MUHAMMED SAİF KHAN
12/20/22

CS 447 - PROJECT REPORT: STROKE PREDICTION

Approaches:

I have used **decision Tree** and **random Forest** to approach this problem.

Based on how a prior set of questions were answered, a **decision tree** is a type of supervised machine learning that is used to categorize or generate predictions. The model is supervised learning in nature, which means that it is trained and tested using data sets that contain the required categorization.

For a more precise prediction, **random forest** produces numerous decision trees that are then combined. The Random Forest model is based on the idea that several uncorrelated models (the various decision trees) work significantly better together than they do separately.

Difficulties:

The parts I found most difficult were:

- The preprocessing and analysis of data, in which I had to find out if there were any null values present in the dataset and removing them; looking for patterns and correlations between certain features of the dataset, such as no. of patients affected by stroke, gender vs stroke, types of smokers, and gender vs smoking types of smokers, and plotting graphs to illustrate these correlations. Below are the graphs of the patterns and correlations mentioned above.

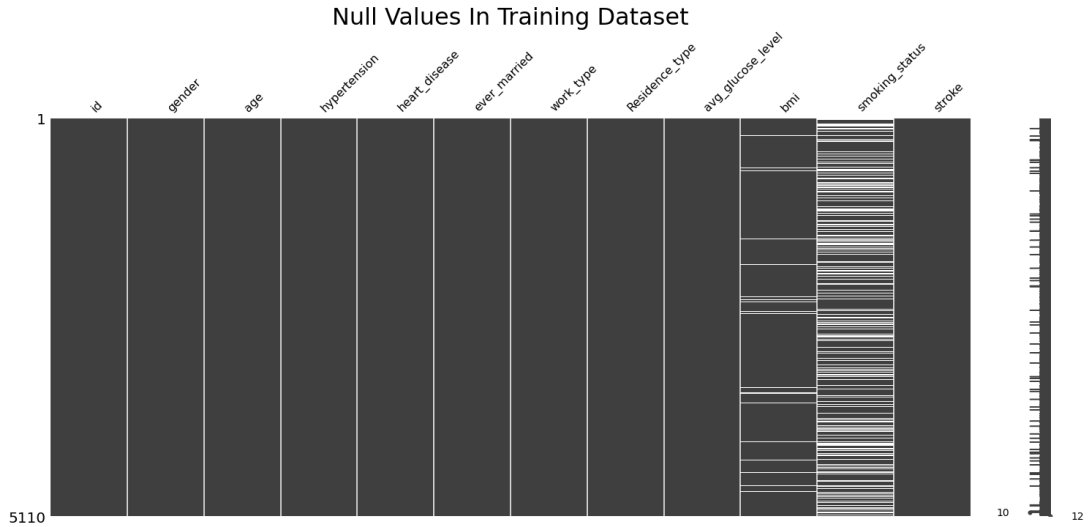

Figure 1: (Before Removing Null Values)

CS 447 - PROJECT REPORT: STROKE PREDICTION

Difficulties:

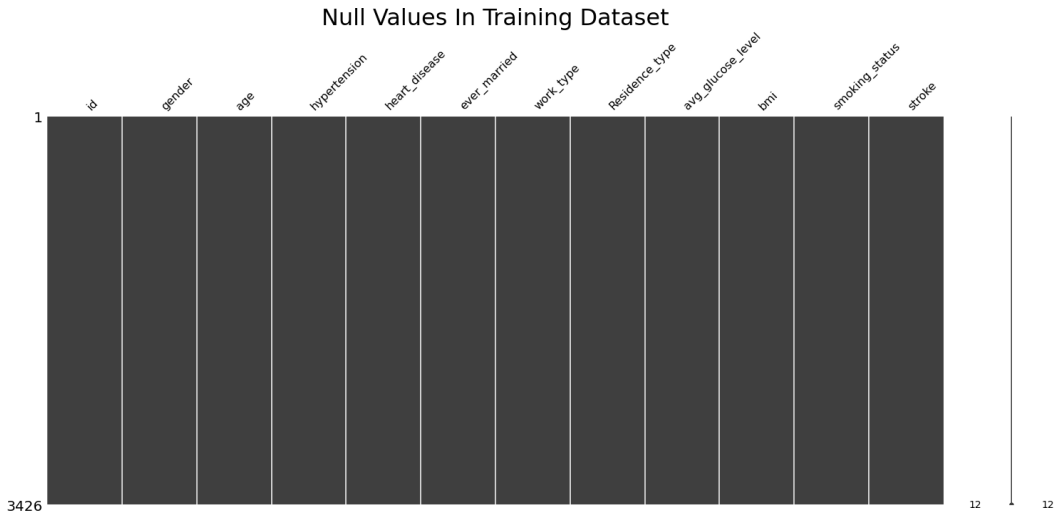

Figure 2: (After Removing Null Values)

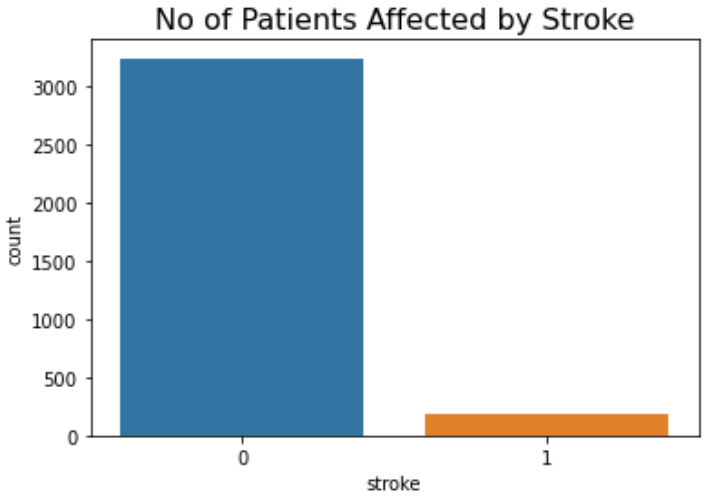

Figure 3: No. of Patients Affected by Stroke

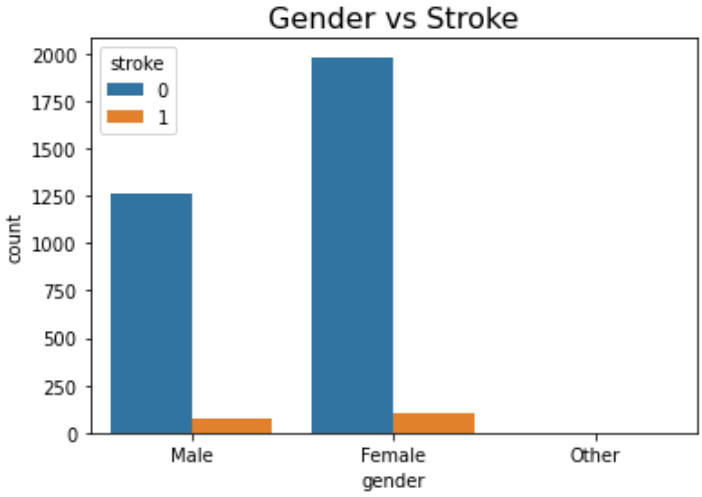

Figure 4: Gender vs Stroke

CS 447 - PROJECT REPORT: STROKE PREDICTION

Difficulties:

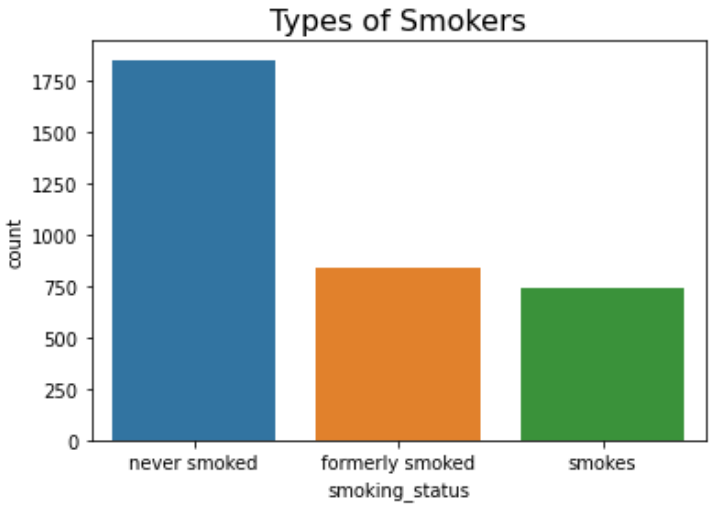

Figure 5: Types of Smokers

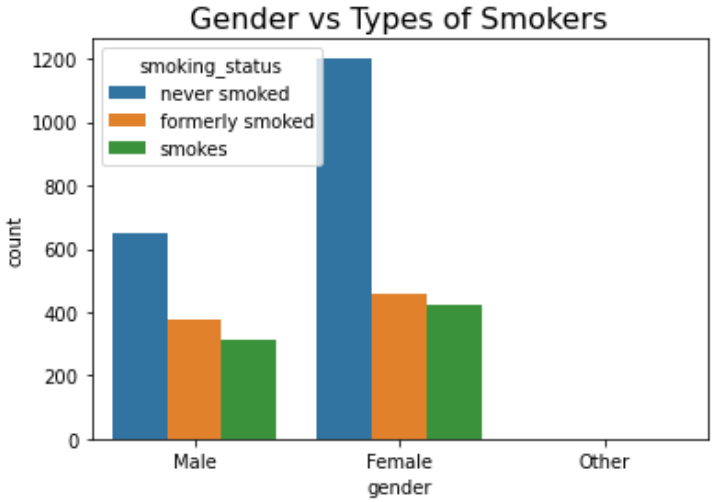

Figure 6: Gender vs Types of Smokers

- Another difficulty I had was making sure of avoiding overfitting, so I first tried to implement normalizing the dataset and then use cross-validation.

CS 447 - PROJECT REPORT: STROKE PREDICTION

Comparison of Approaches:

Implementing both the decision tree and random forest approaches were relatively similar in all aspects because a random forest is in essence just a combination of multiple decision trees, hence the name.

Implementation of both approaches: - I first declared the variables ***dtc*** and ***rfc*** for decision tree classifier and random forest classifier with the commands ***dtc=DecisionTreeClassifier(max_depth=3)*** and ***rfc=RandomForestClassifier(n_estimators=75)***. I played around with many of their parameters and their respective values and ended up just using the **max_depth** parameter for ***dtc*** and setting its value to **3**, and the **n_estimators** parameter for ***rfc*** and setting its value to **75**, since these were the best combinations I found by trial-and-error to give the best results. I then fitted the data to the classifiers for them to estimate/predict the result from the testing dataset and stored these calculated results in the respective variables ***dtc_y_pred*** and ***rfc_y_pred***.

Success rates and performance: - Both approaches have very similar success rates, with random forest having slightly higher accuracy, but also slightly slower running times. The larger the dataset, the bigger the difference between these two metrics become, which means, random forest would have significantly higher success rates than decision tree, but significantly slower run times too with larger datasets.

Increasing Success Rates:

I tried increasing the success rates by shuffling the data and cleaning the dataset, experimenting with the combination of parameters, experimenting with the parameters' values, label encoding and applying normalization, applying PCA, and finally, I used cross-validation to check for underfitting/overfitting.

For example, below is a comparison of the accuracy of random forest before and after applying PCA:

```
======================
Model 2: Random Forest
======================
Random Forest test score: 0.9387755102040817

Random Forest classification report:
              precision    recall  f1-score   support

         0.0       0.94      1.00      0.97       644
         1.0       1.00      0.00      0.00        42

    accuracy                           0.94       686
   macro avg       0.97      0.50      0.48       686
weighted avg       0.94      0.94      0.91       686

Random Forest confusion matrix:
[[644   0]
 [ 42   0]]

Mean 5-Fold Cross Validation score for Random Forest: 0.9454163563235514
```

Figure 7: Random Forest Results Before Applying PCA

CS 447 - PROJECT REPORT: STROKE PREDICTION

Increasing Success Rates:

```
============================
Applying PCA on Random Forest
============================
Random Forest test score after PCA: 0.9463243873978997

Random Forest classification report after PCA:
              precision    recall  f1-score   support

         0.0       0.95      1.00      0.97       810
         1.0       1.00      0.02      0.04        47

    accuracy                           0.95       857
   macro avg       0.97      0.51      0.51       857
weighted avg       0.95      0.95      0.92       857

Random Forest confusion matrix after PCA:
[[810   0]
 [ 46   1]]

Mean 5-Fold Cross Validation score for Random Forest after PCA: 0.9468766359515652
```

Figure 8: Random Forest Results After Applying PCA

Further Questions:

What did you do to solve the unbalanced data if you have in your problem?

- I have not done so.

What did you do to solve missing values, dirty or noisy data problems?

- I removed all rows which contained null values and normalized the dataset.

Did you use dimension transformation like PCA or LDA, why?

- I used PCA because it performs better when there are fewer samples in each class as opposed to LDA, which performs better with big datasets that contain several classes. Also, dimensionality reduction reduces space used up by data and time consumed to run algorithms.

Did you check the underfitting or overfitting possibility and how did you get rid of it?

- I checked for the possibility of underfitting/overfitting by first normalizing the data set and using cross-validation on top of it.

Did you use any regularization?

- No.

Did you implement segmentation / clustering before the classification or prediction steps? Why or why not?

- No, I did not, because I did not follow the KDD project management method (in which it is a requirement to regularize the data).

MUHAMMED SAİF KHAN
12/20/22

CS 447 - PROJECT REPORT: STROKE PREDICTION

Further Questions:

Which project management method did you use (e.g., SEMMA, CRISP-DM, or KDD?) and why did you pick this method?

- I used the CRISP-DM project management method because it is a simple and flexible project management method in a sense that; transitions between stages of this project management method can be reversed without starting from the very beginning.

Which step was the most difficult step and why?

- I found the preprocessing step to be the most difficult because the data was a bit messy, and I needed to clean it then analyze it.

How did you optimize the parameters of your algorithms?

- By trial and error.

What were the best parameters and why?

- For decision tree, the best parameter I found to be was the **max_depth** parameter, and **n_estimators** for random forest.

How did you find these parameters, and do you think you can use the same parameters for other datasets in the future for the same problem?

- As mentioned earlier, I found these by trial and error. These parameters indeed can be used in the future for other datasets.

Libraries Used:

- pandas
- matplotlib.pyplot
- seaborn
- missingno
- statistics
- sklearn

Dataset Source:

https://www.kaggle.com/datasets/fedesoriano/stroke-prediction-dataset