Assignment 2: Using Machine Learning to predict strokes

Student ID: 2047820

February 3, 2022

Contents

1 Introduction

A stroke is a medical condition in which poor blood flow to the brain. It causes parts of the brain, and therefore parts of the body, to stop functioning properly. According to the World Health Organization (WHO), stroke is the 2nd most important cause of death globally, responsible for approximately 11% of all deaths.
The objective of this assignment is to use Machine Learning to predict the occurrence of a stroke based on some parameters, such as: gender, age, previous diseases, or smoking status.
This work was carried out by utilising Random Forest, Artificial Neural Networks, and Linear Regression based algorithms.

2 Tools and libraries

The coding was done on Jupyter Notebook, which was installed locally, using Python as programming language.
Also, some of the libraries available in the Python ecosystem were used. Such as: Numpy, Pandas, Matplotlib, Seaborn, Plotly, Scikit-learn, Tensorflow and Keras.

3 Data analysis: exploration and visualisation

3.1 Dataset

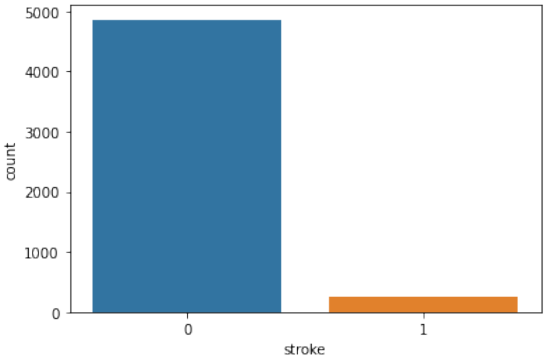
The dataset is relatively small (5110, 12). It consists of 5,110 rows, each of which describes a patient, and 12 columns which represent each of the features.
The feature 'stroke' is the class variable, to be predicted by the models.
The other 11 features describe demographic and health data for each patient.

3.2 Data types

The 11 features were classified as follows:
Categorical features: 'gender', 'hypertension', 'heart_disease', 'ever_married', 'work_type', 'Residence_type', 'smoking_status'.
Numerical features: 'age','avg_glucose_level', 'bmi'.

3.3 Analysing the class feature

The class feature 'stroke' can take values 0 or 1.
The value '1' represents that the patient had a stroke, and '0' otherwise.
These are the number of patients in each class:
Positive (1): 249 (4.87% of total)
Negative (0): 4861 (95.13% of total)
These figures reflect class imbalance.



The chart above shows that an extreme minority of the patients had a stroke.
This imbalance of the data needs to be corrected before making the predictions.

3.4 Analysing the numerical features

Statistics for the numerical features

The table below shows the average values for 'age', 'avg_glucose_level' and 'bmi', by each class of 'stroke':

Average values			
stroke	**age**	**avg_glucose_level**	**bmi**
0	42	104.80	28.82
1	68	132.54	30.47

The 'bmi' values are similar for both classes of patients, being 30.5 and 28.9 for stroke and non-stroke respectively.
The 'avg_glucose_level' values are also similar.
However, patients who suffered a stroke were older (68) than those who did not (42).
It therefore seems that 'age' is an important risk factor in suffering a stroke.

The table below shows the statistics for the numerical features:

statistics	**age**	**avg_glucose_level**	**bmi**
count	5,110	5,110	**4,909**
mean	43.23	106.15	28.89
std	22.61	45.28	7.85
min	**0.08**	55.12	10.30
25%	25.00	77.24	23.50
50%	45.00	91.88	28.10
75%	61.00	114.09	33.10
max	82.00	271.74	**97.60**

Observations:
There is an extremely high value for 'bmi' (97.6). It will be analysed further;
The feature 'bmi' contains missing values as it only presents 4,909 values instead of 5,110;
The feature 'age' contains values below 1, which probably represent babies;

Correlation matrix for the numerical features

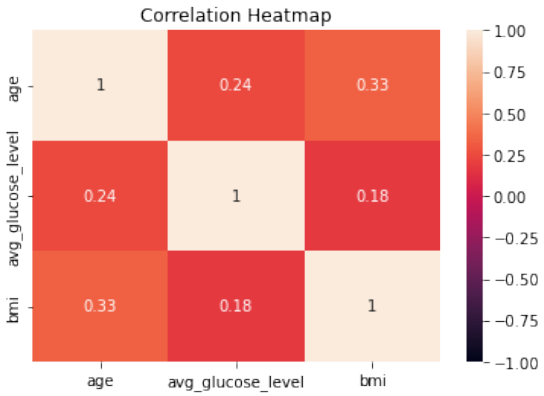
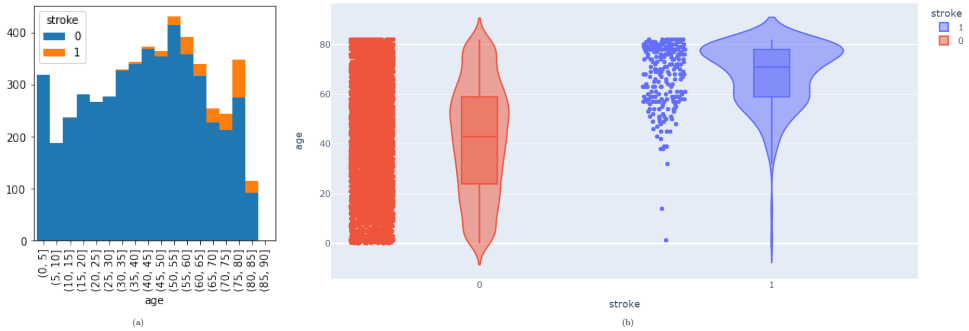


Figure 1: correlation matrix

The three numerical features are positively correlated with each other. Although their correlation is weak (less than 0.35 in all cases); 'age' and 'bmi' are the features with stronger correlation (0.33).

Plotting the numerical features
After the numerical features have been binned, they were analysed in relation to the number of patients who had, or didn't have, a stroke. The results are expressed in the bar charts below.
Violin plots also have been used to summarize the numerical features over the two categories of 'stroke'.

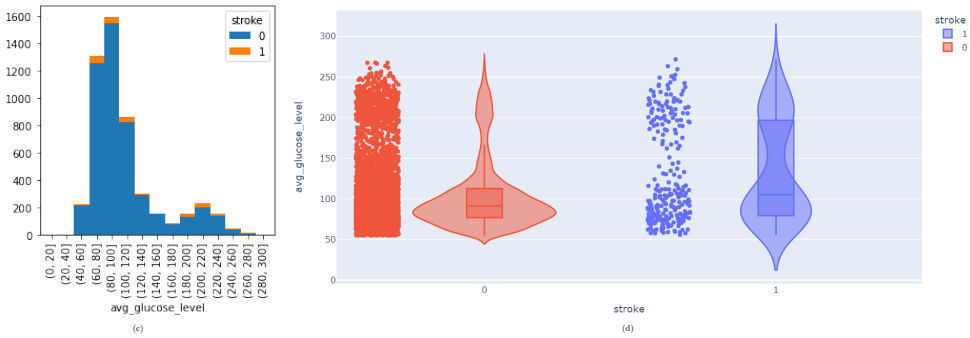
'age'



The bar chart shows that the largest number of incidences of stroke is concentrated between the ages of 75 and 80.

The violin plots also show that the median age for patients who suffered a stroke (71 years) was higher than that for patients who did not (43 years).

'avg_glucose_level'



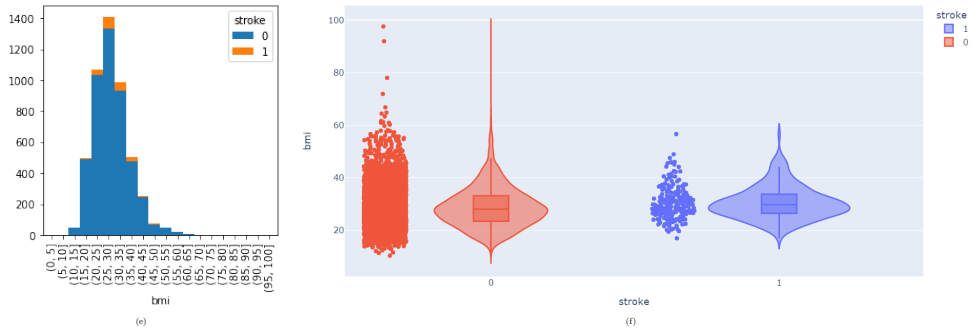
The bar chart shows that the largest number of incidences of stroke is concentrated between 60 and 120 avg_glucose_level.

The violin plots show that the median 'avg_glucose_level' for patients who suffered a stroke (105.22) was slightly higher than that for patients who did not (91.47).

The violin plots also show that the values for of 'avg_glucose_level' for patients who did not suffer a stroke were highly concentrated between 77.12 (q1) and 112.85 (q3).
In contrast, these values were more uniformly distributed for patients who suffered a stroke.

Considering levels of 'avg_glucose_level' under 140 as normal, there is not a clear relationship between this feature and the risk of suffering a stroke.

"bmi'



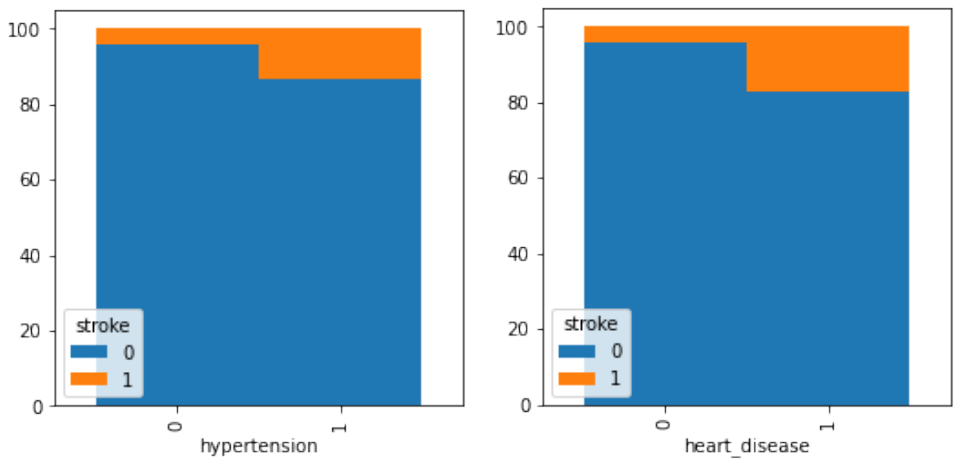
The bar chart shows that the largest number of incidences of stroke is concentrated between the values of 25 and 35, which corresponds to healthy or overweight people.

The violin plots show that the median 'bmi' for patients who suffered a stroke (29.7) was similar than that for patients who did not (28).

Therefore, has not been identified a clear relationship between having an extreme body mass index and the higher risk of suffering a stroke.

3.5 Analysing the categorical features

'hypertension' and 'heart disease'



The stacked bar charts above suggest that patients suffering from hypertension or heart disease are more likely to suffer a stroke.

Hypertension	count	avg ('age')
0	4,612	41
1	498	62

Table 1: Hypertension

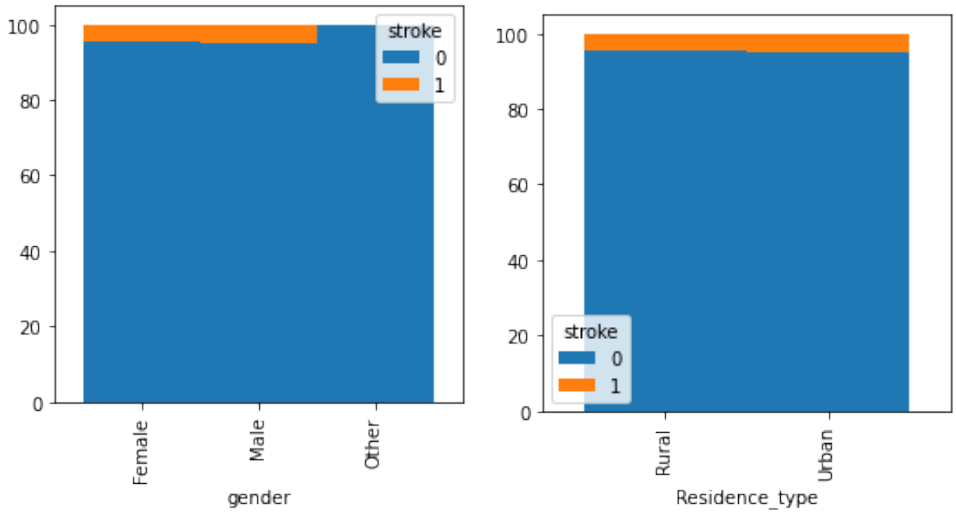
Heart_disease	count	avg ('age')
0	4,834	42
1	276	68

Table 2: Heart_disease

However, the tables above show that patients suffering from hypertension or heart disease were older (62 and 68 years old, respectively) than those who did not suffer from those diseases (41 and 42 years old, respectively).

Thus, 'hypertension' and 'heart disease' do not have direct influence on suffering a stroke. However, the 'age' does.

'gender' and 'residence_type'



The bar charts suggest that the number of patients suffering a stroke were similar between men and women, and also between people living in rural and urban locations.

Gender	count	avg ('age')
Female	2,994	44
Male	2,115	43
Other	1	26

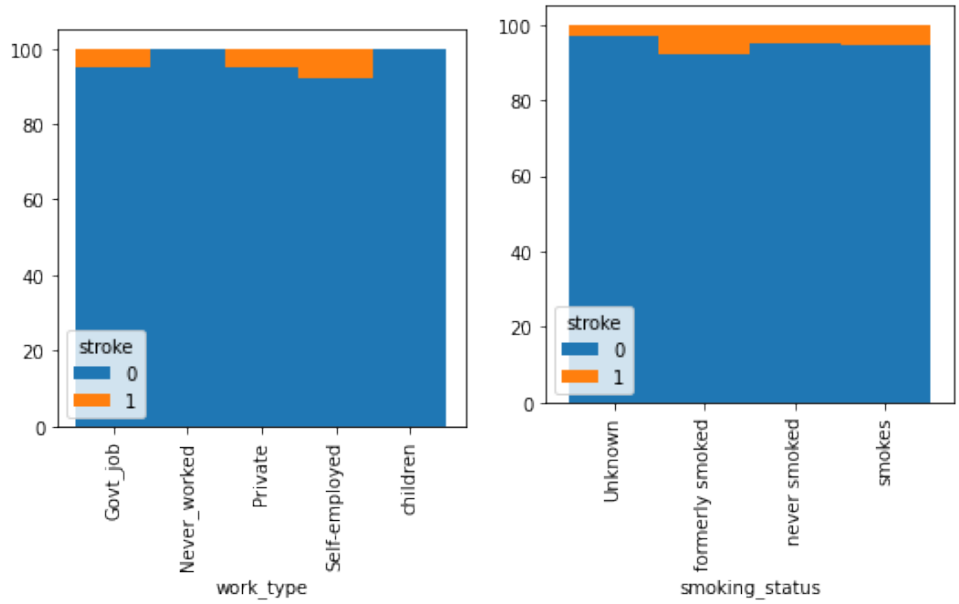
Table 3: Gender

Residence_type	count	avg ('age')
Rural	2,514	43
Urban	2,596	44

Table 4: Residence_type

However, the tables above show that men and women in the dataset are almost the same age. Also, patients living in rural and urban areas are almost the same age.
Thus, 'gender' and 'residence_type' do not have direct influence on suffering a stroke, but 'age' does.

'work_type' and 'smoking_status'



'Smoking' and 'work_type' are other features associated with 'age'. Patients more likely to suffer a stroke were those who formerly smoked and who worked as self-employed, who also result to be the oldest (55 and 60 years old respectively, on average).

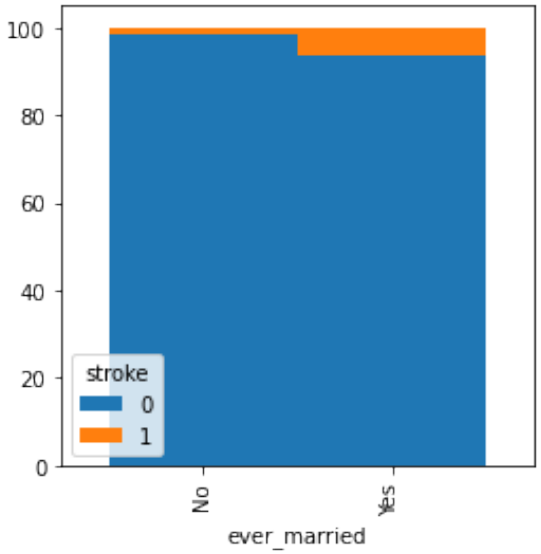
work_type	count	avg ('age')
Govt_job	657	51
Never_worked	22	16
Private	2925	46
Self-employed	819	60
children	687	7

Table 5: Work type

smoking_status	count	avg ('age')
Unknown	1544	30
formerly smoked	885	55
never smoked	1892	47
smokes	789	47

Table 6: Smoking status

'ever_married'



'ever_married' is another feature associated with 'age'.

ever_married	count	avg ('age')
No	1,757	22
Yes	3,353	54

Patients more likely to suffer a stroke were those who were married, who resulted to be the oldest (54 years old, on average).

3.6 Conclusion

As a conclusion, it seems that 'age' is the most important risk factor in suffering a stroke.

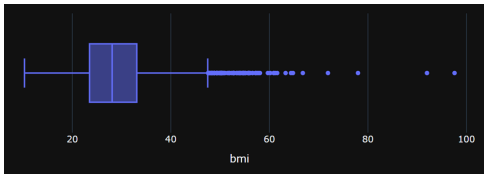
4 Preprocessing

4.1 'id' feature

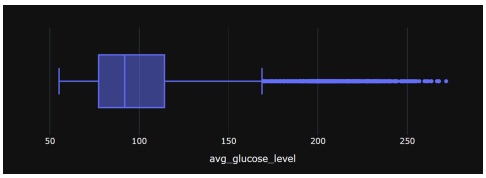
The feature 'id' was dropped as it does not provide any meaningful information related to the patients.

4.2 Extreme values

Several outliers are present in 'bmi' and 'avg_glucose_level', as shown on the boxplots below:



(m)



(n)

These outliers make the distribution highly skewed towards right.
Although they could be dropped, it will lead to loss data of patients who actually had a stroke.
The extreme value of 97.6 for 'bmi' has been analysed further. It corresponds to a young man who did not had a stroke and whose 'smoking_status' was unknown. As this data point is not making any improvement to the model, it was drooped.

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
2128	56420	Male	17.0	1	0	No	Private	Rural	61.67	97.6	Unknown	0

Figure 2: bmi =97.6

4.3 Missing values

'bmi' was the only feature containing missing values.

```
bmi                  201
gender                 0
age                    0
hypertension           0
heart_disease          0
ever_married           0
work_type              0
Residence_type         0
avg_glucose_level      0
smoking_status         0
stroke                 0
dtype: int64
Number of patients who had a stroke and missing bmi values: 40
```

Figure 3: Number of missing values

The number of patients who had a stroke and present missing bmi values was 40, which is a high number considering that only 249 patients had a stroke. Hence, for not losing this information, the missing values for 'bmi' have been replaced for the mean of that feature.

4.4 'age' feature containing decimals

'age' contains decimal places for young patients. Therefore, this column has been rounded and changed to integer, to be consistent with the rest of the values within the dataset.

4.5 Undetermined categories

These are the unique values for the categorical features:

```
['Male' 'Female' 'Other']
[0 1]
[1 0]
['Yes' 'No']
['Private' 'Self-employed' 'Govt_job' 'children' 'Never_worked']
['Urban' 'Rural']
['formerly smoked' 'never smoked' 'smokes' 'Unknown']
```

Figure 4: unique values

These are the undetermined values found for 'smoking_status' and 'gender':

```
#counting undetermined values
print(data['smoking_status'].value_counts())
print(data['gender'].value_counts())
```

```
never smoked       1892
Unknown            1543
formerly smoked     885
smokes              789
Name: smoking_status, dtype: int64
Female    2994
Male      2114
Other        1
Name: gender, dtype: int64
```

Figure 5: undetermined values

Regarding the feature 'gender', only one patient was identified as 'other'. As this single person did not have a stroke, they have been removed from further analysis.

```
Female    2994
Male      2114
Other        1
```

Figure 6: other gender

In contrast, 'smoking_status', included a category called 'unknown'. Due to there were 46 patients in this category who had a stroke, they cannot be removed from consideration.

```
never smoked       1892
Unknown            1543
formerly smoked     885
smokes              789
```

Figure 7: 'smoking_status containing 'unknown' values

These values were imputed by their mode, resulting this distribution:

```
print(data['smoking_status'].value_counts())
```

```
never smoked       3435
formerly smoked     884
smokes              789
```

Figure 8: 'smoking_status without 'unknown' values

4.6 Scaling numerical features

StandardScaler was used. It was noticed that it produced some negative values within the dataframe as a result of removing the mean and then scaling to unit variance. This is how the data looks like after scaling:

	age	avg_glucose_level	bmi
0	1.051047	2.706248	1.010567
1	0.785685	2.121452	0.002445
2	1.625996	-0.005060	0.473775
3	0.254963	1.437276	0.722532
4	1.581769	1.501100	-0.639087

Figure 9: showing first 5 rows after scaling

4.7 Encoding categorical features

Label encoding was performed to convert the string values of the binary features into integers.

```
#Label encoding for binary features
le=LabelEncoder()
gender=le.fit_transform(data['gender'])
Residence_type=le.fit_transform(data['Residence_type'])
ever_married=le.fit_transform(data['ever_married'])
```

Figure 10: Label encoding

However, one-hot encoding was performed to convert the string values of the features that contain more than two categories, such as 'work_type' and 'smoking_status'.

```
#One-hot encoding for multicategory features
ohe = OneHotEncoder()

data['smoking_status'] = pd.Categorical(data['smoking_status'])
dataDummies_smoking_status = pd.get_dummies(data['smoking_status'], prefix = 'smoking_status_encoded')
print(dataDummies_smoking_status)

data['work_type'] = pd.Categorical(data['work_type'])
dataDummies_work_type = pd.get_dummies(data['work_type'], prefix = 'work_type_encoded')
print(dataDummies_work_type)

data.drop("work_type", axis=1, inplace=True)
data.drop("smoking_status", axis=1, inplace=True)

data = pd.concat([data, dataDummies_work_type], axis=1)
data = pd.concat([data, dataDummies_smoking_status], axis=1)
print(data.head())
```

Figure 11: One hot encoding

4.8 Encoding summary

This is a summary of the type of encoding applied to each feature:

Categorical features	Dtype	Unique_values	Action	Comments
gender	Object	2	Label Encoding	Replacing the categories of each feature with dummy numbers (0,1).
Residence_type	Object			
ever_married	Object			
work_type	Object	5	One-Hot-Encoding	The feature contains more than two categories, Label Encoding might not be intuitive.
smoking_status	Object	4	One-Hot-Encoding	
stroke	Integer	2	Not required	
Hypertension	Integer	2	Not required	
heart_disease	Integer	2	Not required	

Figure 12: encoding type

```
 #   Column                                 Non-Null Count  Dtype
---  ------                                 --------------  -----
 0   gender                                 5108 non-null   int32
 1   age                                    5108 non-null   int32
 2   hypertension                           5108 non-null   int64
 3   heart_disease                          5108 non-null   int64
 4   ever_married                           5108 non-null   int32
 5   Residence_type                         5108 non-null   int32
 6   avg_glucose_level                      5108 non-null   float64
 7   bmi                                    5108 non-null   float64
 8   stroke                                 5108 non-null   int64
 9   work_type_encoded_Govt_job             5108 non-null   uint8
 10  work_type_encoded_Never_worked         5108 non-null   uint8
 11  work_type_encoded_Private              5108 non-null   uint8
 12  work_type_encoded_Self-employed        5108 non-null   uint8
 13  work_type_encoded_children             5108 non-null   uint8
 14  smoking_status_encoded_formerly smoked 5108 non-null   uint8
 15  smoking_status_encoded_never smoked    5108 non-null   uint8
 16  smoking_status_encoded_smokes          5108 non-null   uint8
```

Figure 13: data types after encoding

The table above contains the data types which were obtained for each encoded feature. As can be seen, the categorical features have been processed.

4.9 Splitting the dataset into features matrix and output array

```
#Splitting the dataset
y = data['stroke'] #output
X = data.drop('stroke', axis=1) #features
print(X.shape, y.shape)
```

```
X:  (5108, 16)
y:  (5108,)
```

Figure 14: X and y

4.10 Creating training and testing random subsets

After preprocessing the dataset, it was split into 'train' (80%) and 'test' (20%) subsets.

```
#Creating training and testing datasets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size= 0.2, random_state= 0)
print("X_train: ", X_train.shape)
print("y_train: ", y_train.shape)
print("X_test: ", X_test.shape)
print("y_test: ", y_test.shape)
```

```
X_train:  (4086, 16)
y_train:  (4086,)
X_test:  (1022, 16)
y_test:  (1022,)
```

Figure 15: 'train' and 'test' sets

4.11 Handling imbalanced data

SMOTETomek, a combination of oversampling and undersampling techniques, SMOTE and Tomek links respectively, was applied to the training set.

These are the training samples for each category of 'stroke' before and after applying this technique:
Before SMOTETomek:
stroke = 1: 195
stroke = 0: 3891

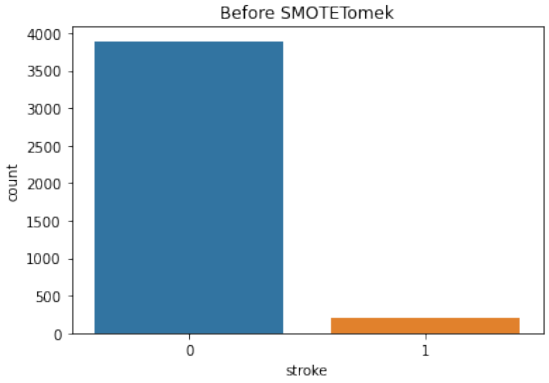


Figure 16: Before SMOTETomek

After SMOTETomek:
stroke = 1: 3883
stroke = 0: 3883

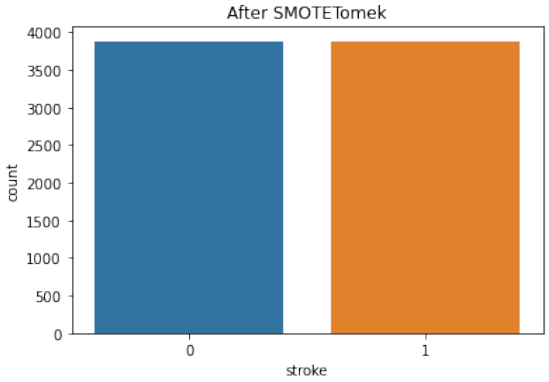


Figure 17: After SMOTETomek

5 Model I: Random Forest

5.1 Implementing a model using the default parameters

Firstly, a model was implemented by using the default parameters, which are shown below:

```
{'bootstrap': True, 'ccp_alpha': 0.0, 'class_weight': None, 'criterion': 'gini', 'max_depth': None,
'max_features': 'auto', 'max_leaf_nodes': None, 'max_samples': None, 'min_impurity_decrease': 0.0, 'm
in_impurity_split': None, 'min_samples_leaf': 1, 'min_samples_split': 2, 'min_weight_fraction_leaf':
0.0, 'n_estimators': 100, 'n_jobs': None, 'oob_score': False, 'random_state': 0, 'verbose': 0, 'warm_
start': False}
```

Figure 18: Parameters by default

These were the evaluation metrics obtained:
Recall: 15%
Accuracy: 89.63%
The confusion matrix shows that this initial model produced 916 correct classifications, whereas 106 patients of one class were incorrectly classified into another class.

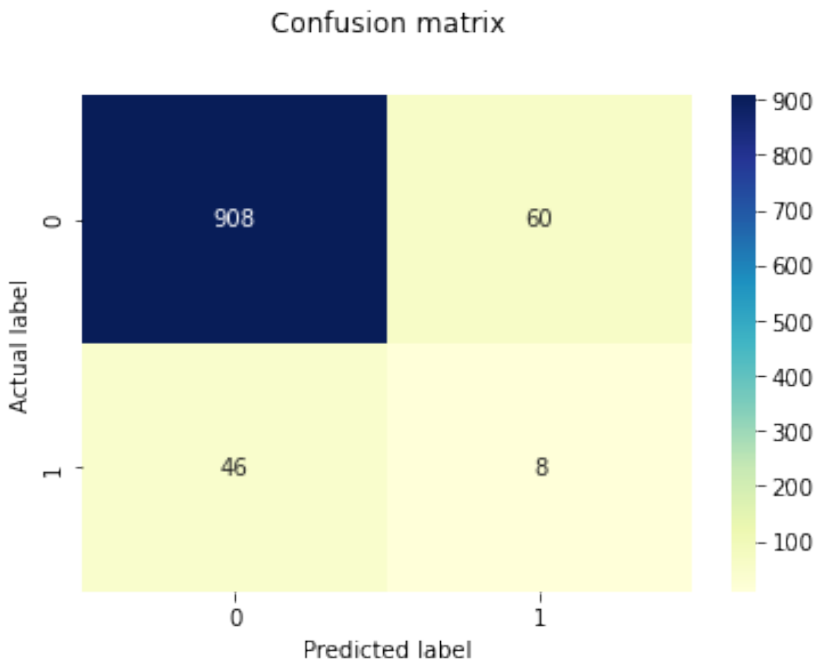


Figure 19: Random Forest confusion matrix

When predicting strokes, inaccurate predictions could have drastic consequences. Hence, an actual 'stroke' patient should not be labelled as a 'non-stroke' because that patient could die if they do not receive treatment (False negative predictions).

However, if a non-stroke person is labelled as 'stroke', the consequences are not so serious (False positive predictions). Thus, this problem requires a model with a high recall, to avoid the false negatives. However, accuracy is not a good evaluation metric.
As the results present good accuracy but poor recall, I have tuned the model by using GridSearchCV technique to improve the values for recall.
However, I had a look at the feature importance first.

5.2 Feature importance and selection

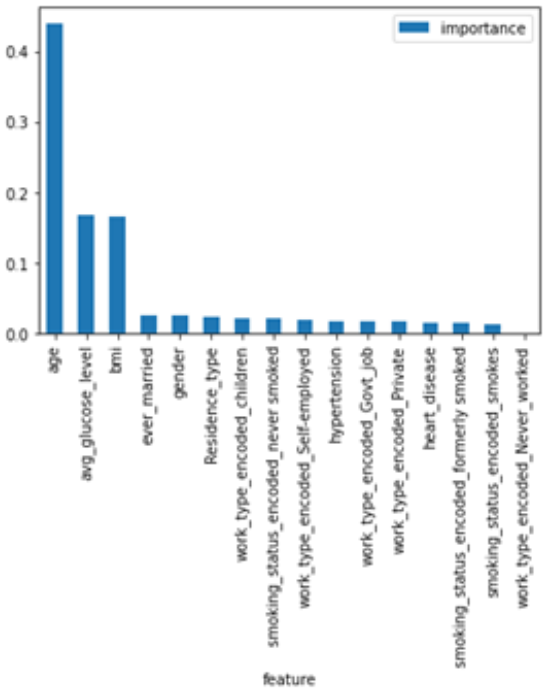


Figure 20: Feature importance

The feature used in the top split ('age') is by far the most important feature, followed by 'avg_glucose_level' and 'bmi'.
These three features seem to be the most helpful ones for predicting if a person will have a stroke or not.
Therefore, the most insignificant feature with zero importance ('work_type_encoded_Never_worked'), was dropped and the model was trained again, which gave these results:
Recall: 9.26%;
Accuracy: 89.82%;
F1 score: 8.77%;

As the recall was low, another model was trained keeping the 3 most important features: 'age', 'avg_glucose_level', and 'bmi'. These were the results obtained:
Recall: 53.70%;
Accuracy: 84.54%;
F1 score: 26.85%;

As the recall increased significantly, these 3 features were considered for further analysis.

5.3 Tuning the hyperparameters of the model

The objective is to minimize false negatives while maximizing the true positives, so the recall is higher.
I have used GridSearch with 3-fold cross-validation, which tries all possible combinations of the parameters defined and uses cross-validation to evaluate the model.

Parameters:
The parameters to be tested were 'n_estimators', which refers to the number of trees in the forest, 'max_depth', which refers to the maximum depth of the tree, and the 'criterion', which measures the quality of a split.

```
parameters = {
  'n_estimators' : [10,50,100,150,200,250,300,350,400,450,500],
  'max_depth': [1,5,10,11,12,15,20],
  'criterion':['gini', 'entropy']
```

Figure 21: Parameters tested

The model was also trained by modifying other parameters such as 'bootstrap', 'min_samples_split' or 'min_samples_leaf'. However, they performed better with their default value:

```
#parameters not to modify, same results
#  'bootstrap': [True, False]
#  'min_samples_split' : [2, 5, 10], # Minimum number of samples required to split a node
#  'min_samples_leaf' : [1, 2, 4] # Minimum number of samples required at each leaf node
```

Figure 22: Parameters not modified

The GridSearchcv scorer was configured in the following way, in order to use the recall as the evaluation metric:

```
scorer = make_scorer(recall_score,average='binary', pos_label=1)#to maximize recall for class1
```

Figure 23: Scorer

These were the best values obtained for the parameters which were tested:

```
Fitting 3 folds for each of 154 candidates, totalling 462 fits
Best parameters: {'criterion': 'entropy', 'max_depth': 12, 'n_estimators': 450}
Best estimator: RandomForestClassifier(criterion='entropy', max_depth=12, n_estimators=450,
                       random_state=42)
```

Figure 24: Best parameters

5.4 Evaluating the optimal model

Recall: 72.22%
Accuracy: 79.26%;
Precision: 16.5%;

Confusion Matrix:

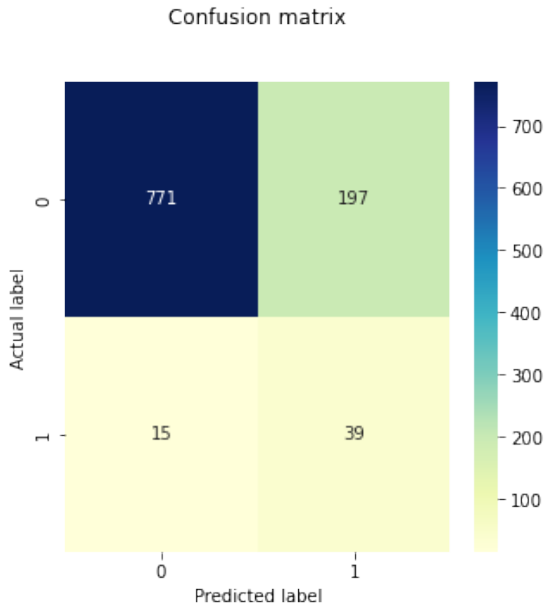


Figure 25: Confusion matrix for the tuned model

The tuned model has correctly predicted 810 cases (79.26% of the total). However, 15 potentially ill patients were incorrectly predicted as 'no-stroke'.
The number of false negatives has decreased from 46 to 15.
Recall has increased from 15% to 72.22%, which is a huge improvement.
Finally, the accuracy of the model is nearly 80%.

The following table summarizes the variations made in the Random Forest model until the optimal model was obtained:

Random Forest Summary	Parameters	Features	Recall (%)	Accuracy(%)
Initial model	Default	All	15.00%	89.63%
Dropped feature importance zero	Default	Dropped the feature with importance = 0.00 'work_type_encoded_Never_worked'	9.26%	89.82%
3 most important features	Default	Only 3 most important features: 'age', 'bmi', 'avg_glucose_level'	53.70%	84.54%
Optimized model	Gridsearchcv: 'criterion' =entropy; 'max_depth' = 12; 'n_estimators' = 450	Only 3 most important features: 'age', 'bmi', 'avg_glucose_level'	72.22%	79.26%

Figure 26: Random Forest Summary

6 Model II: Artificial Neural Network

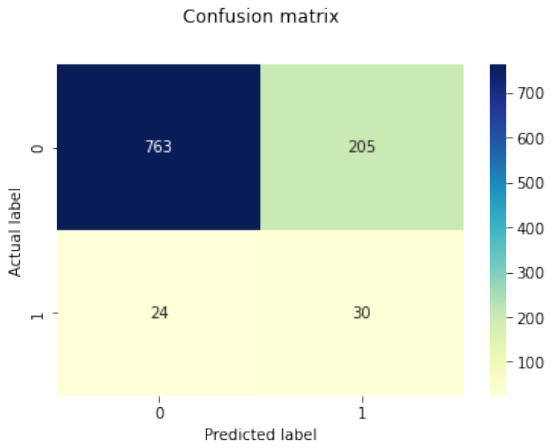
I chose to implement artificial neural networks (ANN), instead of convolutional neural networks, because they are the best type of neural network that fits tabular data.

6.1 No-hidden-layer neural network (implementation and evaluation)

Firstly, I chose a basic architecture, using the sequential API, without any hidden layers. It consisted of one single layer with one node, corresponding to the output node.
These were its characteristics:
Input features: 16
Number of hidden layers: none
Number of nodes in the output layer:1, as it is a binary classification problem
Activation for the output layer: 'sigmoid', as it is a binary classification problem
Optimizer: 'adam', with its default learning rate
Loss:'binary crossentropy', as it is a binary classification problem
Batch size: default (32)

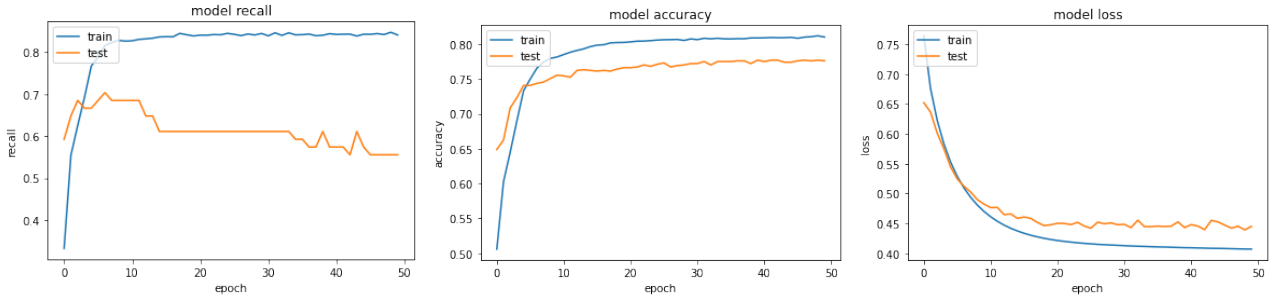
```
ann = Sequential()
ann.add(Dense(1, input_shape=(X_train_smtom.shape[-1], ), activation='sigmoid'))
```

This model was trained for 50 epochs and obtained these evaluation metrics:
Recall: 55.56%; Accuracy: 77.59%; Precision: 12.77%; False negative predictions: 24



The confusion matrix shows that this model correctly predicted 793 cases (77.59% of the total). However, 24 potentially ill patients were incorrectly classified as 'no-stroke'.

The graphs below illustrate the effect of the number of epochs on the performance metrics for both 'train' and 'test' datasets.



(a) (b) (c)

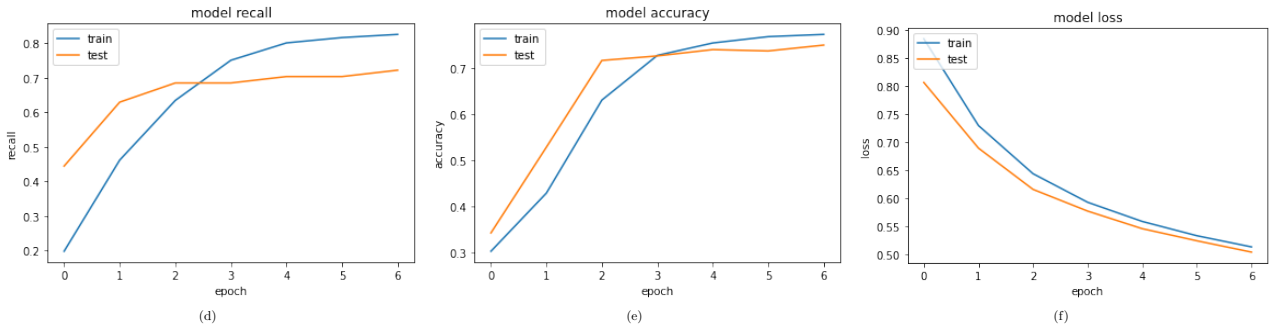
The graph (a) for **'recall'** shows a marked contrast between 'train' and 'test' curves. Both values rise sharply to epoch 7. At that epoch, the 'test' curve reaches a maximum value of 70.37%, then starts decreasing. At the epoch 15, it levels off. However, the 'train' curve keeps increasing slightly.

The graph (b) for **'accuracy'** shows a similar pattern where both curves increase sharply to epoch 7. Then, they both continue rising slightly for the rest of the epoch values. Note that the pattern for 'train' is broadly similar in the two first graphs, whereas there are major differences in the two patterns for 'test'.

The graph (c) for **'loss'** shows that the model is a good fit for the first 7 epochs. After that, the gap between both curves starts increasing, which suggests that the performance of the model gets worse.

This analysis led to reassessment of the original model. However, this time only 7 epochs were used instead of 50. These were the results obtained:
Accuracy: 74.95%; Precision: 13.93%; Recall: 72.22%; False Negative predictions: 15.

The graphs below illustrate the effect of the number of epochs on the performance metrics for both 'train' and 'test' datasets.



(d) (e) (f)

The graph (d) for **'recall'** shows that the model performs better on the 'test' dataset for the first two epochs. After that the 'test' curve levels off and the model performs better on the 'train' dataset.

The graph (e) for **'accuracy'** shows a similar pattern. The model performs better on the 'test' dataset for the first 3 epochs. However, after epoch 3 the 'train' curve continuous to increase, from there, the model performs better on the 'train' set.

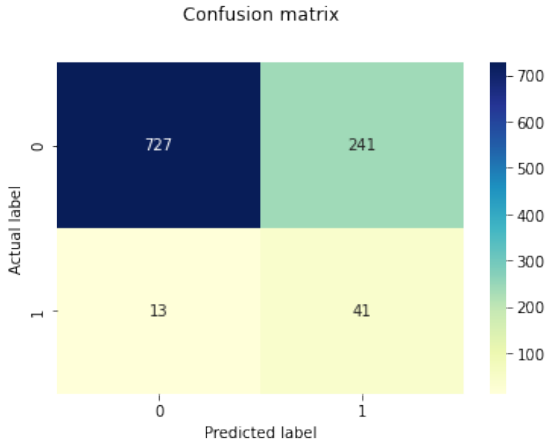
The graph (f) for **'loss'** shows that 'train' loss and 'test' loss are relatively closed from each other, overall, after epoch 3.

Although these graphs may suggest that there is room for improvement by training the model for 3 epochs, I decided to optimize the model already obtained.

6.2 No-hidden-layer neural network (learning rate optimization)

As this model did not have any hidden layers, it was not possible to optimize its neurons or to add dropout layers after the hidden ones. Instead, the recall was increased by using different learning rates for the 'adam' optimizer. The best value found was lr=0.0008.

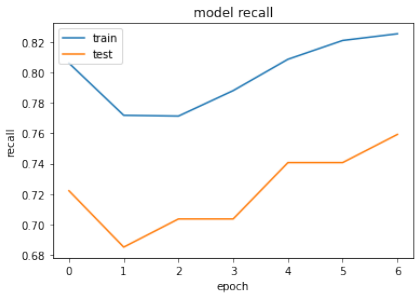
The no-hidden-layer model was trained using 7 epochs and the optimized learning rate as hyperparameters. It gave these results:
Accuracy: 75.15%; Precision: 14.54%; Recall: 75.93%; False negative predictions: 13



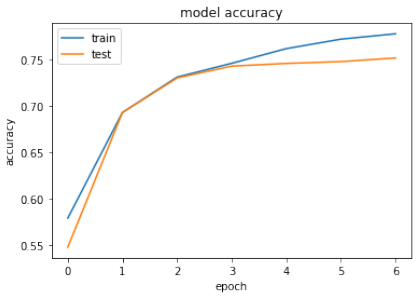
The confusion matrix shows that this model correctly predicted 768 cases (75.15% of the total). However, 13 potentially ill patients were incorrectly classified as 'no-stroke'.

The graphs below illustrate the effect of the number of epochs on the performance metrics for both 'train' and 'test' datasets.

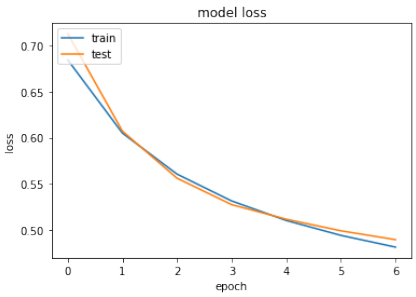
The graph (i) for **'loss'** shows that both curves, representing the 'train' and 'test' datasets, are close to each other, and follow the same trajectory.
This suggests that the model is optimal and the learning rate applied is good (not low and not high).



(g)



(h)



(i)

6.3 No-hidden-layer neural network (conclusion)

Reducing the number of epochs from 50 to 7 and optimizing the learning rate improved the performance of the initial model.
Therefore, there was a substantial improvement for 'recall' from 55.56% to 75.93%.
The following table summarizes the metrics obtained so far:

	Basic model	**Optimizing number of epochs**	**Optimizing learning rate**
Number hidden layers	0	0	0
Epochs	50	**7**	7
Learning rate	0.001	0.001	**0.0008**
Recall	55.56%	72.22%	75.93%
Accuracy	77.59%	74.95%	75.15%
Precision	12.77%	13.93%	14.54%
False negatives	24	15	13

In an effort to obtain a better performance, I built other three artificial neural networks, deeper than the basic one. These were their configurations and results:

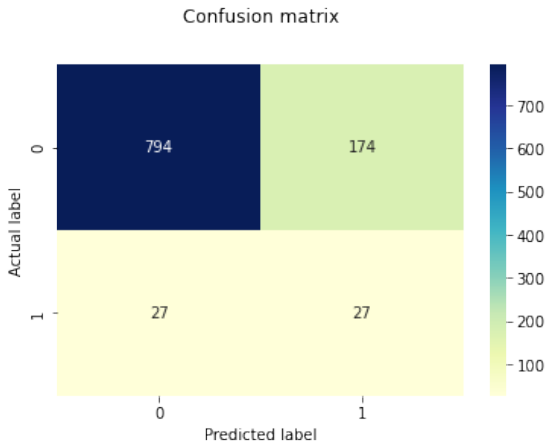
6.4 One-hidden-layer neural network (implementation and evaluation)

Input features:16
Number of hidden layers: one
Number of nodes in the hidden layers: 8, half of the input size (16)
Number of nodes in the output layer: 1, as it is a binary classification problem
Activation for the hidden layer: 'relu' activation function
Activation for the output layer: 'sigmoid', as it is a binary classification problem
Optimizer: 'adam', with its default learning rate
Loss:'binary crossentropy', as it is a binary classification problem
Batch size: default (32)

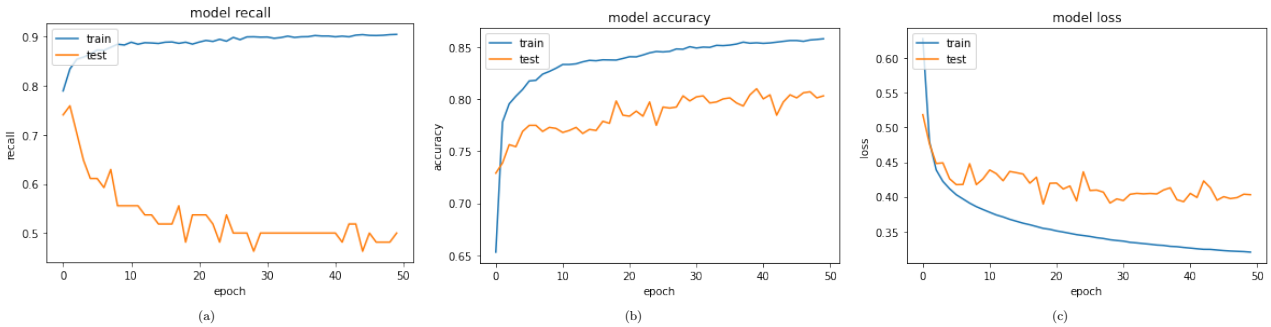
```
ann = Sequential()
ann.add(Dense(8, input_shape=(X_train_smtom.shape[-1], ), activation='relu'))
ann.add(Dense(1, activation='sigmoid'))
```

Figure 27: One-hidden-layer model

This model was trained for 50 epochs and obtained these evaluation metrics:
Accuracy: 80.33%; Precision: 13.43%; Recall: 50.00%; False negative predictions: 27



The confusion matrix shows that this model correctly predicted 821 cases (75.15% of the total). However, 27 potentially ill patients were incorrectly classified as 'no-stroke'. The graphs below illustrate the effect of the number of epochs on the performance metrics for both 'train' and 'test' datasets.



The graph (a) for **'recall'** shows a marked contrast between the 'train' and 'test' curves. Although both curves increase sharply until epoch 2, only 'train' continues increasing slightly after that. In contrast, the 'test' curve decreases sharply until epoch 9. Then it levels off with some fluctuations. This suggests that the model is overfitting.

The graph (b) for **'accuracy'** shows a different pattern. Both datasets increased sharply to epoch 2. Then, continue increasing slightly for the rest of the epoch values. However, the values for 'train' were always higher which also suggests that the model is overfitting. As expected, the graph for **'loss'** shows higher values for 'test' than for 'train', which suggests that the model learnt very well the training dataset that it negatively impacts on the performance of not seen data, such as the 'test' dataset.

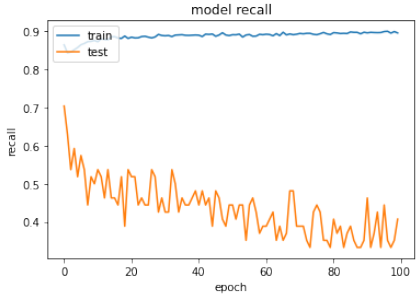
6.5 One-hidden-layer neural network (optimization)

GridSearchcv was used to find the optimal batch size and number of epochs. However, using the recall as a scorer was not the best approach because it generated some 'NAN' values. Therefore, the accuracy was used as scorer for calculating the best hyperparameters.

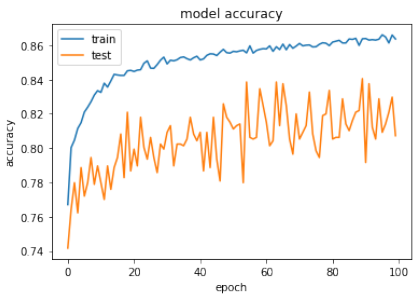
These were the hyperparameters obtained: "Best: 'batch_size': 10, 'epochs':100"

By using these hyperparameters, the accuracy improved slightly to 80.72%. However, the recall decreased from 50% to 40.74%. It did not work well for improving the 'recall'.

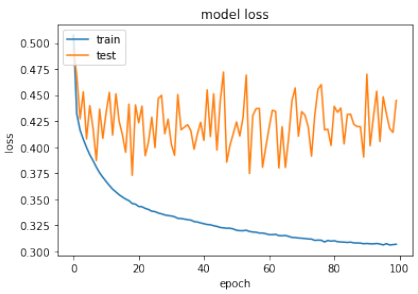
The graphs below illustrate the effect of the number of epochs on the performance metrics for both 'train' and 'test' datasets.
The graphs also suggest overfitting and also noisy movements in the 'test' dataset.



(d)



(e)



(f)

6.6 Two-hidden-layer neural network (implementation and evaluation)

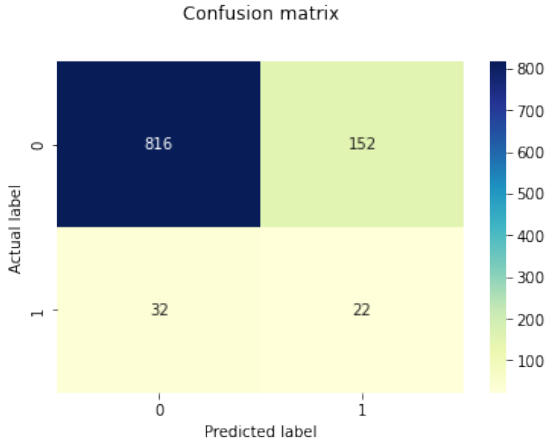
Then, I built another model, deeper than the others.
Input features:16
Number of hidden layers: two
Number of nodes in the hidden layers: the first hidden layer contains 4 neurons and the second one 2.
Number of nodes in the output layer: 1, as it is a binary classification problem
Activation for the hidden layers: 'relu' activation function

Activation for the output layer: 'sigmoid', as it is a binary classification problem
Optimizer: 'adam', with its default learning rate
Loss: 'binary crossentropy', as it is a binary classification problem
Batch size: default (32)

```
ann = Sequential()
ann.add(Dense(4, input_shape=(X_train_smtom.shape[-1], ), activation='relu'))
ann.add(Dense(2, activation='relu'))
ann.add(Dense(1, activation='sigmoid'))
```

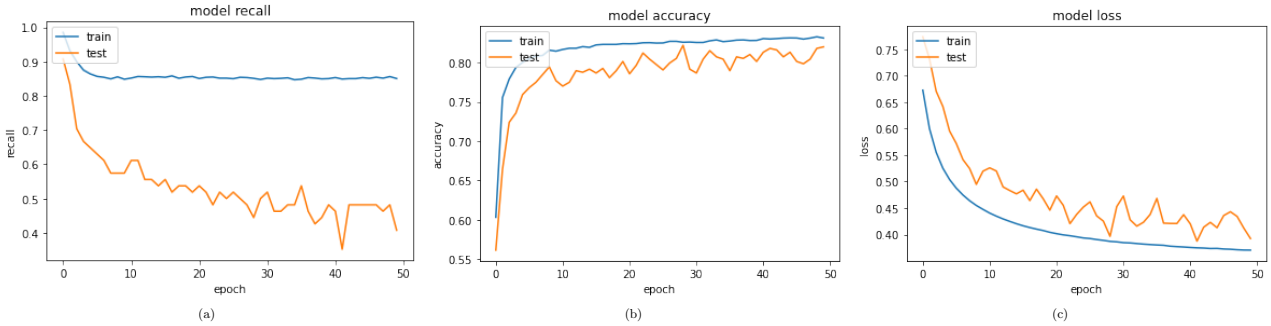
Figure 28: Two-hidden-layer model

This model was trained for 50 epochs and obtained these evaluation metrics:
Accuracy: 82.00%; Precision: 12.64%; Recall: 40.74%; False negative predictions: 32



The confusion matrix shows that this model correctly predicted 838 cases (82% of the total). However, 32 potentially ill patients were incorrectly classified as 'no-stroke'.

The graphs below illustrate the effect of the number of epochs on the performance metrics for both 'train' and 'test' datasets.



These graphs also show overfitting, altough the curves for both 'train' and 'test' datasets are closer than in the previous model.

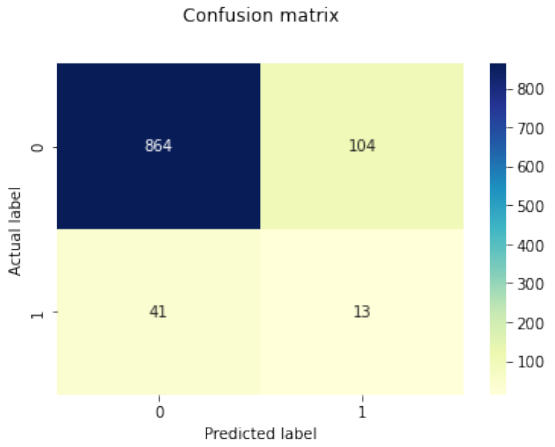
6.7 Three-hidden-layer neural network (implementation and evaluation)

Finally, I tried another neural network, the deepest one. To be able to confirm that deeper neural networks give worse 'recall', altough they improve the 'accuracy'.
Three-hidden-layer model:
Input features:16
Number of hidden layers: three
Number of nodes in the hidden layers: the number of nodes in each hidden layer is 32, which is the double of the input size.
Number of nodes in the output layer:1, as it is a binary classification problem
Activation for the hidden layers: 'relu' activation function
Activation for the output layer: 'sigmoid', as it is a binary classification problem
Optimizer:adam, with its default learning rate
Loss:'binary crossentropy', as it is a binary classification problem
Batch size: default (32)

```
ann = Sequential()
ann.add(Dense(32, input_shape=(X_train_smtom.shape[-1], ), activation='relu'))
ann.add(Dense(32, activation='relu'))
ann.add(Dense(32, activation='relu'))
ann.add(Dense(1, activation='sigmoid'))
```

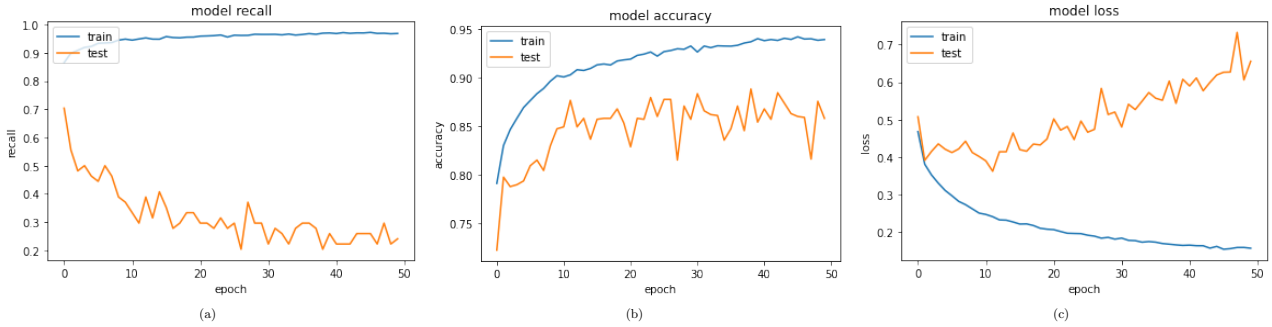
Figure 29: Three-hidden-layer model

This model was trained for 50 epochs and obtained these evaluation metrics:
Accuracy: 85.81%; Precision: 11.11%; Recall: 24.07%; False negative predictions: 41



The confusion matrix shows that this model correctly predicted 877 cases (85.81% of the total). However, 41 potentially ill patients were incorrectly classified as 'no-stroke'.

The graphs below illustrate the effect of the number of epochs on the performance metrics for both 'train' and 'test' datasets. The graphs suggest that this model is not a good fit for the 'test' dataset. The 'recall' and 'loss' graphs do not look very much like an optimal model. Also, there is an increasing gap between both 'train' and 'test' curves in all the graphs.



The following table collects the results for all the artificial neural networks built, to easily compare their performance:

ANN Summary	Hidden layers	Epochs	Learning rate	Recall	Accuracy	Precision	False negative predictions
No-hidden-layer model	None	50	Default (0.001)	55.56%	77.59%	12.77%	24
No-hidden-layer model **(optimized epochs)**	None	7	Default (0.001)	72.22%	74.95%	13.93%	15
No-hidden-layer model **(optimized lr)**	None	7	Optimized (0.0008)	75.93%	75.15%	14.54%	13
One-hidden-layer model	One (8 nodes)	50	Default (0.001)	50%	80.33%	13.43%	27
Two-hidden-layer model	Two (4 and 2 nodes)	50	Default (0.001)	40.74%	82%	12.64%	32
Three-hidden-layer model	Three (32 nodes each)	50	Default (0.001)	24.07%	85.81%	11.11%	41

In terms of accuracy, the deepest model is the one which performs worse, although its accuracy was the highest.
In contrast, the model which produces the best recall and, therefore, the smaller number of false negative predictions, is the simplest one, with no hidden layers, 7 epochs and the optimized learning rate.

7 Linear Regression based models

7.1 Linear Regression (implementation)

Firstly, I have changed the encode of the response variable 'stroke' as follows, to achieve a better fit into the linear regression model:
stroke = 1
'no stroke' = -1 (instead of 0)
Regarding the coefficients, the table below shows that only the coefficients for 'age' and 'avg_glucose_level' have a positive sign.

0	gender	-0.130590
1	age	0.760543
2	hypertension	-0.357312
3	heart_disease	-0.091777
4	ever_married	-0.229491
5	Residence_type	-0.100632
6	avg_glucose_level	0.027630
7	bmi	-0.019516
8	work_type_encoded_Govt_job	-0.937600
9	work_type_encoded_Never_worked	-0.437558
10	work_type_encoded_Private	-0.614367
11	work_type_encoded_Self-employed	-0.840831

Figure 30: Coefficients

The following formula was applied. It allows linear models to be used for binary classification tasks, as this one, by thresholding the predicted value at 0.5 (following the general rule):

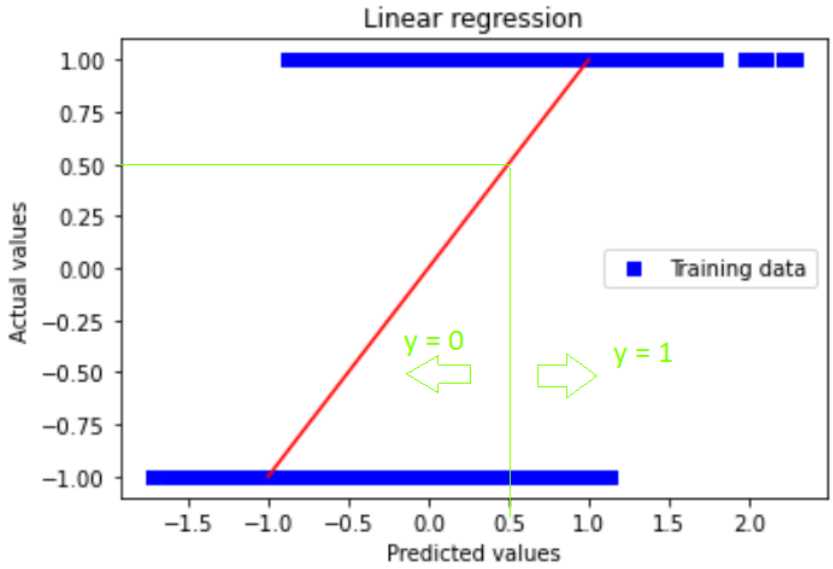
$$\hat{y} = w[0] * x[0] + w[1] * x[1] + ... + w[p] * x[p] + b > 0.5$$

Figure 31: Formula

Therfore:
If output > 0.5, prediction = 'stroke' (class =1)
If output < 0.5, prediction = 'no stroke' (class =-1)

The green line, on the chart below, represents the threshold at 0.5. Samples to the right of that point would be predicted as 'stroke' whereas samples to the left of that point would be predicted as 'no stroke'.
The graph suggests that many training samples will be incorrectly classified, therefore, the regression model is not the best fit.

On the other hand, by looking at the training data, it is not clear what a good threshold would be for the regression line to separate the two classes.



Making predictions

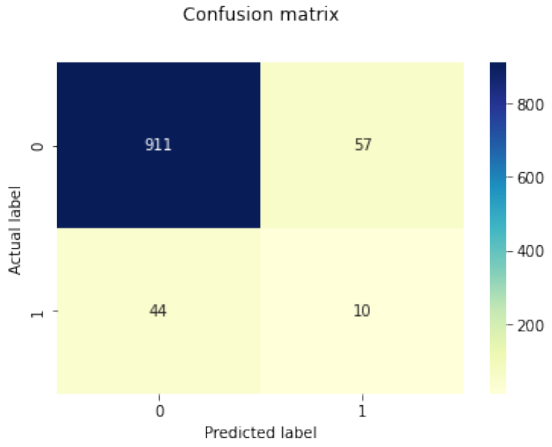
The predictions of the regressor can take values below -1 or greater than 1, even though the values of the classes are 1 and -1. This also indicates that linear regression is not the best model for this problem.

```
array([ 0.65536423, -1.09056555, -0.17581098, ...,  0.71324621,
        0.5311548 , -0.13728498])
```

Figure 32: Example for predictions below -1

7.2 Linear Regression (evaluation)

The confusion matrix shows that this model has correctly predicted 921 cases (90% of the total). However, 44 potentially ill patients were incorrectly classified as 'no-stroke'.



Evatuation metrics:
Accuracy: 90.11%
Precision: 14.92%
Recall: 18.52%
F1 score: 16.53%
Recall resulted quite low, therefore, a Logistic Regression model was built instead.

7.3 Logistic Regression (implementation)

This model was built using the default parameters.

Making predictions

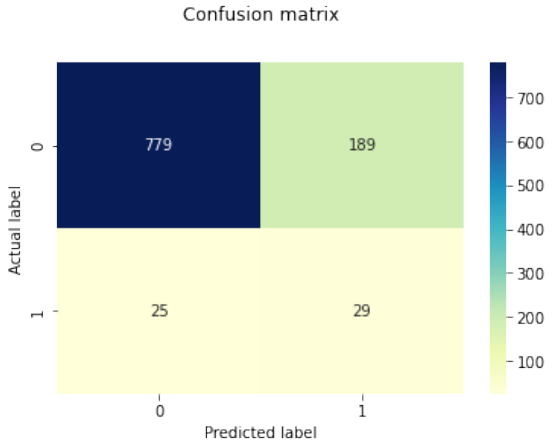
```
#Predictions on test data
y_pred_log =log.predict(X_test)
y_pred_log
```

```
array([1, 0, 0, ..., 1, 1, 0], dtype=int64)
```

This model seems to be more appropriate than linear regression since it gives predicted values between 0 and 1, but no bigger than 1 or less than 0 (as happened using linear regression).

7.4 Logistic Regression (evaluation)

The confusion matrix shows that this model has correctly predicted 808 cases (79% of the total). However, 25 potentially ill patients were incorrectly classified as 'no-stroke'.



The logistic regression algorithm obtained the following values for the evaluation metrics:
Recall: 53.70%;
Precision: 13.30%;
Accuracy: 79%;
F1 score: 21.32%;
False negative predictions: 25

7.5 Logistic Regression (optimization)

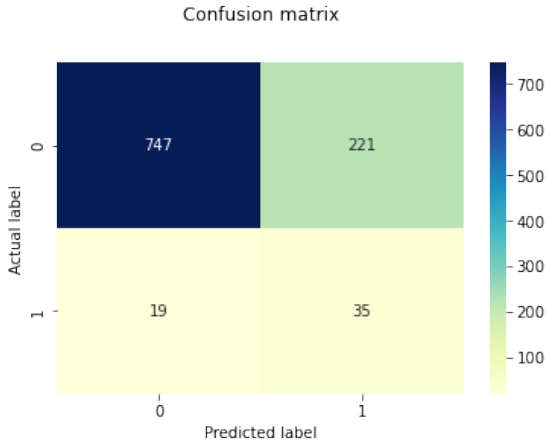
Hyperparameter optimization was carried using GridSearchCV with 5-fold cross-validation. These were the parameters tested:

```
solvers = ['newton-cg', 'lbfgs', 'liblinear']
penalty = ['l2']
c_values = [100, 10, 1.0, 0.1, 0.01]
```

This is the best hyperparameter configuration, which achieved the highest 'recall':

```
Best Hyperparameters: {'C': 0.01, 'penalty': 'l2', 'solver': 'liblinear'}
```

These were the results obtained:



Recall: 64.81%;
Precision: 13.67%;
F1 score: 22.58%;
Accuracy: 76.52%;
False negative predictions: 19;

The optimized model correctly predicted 782 cases (76.52% of the total). However, 19 potentially ill patients were incorrectly predicted as 'no-stroke'.
The initial model has been improved as expected. The number of false negatives has decreased from 25 to 19.
In fact, the recall score has been improved from 53.70% to 64.81%.

8 Conclusion

As explained above, when predicting if a patient will have a stroke, inaccurate predictions could have drastic consequences. Hence, an actual 'stroke' patient should not be predicted as a 'non-stroke' because that patient could die if they do not receive treatment (False negative predictions). However, if a non-stroke patient is predicted as 'stroke', the consequences will not be so serious (False positive predictions). Thus, this problem requires a model which produces high recall as evaluation metric, to avoid the false negatives. However, accuracy is not a good evaluation metric for this case. Therefore, the focus here has been on recall as the metric for choosing which model performs better.

This table summarises the results of the best three classifiers.

Model	Characteristics	Recall	False negative predictions	Accuracy	Precision
Random Forest	Based on the 3 most important features: 'age', 'avg_glucose_level', 'bmi' GridSearch best hyperparameters: 'criterion': 'entropy' 'max_depth': 12 'n_estimators': 450	72.22%	15	79.26%	16.50%
ANN	No-hidden-layer model 7 epochs 32 batch_size optimized learning rate (0.0008)	75.93%	13	75.15%	14.54%
Logistic Regression	GridSearch best hyperparameters: 'C': 0.01 'penalty': 'l2' 'solver': 'liblinear'	64.81%	19	76.52%	13.67%

Notice that the table only contains one model of each category. In each case, this chosen model was the one with the best performance after analysis and optimisation.

To conclude, the most appropriate model for the prediction of strokes is the artificial neural network built using the characteristics described in the table. This model performed better on the data in comparison to the other models.

9 Challenges

I had to overcome some interesting challenges while working on this assignment.

Firstly, it was clearly an unbalanced dataset, so I studied and coded some of the techniques to resample the data. Once the training data was resampled, the testing data was still unbalanced. The most common metric for classification problems such as this one is 'accuracy', however, I considered that the most important metric would be 'recall', as the main objective is to reduce the level of false negatives. This would have the effect of preventing people from dying due to lack of necessary medical treatment. I therefore tried to tune the models in this direction.

Secondly, building a linear regression model for this binary classification was a challenging task. Lots of reading was necessary in order to understand how to set the threshold and also to plot this along with the two classes for 'stroke'.

Thirdly, reaching an optimal artificial neural network has been challenging. I started by building a basic model using sequential API, without any hidden layers. This was not a perfect model so I built deeper and deeper models, increasing the number of hidden layers and also the number of nodes on each layer.
GridSearchCV was used to optimize the deepest models. However, this was only effective when the scorer was 'accuracy'; 'recall' did not respond very well, resulting in some 'NAN' values. I therefore used 'accuracy' for this purpose and then calculated the 'recall' values for the models.
However, the simplest model performed better than the more complex ones and this performance was further improved by optimizing its learning rate.

10 References

BOOK: "Hands-On Machine Learning with-Scikit-Learn, Keras and Tensorflow". By Aurelien Geron.
BOOK: "Deep Learning with Python". By Francois Chollet.
BOOK: "Introduction to Machine Learning with Python". By Andreas C. Muller and Sarah Guido.
BOOK "Regression Analysis with Python". By Luca Massaron and Alberto Boschetti.
https://www.kaggle.com/shrutimechlearn/deep-tutorial-1-ann-and-classification
https://machinelearningmastery.com/hyperparameter-optimization-with-random-search-and-grid-search