Supervised Learning on Stroke Prediction Dataset

1st Beatriz Gonçalves
*DETI**
University of Aveiro
Machine Learning Foundations
Prof. Petia Georgieva
(50% of work load)

2nd Tiago Nazário
*DETI**
University of Aveiro
Machine Learning Foundations
Prof. Petia Georgieva
(50% of work load)

*DETI: Department of Electronics, Telecommunications and Informatics

***Abstract*—In this project a database with 12 variables was analyzed [16], some of these variables being parameters commonly associated with the possibility of stroke in a person, others representing medical test results, and one variable representing the individual's stroke history. This project concerns a supervised learning problem. The main goal is to learn how to classify examples in terms of the concept under analysis using different learning algorithms. We used two machine learning models, Decision Tree Classifier (DT) (top accuracy of, approximately, 94%) and Support Vector Machines (SVM) (top accuracy of, approximately, 84%) and two deep learning models, Multilayer Perceptron (MLP), and Recurrent Neural Network (RNN). Then, we compared them using appropriate evaluation metrics.**

***Index Terms*—possibility of stroke, supervised learning, machine learning, deep learning, DT, SVM, MLP, RNN**

I. Introduction

The brain is an essential organ for a living being, so it is vital to take precautions to avoid having a stroke.

According to the World Health Organization (WHO) stroke is the second leading cause of death globally, responsible for approximately 11% of total deaths, and the third leading cause of disability. One in four people are in danger of stroke in their lifetime. Lifestyle risk factors for stroke include being overweight or obese, physical inactivity, tobacco use and alcohol abuse. [15] Hence, the prediction of an event like this is quite relevant for medicine and for the patients themselves.

Taking that into account, we aimed to develop four different models for predicting the occurrence of stroke in patients. We used two machine learning models, Decision Tree Classifier (DT) [13] and Support Vector Machines (SVM) [17], and two deep learning models, Multilayer Perceptron (MLP) [7] and Recurrent Neural Network (RNN) [9]. We collected a dataset that contains information about patients who have had a stroke and those who have not, and it includes features such as age, sex, blood pressure, body mass index, and others. These features are used as input for the machine learning models to achieve our goal of predicting whether a patient is at risk for a stroke. The dataset is provided by an user named "Federico Soriano" and it's labeled, it can be used for supervised learning problems. [16]

We first focused on data pre-processing. This step is crucial as it helps to clean and prepare the data for further analysis. In our case, we first checked for missing values and then handled them accordingly. We also made sure that the data was in the correct format and that all the features were of the same scale.

Next, we moved on to data visualization. This step helped us to better understand the (statistical) distribution of our data and identify any patterns or trends. We used various visualization techniques such as histograms, box plots, and scatter plots to gain insights into our data.

Finally, we applied the supervised learning techniques that we talked about before.

It is important to note that these three steps, data pre-processing, visualization, and supervised learning, should always be used together. This is because data pre-processing ensures that the data is in the correct format for analysis, visualization helps to understand the data and identify patterns, and finally, supervised learning helps to make predictions or classify the data. Without one of these steps, the analysis would not be as effective and accurate. [25]

II. State of art review

Existing works in the literature have investigated various aspects of stroke prediction. Jeena et al, provides a study of various risk factors to understand the probability of stroke [22]. It used a regression-based approach to identify the relation between a factor and its corresponding impact on stroke.

In Hanifa and Raja [21], an improved accuracy for predicting stroke risk was achieved using radial basis function and polynomial functions applied in a non-linear support vector classification model. The risk factors identified in this work were divided into four groups — demographic, lifestyle, medical/clinical and functional. Similarly, Luk et al, studied Chinese subjects to understand if age has an impact on stroke rehabilitation outcomes [23].

In [20] a methodology for designing effective binary classification ML models for stroke occurrence is presented. Since class balancing is crucial for the design of efficient methods in stroke prediction, the synthetic minority over-sampling technique (**SMOTE**) method was applied. Then, various models are developed, configured and assessed in the balanced dataset. For our purpose, **naive Bayes**, **logistic regression**, **stochastic gradient descent (SGD)**, **K-NN**, **decision trees**, **random forests** and **multilayer perception** were evaluated. In addition, the majority voting and stacking methods were applied, with the latter being the main contribution of the current study.

The experiments revealed the efficacy of the stacking method against the single models and the voting, achieving a high AUC, precision, recall, F-measure and accuracy.

The aim of [24] is to classify state-of-arts on ML techniques for brain stroke into 4 categories based on their functionalities or similarity, and then review studies of each category systematically. A total of 39 studies were identified from the results of ScienceDirect web scientific database on ML for brain stroke from the year 2007 to 2019. Support Vector Machine (SVM) is obtained as optimal models in 10 studies for stroke problems. Besides, maximum studies are found in stroke diagnosis although number for stroke treatment is least thus, it identifies a research gap for further investigation. Similarly, CT images are a frequently used dataset in stroke. Finally SVM and Random Forests are efficient techniques used under each category. The present study showcases the contribution of various ML approaches applied to brain stroke.

In [19] was used 10-fold cross-validation to perform the train and test approach. To train models, four different machine learning classifiers were used (naïve Bayes, BayesNet and random forest). The performance of the algorithms was evaluated and compared for stroke prediction using lab test results as features. Model accuracy was evaluated based on the following measures: recall or sensitivity, specificity, positive predictive value (PPV), negative predictive value (NPV), accuracy, and area under the curve (AUC) (or area under the receiver operating characteristic [ROC] curve) to compare the four classifiers.

The results from the various techniques are indicative of the fact that multiple factors can affect the results of any conducted study. These various factors include the way the data was collected, the selected features, the approach used in cleaning the data, imputation of missing values, randomness and standardization of the data will have an impact on the outcome of any study carried. Therefore, it is important for the researchers to identify how the different input factors in an electronic health record are related to each other, and how they impact the final stroke prediction accuracy.

III. Data description and Pre-processing

Our dataset contains clinical features for predicting stroke events. The data contains about 12 risk factors (columns/features) and 5110 patients (rows/observations). The variables used here can be described as follows.

A. Data description

1) **id:** unique identifier;
2) **gender:** gender of the patient: ”Male”, ”Female” or ”Other”;
3) **age:** age of the patient;
4) **hypertension:** 0 if the patient doesn’t have hypertension, 1 if the patient has hypertension;
5) **heart_disease:** 0 if the patient doesn’t have any heart diseases, 1 if the patient has a heart disease;
6) **ever_married:** patient is or was ever married: ”No” or ”Yes”;
7) **work_type:** The type of work the patient has: ”children” (if the patient is under age), ”Govt jov”, ”Never worked”, ”Private” or ”Self-employed”;
8) **Residence_type:** type of residence the patient lives in: ”Rural” or ”Urban”;
9) **avg_glucose_level:** average glucose level in patient’s blood;
10) **bmi:** body mass index;
11) **smoking_status:** Smoking status of patient: ”formerly smoked”, ”never smoked”, ”smokes” or ”Unknown”;
12) **stroke:** 1 if the patient had a stroke or 0 if not.

B. Pre-processing

Data pre-processing is a crucial step in the data analysis process. It is the process of cleaning, transforming, and organizing the data before it is fed into a machine learning model. The goal of data pre-processing is to make sure that the data is in a format that is suitable for analysis and modeling. This step is necessary to ensure that the data is accurate, consistent, and free of errors.

Before visualization, we do some initial pre-processing steps. Our first step is to select information (feature selection). Since our goal is, within these features, to predict whether a patient is at risk for a stroke, we will select the information that we think is important and least important for this prediction.

Analyzing the meaning of each column, listed above, we decided that the ”id” column has no influence on the stroke prediction, this is just an identification feature that will not affect our prediction. Second, it is a feature that makes no sense to be normalized. In that case, it would end up influencing our models, having more weight in them than what we want.

We then proceeded to verify the existence of null values. The output shows that there are 201 missing values in the ”bmi” column. This missing data can have an impact on the model’s performance and should be handled accordingly, for example, by removing the rows with missing data or imputing the missing values with a suitable value such as the mean or median of the column. This decision should be made based on the overall size of the dataset and the proportion of missing values. Losing 201 rows would end up greatly reducing our sample size. Therefore, we chose to replace the null values with the average of this column.

Regarding repeated rows, we have verified that there are none and that, therefore, the dataset has 5110 unique rows.

After visualization, further pre-processing steps were done. We started by counting the values in the ”gender” column, for example. We only got one sample classified as ”other”. It will not have any contribution to our models, so we dropped this row.

We also have ages, in the ”age” column that are not integer numbers, and thus do not have an intuitive interpretation. We have no information on how this variable is measured, so it is best to take everything as years and round it to the nearest integer number.

We have some columns with strings, such as ”gender”, ”ever_married”, ”residence_type”, and ”smoking_status”. So we need to encode it before applying it to the models. We start by encoding ”gender” and ”ever_married”, since they are classified as ”female”, ”male” and ”yes”, ”no”, respectively. So we classified ”female” and ”yes” as 1, and ”male” and ”no” as 0.

Regarding the ”smoking_status” feature, we have four classification possibilities ”formerly smoked”, ”never smoked”, ”smokes” or ”Unknown”. We will group ”formerly smoked” and ”smokes” into the same category, to simplify our analysis. This will reduce the number of categories in the smoking status column, making it easier to analyze and interpret the data. Now, for the ”Unknown” category, one option could be to remove these rows, but we don’t want to lose any more information from our dataset. So, another option could be to replace the unknown values with the (statistical) mode (most common value) of the ”smoking_status” column. It is not clear what the true value for these unknown observations is, this is one assumption that we will make. This way, we will join the ”formerly smoked” and ”smokes” classes and classify them as 1, and join the ”never smoked” and ”Unknown” classes and classify them as 0.

For the ”Residence_type” feature we can just classify it as zeros and ones, because it only has 2 possible classifications ”Urban” and ”Rural”. This way, we will classify ”Urban” as 1, and ”Rural” as 0.

Regarding the ”work_type” feature, we have more than two classes. We have the classes: ”children”, ”Govt_jov”, ”Never_worked”, ”Private” or ”Self-employed”. We then have to figure out how to encode this feature. It would not make sense to use ”*.LabelEncoder()*”. It is a class in the scikit-learn library in Python that is used to convert categorical data, or data that can be divided into categories, into numerical data. It assigns a unique integer value to each category. [12] However, we would end up assigning codes to the ”work_type” classes randomly, and we would not be able to understand and identify what each code corresponds to. Since we don’t want to do this ”blindly”, we will encode this differently.

For each of these classes we will create a column, that is, we will add five more columns to our dataset: ”children”, ”Govt_jov”, ”Never_worked”, ”Private” and ”Self-employed”. If the sample belongs to the ”children” class, for example, then we will return 1 in this column for ”childern” and 0 in the remaining four, and so on. Finally, we drop the original ”work-type” column.

The ”stroke” variable, the one we want to predict, is extremely unbalanced. We have 4860 samples with stroke = 0 and only 249 with stroke = 1. Trying to create a sub-dataset with the same number of stroke and non-stroke cases is not possible, since we would loose a lot of information and get a much smaller dataset to later apply in machine learning and deep learning algorithms.

One solution for this problem is the SMOTE (Synthetic Minority Oversampling Technique). It is a technique used to balance class distribution in a dataset. It is used to oversample the minority class by creating synthetic samples. The algorithm works by finding the k nearest neighbors for each minority class sample, and then generating new samples by randomly selecting one of the k nearest neighbors and taking the difference between that sample and the original sample, multiplying it by a random number between 0 and 1, and then adding it to the original sample. This new sample is then added to the dataset, thereby increasing the number of minority class samples. The goal of SMOTE is to balance the class distribution by oversampling the minority class, which can help improve the performance of machine learning models on imbalanced datasets. [14] [4]

Some advantages of SMOTE include:

1) It can effectively balance class distribution by oversampling the minority class;
2) It can reduce the impact of the class imbalance problem on the model’s performance;
3) It can help improve the model’s ability to generalize to new minority samples.

Disadvantages of SMOTE include:

1) It can lead to overfitting if the synthetic samples are too similar to the original minority samples;
2) It can be computationally expensive if the size of the dataset is large;
3) It can lead to a loss of diversity in the minority class if the synthetic samples are not diverse enough.

Now, our next step is normalization. Since our features are measured at different scales we have to normalize our data. This is because scales with higher numbers end up having more importance in our models, while smaller numbers end up having less, and this will end up misleading our models.

Columns that are already classified as 0 and 1 make no sense to be normalized. This happens with ”gender”, ”hypertension”, ”heart_disease”, ”ever_married”, ”Residence_type”, ”smoking_status”, ”stroke”, ”children”, ”Govt_job”, ”Never_worked”, ”Private” and ”Self_employed” columns.

This way, normalization was done to ”age”, ”avg_glucose_level” and ”bmi”. This was done using the max-min normalization technique, which scales the values between 0 and 1. [6]

Finally, we split our data into training and test set. Since our goal was to predict the ”stroke” column, we don’t want to train this column, we want it to be our predictable one. This way, we dropped and saved the column we want to predict.

We set side **20%** for test data. This is to ensure that we don’t commit any data snooping bias. In short, 20% of our data was used for testing, and the remaining 80% for training.

Since, as expected, our ”stroke” variable after splitting, is very unbalanced, we applied the SMOTE technique that we talked before. It is also important to note that it is generally recommended to only use SMOTE on the training set, and not the test set, so that the model is not trained to expect the over sampled data during evaluation. After applying this we got 7800 rows in our training set, ending with 50% of the data

with stroke=1 and other 50% with stroke=0 in our dependent column.

IV. Data Visualization

In the previous section, Pre-processing, we analysed the data provided and made a few changes in order to easily have a better approach for our solution of the problem. Now, we will provide some different types of graphics and plots for a user friendly comprehension of the data to study as well as some statistical interpretation of our dataset.

A. Correlation Matrix

A correlation matrix is a symmetric matrix that shows the correlation coefficient between the different variables in the dataset. The correlation coefficient is a value between -1 and 1 that measures the strength and direction of the linear relationship between two variables. A value of 1 indicates a perfect positive correlation, a value of -1 indicates a perfect negative correlation, and a value of 0 indicates no correlation. [2]

We will analyse the correlation matrix between our numerical variables:

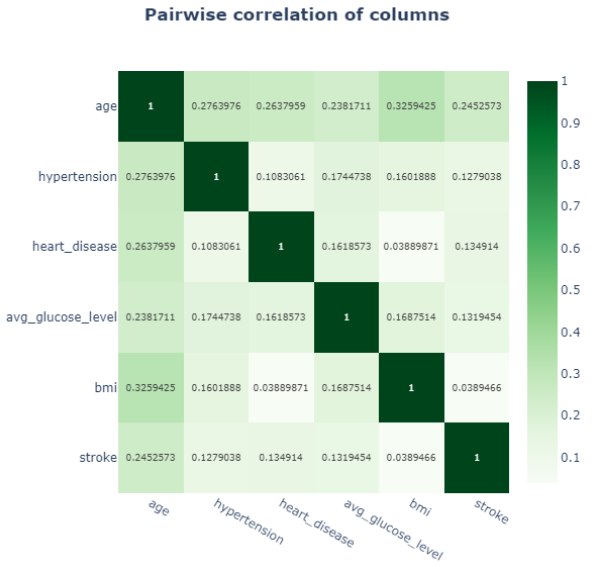

Fig. 1. Correlation Matrix.

From Fig.1 the following observations can be made:

1) The variable "**age**" has a relatively strong positive correlation with "**avg_glucose_level**" (0.238), "**bmi**" (0.326) and a moderate positive correlation with "**hypertension**" (0.276) and "**heart_disease**" (0.264);
2) The variable "**hypertension**" has a moderate positive correlation with "**avg_glucose_level**" (0.174) and a weak positive correlation with "**heart_disease**" (0.108) and "**bmi**" (0.160);
3) The variable "**heart_disease**" has a moderate positive correlation with "**avg_glucose_level**" (0.162) and a weak positive correlation with "**hypertension**" (0.108) and "**bmi**" (0.038);
4) The variable "**avg_glucose_level**" has a moderate positive correlation with "**bmi**" (0.169) and weak positive correlation with "stroke" (0.132);
5) The variable "**bmi**" has a weak positive correlation with "**stroke**" (0.038);
6) The variable "**stroke**" has a moderate positive correlation with "**age**" (0.245) and "**heart_disease**" (0.135) and weak positive correlation with "**avg_glucose_level**" (0.132).

It's important to note that correlation does not imply causation, so these findings should not be interpreted as indicating that one variable causes another.

The relatively strong positive correlation between "age" and "avg_glucose_level" could indicate that older people tend to have higher glucose levels. The moderate positive correlation between "age" and "hypertension" and "heart_disease" could indicate that these conditions are more common in older people. The moderate positive correlation between "hypertension" and "avg_glucose_level" could indicate that people with hypertension tend to have higher glucose levels. The weak positive correlation between "heart_disease" and "avg_glucose_level" could indicate that people with heart disease tend to have slightly higher glucose levels. The moderate positive correlation between "stroke" and "age" and "heart_disease" could indicate that these conditions are more common in people who have had a stroke.

B. Feature's Distribution

We analyzed the distribution of all our features (again, all non-strings), to see some of their behavior. The distributions are represented in Fig.47 in Appendix.

There we can see how our dataset is balanced. For the "age" feature we can see that the features are more or less evenly distributed, from 0 to 82 years.

For the "hypertension" feature, we can observe that we have much more samples with hypertension=0 than with hypertension=1. This is clearly an unbalanced feature.

For the "heart_disease" feature we can also see a major unbalanced distribution, as we have much more observations with heart_disease=0 than heart_disease=1.

The observations for the "avg_glucose_level" feature range between 0 and 272. We can see that we have a major concentration of population between 65 and 100, approximately.

Regarding the "bmi" feature, we can see a bigger distribution between 20 and 25, approximately.

Now, regarding the variable we want to predict, we can see that we have many more patients who have not suffered from "stroke" than users who have suffered from "stroke".

C. Identifying outliers

We can also represent the distribution of the numerical variables using boxplots. These represent the interquartile range (IQR), the whiskers representing the minimum and maximum

values (excluding outliers), and the dots representing outliers or extreme values. This plot can be used to identify outliers and check the distribution of the numerical variables in the dataset. [5]

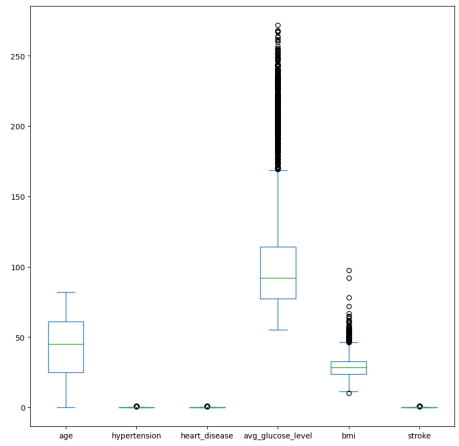

Fig. 2. Box plot.

Due to the high number of dots (that could be outliers) in the variable "**avg_glucose_level**", we decided to explore a little further. We then conclude that for this particular variable we have about 1277 rows with values above the third quartile (75%), which is still a considerable part of our data. Therefore, we didn't remove them.

D. String Features Distribution

Here we will see the distribution only for some string features.

1) Smoking Status: Let's see the distribution for the smoking status.

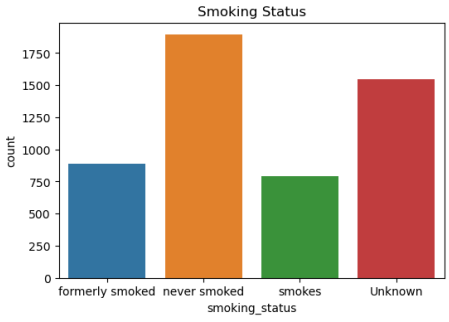

Fig. 3. Smoking Status.

This count plot will display the number of occurrences for each category in the 'smoking_status' column. This can be useful to see the distribution of smoking status among the observations in the data. We can see that we more non-smokers in our data, for example.

2) Stroke Status: Let's see the distribution for the stroke status, that is the dependent variable.

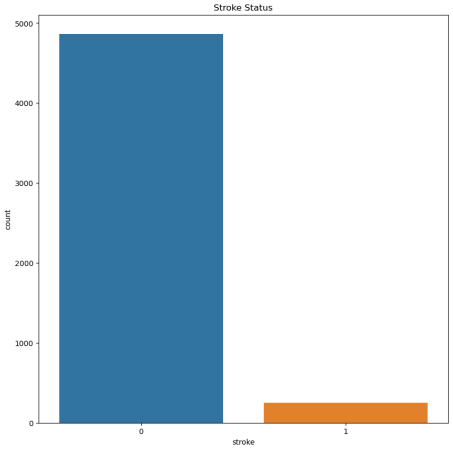

Fig. 4. Stroke Status.

We have seen earlier, in section Pre-processing, that the count of stroke=0 values is much higher than the count of stroke=1 values. Here we can check this distribution visually, confirming that our dataset is not balanced.

E. Correlation distribution

1) Age Vs. Avg_glucose_level : From the correlation matrix (Fig.1) there is an indication that older people tend to have higher glucose levels. We decided to look at the distribution of "**avg_glucose_level**" as a function of "**age**", Fig.48 in Appendix. From this distribution we can once again confirm the tendency for "**avg_glucose_level**" to increase as "**age**" also increases.

2) Age Vs. Hypertension Vs. Heart_disease: Another inference made was that the moderate positive correlation between "age" and "hypertension" and "heart_disease" could indicate that these conditions are more common in older people.

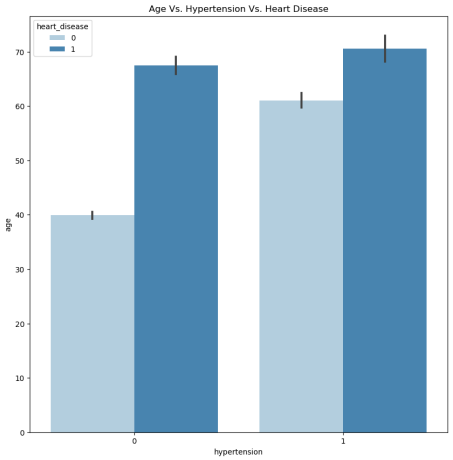

Fig. 5. Age Vs. Hypertension Vs. Heart_disease.

This bar plot shows the relationship between "hypertension", "age", and "heart disease". The x-axis represents hypertension, the y-axis represents the mean of the age for each category of hypertension and heart disease. The hue represents

the heart disease, and it's divided into two categories "Yes" and "No".

This bar plot can help us understand the relationship between hypertension, age, and heart disease. It can show us the mean age of patients who have hypertension, heart disease, and both, and help us understand if hypertension and heart disease are related to age or not.

We can confirm the previous inference that the average age is higher when "heart_disease" is equal to 1, that is, when the user has some heart disease.

3) Hypertension Vs. Avg_glucose_level: From the correlation matrix of Fig.1 we supposed that the moderate positive correlation between "hypertension" and "avg_glucose_level" could indicate that people with hypertension tend to have higher glucose levels.

To represent this we use a violin plot. A violin plot is a way to visualize the distribution of a continuous variable by a categorical variable. It combines the best aspects of a boxplot and a kernel density estimate (KDE) plot.

In the violin plot, the violin shape is generated based on the kde (kernel density estimate) of the data, showing the density of the data points in the y axis.

It has a few key elements that can be used to interpret the distribution of the data:

1) **The width of the violin:** The width of the violin represents the number of observations in that category. A wider violin indicates that there are more observations in that category;
2) **The height of the violin:** The height of the violin represents the density of the observations. A taller violin indicates that the observations are more dense in that area;
3) **Comparison between the violins:** We can compare the distribution of the data between the different categories (in this case hypertension) by looking at the shape, height and width of the violins.

[18]

This way, we got the Fig.49 in the Appendix.

In this case, we can see that the height of the violin is different for hypertension = 0 and hypertension = 1, this means that the density of the observations is different for these two groups. If we want to compare the distributions of "avg_glucose_level" in the two groups, we can look at the shape, height and width of the violins in each group. A taller and wider violin in one group indicates that the observations are more dense and there are more observations in that group.

We can see that the height of the violin is taller for hypertension = 1 than hypertension = 0, this means that the density of the observations is higher for people with hypertension. Therefore, we can say that people with hypertension tend to have higher glucose levels.

However, it's important to note that this plot only shows the relationship between hypertension and average glucose level, it doesn't account for other factors that could influence this relationship.

4) Heart_disease Vs. Avg_glucose_level: The weak positive correlation between "heart_disease" and "avg_glucose_level" could indicate that people with heart disease tend to have slightly higher glucose levels.

Again, we are comparing a continuous variable with a categorical variable. This way, we will use again the violin plot, which can be seen in Fig.50 in Appendix.

We can see that the comparison between the violins is very similar to the one we did before. The height of the violin is taller for heart_disease = 1 than heart_disease = 0, this means that the density of the observations is higher for people with heart_disease. Therefore, we can say that people with heart_disease tend to have higher glucose levels.

Again, it's important to note that this plot only shows the relationship between hypertension and average glucose level, it doesn't account for other factors that could influence this relationship.

5) Stroke Vs. Age Vs. Heart Disease: We also said that the moderate positive correlation between "stroke" and "age" and "heart_disease" could indicate that these conditions are more common in people who have had a stroke. Let's try to confirm this with another visualization:

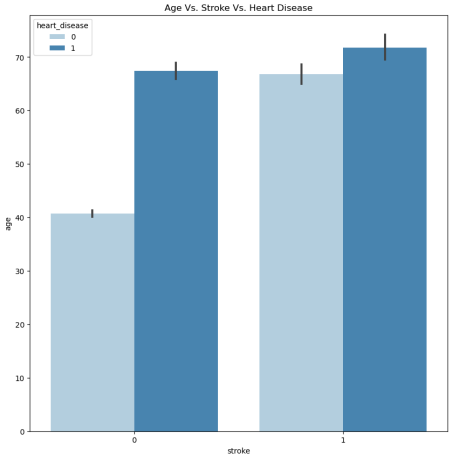

Fig. 6. Stroke Vs. Age Vs. Heart Disease.

The x-axis represents the variable "stroke" and is divided into two categories: 0 and 1, representing whether the patient had a stroke or not. The y-axis represents the variable "age" and shows the average age of the patients for each category of "stroke" and "heart_disease". The hue parameter is set to "heart_disease" which will divide the bars into two different colors, representing whether the patient has heart disease or not. The palette parameter is set to "Blues" which will change the color of the bars to shades of blue.

This bar plot can help us understand the relationship between stroke, age, and heart disease. It can show us the average age of patients who had a stroke or not and whether they have heart disease or not. We can actually see a moderate positive correlation between these variables.

F. More Feature Relations

We can also explore some different relations between our features.

1) Work Type Vs. Stroke: For the ”work_type” and ”stroke” relation we got:

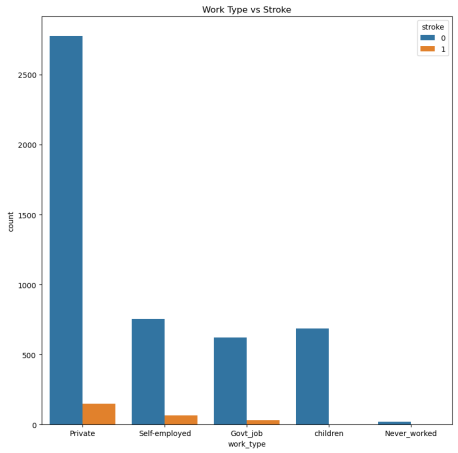

Fig. 7. Work Type Vs. Stroke.

The x-axis represents the variable ”work_type” and is divided into different categories such as ”children”, ”Private”, ”Self-employed”, ”government” and ”Never_worked”. The y-axis represents the count of observations for each category of ”work_type” and ”stroke”. The hue parameter is set to ”stroke” which will divide the bars into two different colors, representing whether the patient had a stroke or not.

This count plot can help us understand the relationship between work_type and stroke. It can show us the count of patients who had a stroke or not for each work_type and understand if certain work_type are more prone to stroke or not. In this case, we can see that ”Private” work type and ”Self-employed” work type are more prone to have a stroke than the other work type. But they are also the ones that have more observations.

All these previous visualization were done with non-normalized data. However, if we want to compare the proportions of stroke cases among different work types, it would be better to normalize the data by dividing the count of observations in each category by the total number of observations in that category. This will give us the proportion of observations in each category, which can be useful for making meaningful comparisons between categories:

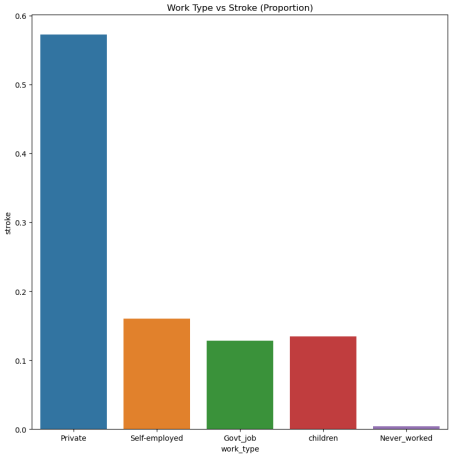

Fig. 8. Work Type Vs. Stroke (Proportion).

This shows the relationship between the categorical variable ”work_type” and the categorical variable ”stroke” in terms of proportion.

The x-axis represents the variable ”work_type” and is divided into different categories such as ”children”, ”Private”, ”Self-employed”, ”government” and ”Never_worked”. The y-axis represents the proportion of observations for each category of ”work_type” and ”stroke”. The estimator parameter is set to a lambda function that normalizes the data by dividing the count of observations in each category by the total number of observations in the dataset.

This bar plot can help us understand the relationship between work_type and stroke in terms of proportion. It can show us the proportion of patients who had a stroke or not for each work_type. It can help you understand if certain work_types are more prone to stroke or not. We can still see, with the normalized data, that the ”Private” and ”Self-employed” work types are the ones with the highest proportion of strokes.

2) Residence Type Vs. Stroke: Let’s explore the number of strokes for each residence type:

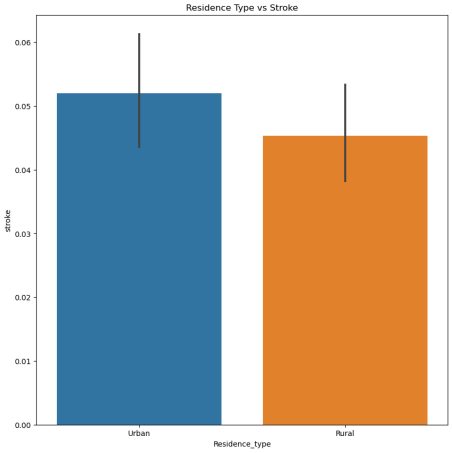

Fig. 9. Residence Type Vs. Stroke.

The x-axis represents the variable ”Residence_type” and is divided into different categories such as ”Rural”, ”Urban” . The y-axis represents the mean of the observations for each category of ”Residence_type” and ”stroke” .

This bar plot can help us understand the relationship between Residence_type and stroke. It can show us the mean of patients who had a stroke or not for each Residence_type, and help us understand if certain Residence_types are more prone to stroke or not.

However, as before, it is important to consider normalizing the data when we’re comparing different categories of a variable, especially when the number of observations in each category is not the same. Normalizing the data will help us compare the proportion of observations in each category rather than the raw count of observations.

In this case, we can use same estimator as we did in the previous plot:

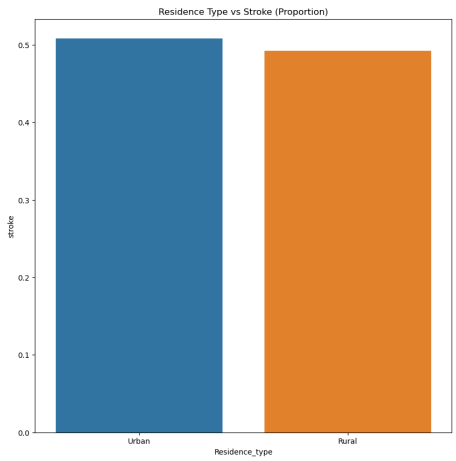

Fig. 10. Residence Type Vs. Stroke (Proportion).

This will give us the proportion of stroke cases among different Residence types. Unlike what we saw in the plot where the data was not normalized, we can see that the proportion of strokes for the different residence types is very similar.

3) Ever Married Vs. Stroke: Let’s explore the number of strokes for married and unmarried people, since this also seems to be a risk factor.

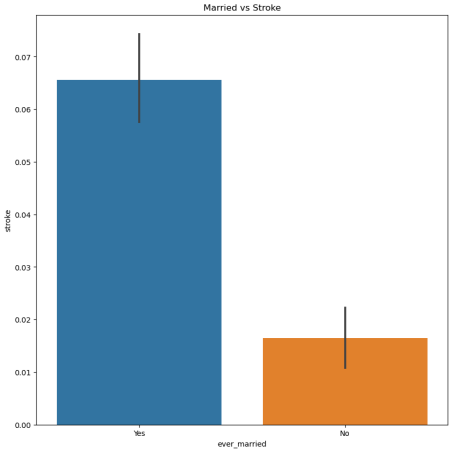

Fig. 11. Ever Married Vs. Stroke.

The x-axis represents the variable ”ever_married” and is divided into two categories ”Yes” and ”No”. The y-axis represents the mean of the observations for each category of ”ever_married” and ”stroke”.

This bar plot can help us understand the relationship between ever_married and stroke. It can show you the mean of patients who had a stroke or not for each category of ever_married, and understand if being married is actually a risk factor for stroke or not.

As mentioned before it’s important to normalize the data to compare the proportion of observations in each category rather than the raw count of observations. We will use the same estimator parameter as before:

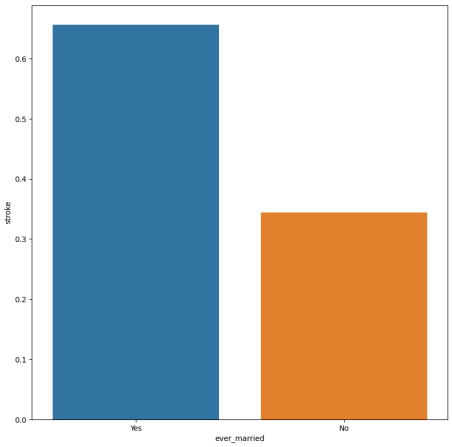

Fig. 12. Ever Married Vs. Stroke (Proportion).

From the bar plot, we can see that the proportion of stroke cases among the ”Yes” category is higher than the proportion of stroke cases among the ”No” category. However, it’s important to keep in mind that the proportion of stroke cases alone doesn’t necessarily mean that being married is a risk factor for stroke. There might be other factors that are associated with both being not married and having a stroke, such as age, lifestyle, etc.

It's also important to consider the sample size, it's possible that the dataset we're working with is not representative and has more observations in the "Yes" category than the "No" category, that could be the reason why the bar is twice the size. We actually have 3353 observation for the "Yes" category, and 1757 for the "No" category.

4) Gender Vs. Stroke: Let's explore the number of strokes by gender.

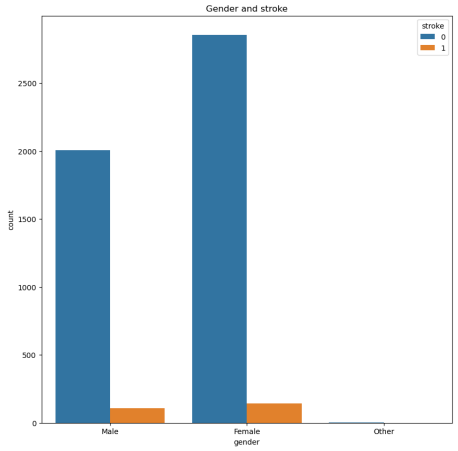

Fig. 13. Gender Vs. Stroke.

This count plot shows the relationship between gender and stroke. The x-axis represents gender and the y-axis represents the number of observations for each category of gender and stroke. The hue represents the stroke, and it's divided into two categories "Yes" and "No".

This count plot can help us understand the relationship between gender and stroke. It can show us the number of patients of each gender who have had a stroke or not, and help us understand if there is a relationship between gender and stroke or not.

It's important to keep in mind that the sample size for each category is important, it could be that one category has more observations than the other and that could be the reason why one category is bigger than the other.

Again, it would be a good idea to normalize the data in this case. Count plots are useful for showing the frequency of observations in each category, but the raw count can be misleading if the sample size is not equal for each category. By normalizing the data, we can compare the proportion of observations in each category, rather than the raw count.

For example, in this case, we can calculate the proportion of stroke cases among men and women, and compare the proportions rather than the raw count.

Again, we will use the same estimator parameter as before:

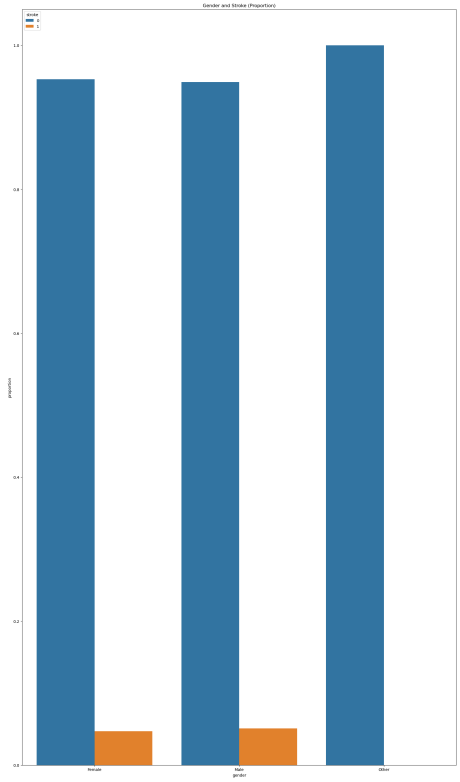

Fig. 14. Gender Vs. Stroke (Proportion).

From this graph, it can be seen that the proportion of stroke occurrences is similar in females and males. The proportion of cases where stroke does not occur is higher in both males and females as compared to the cases where stroke does occur.

5) Hypertension Vs. Stroke: Let's explore the number of strokes related to the occurrence of hypertension:

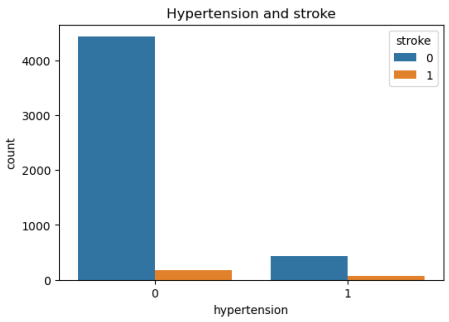

Fig. 15. Hypertension Vs. Stroke.

This graph shows the count of occurrences of hypertension and stroke for each category (0 or 1) in the dataset. It can be seen that the majority of individuals with hypertension did not have a stroke.

One more time, it would be a good idea to normalize the data in order to better compare the proportions of stroke occurrences between the two groups. This can be done by calculating the proportion of stroke occurrences for each group (hypertension=0 and hypertension=1) relative to the

total number of observations in that group. This way, we can directly compare the proportion of stroke occurrences between the two groups and make more accurate conclusions:

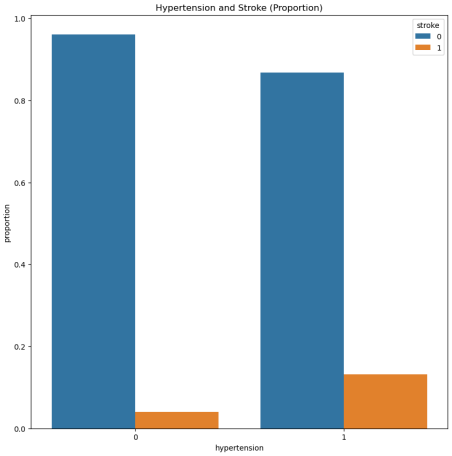

Fig. 16. Hypertension Vs. Stroke (Proportion).

Now we can see that a larger number of individuals who did have a stroke, also had hypertension compared to those who did not have a stroke. This suggests that hypertension may be a risk factor for stroke.

After this visualization process we already have a much better understanding of the data we are dealing with, as well as the relationships of our variables. We are then able to apply our supervised learning models.

V. Implemented Models

After having pre-processed the data we proceeded to the implementation of ML Models.

For our solution we implemented 2 machine learning algorithms:

*1) **Decision Tree Classifier**:* The algorithm creates a tree-like model of decisions and their possible consequences. It starts with a single node, called the root node, that represents the entire dataset. The root node is then split into several branches, each representing a potential decision or feature. Each branch then splits into more branches, and so on, until the tree reaches its leaves, which represent the final outcomes or predictions. [13]

*2) **Support Vector Machines (SVMs)**):* as well as decision trees, are a type of supervised machine learning algorithm that can be used for both classification and regression tasks. The goal of an SVM is to find the best boundary (or "hyperplane") that separates the data into different classes or groups. The best boundary is the one that maximizes the margin, which is the distance between the boundary and the closest data points from each class. These closest data points are called "support vectors." [17]

For our solution we also implemented 2 deep learning algorithms:

*3) **Multilayer Perceptrons Classifier (MLP)**:* is a type of artificial neural network that is composed of multiple layers of interconnected "neurons." The input layer receives the raw data, and each subsequent layer applies mathematical transformations to the data, allowing the network to learn increasingly complex features of the data. The final layer, known as the output layer, generates the network's predictions. [7]

*4) **Recurrent Neural Networks (RNN)**:* is a type of artificial neural network that have a "memory" which allows them to process sequences of data, such as time series or natural language. RNNs are different from traditional feedforward neural networks in that they have feedback connections that allow information to flow across multiple time steps. There are different types of RNNs, such as Long Short-Term Memory (LSTM), which was the one that we implemented in our work. [9]

VI. Application and Analysis of the ML Models

Let's divide the application of our ML models into two parts. In the first part, we apply the ML models using the **default values**, that is, without setting the **hyper-parameters**. In the second part we choose the best hyper-parameters for our models and see if there were any improvements in their performance.

For each model we apply, we also do k-fold cross-validation [3]. In addition, we present a classification report performance metrics [1], a confusion matrix [10], and a ROC curve [11].

K-fold Cross-Validation is when the dataset is split into a K number of folds and is used to evaluate the model's ability when given new data. K refers to the number of groups the data sample is split into. For example, if you see that the k-value is 5, we can call this a 5-fold cross-validation. Each fold is used as a testing set at one point in the process. The process is the following: we choose your k-value, split the dataset into the number of k folds, start off with using your k-1 fold as the test dataset and the remaining folds as the training dataset. Then we train the model on the training dataset and validate it on the test dataset and save the validation score. We keep repeating steps 3 – 5 but changing the value of your k test dataset. So we chose k-1 as our test dataset for the first round, we then move onto k-2 as the test dataset for the next round. By the end of it we would have validated the model on every fold that you have. Finally, we just average the results that were produced in step 5 to summarize the skill of the model.

For the classification report, we analysed the performance metrics:

1) **Precision:** tells us, out of all the positive predicted, what percentage is truly positive. The precision value lies between 0 and 1;
2) **Recall:** tells us, out of the total positive, what percentage are predicted as positive. It is the same as TPR (true positive rate);
3) **F1-score** that is the harmonic mean of precision and recall. It takes both false positive and false negatives into

account. Therefore, it performs well on an imbalanced dataset.

Precision, recall, and F1 score are all broken down by class, and then a macro average and weighted average are given for each.

1) **Macro average:** is the usual average we're used to seeing. Just add them all up and divide by how many there were;
2) **Weighted average:** considers how many of each class there were in its calculation, so fewer of one class means that it's precision/recall/F1 score has less of an impact on the weighted average for each of those things.
3) **Support:** tells how many of each class there were.

The confusion matrix represents, on the principal diagonal, the true positives and true negatives, that is, the values that the model predicted as having a stroke or not and actually are. The remaining two values are false positives or false negatives, that is, cases where the model predicted wrong class.

Finally, we also wanted to analyze the Receiver Operating characteristic curve. The ROC curve is created by plotting the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings. The true-positive rate is also known as sensitivity, recall or probability of detection.The false-positive rate is also known as probability of false alarm and can be calculated as (1 specificity). Classifiers that give curves closer to the top-left corner indicate a better performance. As a baseline, a random classifier is expected to give points lying along the diagonal (FPR = TPR).

A. Using Default Values

In this first part of our algorithm implementation we used the **default values** of each algorithm classifier. Then we compared the results with the original "**stroke**" column.

*1) **Decision Tree Classifier**:* Here we present the decision tree model where we obtained these results regarding accuracy:

Accuracy for training set: 1

Accuracy for testing set: 0.82

Clearly, analyzing the accuracies we obtained, we were facing a case of **overfitting** for this model.

This happens when the model completely fits the training data but fails to generalize the testing unseen data. Overfit condition arises when the model memorizes the noise of the training data and fails to capture important patterns. It is important to check the accuracy on the test set or use cross validation techniques to get a better idea of the model's performance on unseen data.

There are various techniques to prevent the decision tree model from overfitting, we will use "**Pruning**". By default, the decision tree model is allowed to grow to its full depth. **Pruning** refers to a technique to remove the parts of the Decision Tree to prevent growing to its full depth. By tuning the hyperparameters of the Decision Tree model, the trees are pruned and prevented from overfitting.

There are two types of pruning, **Pre-pruning** and **Post-pruning**. **Pre-pruning** is what we did in the second part that we talked about above, we will clarify this one in more detail later.

So, here, we applied **Post-pruning**. We will come back to this shortly after analysing the **Classification report** for the test set, **Confusion matrix** and **Roc curve**.

Classification Report:

Classification report:

	precision	recall	f1-score	support
0	0.95	0.85	0.90	960
1	0.11	0.27	0.15	62
accuracy			0.82	1022
macro avg	0.53	0.56	0.53	1022
weighted avg	0.90	0.82	0.85	1022

Fig. 17. Classification Report Decision Tree Classifier

Confusion Matrix:

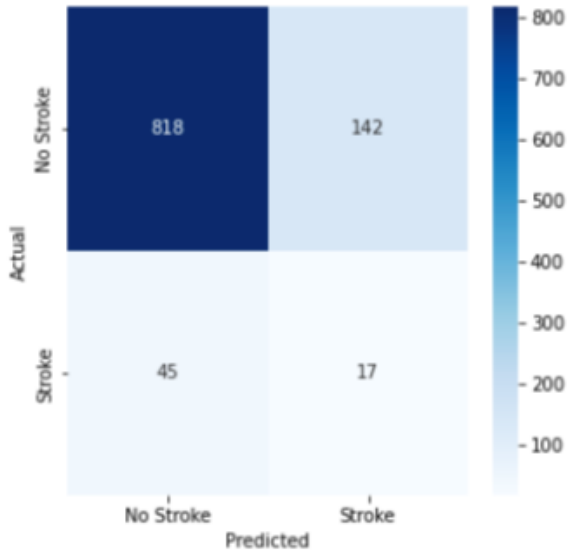

Fig. 18. 1st Confusion Matrix Decision Tree.

ROC curve:

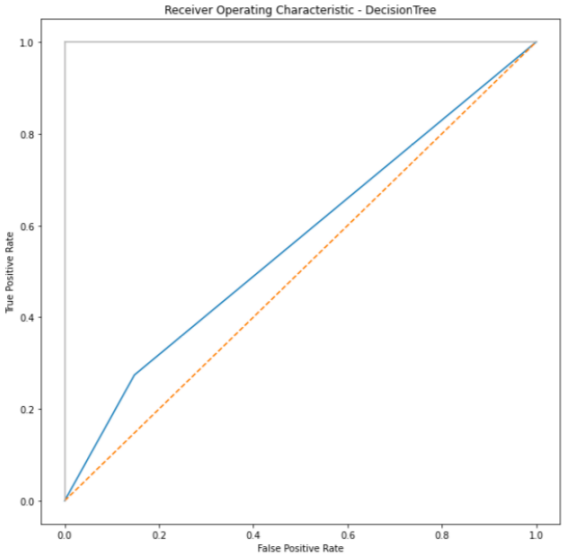

Fig. 19. 1st ROC Decision Tree.

The **AUC (Area Under the Curve)** is also a metric to measure the performance of a classifier, with a value of 1 indicating perfect performance, and a value of 0.5 indicating a random classifier.

AUC for Decision Tree Classifier: 0.54

In this case, we got an AUC score of, approximately, 0.55, which is not good, since it indicates a random classifier.

K-fold Cross Validation:

```
Fold:  1, Training/Test Split Distribution: [3510 3510], Accuracy: 0.846
Fold:  2, Training/Test Split Distribution: [3510 3510], Accuracy: 0.890
Fold:  3, Training/Test Split Distribution: [3510 3510], Accuracy: 0.887
Fold:  4, Training/Test Split Distribution: [3510 3510], Accuracy: 0.885
Fold:  5, Training/Test Split Distribution: [3510 3510], Accuracy: 0.888
Fold:  6, Training/Test Split Distribution: [3510 3510], Accuracy: 0.887
Fold:  7, Training/Test Split Distribution: [3510 3510], Accuracy: 0.885
Fold:  8, Training/Test Split Distribution: [3510 3510], Accuracy: 0.900
Fold:  9, Training/Test Split Distribution: [3510 3510], Accuracy: 0.849
Fold: 10, Training/Test Split Distribution: [3510 3510], Accuracy: 0.865

Cross-Validation mean accuracy: 0.878 +/- 0.017

Cross-Validation top accuracy: 0.900
```

Fig. 20. K-fold Cross Validation Decision Tree

Again, it's important to note that the decision tree classifier is prone to overfitting, which means that it can have a high accuracy on the training set but low accuracy on the test set. That's why it's important to perform cross-validation to evaluate the performance of the model in different sets of data.

In this case, the accuracy for each fold varies between 0.837 and 0.899, with a mean accuracy of 0.877 and a standard deviation of 0.019. The highest accuracy was 0.899. This means that on average, the Decision Tree Classifier model is able to correctly predict the stroke occurrence with an accuracy of **87.7% +/- 1.9%** across all the 10 folds of the cross validation.

For better reading and understanding of this implemented model in our data, we looked for **Feature Importance** of the input features using the **feature_importances** attribute of the DecisionTreeClassifier object.

This way,for better accuracy, it made sense to drop the features with less importance. So we decided to test the model only with features which importance is higher than 0.01. Where we got the following results:

Accuracy for training set: 1

Accuracy for testing set: 0.87

This means that the decision tree classifier is able to accurately predict the target variable 87% of the time when using the test set after removing features that have an importance less than 0.01. We should keep in mind that the accuracy score alone is not always a good metric to evaluate the performance of a model, it is important to also consider other evaluation metrics such as precision, recall, F1-score and confusion matrix.

Classification Report:

```
Classification report:
              precision    recall  f1-score   support

           0       0.95      0.91      0.93       962
           1       0.11      0.17      0.13        60

    accuracy                           0.87      1022
   macro avg       0.53      0.54      0.53      1022
weighted avg       0.90      0.87      0.88      1022
```

Fig. 21. Classification Report Decision Tree with reduce feature set

This is the Classification Report for the Decision Tree Classifier model with the reduced feature set. We saw that accuracy for the test set is 0.87, which is an improvement over the previous model. However, the precision and recall for the positive class (class 1) are still very low, indicating that the model is not performing well for that class. The

f1-score is also low, which means that precision and recall are both low. This means that the model is not doing a good job of predicting the positive class and that it is more likely to predict negative instead of positive.

Confusion Matrix:

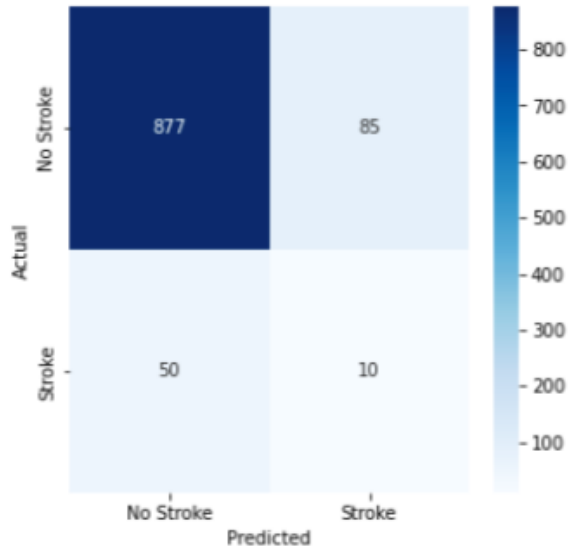

Fig. 22. Confusion Matrix Decision Tree with reduce feature set

The confusion matrix shows that the Decision Tree Classifier has correctly classified 880 samples as class 0 (non-default) and 10 samples as class 1 (default) in the test set. However, it has also misclassified 82 samples as class 1 (default) when they actually belong to class 0 (non-default) and 50 samples as class 0 (non-default) when they actually belong to class 1 (default). This can be seen in the precision, recall, and f1-score values in the classification report, where the precision and recall values for class 1 (default) are low, indicating poor performance in identifying samples of that class.

ROC Curve:

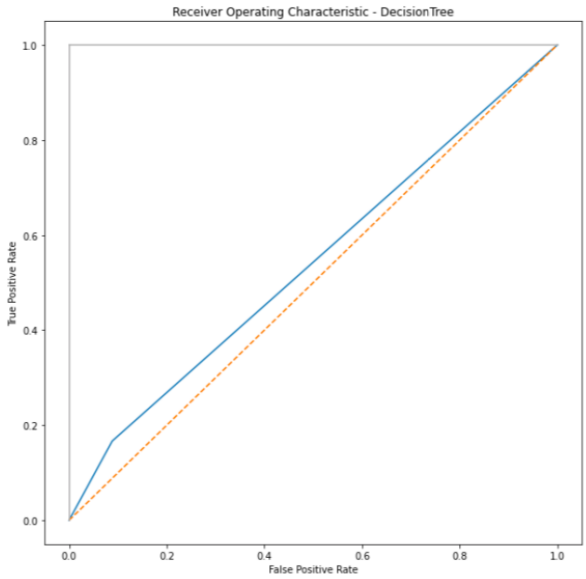

Fig. 23. Decision Tree ROC curve with reduce feature set

AUC for Decision Tree Classifier: 0.53

In this case, the AUC value of, approximately, 0.531 is relatively low (lower than before), indicating that the Decision Tree Classifier is not performing well in distinguishing between the positive and negative classes.

K-fold Cross Validation:

```
Fold:  1, Training/Test Split Distribution: [3508 3508], Accuracy: 0.812
Fold:  2, Training/Test Split Distribution: [3508 3508], Accuracy: 0.903
Fold:  3, Training/Test Split Distribution: [3508 3508], Accuracy: 0.905
Fold:  4, Training/Test Split Distribution: [3508 3508], Accuracy: 0.913
Fold:  5, Training/Test Split Distribution: [3508 3508], Accuracy: 0.909
Fold:  6, Training/Test Split Distribution: [3508 3508], Accuracy: 0.946
Fold:  7, Training/Test Split Distribution: [3508 3509], Accuracy: 0.891
Fold:  8, Training/Test Split Distribution: [3508 3509], Accuracy: 0.877
Fold:  9, Training/Test Split Distribution: [3509 3508], Accuracy: 0.928
Fold: 10, Training/Test Split Distribution: [3509 3508], Accuracy: 0.946


Cross-Validation mean accuracy: 0.903 +/- 0.037

Cross-Validation top accuracy: 0.946
```

Fig. 24. Decision Tree K-fold cross Validation with reduce feature set

We saw already that the accuracy of 1 for the training set is a sign of overfitting, which means that the model is too complex and is able to perfectly predict the training set but does not generalize well to new data. The high accuracy for the training set and lower accuracy for the test set confirms this. The cross-validation results also show that the model's performance is variable and not consistent across different folds. So overall, the model's performance on the test set is not very good, specifically for the minority class (class 1).

The AUC is also low.

The second model, which has removed the features with low importance, has a higher mean accuracy in cross validation (0.905) compared to the first model (0.877), and also a higher top accuracy (0.945) compared to the first model (0.899). This suggests that removing the features with low importance improves the model's performance. However, it is important to note that the AUC for the second model is lower (0.53) compared to the first model (0.5576). This means that the second model has a lower ability to distinguish between the two classes. So it's better to use the first model as it have a better AUC and a good accuracy.

Post Pruning

Going back to the over fitting problem mentioned above, as we said we are doing **post-pruning**. It allows the decision tree model to grow to its full depth, then removes the tree branches to prevent the model from overfitting. **Cost complexity pruning (ccp)** is one type of post-pruning technique. In case of cost complexity pruning, the **ccp_alpha** can be tuned to get the best fit model.

Scikit-learn package comes with the implementation to compute the **ccp_alpha** values of the decision tree using function **cost_complexity_pruning_path**(). With the increase in **ccp_apha** values, more nodes of the tree are pruned. [8]

AUC-ROC:

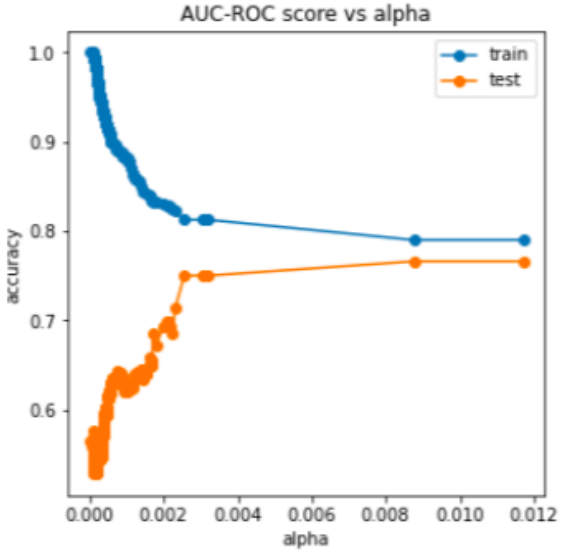

Fig. 25. AUC-ROC Decision Tree

The **ccp_alpha** parameter controls the amount of regularization applied to the tree, with a larger value of **ccp_alpha** resulting in a simpler, more regularized tree. The code is training a decision tree classifier for each value of **ccp_alpha** and appending it to a list of classifiers. Then, it is plotting the train and test AUC-ROC score for each classifier, with the x-axis representing the **ccp_alpha** value and the y-axis representing the accuracy. This allows us to visualize the trade-off between model complexity and accuracy and find the optimal value of **ccp_alpha** for the given dataset. Ideally, we would want to choose an alpha that results in a good balance between performance on the training set and the test set, without overfitting. This will typically be when the performance on the training set and the test set are similar and both are high.

*2) **Support Vector Machines (SVMs)**:* Here we present the SVMs model where we obtained these results regarding accuracy:

Accuracy for training set: 0.84

Accuracy for testing set: 0.75

This output shows the accuracy of the SVMs on the training set and the test set. The training set accuracy is, approximately, 0.84 and the test set accuracy is, approximately, 0.75.

Classification Report:

Classification report:

	precision	recall	f1-score	support
0	0.97	0.75	0.85	960
1	0.15	0.69	0.25	62
accuracy			0.75	1022
macro avg	0.56	0.72	0.55	1022
weighted avg	0.92	0.75	0.81	1022

Fig. 26. Classification Report SVMs Classifier

The classification report shows that the model is performing well in terms of precision for class 0 (97%) but not so well for class 1 (15%). The recall is also quite low for class 1 (69%) but good for class 0 (75%). The F1-score for class 0 is good (85%) but not for class 1 (25%). This suggests that the model is having trouble correctly identifying class 1 instances. The overall accuracy of the model is 75%.

Confusion Matrix:

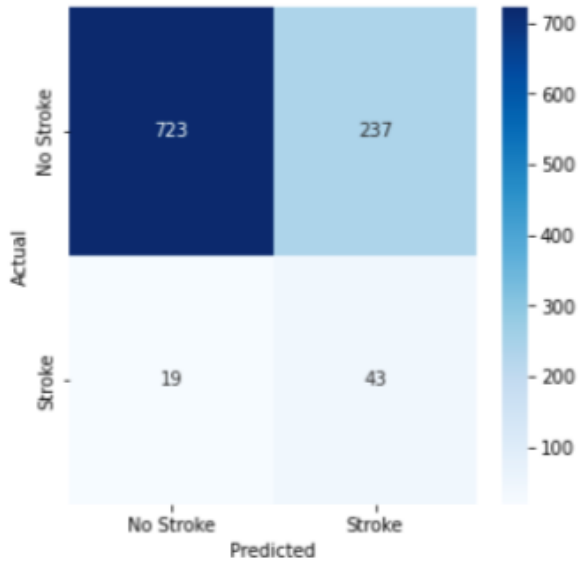

Fig. 27. Confusion Matrix SVMs Classifier

1) True positives (TP) are the number of instances that were correctly classified as class 1. In this case, it's 43;
2) False positives (FP) are the number of instances that were incorrectly classified as class 1. In this case, it's 237;
3) True negatives (TN) are the number of instances that were correctly classified as class 0. In this case, it's 723;
4) False negatives (FN) are the number of instances that were incorrectly classified as class 0. In this case, it's 19.

ROC Curve:

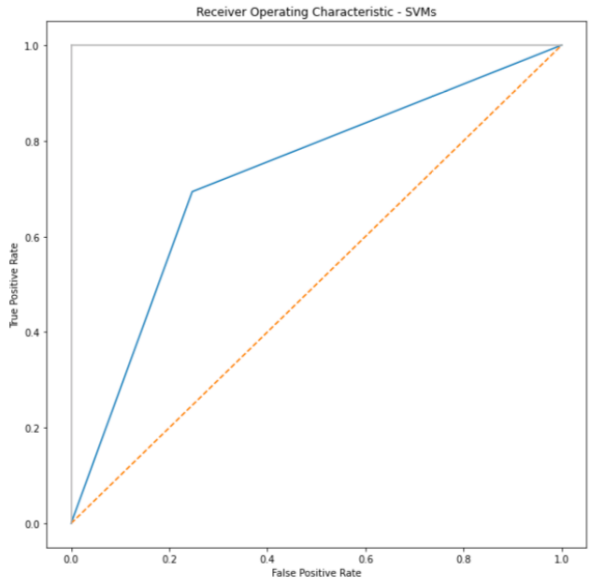

Fig. 28. ROC Curve SVMs Classifier

AUC for Support Vector Machines: 0.73

This value ranges from 0 to 1, where a value of 1 represents a perfect classifier and a value of 0.5 represents a random classifier. A value of **0.723** suggests that the classifier is performing well, but there is still room for improvement.

K-fold Cross Validation:

```
Fold:  1, Training/Test Split Distribution: [3510 3510], Accuracy: 0.796
Fold:  2, Training/Test Split Distribution: [3510 3510], Accuracy: 0.832
Fold:  3, Training/Test Split Distribution: [3510 3510], Accuracy: 0.844
Fold:  4, Training/Test Split Distribution: [3510 3510], Accuracy: 0.827
Fold:  5, Training/Test Split Distribution: [3510 3510], Accuracy: 0.832
Fold:  6, Training/Test Split Distribution: [3510 3510], Accuracy: 0.813
Fold:  7, Training/Test Split Distribution: [3510 3510], Accuracy: 0.846
Fold:  8, Training/Test Split Distribution: [3510 3510], Accuracy: 0.831
Fold:  9, Training/Test Split Distribution: [3510 3510], Accuracy: 0.832
Fold: 10, Training/Test Split Distribution: [3510 3510], Accuracy: 0.836


Cross-Validation mean accuracy: 0.829 +/- 0.014

Cross-Validation top accuracy: 0.846
```

Fig. 29. K-Cross Validation SVMs Classifier

It shows that the mean accuracy is 0.829, with a standard deviation of 0.014. The top accuracy across all the folds is 0.846. It's important to note that the cross validation results might be different than the accuracy score of the model trained with all the data, as the model is trained and tested on different subsets of the data during the cross validation.

Support Vector Machines (SVMs) do not have a direct way to calculate feature importance, as they do not inherently use feature importance for their decision boundary construction. We will apply recursive feature elimination (RFE) to check if we have improvements of our model's performance. This is a technique that recursively removes features and builds the model on the remaining features. It then calculates the feature importance based on the improvement in model performance with the addition of each feature.

By default, SVC uses the "**rbf**" kernel which does not provide feature importances. Therefore, RFE cannot be used with SVC in its default configuration.

Besides the scaling the data, handling the imbalanced dataset (that we have already done), and hyperparameter tunning (that we will be doing later), we can use esemble methods to try to improve the accuracy of the SVMs model. We do this by combining multiple models. This can improve the accuracy of your SVM model. Ensemble methods such as bagging can be used to improve the performance of the SVM model.

We decided to do a **BaggingClassifier** to try to improve the performance of the default SVM model. The **BaggingClassifier** is an ensemble method that uses multiple instances of the base estimator (in this case, the SVC) to make predictions. It works by training multiple instances of the base estimator on different subsets of the training data and then averaging (for regression) or voting (for classification) the predictions made by each instance.

The code instantiates both the **SVC** and **BaggingClassifier**, fits the **BaggingClassifier** to the training data, makes

predictions on the test set, and then prints the accuracy score of the predictions. The default hyperparameters used are:

1) **base_estimator**=None (but we chose SVM);
2) **n_estimators**=10: The number of base estimators in the ensemble;
3) **max_samples**=1.0: The number of samples to draw from X to train each base estimator;
4) **bootstrap**=True: Whether samples are drawn with replacement;
5) **oob_score**=False: Whether to use out-of-bag samples to estimate the generalization accuracy;
6) **warm_start**=False: When set to True, reuse the solution of the previous call to fit and add more estimators to the ensemble, otherwise, just fit a whole new ensemble;
7) **n_jobs**=None: The number of jobs to run in parallel for both fit and predict. None means 1 unless in a joblib.parallel_backend context. -1 means using all processors;
8) **random_state** =None: Control the randomization of the estimator;
9) **verbose**=0: The verbosity level;
10) **class_weight**= None: Weights associated with classes in the form class_label: weight;
11) **max_depth** = None: The maximum depth of the base estimators. If None then nodes are expanded until all leaves contain less than min_samples_split samples.

By using the BaggingClassifier, the performance of the SVM model should be improved, as the predictions are made by multiple instances of the base estimator, which should decrease the variance of the predictions and improve the overall accuracy.

We can see that the accuracy of the test has improved but very little (about tenths). Therefore, we will continue with our previous model (without the ensemble methods).

Comparison of the accuracy obtained by our ML models

In order to compare the accuracy of both of our ML models. We got this bar plots :

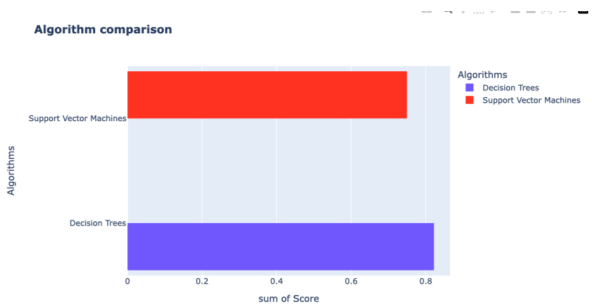

Fig. 30. Comparison of our Models

Voting Classifier

Then, we used the **Voting Classifier** to train the 2 models.

A Voting Classifier is a machine learning model that trains on an ensemble of numerous models and predicts an output based on their highest probability of chosen class as the output.

It simply aggregates the findings of each classifier passed into Voting Classifier and predicts the output class based on the highest majority of voting. The idea is instead of creating separate dedicated models and finding the accuracy for each them, we create a single model which trains by these models and predicts output based on their combined majority of voting for each output class.

Final Accuracy Score: 0.82

K-fold Cross Validation

Cross-Validation mean accuracy: 0.877 +/- 0.017

Cross-Validation top accuracy: 0.901

K-fold cross-validation was applied to the MLP classifier on the dataset and the results show that the model has achieved a good performance, with a mean accuracy of 0.877 and a standard deviation of 0.017 The top accuracy was 0.867 across different which suggests that the model is generalizing well to new data.

ROC curve

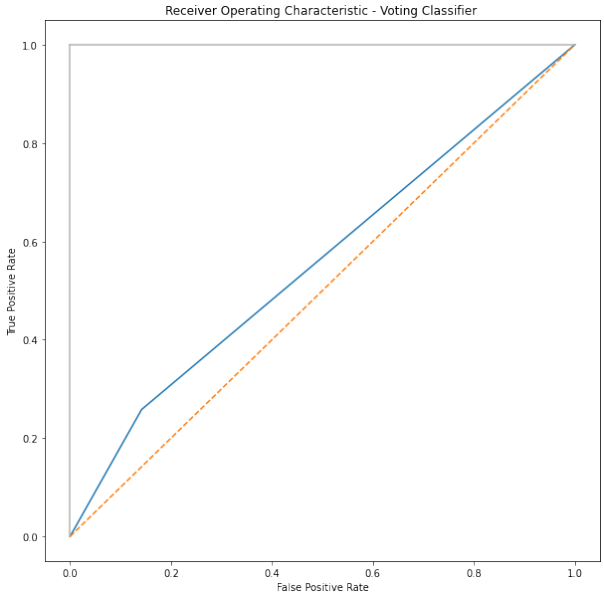

Fig. 31. Final ROC Curve

AUC for Voting Classifier: 0.56

B. Parameter Tuning using GridSearchCV

In this second part of our algorithm implementation we implemented GridSearch for the algorithms. This means that, for each algorithm used, we define a group of parameters and run all the algorithms with those parameters. Then we can see which is the best parameter combination by choosing the one with the best score.

1) Decision Tree Classifier: Parameters Tested:

1) **Criterion:** The function to measure the quality of a split. Supported criteria are "gini" for the Gini impurity and "log_loss" and "entropy" both for the Shannon information gain.
2) **Splitter:** The strategy used to choose the split at each node. Supported strategies are "best" to choose the best split and "random" to choose the best random split.
3) **Max_depth:** The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.
4) **Max_features:** The number of features to consider when looking for the best split.

In this part of the project we applied the before mentioned **Pre-pruning**. We chose the best parameters for the model using GridSearchCV.

In order to have the the best hyper-parameter tuning possible, but taking into account the possibility of the occurence of overfitting we decided to do a small loop of 4 steps in which we tested the model with different parameters. In other words we keep changing the parameters to find the ones who have a higher performance, but not making it to many times to prevent overfitting. We can see the graphs that demonstrate the performance of each step in Fig.**??** to Fig.54 in Appendix.

Classification Report

Classification report:

	precision	recall	f1-score	support
0	0.95	0.90	0.92	960
1	0.12	0.23	0.16	62
accuracy			0.86	1022
macro avg	0.53	0.56	0.54	1022
weighted avg	0.90	0.86	0.87	1022

Fig. 32. Classification report for Decision Tree with hyper-parameter tuning

After the fourth step we reached the previous classification report. It shows that the model is almost perfect in terms of precision for class 0 (99%) but quite bad for class 1 (14%). The recall, on the opposite of what has happened until this stage has a higher value for class 1 (87%) than class 0 (66%) The F1-score for class 0 is good (79%) but not for class 1 (24%). The overall accuracy of the model is 67%.

Confusion Matrix

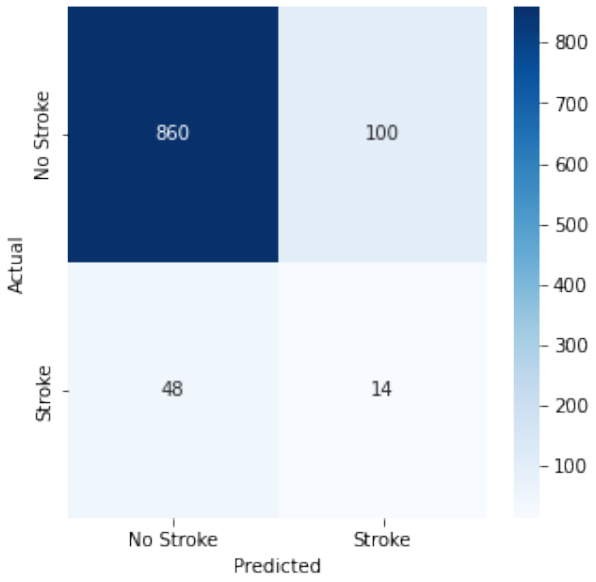

Fig. 33. Confusion matrix for Decision Tree with hyper-parameter tuning

Here 860 instances were correctly classified as not having a stroke (true negatives), while 14 instances were correctly classified as having a stroke (true positives). There were 48 instances that were incorrectly classified as not having a stroke (false negatives) and 100 instances that were incorrectly classified as having a stroke (false positives).

AUC for Multi-Layer Perception: 0.56

ROC curve

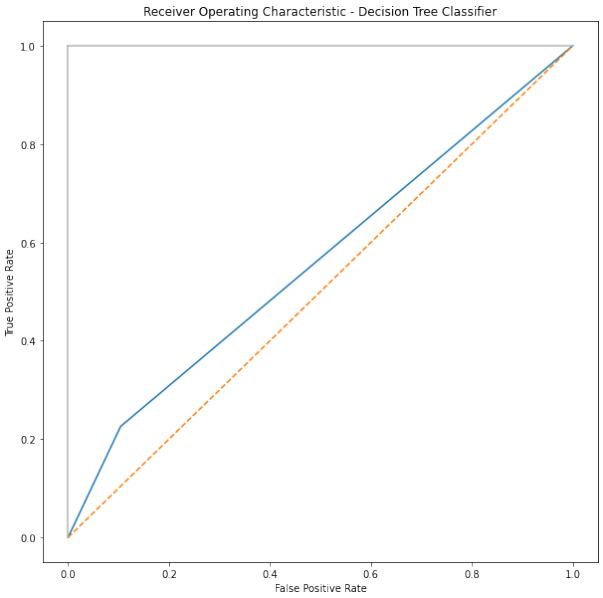

Fig. 34. ROC Curve for Decision Tree with hyper-parameter tuning

2) Suppor Vector Classifier: The main hyperparameters that can be tuned in a Support Vector Machine (SVM) model are:

1) **C:** controls the trade-off between maximizing the margin and minimizing the misclassification error. A smaller value of C will result in a larger margin, but may also result in more misclassifications
2) **kernel:** specifies the kernel type to be used in the algorithm. Common kernels include 'linear', 'poly', 'rbf' and 'sigmoid'.
3) **degree:** degree of the polynomial kernel function ('poly'). Ignored by all other kernels.
4) **gamma:** parameter for the rbf, poly and sigmoid kernel. High value of gamma will try to exact fit the as per training data set, which causes over-fitting.
5) **class_weight:** Weights associated with classes in the form class_label: weight. If not given, all classes are supposed to have weight one
6) **coef0:** independent term in kernel function. It is only significant in 'poly' and 'sigmoid'.
7) **probability:** boolean, whether to enable probability estimates. This must be enabled prior to calling fit, and will slow down that method.
8) **tol:** Tolerance for stopping criterion.

The hyperparameters such as **coef0**, **probability**, and **tol** are less likely to have a significant impact on the performance of the model and may not be necessary to include in the grid search. We will include the other hyperparameters listed.

Even though the implementation of the model is complete and without any error, it was not possible to fulfill the goal to implement GridSearchCV because SVM is a very computationally expensive model. Even with the reduction of the hyperparameters it was impossible to run the model with tuning. As a consequence we were not able to do the confusion matrix, classification report, ROC curve and AUC score. Moreover, the relevant comparison with the Decision Tree GridSearch as well as the Voting Classifier joining both predictions became hopeless.

VII. Application and Analysis of the Deep Learning Models

After applying machine learning algorithms the next step was implementing Deep Learning (DL). Again the approach was to divide the application of the algorithms in 2 parts. The first with the default values of the algorithms and then choosing the a set of hyper-parameters for our models and see if there were any improvements in their performance.

One more, for each model we apply, we also do k-fold cross-validation [6]. In addition, we present a classification report performance metrics [7], a confusion matrix [8], and a ROC curve. Only in the implementation of RNN with default values the confusion matrix and the ROC curve is not presented due to the fact that the library used is different from every other algorithm, which lead to an unsuccessful try of getting the confusion matrix.

A. Using default values

Here the main goal was to have a look on how the algorithm performs without changing any of its parameters.

*1) **Multi-Layer Perception**:* Here we present the Multi-Layer Perception where we obtained these results regarding accuracy:

Accuracy for training set: 0.86

Accuracy for testing set: 0.78

The results show that the classifier achieved an accuracy of 0.86 on the training set and an accuracy of 0.78 on the test set.

These results indicate that the classifier has achieved a good performance on both the training and test sets, with a difference between the two improved comparing to most of previous results. This means that, even though the possibility of some overfitting is not discarded, the model is more well prepared for unseen data as it is for test data.

K-fold Cross Validation

Cross-Validation mean accuracy: 0.845 +/- 0.008

Cross-Validation top accuracy: 0.854

K-fold cross-validation was applied to the MLP classifier on the dataset and the results show that the model has achieved a good performance, with a mean accuracy of 0.845 with a

standard deviation of 0.008 and a top accuracy of 0.854 across different subsets of the data, with a relatively small variation between the different folds. This suggests that the model is generalizing well to new data.

Classification Report

Classification report:

	precision	recall	f1-score	support
0	0.96	0.80	0.87	960
1	0.15	0.55	0.23	62
accuracy			0.78	1022
macro avg	0.56	0.67	0.55	1022
weighted avg	0.92	0.78	0.83	1022

Fig. 35. Classification report for MPL with default values

This is the Classification Report for the MPL Classifier model. We saw that accuracy for the test set is 0.78. However, the precision and recall for the positive class (class 1) continue to have a very low value, indicating that the model is not performing well for that class. The f1-score is also low, which means that precision and recall are both low. This means that the model is not doing a good job of predicting the positive class and that it is more likely to predict negative instead of positive.

Confusion Matrix

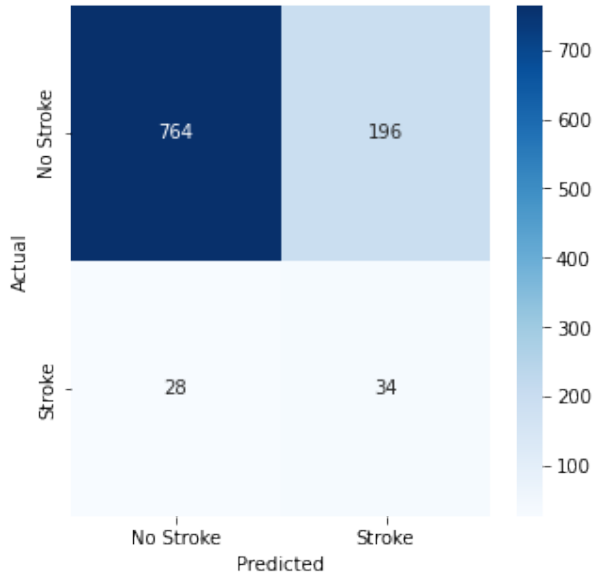

Fig. 36. Confusion matrix for MPL with default values

In this case, 753 instances were correctly classified as not having a stroke (true negatives), while 39 instances were correctly classified as having a stroke (true positives). There were 23 instances that were incorrectly classified as not having a stroke (false negatives) and 207 instances that were incorrectly classified as having a stroke (false positives).

AUC for Multi-Layer Perception: 0.67

ROC curve

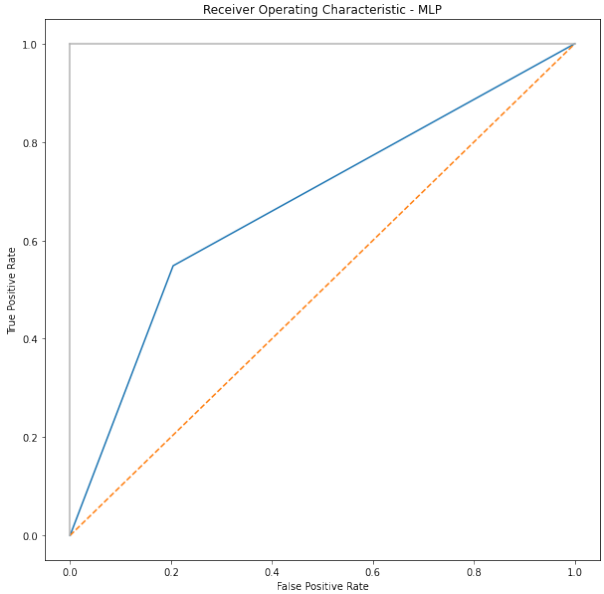

Fig. 37. ROC Curve for MPL with default values

*2) **Recurrent Neural Networks**:* Here we present the Recurrent Neural Networks where we obtained these results regarding accuracy:

Accuracy for training set: 0.81

Accuracy for testing set: 0.79

The results show that the classifier achieved an accuracy of 0.81 on the training set and an accuracy of 0.79 on the test set.

These results indicate that the classifier has achieved a good performance on both the training and test sets, with a very small difference between them, which means that the model is handling good unknown data such as the test data.

Algorithm Comparison

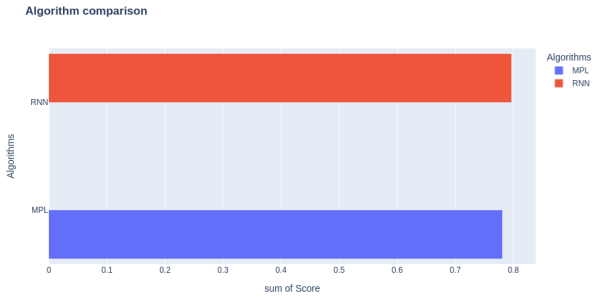

Fig. 38. Comparing Results for default values

As we can see the algorithms have a similar result with a little advantage for RNN (79%).

B. Parameter Tuning using GridSearchCV

Here the main goal was to, for each algorithm used, define a group of parameters and run all the algorithms with those parameters. Then we can see which is the best parameter combination by choosing the one with the best score. For each model it was used a cv equal to 10 for k-fold cross validation.

1) Multi-Layer Perception: Parameters tested:

1) **Hidden Layer Sizes:** This parameter is used to specify the number of neurons in the hidden layers of a multi-layer perceptron (MLP) neural network. It is a tuple of integers, where each integer corresponds to the number of neurons in a specific hidden layer. For example, hidden_layer_sizes=(50,50) would create an MLP with two hidden layers, each containing 50 neurons.
2) **Activation:** This parameter specifies the activation function used in the hidden layers of the MLP. Activation functions are mathematical functions that are applied to the output of each neuron in order to introduce non-linearity into the model. Common choices include 'relu', 'tanh', and 'sigmoid'.
3) **Solver:** This parameter is used to specify the algorithm used to optimize the MLP's weights. It can take values 'lbfgs', 'sgd', 'adam'. 'lbfgs' is an optimizer in the family of quasi-Newton methods, 'sgd' stands for Stochastic Gradient Descent, and 'adam' is an optimizer that uses an adaptive learning rate.
4) **Alpha:** This parameter is used to add a regularization term to the cost function of the MLP. Regularization helps to prevent overfitting by adding a penalty term to the cost function that discourages the model from assigning too much weight to any one feature. The alpha parameter controls the strength of the regularization term, with higher values resulting in stronger regularization.

5) **Learning Rate:** This parameter controls the step size at which the optimizer makes updates to the model's weights. A small learning rate may result in slow convergence, while a large learning rate may result in the optimizer overshooting the optimal solution. The value of the learning rate can be adjusted accordingly during the training process.

Accuracy for testing set: 0.76

The results show that the classifier achieved an accuracy of 0.76 on the test set.

Classification Report

Classification report:

	precision	recall	f1-score	support
0	0.97	0.77	0.86	960
1	0.15	0.63	0.24	62
accuracy			0.76	1022
macro avg	0.56	0.70	0.55	1022
weighted avg	0.92	0.76	0.82	1022

Fig. 39. Classification report for MPL with hyper-parameter tuning

The classification report shows that the model is performing well in terms of precision for class 0 (97%) but not so well for class 1 (15%). The recall could be better for class 1 (63%) but good for class 0 (77%). The F1-score for class 0 is good (86%) but not for class 1 (24%). This suggests that the model is having trouble correctly identifying class 1 instances. The overall accuracy of the model is 76%.

Confusion Matrix

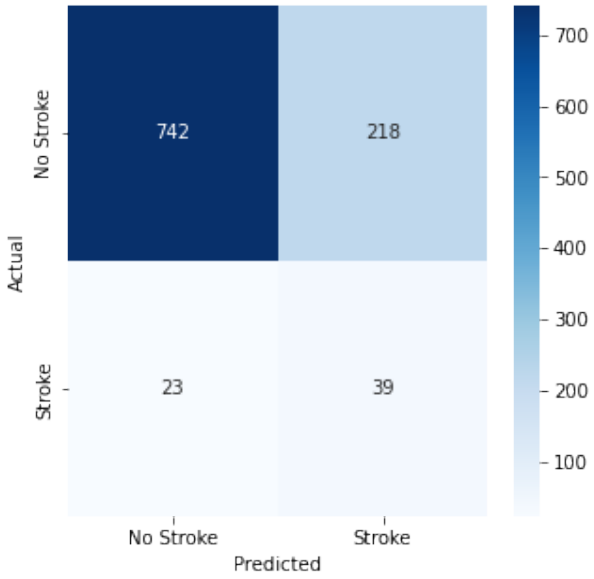

Fig. 40. Confusion matrix for MPL with hyper-parameter tuning

In this case, 742 instances were correctly classified as not having a stroke (true negatives), while 39 instances were correctly classified as having a stroke (true positives). There were 23 instances that were incorrectly classified as not having a stroke (false negatives) and 218 instances that were incorrectly classified as having a stroke (false positives).

AUC for Multi-Layer Perception: 0.70

ROC curve

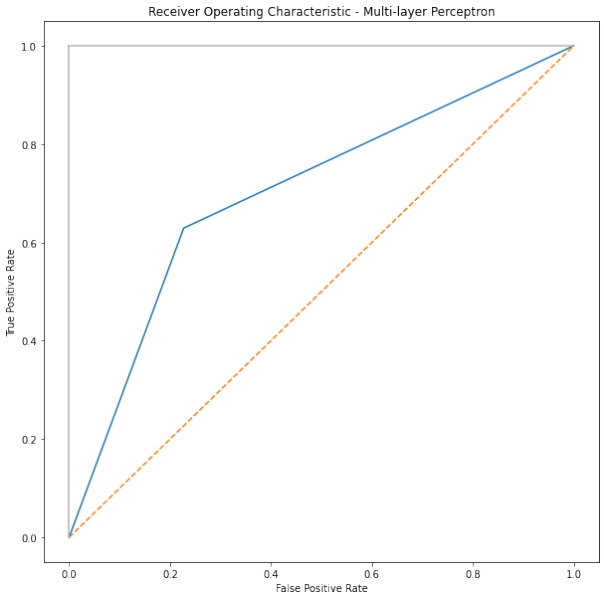

Fig. 41. ROC Curve for MPL with hyper-parameter tuning

2) Recurrent Neural Networks: Parameters tested:

For this hyper-parameter tuning we choose the tune the **neurons**. A neuron in a neural network is a processing unit that receives input, performs a computation, and produces an output. In an RNN, each neuron in a layer receives input from the previous time step, and its output is passed to the next time step. The number of neurons in a layer determines the number of computations that can be performed in parallel, which can affect the model's ability to learn complex patterns in the input data.

Increasing the number of neurons in a layer can increase the capacity of the model, allowing it to learn more complex patterns and representations. However, this also increases the risk of overfitting, as the model may start to memorize the training data instead of generalizing to new data.

On the other hand, decreasing the number of neurons in a layer can reduce the capacity of the model, making it less prone to overfitting but also making it less able to learn complex patterns.

Accuracy for testing set: 0.62

The results show that the classifier achieved an accuracy of 0.62 on the test set.

Classification Report

Classification report:

	precision	recall	f1-score	support
0	0.99	0.66	0.79	960
1	0.14	0.87	0.24	62
accuracy			0.67	1022
macro avg	0.57	0.77	0.52	1022
weighted avg	0.94	0.67	0.76	1022

Fig. 42. Classification report for RNN with hyper-parameter tuning

The classification report shows that the model is almost perfect in terms of precision for class 0 (99%) but quite bad for class 1 (14%). The recall, on the opposite of what has happened until this stage has a higher value for class 1 (87%) than class 0 (66%) The F1-score for class 0 is good (79%) but not for class 1 (24%). The overall accuracy of the model is 67%.

Confusion Matrix

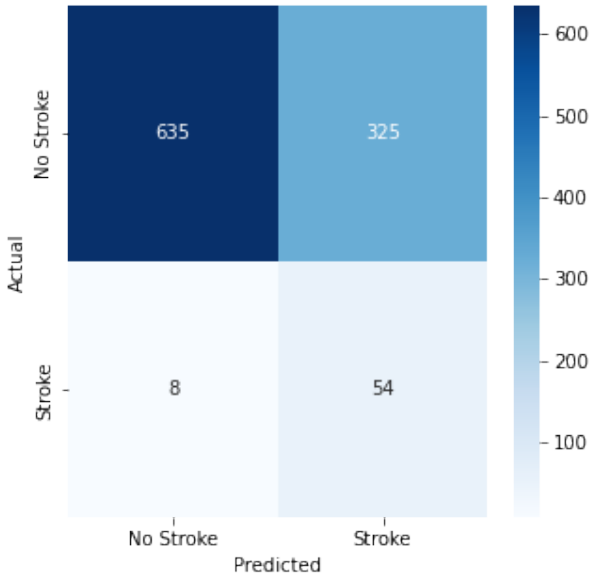

Fig. 43. Confusion matrix for MPL with hyper-parameter tuning

In this case, 635 instances were correctly classified as not having a stroke (true negatives), while 54 instances were correctly classified as having a stroke (true positives). There were 8 instances that were incorrectly classified as not having a stroke (false negatives) and 325 instances that were incorrectly classified as having a stroke (false positives).

AUC for Multi-Layer Perception: 0.77

ROC curve

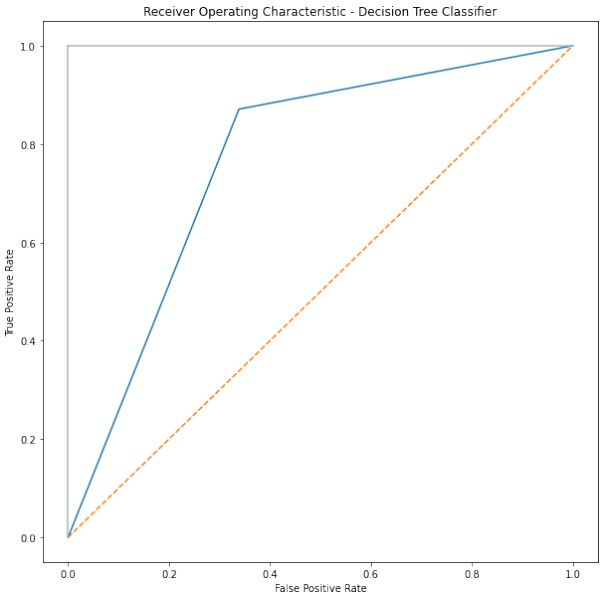

Fig. 44. ROC Curve for RNN with hyper-parameter tuning

Algorithm Comparison

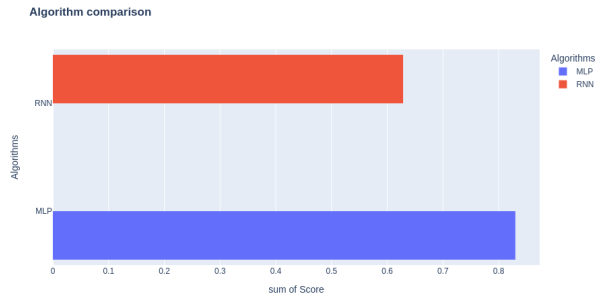

Fig. 45. Comparing Results for hyper-parameter tuning

In contrast with the comparison of the default values there is a major different between performance of both algorithm with a large advantage of th MLP (86%)

VIII. Future work and Novelties

In this following part, we will try to sum up some of the improves we would do to improvements our model and work done. For this, we will take into account the section of the state of the art review as well as the references in it used.

Example 1

In Jeena and Kumar [22] approach, the database was successfully trained and tested using Support Vector Machine (SVM) like we did. We used GridSearch to look for the best kernel functions. In this particular example, the linear kernel gave them an accuracy of 90%, much higher than ours, even though we used the same ML algorithm.

Example 2

The study of Luk, Cheung, Ho and Li [23], examines the predictors of a good outcome after rehabilitation in Chinese stroke patients with special attention to age as a factor. Functional independence measure (FIM) 90 was used to define a good outcome. They decided, unlike us, to go for a relatively simpler way to predict the age factor in stroke. Multivariate regression analysis was used to assess the independent predictors of a good outcome. Multivariate regression is a technique used to measure the degree to which the various independent variable and various dependent variables are linearly related to each other.

Example 3

In the research work of Elias and Maria [20], the aid was also machine learning (ML), several models were developed and evaluated to design a robust framework for the long-term risk prediction of stroke occurrence. The main difference between our approach and there approach was the ML algorithms used. They did a stacking method that achieved a high performance that was also validated by various metrics, such as AUC, precision, recall, F-measure and accuracy, all of them used by us. The experiment results showed that the stacking classification outperforms our methods, with an AUC of 98.9%, F-measure, precision and recall of 97.4% and an accuracy of 98%. Maybe a stacking method is a path we should follow in a next analysis.

Example 4

The report of Manisha, Eduardo and Joana [24] is the closest to our approach. They applied ML algorithms and also deep learning, just like us. They conclude the project by stating that SVM and Random Forests are efficient techniques used under each category. Although we have in common the application of SVM, we opted for Decision Tree Classifier when perhaps we could have decided for a more accurate algorithm like they did.

IX. Conclusions

In this paper, we presented a detailed analysis of patient's attributes for stroke prediction. We have covered every required criteria content. First the dataset was pre-processed, all data cleaning and pre-processing revealed to be very important to improve our model accuracy. We continue with data visualisation, all the plots and graphics were a major assistance to decide how to manipulate data. Data visualization helps people to see, interact with, and better understand the data. Whether simple or complex, the right visualization can bring everyone on the same page, regardless of their level of expertise.

Supervised learning techniques such as decision tree classifier and support vector machine were applied to the data, and the best model was chosen based on its performance on the test set. Additionally, deep learning techniques such as RNN

and MLP were also implemented and their performance was compared with the traditional machine learning models:

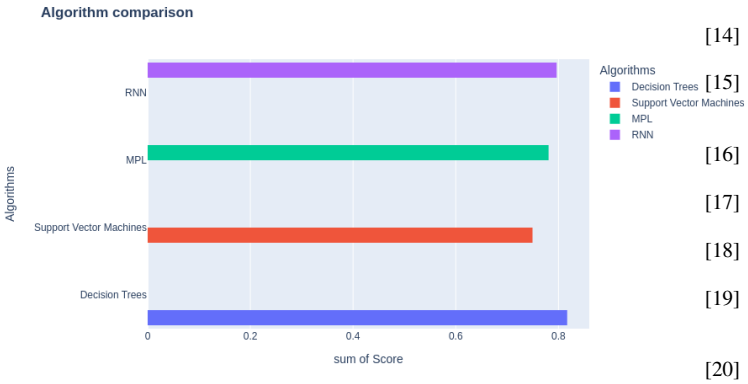

Fig. 46. Comparison between ML models and deep learning models.

Since with gridsearch the increase in accuracy was not significant, overall, we compared the machine learning and deep learning models with the default hyperparameters.

We can see that DT was the best algorithm followed by the deep learning algorithms, RNN and MPL, respectively. The algorithm with the worst score was the SVM.

Concluding, this project highlights the importance of using data pre-processing, visualization and supervised learning together in order to achieve the best performance in predicting stroke and also how to tune the models to get the best accuracy and try to prevent them from overfitting.

REFERENCES

[1] Classification report in machine learning. https://thecleverprogrammer.com/2021/07/07/classification-report-in-machine-learning/. Accessed: 2022-12-18.

[2] Correlation matrix: Definition. https://www.statisticshowto.com/correlation-matrix/. Accessed: 2022-12-17.

[3] A gentle introduction to k-fold cross-validation. https://machinelearningmastery.com/k-fold-cross-validation/. Accessed: 2022-12-18.

[4] Handling imbalanced data with smote. https://www.linkedin.com/pulse/handling-imbalanced-data-smote-fabio-rebecchi/. Accessed: 2022-12-17.

[5] Interquartile range. https://www.scribbr.com/statistics/interquartile-range/. Accessed: 2022-12-18.

[6] Max-min normalization technique. https://towardsdatascience.com/everything-you-need-to-know-about-min-max-normalization-in-python-b79592732b79. Accessed: 2022-12-17.

[7] Neural network models (supervised). https://scikit-learn.org/stable/modules/svm.html. Accessed: 2022-12-17.

[8] Post pruning decision trees with cost complexity pruning¶. https://scikit-learn.org/stable/auto_examples/tree/plot_cost_complexity_pruning.html. Accessed: 2022-12-18.

[9] Redes neurais recorrentes (rnn) com keras. https://www.tensorflow.org/guide/keras/rnn. Accessed: 2022-12-17.

[10] sklearn.metrics.confusion_matrix. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html. Accessed: 2022-12-18.

[11] sklearn.metrics.confusion_matrix. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html. Accessed: 2022-12-18.

[12] sklearn.preprocessing.labelencoder. https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html#sklearn-preprocessing-labelencoder. Accessed: 2022-12-17.

[13] sklearn.tree.decisiontreeclassifier. https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html. Accessed: 2022-12-17.

[14] Smote. https://imbalanced-learn.org/stable/references/generated/imblearn.over_sampling.SMOTE.html. Accessed: 2022-12-17.

[15] Stroke, cerebrovascular accident. https://www.emro.who.int/health-topics/stroke-cerebrovascular-accident/index.html. Accessed: 2022-12-17.

[16] Stroke prediction dataset. https://www.kaggle.com/datasets/fedesoriano/stroke-prediction-dataset/discussion/232298. Accessed: 2022-12-17.

[17] Support vector machines. https://scikit-learn.org/stable/modules/svm.html. Accessed: 2022-12-17.

[18] Violin plots 101: Visualizing distribution and probability density. https://mode.com/blog/violin-plot-examples/. Accessed: 2022-12-18.

[19] Luo J. Alanazi EM, Abdou A. Machine Learning Algorithms: Development and Evaluation of Prediction Models. *JMIR formative research*, 2021.

[20] Elias Dritsas and Maria Trigka. Stroke risk prediction with machine learning techniques. *Sensors*, 22(13), 2022.

[21] SM Hanifa and K Raja-S. Stroke risk prediction through non-linear support vector classification models. *International Journal of Advanced Research in Computer Science*, 1(3):47–53, 2010.

[22] R S Jeena and Sukesh Kumar. Stroke prediction using svm. In *2016 International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCICCT)*, pages 600–602, 2016.

[23] Ho Li Luk, Cheung. Does Age Predict Outcome in Stroke Rehabilitation? A Study of 878 Chinese Subjects. *Cerebrovascular Diseases*, 2006.

[24] Manisha Sanjay, Eduardo Ferme, and Joana Camara. Machine Learning for Brain Stroke: A Review. *Cerebrovascular Diseases*, 2006.

[25] Alfredo Vellido. The importance of interpretability and visualization in machine learning for applications in medicine and health care. *Neural Computing and Applications*, 2020.

APPENDIX

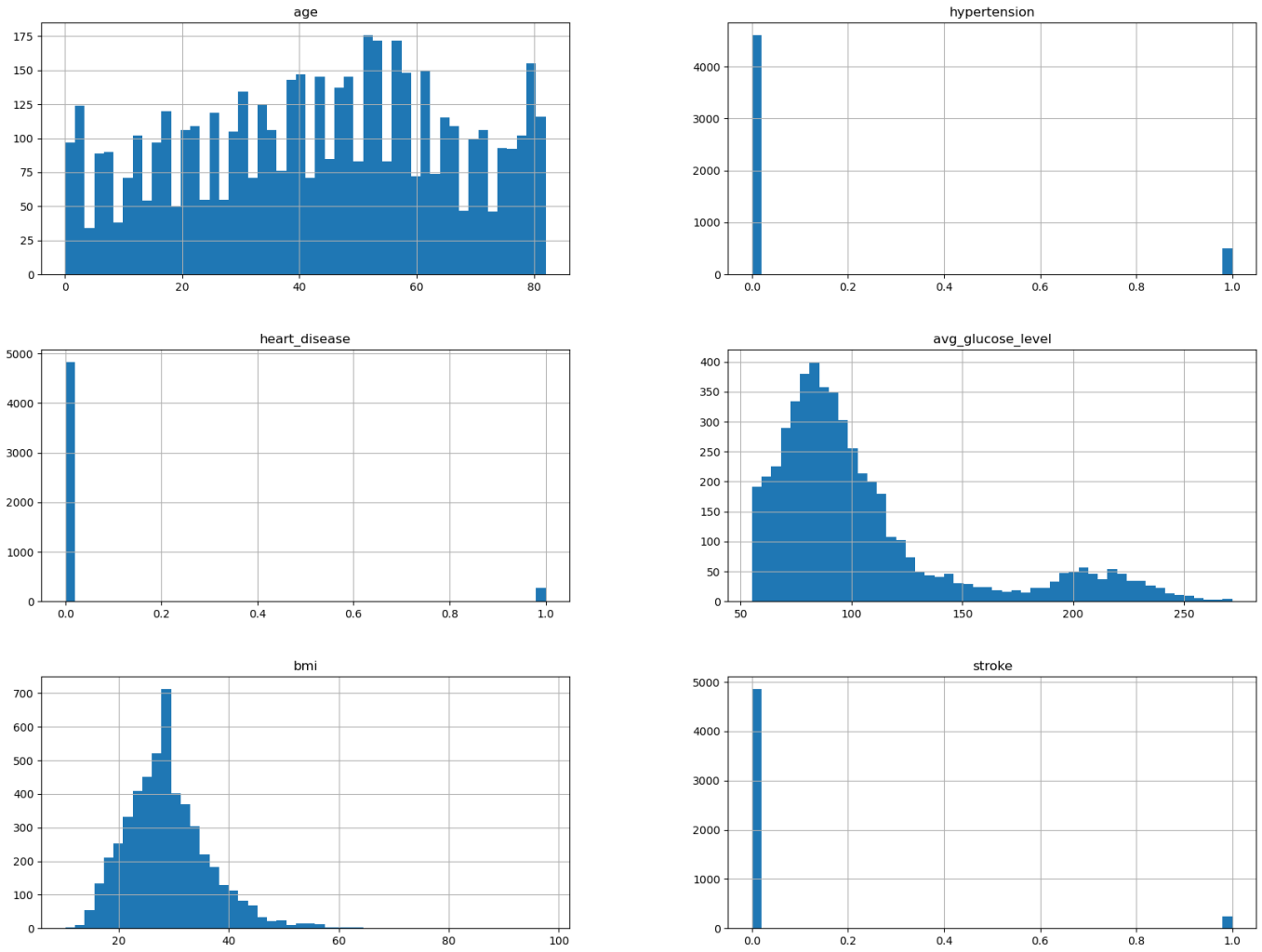

Fig. 47. Features Distribution.

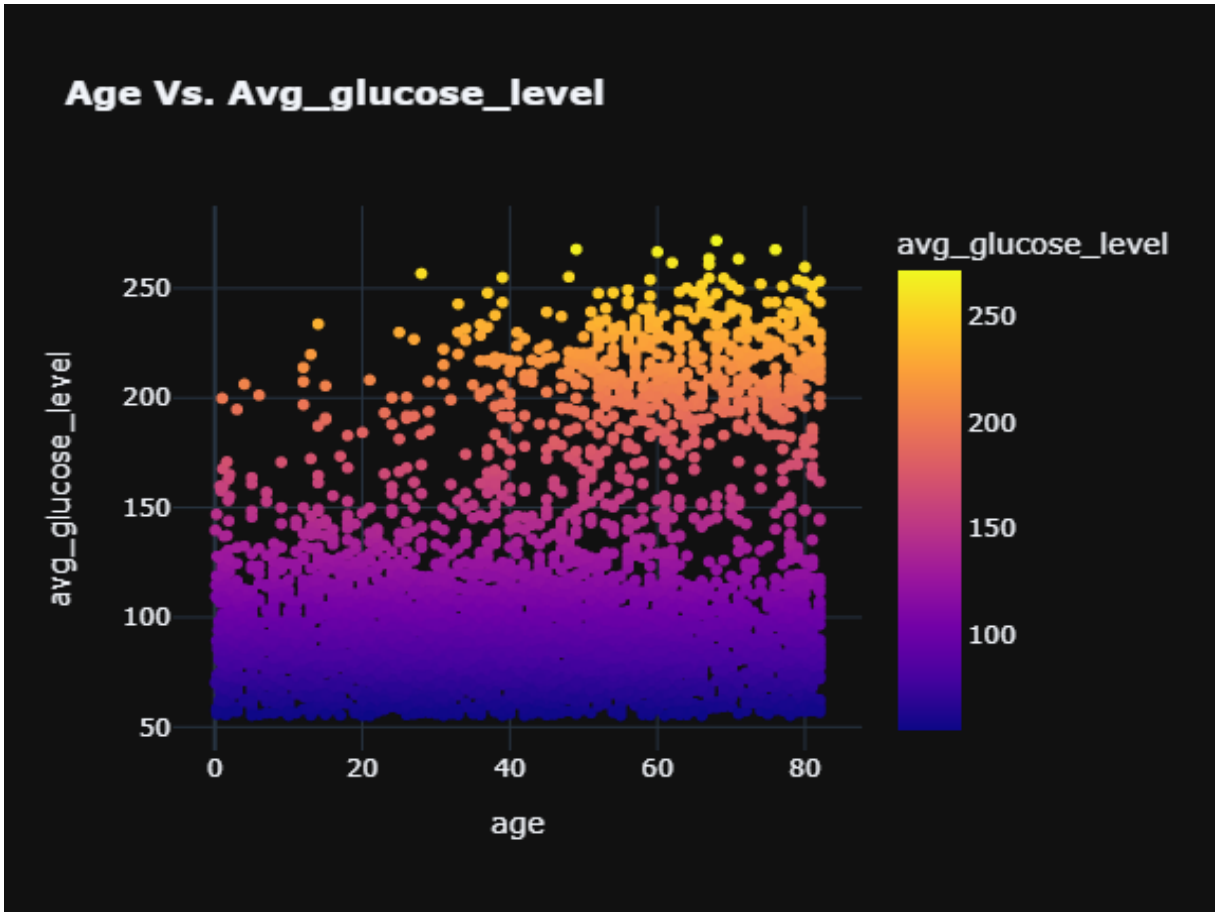

Fig. 48. ”age” and ”avg_glucose_level” Distribution.

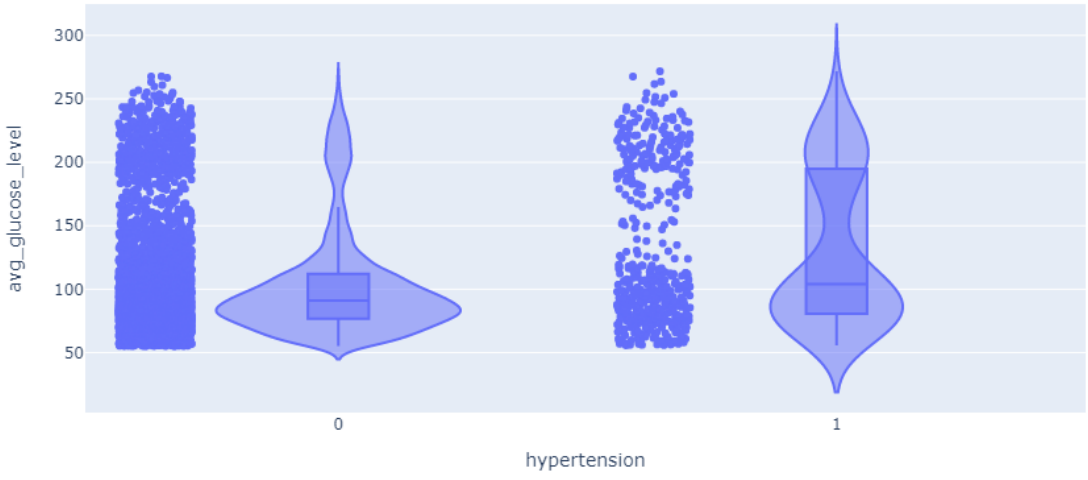

Fig. 49. ”Hypertension” Vs. ”Avg_glucose_level”.

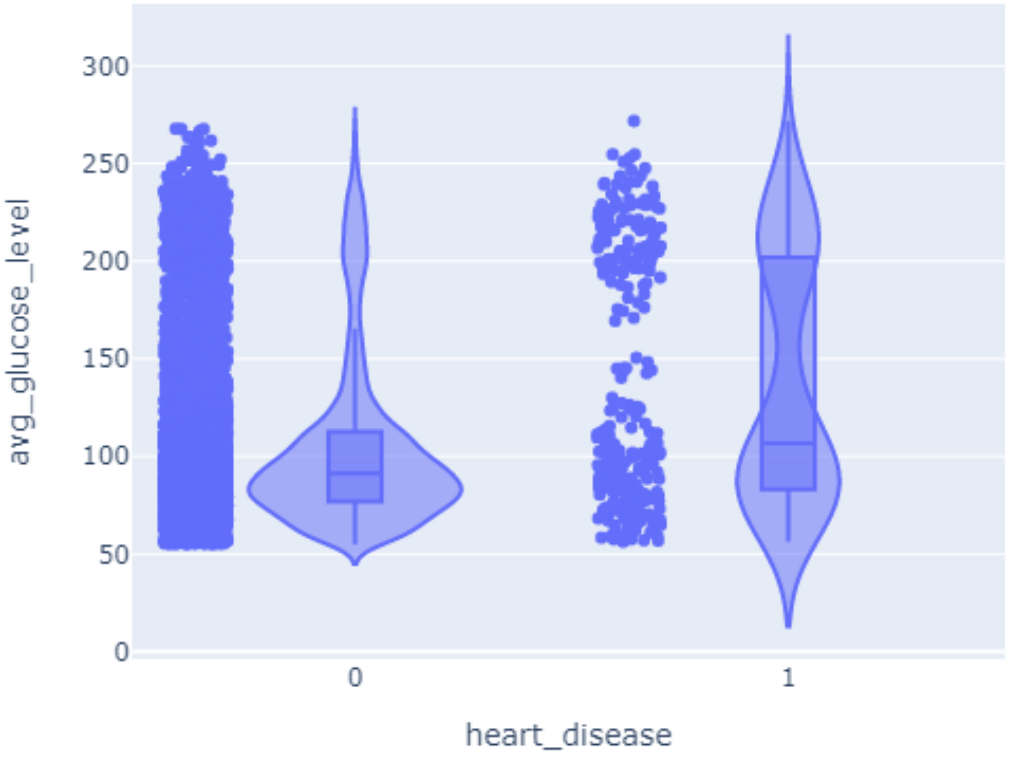

Fig. 50. ”Heart_disease” Vs. ”Avg_glucose_level”.

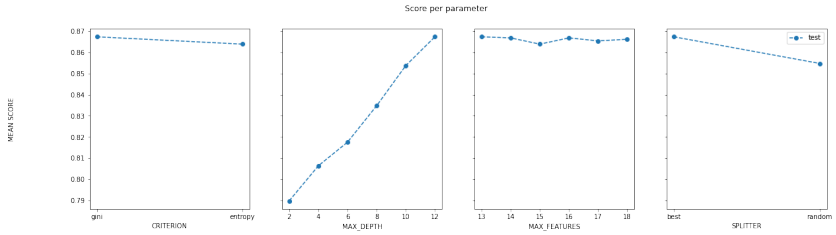

Fig. 51. Score per parameter step 1

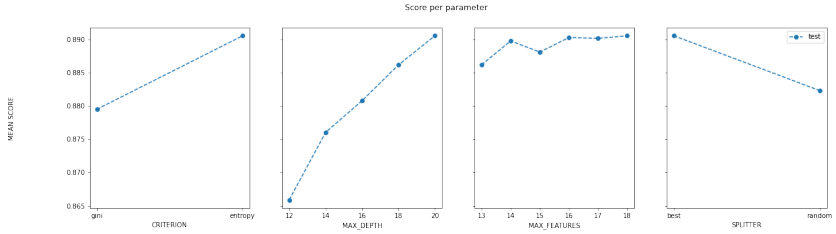

Fig. 52. Score per parameter step 2

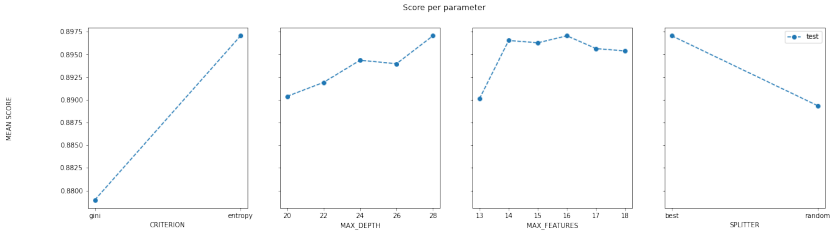

Fig. 53. Score per parameter step 3

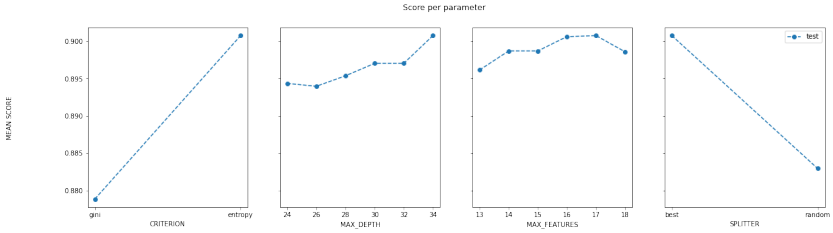

Fig. 54. Score per parameter step 4