Chapter 1. Baseline model

Section 1: Data Analysis

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.
Subsequently, we replaced the variable columns that contained categorical values with one-hot-encoded columns. 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. Besides, for the 'gender' feature the category 'other' was removed since this category contained only one sample, and removing this category allows for gender to be one-hot encoded with male as 0 and female as 1.
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 model evaluation.
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).
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.

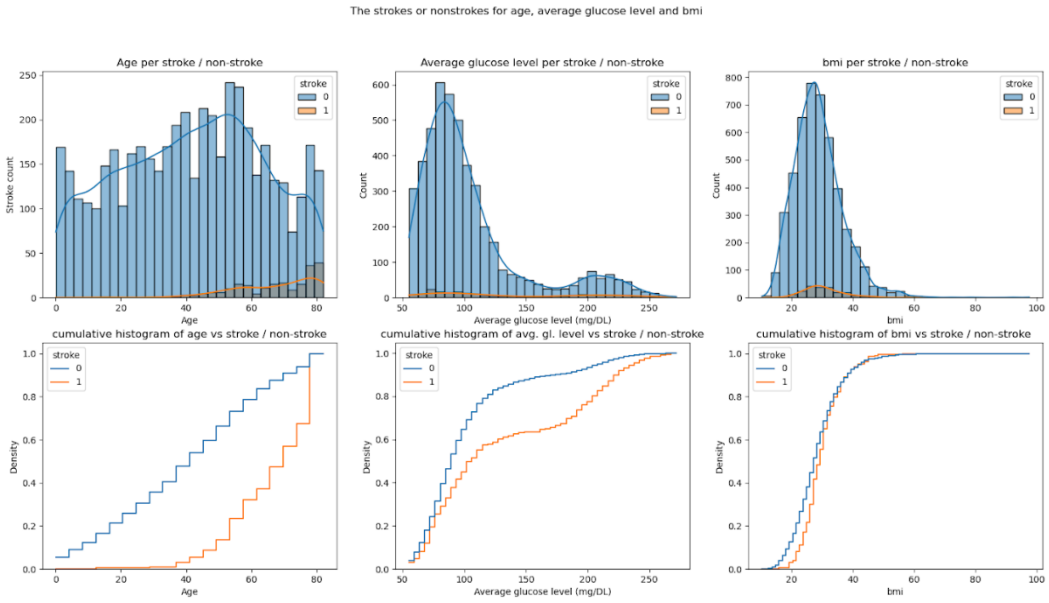


Figure 1: *Normal and cumulative histograms, displaying the age, average glucose level and BMI against the stroke occurrence.*

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. A category where there were noteworthy differences were hypertension (relatively there are more people with hypertension that have a stroke than without hypertension).

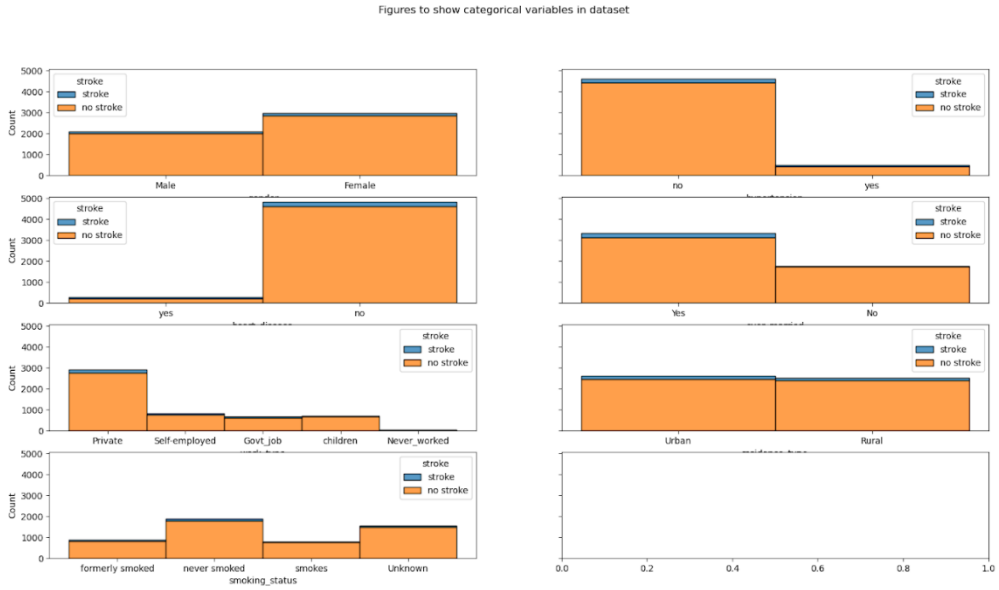


Figure 2: *Stroke and non-stroke samples per categorical data feature*

Section 2: Data Pipeline

To begin with, we made 3 separate base models to see how well they would be able to classify the patients that were at risk for a stroke. We created a k-Nearest Neighbor classifier model, a decision tree and a 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 be stacked.
The k-Nearest Neighbor algorithm classifies unknown data points based on similarity to other, known data points. We did 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 Neighbors

For K-Nearest Neighbors, 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 k-NN classifier, for example the Jaccard distance metric. 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 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 split at 0.5 for a prediction of 0 for no stroke and 1 for stroke.

Section 3: Model Training

K-Nearest Neighbors

The nearest neighbors model is relatively simple; the only parameters that can be set is the amount of neighbors that should be considered for classification, the metric used for distance and the

weights function to determine how a point in a 'neighborhood' 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 neighbors 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

Cost is defined by the binary cross-entropy. We tried to train the model using 30 epochs. For predictions, one sigmoid output layer is used. There were no additional parameters to set (yet).

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 balanced accuracy score 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 Neighbors

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: 94.81%
Numerical data accuracy: 94.70%
All data accuracy: 94.60%

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: 50.00%
sensitivity: 0.0
specificity: 1.0

TN: 931	FP: 6
FN: 51	TP: 1

Numerical data
balanced accuracy: 50.67%
sensitivity: 0.0196
specificity: 0.9935

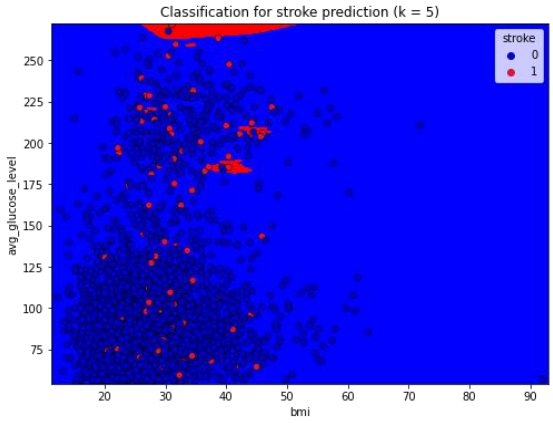
TN: 925	FP: 6
FN: 50	TP: 1

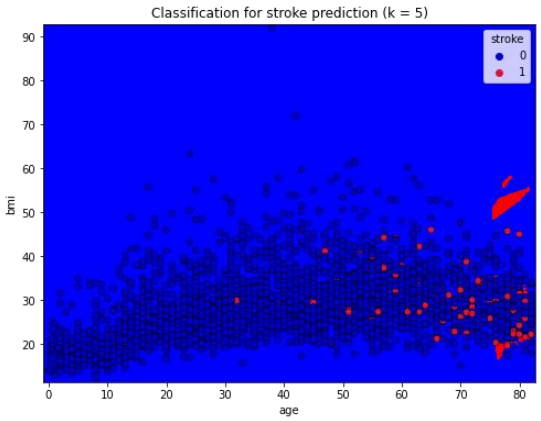
All data
balanced accuracy: 50.82%
sensitivity: 0.0196
specificity: 0.9978

TN: 929	FP: 2
FN: 50	TP: 1

This shows that the categorical data do a somewhat better job than the numerical data, but all predictions are only slightly above chance level (50.00% for balanced accuracy) and sensitivity is extremely low.

Decision Boundaries for numerical data at k=5.





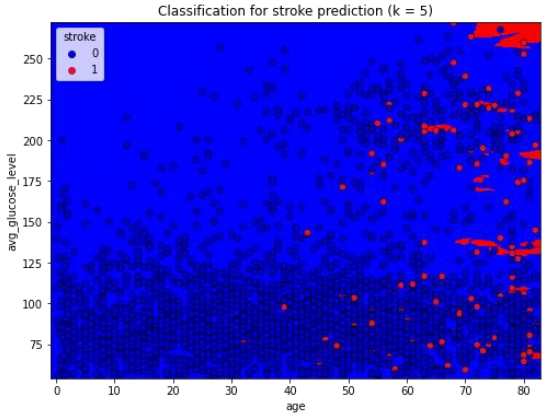


Figure 3: The figure above shows the decision boundaries for the different numerical data measures. The scatteredness of the stroke prediction areas seem to confirm the 'randomness' suggested by the low accuracies of the test predictions. Besides, the distributions of stroke and non-stroke samples largely overlap, showing the difficulty of separating these data using neighbor 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

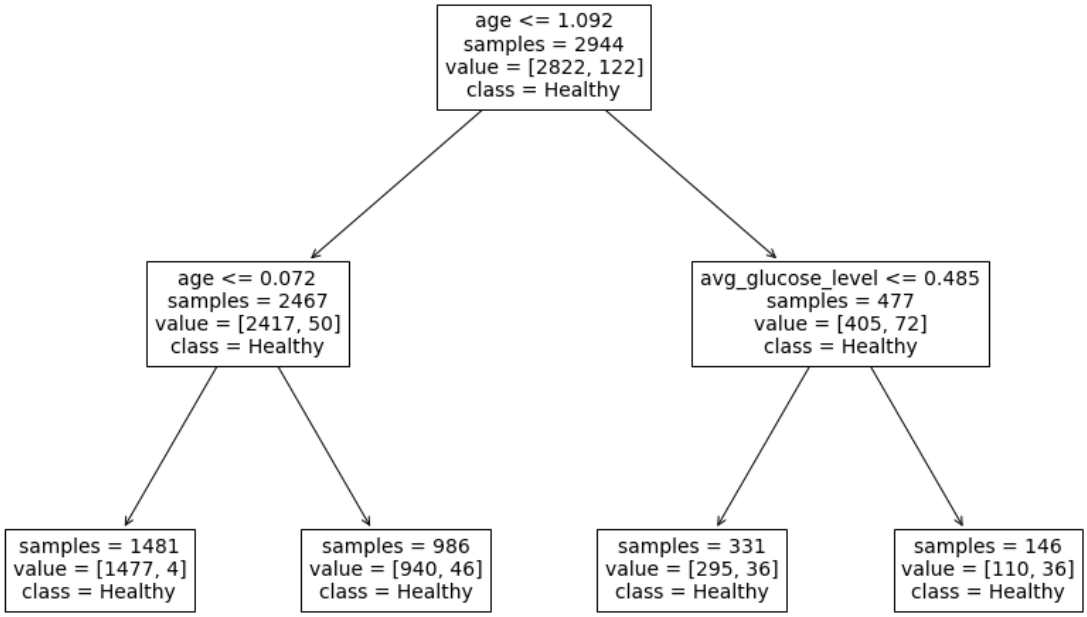


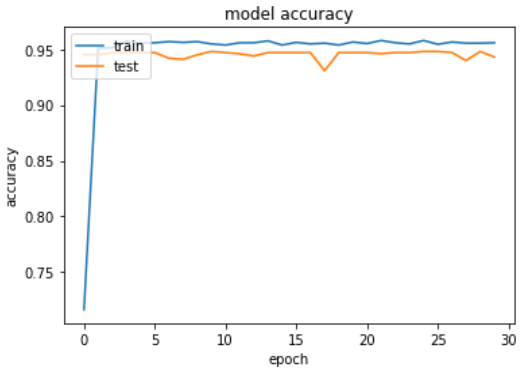
Figure 4: Decision tree for a maximum depth of 2 and without minimum split restrictions

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"]]
```

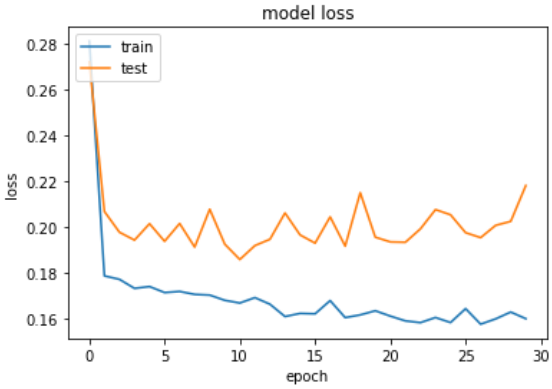


Figure 5: *Training and testing accuracy and loss of the baseline neural network*

Train metrics:
accuracy: 95.5 %
balanced accuracy: 51.8 %
confusion matrix:

TN: 2807	FP: 15
FN: 117	TP: 5

Test metrics:

accuracy: 94.3 %
balanced accuracy: 52.5 %
confusion matrix

TN: 923	FP: 8
FN: 48	TP: 3

Chapter 2. Actual model

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 maximum 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.

Section 2: Data Pipeline

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 every feature of the data, and used the default of 100 trees in the forest.
Third, we implemented a forest that is an ensemble of different resampled datasets; 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, based on tips 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

For our neural network, the categorical data representations were altered slightly. Instead of using 0 and 1 classes, we changed the 0 values to -1. Since the model uses a weighted sum of the input values to calculate the activation of subsequent layers, using many values of exactly 0 may impair training of the model.

For our further optimizations we made use of ideas from the following sources:
https://www.kdnuggets.com/2017/06/7-techniques-handle-imbalanced-data.html
https://reference.wolframcloud.com/language/tutorial/NeuralNetworksExampleWeighting.html

Data augmentation

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. The generation of new samples is based on the k-Nearest Neighbours algorithm. Resampling the numerical features is pretty straight forward, the numerical value is the distance between the data samples. 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.

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 the specification of the distribution of weights and using a fixed seed for pseudo-randomization. Using these fixed initial weights can help choosing the ideal parameter for our model during model optimization.

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 then false positives. 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

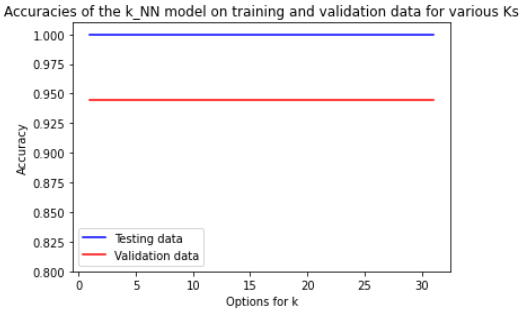
For the neural network, we have implemented a 5-fold cross-validation pipeline for better validation of our model outcomes. Instead of using one split of the training and validation data, this pipeline iteratively splits 80% of the data into 5 (stratified) folds, each time using 1-fold for validation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

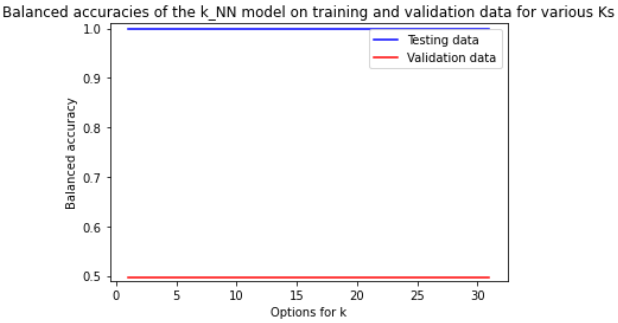
Section 3: Model Training

K-Nearest Neighbors

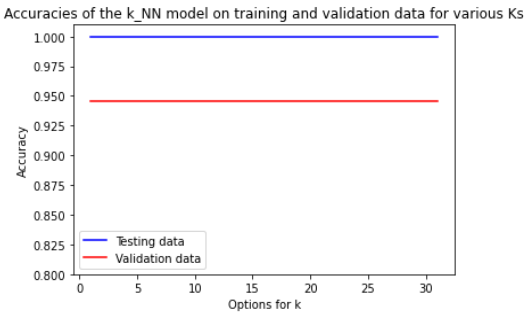
For the model training step for K-Nearest Neighbors we tried to plot the accuracies and balanced accuracies for the various options for k. We tried every value for k between 1 and 33, incremented in steps of two so that the number of neighbors would remain uneven.

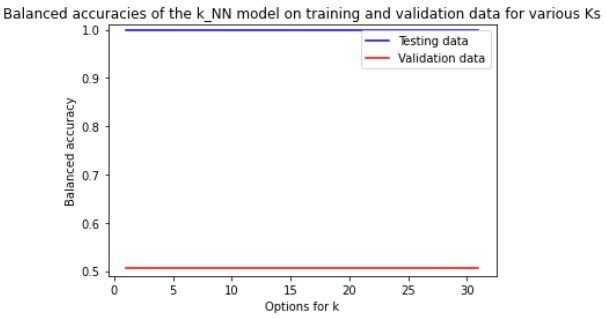
All data



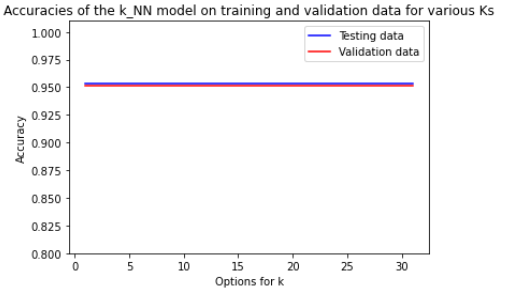


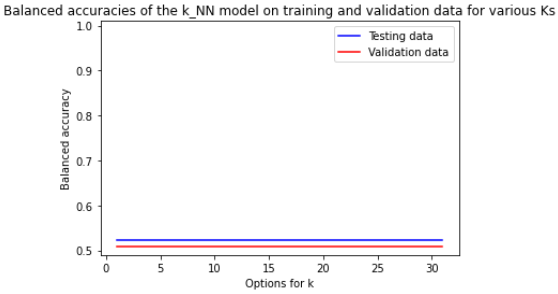
Only the numeric features





Only the categorical features



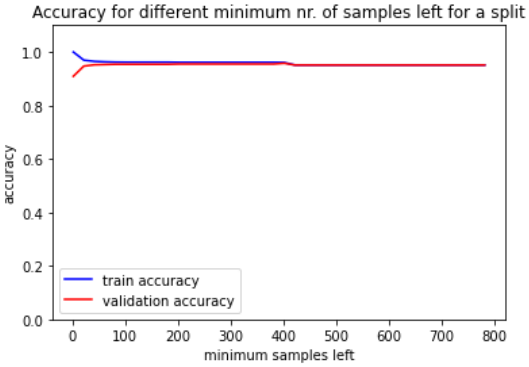


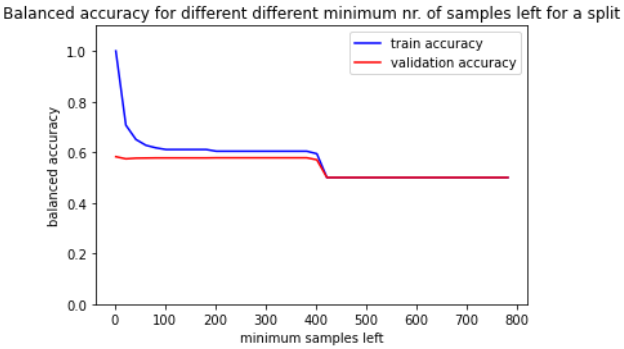
As visible in the figures above, the k did not influence the accuracies for any of the feature combinations. As the accuracies do not change, we will not be reporting the accuracies in section 4 for k-NN in this chapter; K-Nearest Neighbours is not a model that is fit to classify unbalanced, complex data like this. Therefore, we will be dropping the algorithm for the third version of our models.

Decision tree

The basic decision tree was optimized by adding a maximum depth. This can be done at different moments in the process: during and after. We tried three methods during the forming of the tree: minimal sample size, max tree depth and significance threshold and one method after creating the entire tree: pruning.

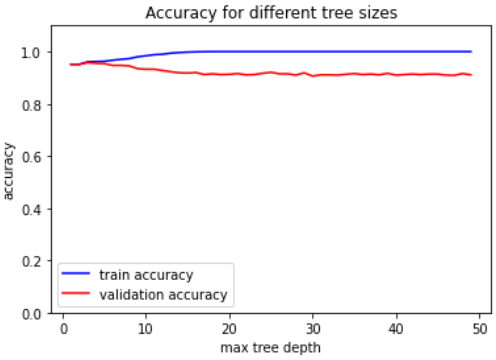
First, we tried to alter the number of samples that should be left over in a batch before it could be called a leaf node. We tried samples left between 0 and 800, as we have quite a large dataset. We plotted the (balanced) accuracy for the training and validation data for these various minimum sample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

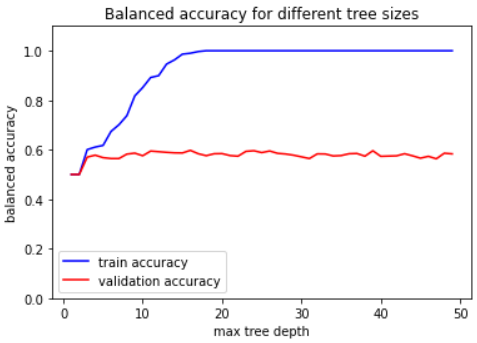




The figure above shows that 2 is the best split size for the balanced accuracy (= 0.58) for the validation data, which is a bit odd as this is basically where the model is still overfitting quite a lot, when looking at the accuracy for the training data. The balanced accuracy is also still not particularly high; it just seems to decrease for a bigger minimum number 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.

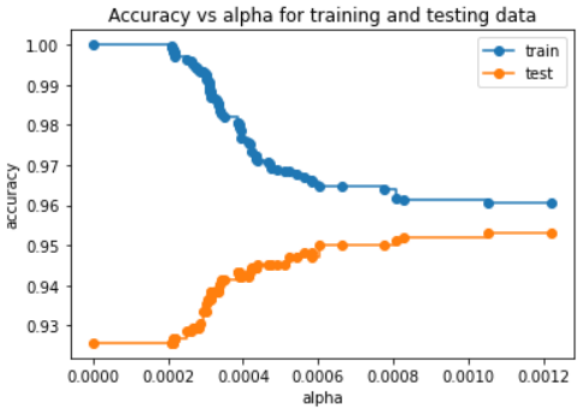
Second, we tried to limit the tree depth for the model, trying tree depths between 1 and 50.

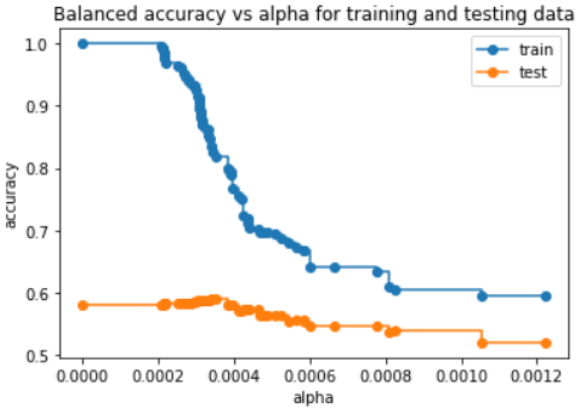




The plot and calculations showed that 16 was the optimal tree depth for a maximum balanced validation accuracy (0.597); there is however not a clear peak, as the accuracy seems to oscillate somewhat.

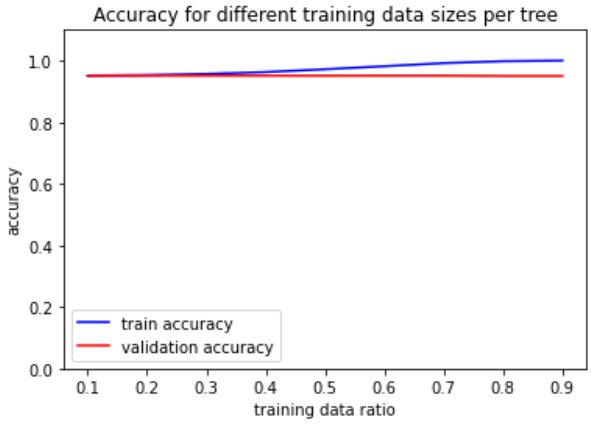
Furthermore, we tried to prune the tree after it was completed with different parameters. We used the cost complexity parameter (alpha) to measure the pruning. The higher alpha, the more leaves are pruned. An alpha of 0.007 left our tree with just one node. We plotted the accuracy and balanced accuracy (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.

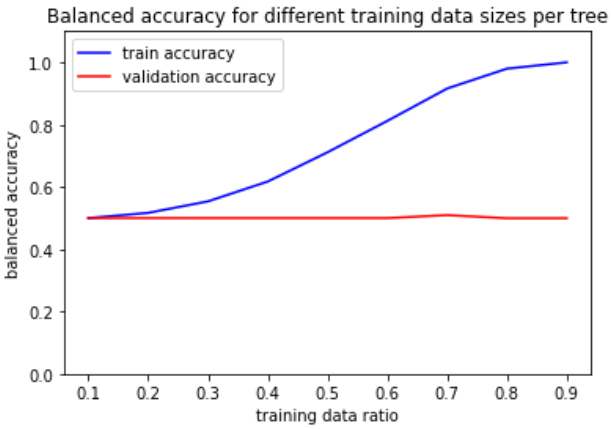




Random forest

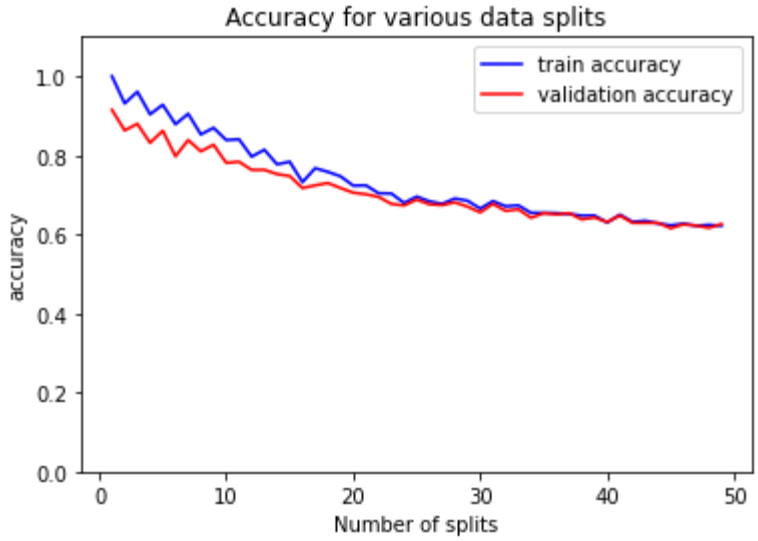
For the random forest, we plotted the accuracy of the forest on the validation data for various ratios of the training data that each tree was trained on. As seen, the training accuracy increased as the trees were trained on a larger part of the training data, but the validation accuracy did not. The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

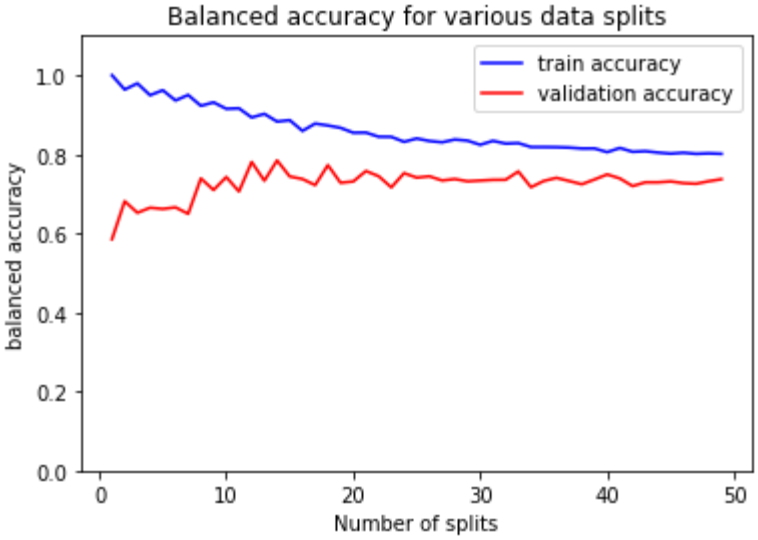




Ensemble forest

For the ensembled 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.





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

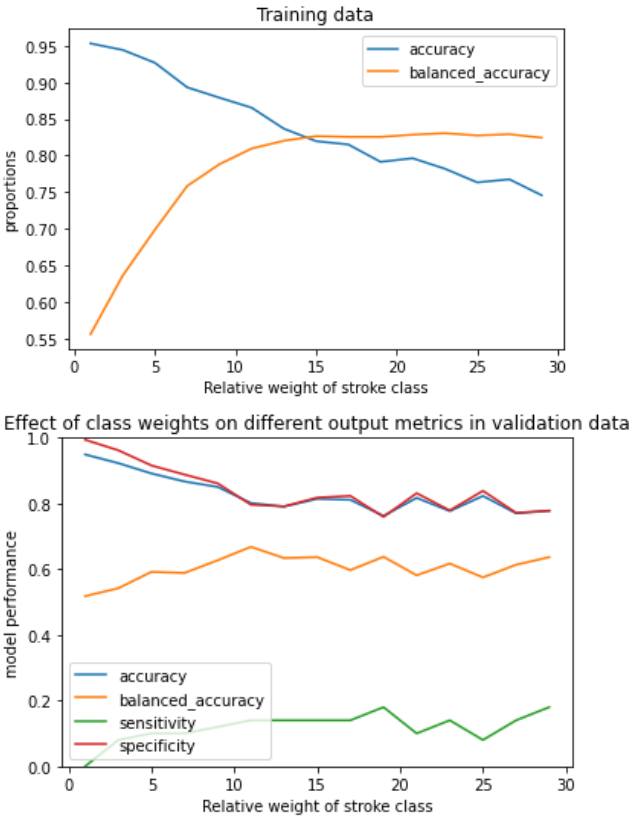
Neural networks

Class weight changes for loss function

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 around 12, 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.



Data augmentation

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.

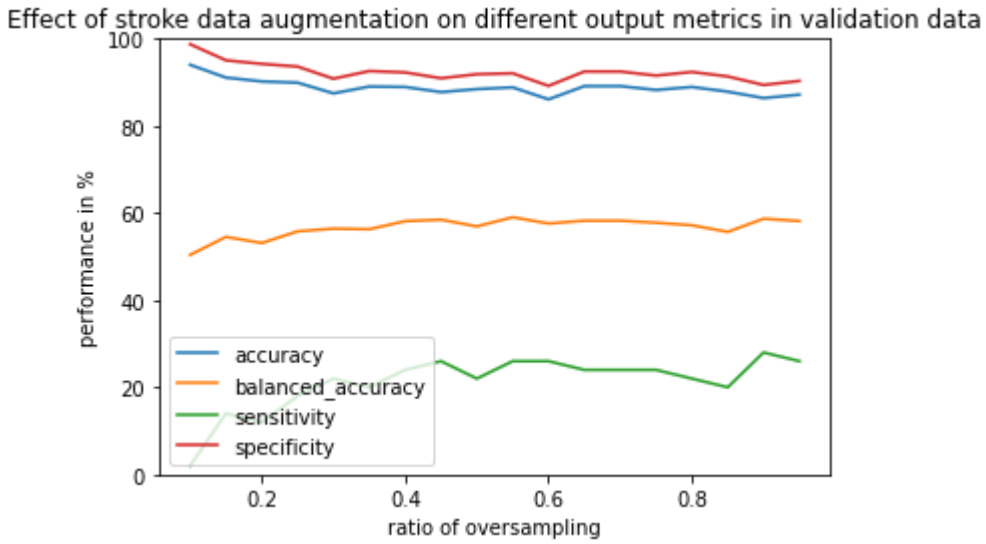


Figure 6: *Plot of the ratio of oversampling against the performance of the neural network*

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 find that the model might have 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.

Section 4: Model Evaluation

K-Nearest Neighbors

No changes, see section 4 chapter 1.

Decision tree

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

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:

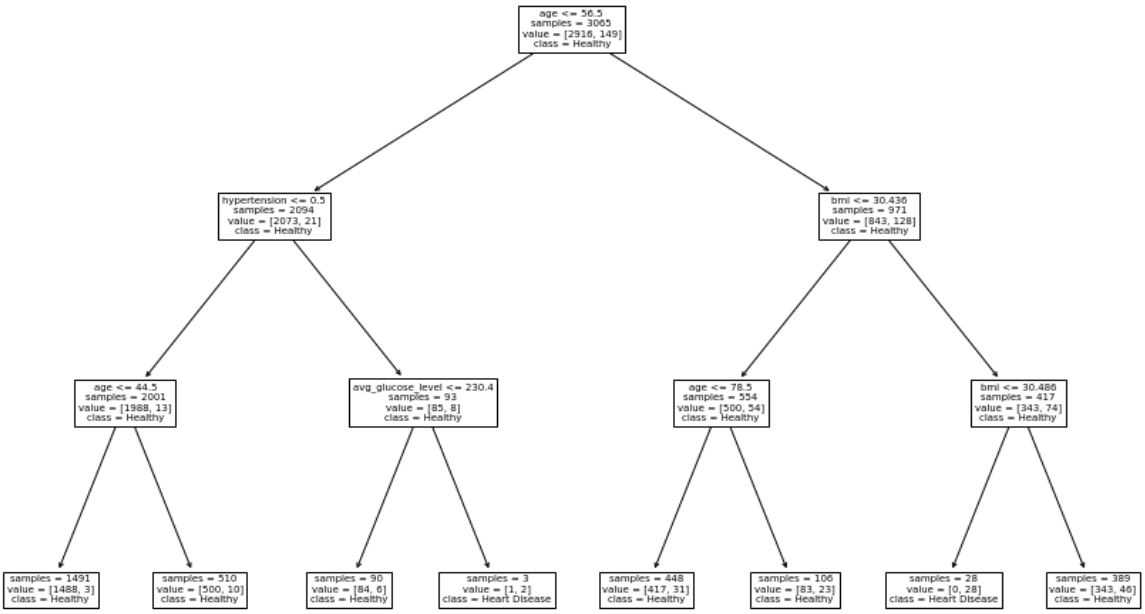
TN: 2915	FP: 1
FN: 52	TP: 97

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.



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.
However, the ensembled 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

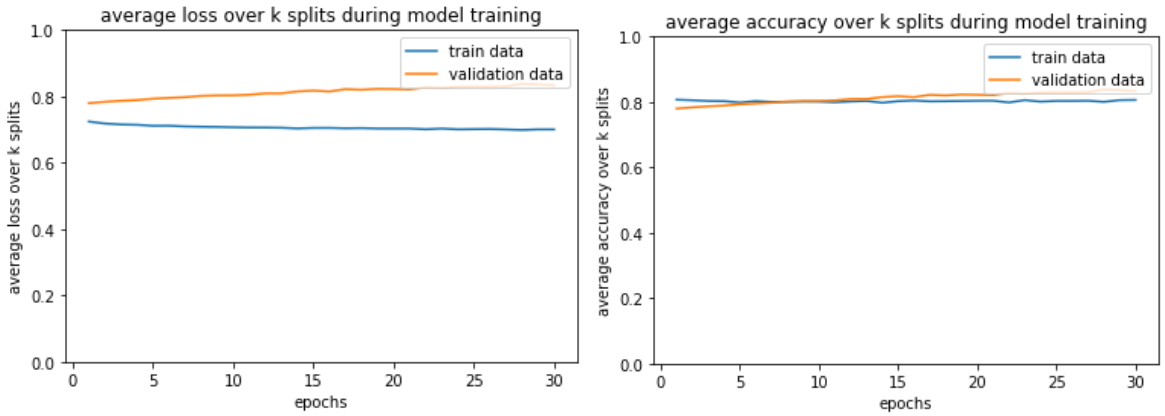
This model seems promising, and is something we should try to develop further.

Neural Network

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.

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



final average validation accuracy: 80.15%
final average validation accuracy balanced: 80.44%
final average sensitivity: 0.7747
final average specificity: 0.7450

The model performance of 80.44% 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.

Dealing with imbalanced data: Class weight and Data oversampling

Both class weight changes and stroke data augmentations 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 lead to an overshoot in false positives, which actually reduces the model performance measured by the balanced accuracy. In simpler terms, the specificity of the model is reduced too much when too much oversampling and class weights are used.
For now, we have not implemented class weight changes and data oversampling in the same model, but our next goal is to find the optimal values when using both imbalance methods to increase our balanced accuracy even further.

References

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

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

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