NN we have plotted 2 numerical values at different oversampling ratios.

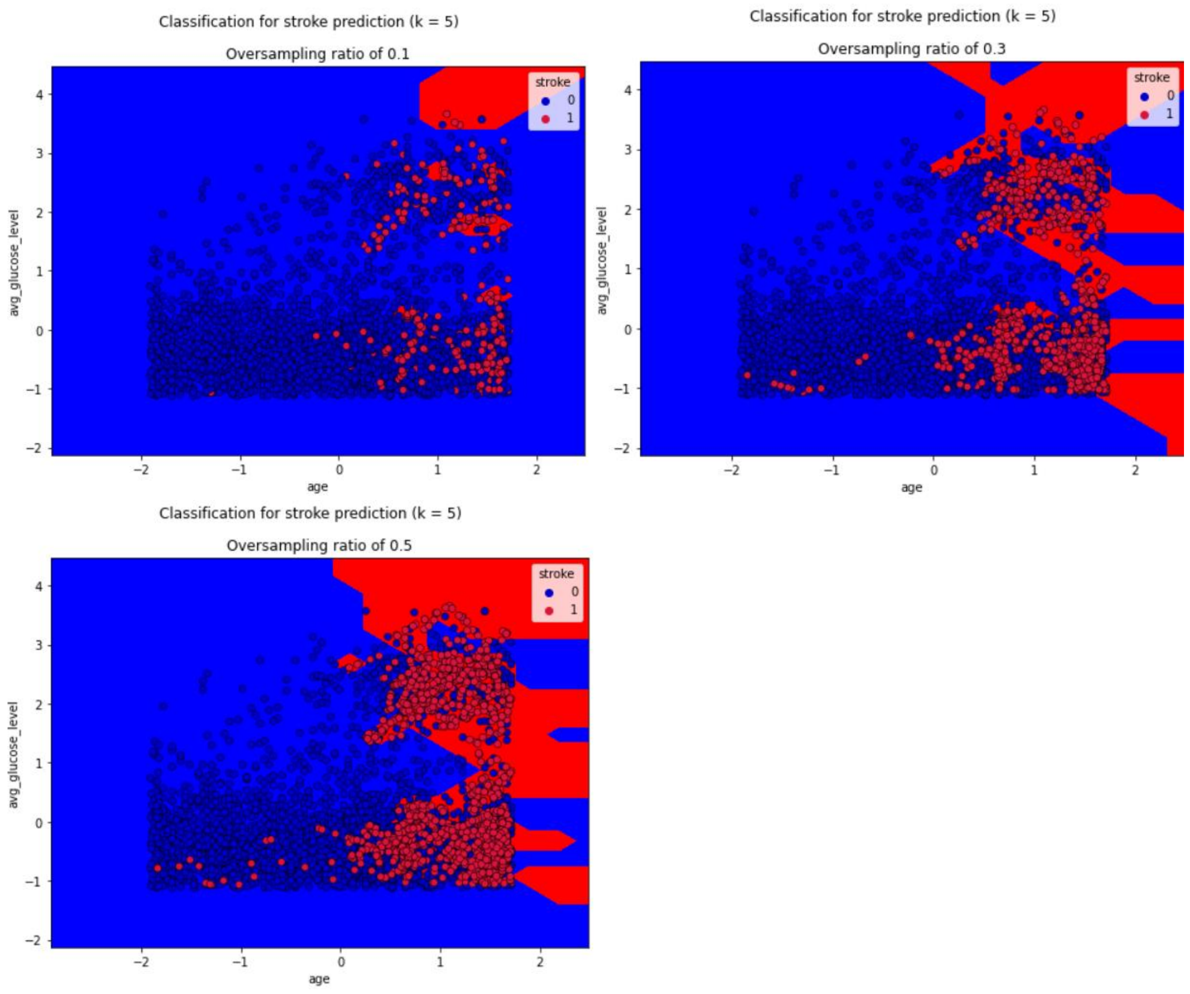


Figure 20.1, 20.2, 20.3: *k-NN visualization for oversampling ratios of 0.1, 0.3 and 0.5.*

Decision Tree

Class weights & features for decision trees and random forests

For the standard decision tree, we tried altering the class weights to deal with the imbalanced data, by trying different class weights for the stroke and non-stroke class, so that gain for the stroke class would be more important by setting the weights for the stroke class to be higher (Brownlee, 2020). We tried different weights and plotted the effect on the balanced accuracy. The non-stroke weight was kept constant at a weight of 1.

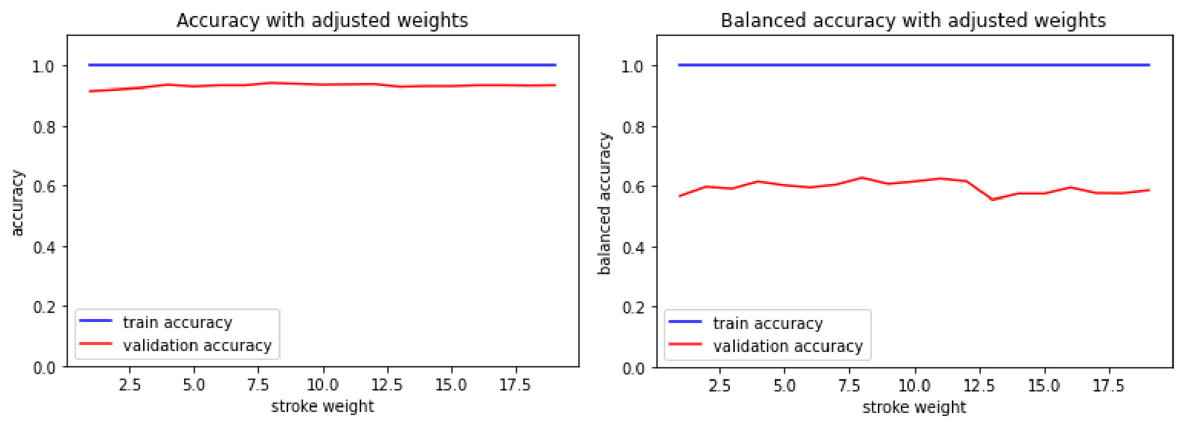


Figure 21.1, 21.2: *Performance with different stroke weight classes*

The best weight was 8 for the stroke class, with a balanced accuracy on the validation data of 0.63 for an individual tree. However, as you can see, the impact of the weight was not immense on the balanced accuracy; as we are manually adjusting the stroke-non-stroke data ratio in the resampled forest, we did not incorporate this altered stroke weight for the decision trees in the resampled forest. We briefly tried the adjusted stroke weight in the random forest, as well as the fact that we were training the every tree in the model on every feature now, but the same problem that we had in the previous version of the model remained; the balanced accuracy for the random forest was still 50%, as the majority vote system of the random forest is still unlikely to ever classify a case as at risk for stroke. Therefore, there is nothing to report for the model evaluation for the random forest.

Resampled forest

We tried different split sizes for the resampled forest for the previous version of our model; for this model, we tried to apply oversampling to the resampled forest as well. We first tried this with a fixed split size of 10, and we tried different oversampling ratios between 0.2 and 1.1 and plot the accuracy and balanced accuracies on the training and validation data:

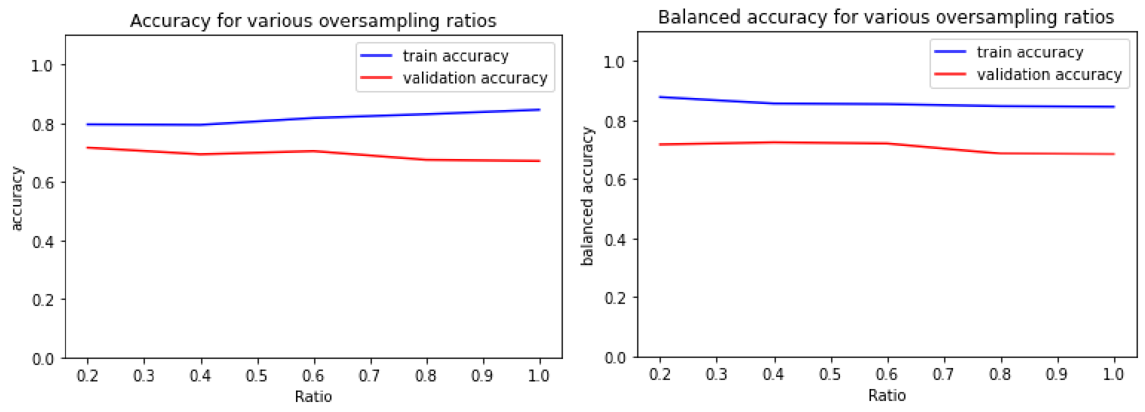


Figure 22.1, 22.2: *performance of resampled random forest for different oversampling ratios*

The optimal oversampling ratio was 0.4, with a balanced accuracy on the training of 0.72.

We do not know that the fixed split size of 10 would be optimal, however; therefore, it would be good to find the combination of the best split and best oversampling ratio that led to the highest balanced accuracy. For this, we tried splits in the range 0 to 30 and oversampling ratios between 0.2 and 1. Results are plotted in the following graph:

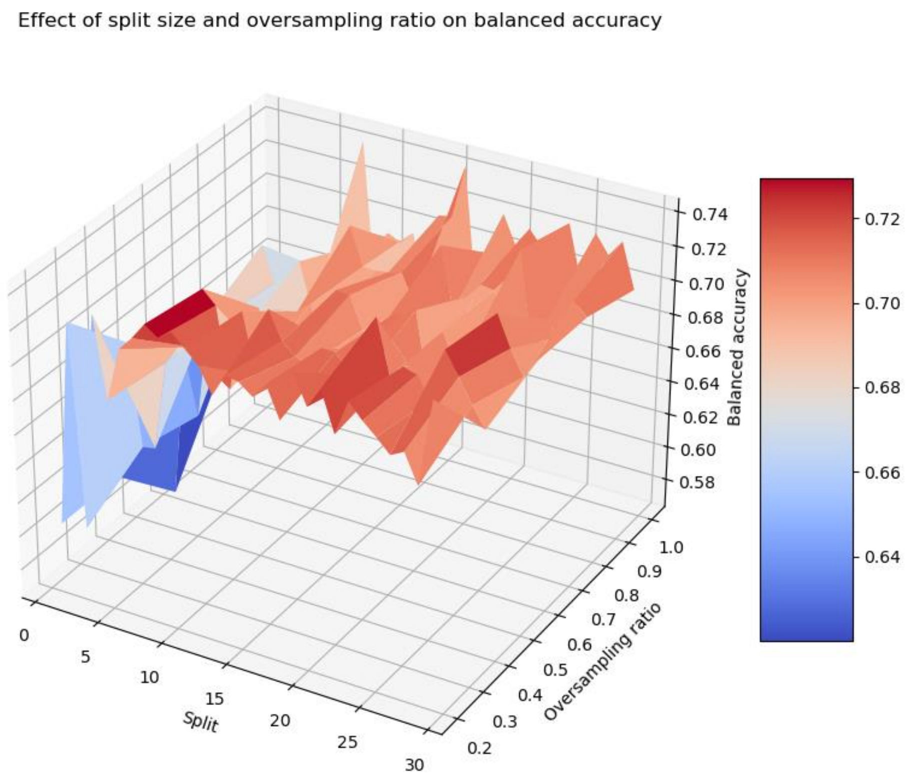


Figure 23: *3D plot of performance of random forest for different oversampling ratios and split sizes*

The optimal values were an oversampling ratio of 1.0 and a split size of 16. This led to a balanced validation accuracy of 0.74.

For the current run of the data, the validation accuracy for the resampled forest without any oversampling applied and a split of 17, the balanced validation accuracy was 0.77. This means the best model to use is actually the model that did not use oversampled input data.

Neural Network

For the final improvements of the neural network, the optimal combination of hyperparameters ought to be found. In the end, there were many different parameters in which we could change. As for the architecture of the neural network, we could change the number of layers, the number of nodes in those layers and the dropout rate. We can also vary the stroke / non-stroke ratio with oversampling. Besides, we can use a certain dropout rate, class weight ratio and vary in the number of epochs. In order to find the optimal combination, we organised a table with the possible promising parameters combinations and their performance, table … . Here we kept track of all the models we tried out. All models were validated using 5-fold cross-validation.

The first focus for optimizing our model went to attempts on changing the number of layers and nodes. As discussed in the previous chapter, for our basic model architecture (1 hidden layer with 25 nodes), the optimal imbalance hyperparameters were class_weight=15 without data augmentations.
With these settings fixed, we played with the architecture of the model, including the amount of layers, nodes and addition of dropout with different dropout rates.

Our ‘best’ neural network:

We found that under certain conditions (see table), the network worked better than others.
The highest accuracy was found using the following model with 2 hidden layers:
Input layer: 22 nodes (one for each feature, one-hot encoded for categorical data)
Dropout layer 1: dropout rate = 0.3
Hidden layer 1: 25 nodes, ReLU activation
Dropout layer 2: dropout rate = 0.3
Hidden layer 2: 15 nodes, ReLU activation
This model was validated using 5-fold cross-validation, metrics are averaged over all 5 iterations:
Accuracy: 77.5%
Balanced accuracy: 76.79%
Sensitivity: 0.7600
Specificity: 0.7758

Confusion matrix:

TN: 603.6	FP: 174.4
FN: 9.6	TP: 30.4

Using the model described above, we went on to search for the optimal hyperparameters for oversampling and class weights, since the optimized model might require different setting of these parameters.

Just like in the previous chapter, we plotted the performance against differing oversampling ratios and class weight ratios (Figure …, below). Since the model has to be trained and tested many times to get a good overview of the many possibilities, computational time increases enormously, to an estimated 3 hours) when trying 15 class weights and 15 oversampling ratios. Therefore, for this investigation we chose not to use k-fold cross-validation which reduced the computational time 5 times.

The model performed best with an oversampling ratio of 0.6 and a class weight ratio of 1, which means there is no extra weight to a stroke sample, at a balanced accuracy on the validation data of 0.73. However, the model also performed well around a class weight of 15, with an oversample ratio of 0.05, which is approximately equal to applying no oversampling, since the natural data has this class ratio already. Even though the model performed better with the oversampling ratio of 0.6, the spike in the graph is not as robust as the one where the class weight ratio is around 15. The spike at oversampling = 0.6 is likely an outlier due to the lack of k-fold cross-validation in this analysis.

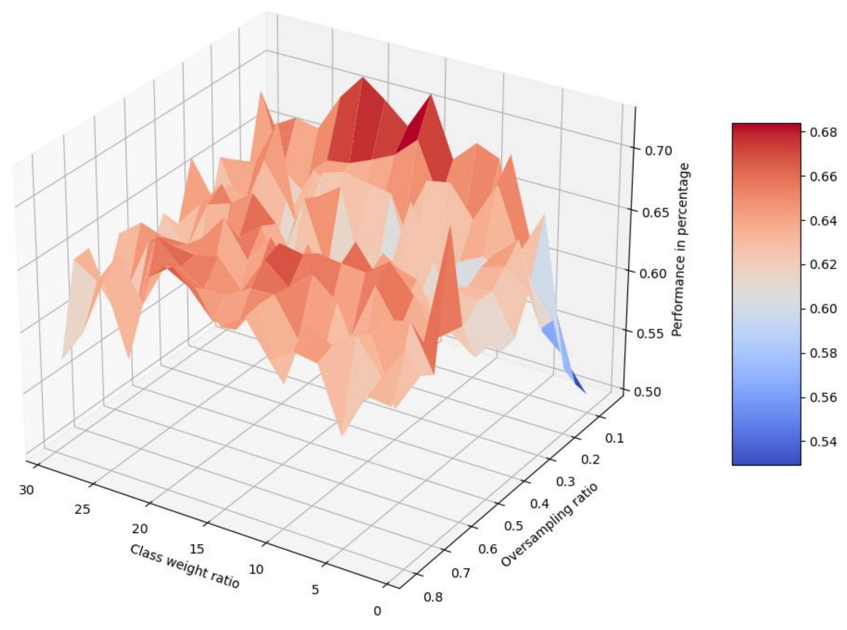


Figure 24: *Balanced accuracy using different class weights and oversampling ratios for our 'best model'.*

Undersampling of non-stroke data

One final method we tried for our deep neural network was applying undersampling of the non-stroke data, in a manner similar to the random forest method with decision trees. Instead of ensembling decision trees, different neural networks were created and all trained on a subset of the training data. Each time using all stroke data and only a subset of the non-stroke data. The predictions for the validation set were then combined using a majority vote system.
Unfortunately, this led to a large over classification of strokes, resulting in a poor specificity.

For the final model and without any oversampling or class weights applied, the performance is described below:

validation accuracy: 53.03%
validation accuracy balanced: 71.51%
sensitivity: 0.9200
specificity: 0.5103
Confusion matrix:

TN: 496	FP: 476
FN: 4	TP: 46

Combined model

The model parameters for the individual models were chosen based on previous experiences in model training. The only choice that had to be made was whether to use the or-function or the majority vote for combining model predictions.
The or-function achieved a balanced validation accuracy of 75.7%
The majority vote achieved a balance validation d accuracy of 74.5%
Therefore, in the model evaluation the or-function will be used to get the metric for the test data.

Section 4: Model evaluation

K-Nearest Neighbours

We predicted the classes for the testing data using a model that was trained on only the numeric features, on oversampled data with a ratio of 0.9.
Train metrics:
Accuracy: 100 %
Balanced accuracy: 100 %
Sensitivity: 1.00
Specificity: 1.00
Confusion matrix:

TN: 2916	FP: 0
FN: 0	TP: 2624

Test metrics:
Accuracy: 81.8 %
Balanced accuracy: 68.6 %
Sensitivity: 0.54
Specificity: 0.83
Confusion matrix:

TN: 809	FP: 163
FN: 23	TP: 27

This model, with oversampling, is more accurate than the past model without oversampling. However, it is not quite as accurate as the neural network and resampled forest.

Decision Tree

Based on the accuracies in model training, the best model to use for predicting the classes is the resampled forest with an optimized number of splits. This model reaches a balanced training accuracy of about 88% and a balanced testing accuracy of around 76%; exact numbers and specificity are reported in the model evaluation of chapter 2.
We will report the accuracies for the model that uses oversampling for comparison reasons:
The resampled forest that is trained on oversampled data, with an oversampling ratio of 1.0 and a split size of 16 (training data split 16 ways; 16 trees that make up the forest) achieved the following accuracies:

Train metrics:
Accuracy: 82.2 %
Balanced accuracy: 82.2 %
Sensitivity: 1.00
Specificity: 0.64
Confusion matrix:

TN: 1878	FP: 1039
FN: 0	TP: 2916

Test metrics:
Accuracy: 65.3 %
Balanced accuracy: 73.2 %
Sensitivity: 0.82
Specificity: 0.64
Confusion matrix:

TN: 626	FP: 346
FN: 9	TP: 41

This model has more false positives than the resampled forest without oversampling we used in chapter 2; this is logical, as this model was trained on more stroke data from oversampling *and* resampled datasets that consisted out of stroke data every time and only a small part of the non-stroke data. As far as the model knows, there are a lot more stroke patients in this dataset than there truly are, so it classifies more of the cases as stroke patients as well.

Neural Network

Using the testing data on our 'best model' we have found the following performance metrics.

Test metrics:
Accuracy: 78.57 %
Balanced accuracy: 74.51 %
Sensitivity: 0.7000
Specificity: 0.7901
Confusion matrix:

TN: 768	FP: 204
FN: 15	TP: 35

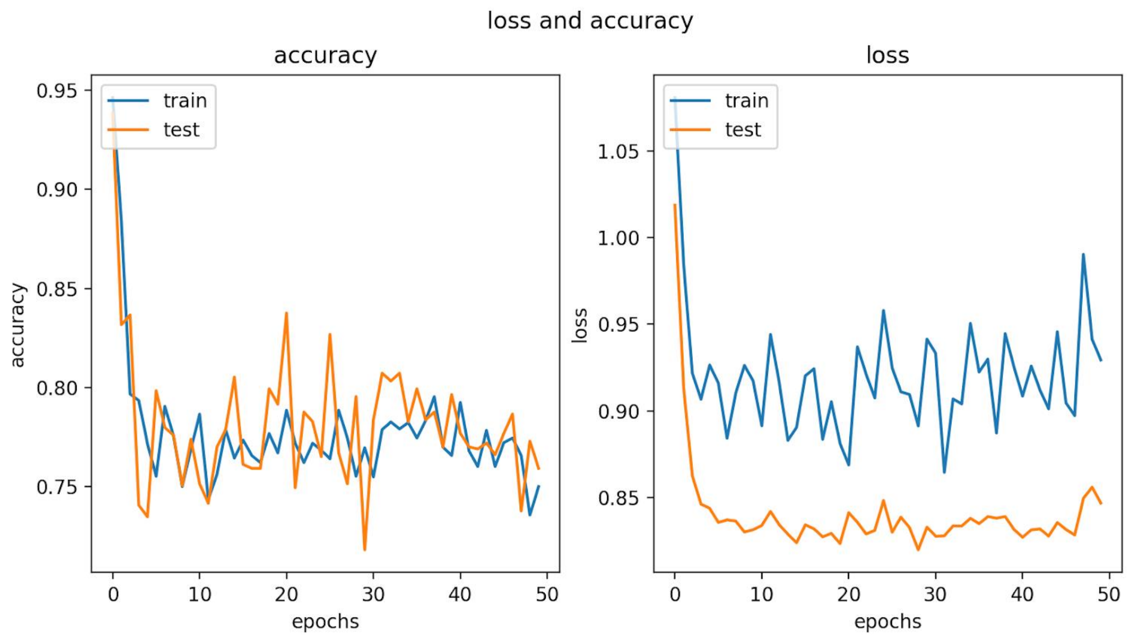


Figure 25.1, 25.2: *Accuracy and loss of our final model against the number of epochs*

The performance on the test data was slightly less then the balanced validation accuracy as reported in the previous section (76,79%). However, the model has improved compared to the actual model, which had a balanced accuracy of 73.09%. Luckily, the model does not seem to overfit anymore, but might be a bit too generalizing instead. Nevertheless, this has led to the best performance on data classification, with only 15 false negatives but relatively many (204) false positives.

Combined model

The k-Nearest Neighbours with a balanced accuracy of 68%, the Resampled Forest with a balanced accuracy of 76% and the Deep Neural Network with a balanced accuracy of 74% on this split data were combined. When the three models were combined using an OR function this resulted in a balanced accuracy of 77.3%. So, combining the model results in a slight increase in the accuracy.

Test metrics:
Accuracy: 71.3%
Balanced accuracy: 77.3 %
Sensitivity: 0.84
Specificity: 0.71
Confusion matrix:

TN: 687	FP: 285
FN: 8	TP: 42

Conclusion

Latere aanpassingen:

- Images map weghalen, of alles daar ook voor de zekerheid inzetten
- Eerdere versies van verslag uit github halen?
- Oude versies notebooks etc. uit de code map op github halen.

Appendix

Table 1: *Ratio of people with stroke and without stroke per feature of each category.*

Category	**Feature**	**Ratio (samples with stroke / total samples)**
Gender	Male	0.051
	Female	0.047
Hypertension	Yes	0.133
	No	0.040
Heart disease	Yes	0.170
	No	0.042
Married	Yes	0.066
	No	0.017
Work type	Private	0.051
	Self-employed	0.080
	Government job	0.050
	Children	0.003
	Never worked	0.000
Residence type	Urban	0.052
	Rural	0.045
Smoking status	Never smoked	0.048
	Formerly smoked	0.080
	Smokes	0.053
	Unknown	0.030

Table 2: *Performance of the trained and tested neural network combinations in architecture and data input.*

Column explanation:
Number: ID of the network layout, for referencing
N. layers: Number of layers in the neural network
N. Nodes: Number of nodes per layer. (sequential)
N. epochs: Number of epochs ran per complete evaluation
Oversampling ratio: Oversampling ratio = count stroke samples / count non-stroke samples. Used in the training phase.
Class weight: extra weight to cost calculation stroke sample. (non-stroke: 0, stroke: 1 * class weight).
Dropout: dropout rate * 100 % is the percentage of nodes dropped per layer. (sequential)
Accuracy: Accuracy of the model in percentage
Sensitivity: True positives / False negatives + True positives
Specificity: True negatives / False positives + True negatives
Con. mat.: Confusion matrix: [[True negatives, False positives] [False negatives, True Positives]]
Bal. accuracy: sensitivity + specificity / 2

Row explanation:
Row number 1: The baseline model
Row2-15: Variations on the baseline model

Bal. accuracy (%)	62,5	75,7	75,9	76,2	75,7	73,3	75,5	68	63,9	75,4
Con. mat.		[[629.8, 148.2] [11.6, 28.4]]	[[639.2, 138.8] [11.2, 28.8]]	[[633.6, 144.4] [11.6 ,28.4]]	[[650, 128] [14.8, 25.2]]	[[647, 131] [14,6 25,4]]	[[604, 174] [11, 29]]	[[661, 311] [16, 34]]	[[641, 331] [19, 31]]	[[625.8 ,152.2] [11.8, 28.2]]
Specificity (%)	79	82,0	81	81	83,5	83,1	77,6	68	65	80
Sensitivity (%)	14	69,5	71	71	63	63,5	73,5	68	62	71
Accuracy (%)	79,2	80,4	81,6	80,9	83	82,2	77,2	68	65,6	79,9
Dropout	None	None	None	None	None	0,1, 0,1	0,1 0,1	0, 0,1	0, 0,1	0, 0,2
Class weight	Optimum : 15	15	15	15	15	15	15	15	15	15
oversampling ratio	Optimum : 0.4	None	None	None	None	None	None	0.4	0.4	0.4
N. epochs	50	50	50	50	50	50	50	50	50	60
N. nodes	25	20	25	25, 5	25, 15	25, 5	25, 25	25, 5	25,15	25, 15
N. layers	1	1	1	2	2	2	2	2	2	2
Number	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*

76,2	76,4	76,7	75,9	74,4
[[598.4, 179.6] [9.8, 30.2]]	[[586.4, 191.6] [9, 31]]	[[603.6, 174.4] [9.6, 30.4]]	[[582.8, 195.2] [9.2, 30.8]]	[[602, 175.4] [11.4, 28.6]]
76,9	75,4	77,5	74,9	77,4
75,5	77,5	76	77	71,5
76,8	75,5	77,5	75,0	76,7
0,2 0,2	0,3 0,3	0,3 0,3	0.3, 0.3, 0.3	0.3, 0.3, 0.3
15	15	15	15	15
None	None	None	None	None
50	50	50	70	70
25, 5	25, 5	25, 15	25, 25, 5	25, 25, 15
2	2	2	3	3
11	*12*	*13*	*14*	*15*

References

Ambielli, B. (2017, 29 october). Gini Impurity (With Examples). https://bambielli.com/til/2017-10-29-gini-impurity/

Brownlee, J. (2021, 29 january). Cost-Sensitive Decision Trees for Imbalanced Classification. https://machinelearningmastery.com/cost-sensitive-decision-trees-for-imbalanced-classification/

Palacios, F.S. (2021, Januari). Stroke Prediction Dataset: 11 clinical features por predicting stroke events. Retrieved June 14, 2021 from https://www.kaggle.com/fedesoriano/stroke-prediction-dataset

Tay, K. (2020, 23 january). What is balanced accuracy?. https://statisticaloddsandends.wordpress.com/2020/01/23/what-is-balanced-accuracy/

Yu, W. & Radewagen, R. (2017). 7 Techniques to Handle Imbalanced Data.
https://www.kdnuggets.com/2017/06/7-techniques-handle-imbalanced-data.html

Example-Weighted Neural Network Training—Wolfram Language Documentation. (2021). Wolfram. https://reference.wolframcloud.com/language/tutorial/NeuralNetworksExampleWeighting.html

data augmentation:
SMOTE-ENC: A Novel SMOTE-Based Method to Generate Synthetic Data for Nominal