Predicting A Stroke Using Machine Learning

GROUP-09

SUDARSHAN SAHA BISHAL
ID: 20-42915-1
Department: CSE
American International University-Bangladesh
Email: 20-42915-1@student.aiub.edu

FABIHA TASNIM
ID: 20-43426-1
Department: CSE
American International University-Bangladesh
Email: 20-43426-1@student.aiub.edu

HIMEL DATTA
ID: 19-41576-3
Department: CSE
American International University-Bangladesh
Email: 19-41576-3@student.aiub.edu

MD. SHAHRIAZ ZAMAN
ID: 18-38692-3
Department: CSE
American International University-Bangladesh
Email: 18-38692-3@student.aiub.edu

Abstract— A stroke occurs when a blood artery supplying the brain with nutrition and oxygen becomes clogged or ruptures, resulting in a blockage of the street. Symptoms may appear when the brain's blood supply and other nutrients are cut off. The World Health Organization (WHO) cites stroke as one of the leading global causes of death and impairment. Delicate problems are validated using machine learning to the point where it can predict their patterns with the fewest probable errors. Several machine learning models have been created to estimate the probability of developing a stroke. To develop four distinct models for accurate prediction, this study employs a variety of physiological characteristics and machine learning algorithms, including Support Vector Machine, Logistic Regression, Naive Bayes Classifier, and Random Forest Classification. Since Logistic Regression had the most excellent recall and good accuracy, it was initially chosen as the technique for the imbalanced dataset. However, Support Vector Machine proved the most effective method with the most excellent f1 score after balancing the dataset using SMOTE. The open-access Stroke Prediction dataset was utilized in the method's development. Several model comparisons have proved the robustness of the models, and the research analysis may be used to infer the strategy.

Keywords— machine learning, smote, stroke, support vector machine, random forest, logistic regression, naïve bayes classifier, k-nearest neighbor, confusion matrix.

1. PROJECT OBJECTIVE

Is it feasible to develop a model that can generate reliable predictions for stroke? The model in question must possess sufficient strength to assure the outcomes. This implies that the model's results should not solely rely on chance. Moreover, it is imperative to enhance the level of performance through all practical methods to the greatest extent achievable. The challenge lies in the limited number of stroke diagnoses within the dataset. Thus, it is imperative to devise a solution to tackle this issue.

The current focus pertains to the domain of supervised learning. If an appropriate model can be established, it ought to be implemented in practical settings to enable individuals to evaluate their susceptibility to a stroke.

2. PROJECT METHODOLOGY

2.1 Data Collection Procedure

There exist two distinct categories of stroke. The topic of interest is ischemic strokes. Those above are cerebral vascular accidents resulting from the occlusion of an arterial vessel or, in uncommon cases, a venous vessel. The absence of differentiation between the two distinct types in the dataset poses a challenge for model training. Therefore, there exists a degree of variability among strokes within the classification.

The corpus was obtained from Kaggle's Library using comma-separated values (CSV). The dataset exhibits an imbalance as specific attributes contain missing values. The sample data shows a slight gender imbalance, with a higher proportion of females than males. Upon visual inspection, the age distribution appears to follow a normal distribution, albeit with a notable degree of variance, as evidenced by a visible fat tail.

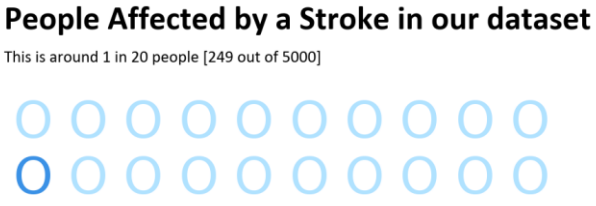

Figure-01: People affected by a stroke in the dataset.

2.2 Data Validation Procedure

Upon assessing the attributes of patients with and without stroke, it is evident that age is a crucial factor; as individual ages, their susceptibility to stroke increases. Although less apparent, there are variations in typical glucose levels and BMI. The dataset comprises 5110 instances, each characterized by 12 distinct features. The dataset consisted of 201 samples that lacked BMI values. Instead of utilizing a simplistic approach such as mean or median imputation, a fundamental decision tree model was employed. This model relied on the age and gender of all other samples to reasonably estimate the absent values, thereby enhancing the study's accuracy.

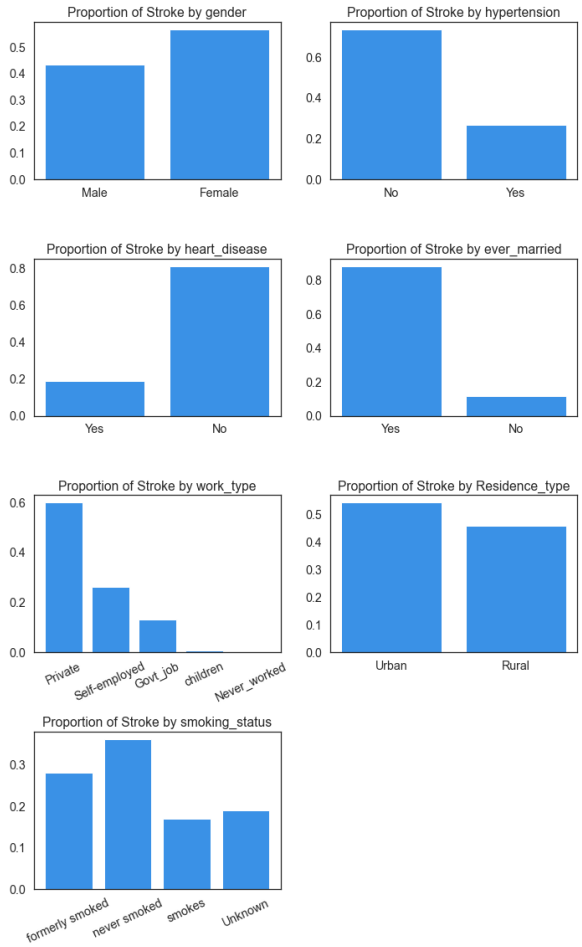

Figure-02: Relations between proportion of strokes and different attributes.

2.3 Data Preprocessing and Normalization

```
Preprocessed Data:
      id   age  hypertension  heart_disease  avg_glucose_level        bmi  \
0   9046  67.0             0              1             228.69  36.600000
1  51676  61.0             0              0             202.21  28.893237
2  31112  80.0             0              1             105.92  32.500000
3  60182  49.0             0              0             171.23  34.400000
4   1665  79.0             1              0             174.12  24.000000

   stroke
0       1
1       1
2       1
3       1
4       1

Normalized Data:
         id       age  hypertension  heart_disease  avg_glucose_level  \
0  0.123214  0.816895           0.0            1.0           0.801265
1  0.708205  0.743652           0.0            0.0           0.679023
2  0.426015  0.975586           0.0            1.0           0.234512
3  0.824928  0.597168           0.0            0.0           0.536008
4  0.021929  0.963379           1.0            0.0           0.549349

        bmi  stroke
0  0.301260     1.0
1  0.212981     1.0
2  0.254296     1.0
3  0.276060     1.0
4  0.156930     1.0
```

Figure-03: Preprocessed and Normalized Data.

2.4 Feature Extraction Procedure

In feature extraction, creating dummy variables is a technique used to convert categorical variables into numerical variables. Dummy variables are binary variables representing a particular category's presence or absence. For example, if a dataset has a categorical variable called "color" with possible values "red," "green," and "blue," dummy variables would create three new binary variables: "color_red," "color_green," and "color_blue." Each variable would take on a value of 1 if the corresponding category is present and 0 otherwise. This technique is helpful for machine learning algorithms that require numerical inputs and can improve the model's accuracy. The *dummies* function was created for the dataset used here, which creates dummies from categorical features of type string. The parts are *gender, ever_married, work_type, Residence_type,* and *smoking_status.*

```
[190]:
def dummies(df, feats: list, dropFirstAll=False) -> pd.DataFrame:

    df=df.copy()

    for f in feats:
        if len(df[f"{f}"].unique()) > 2:
            dropFirst = False
        else:
            dropFirst = True

        if dropFirstAll:
            dropFirst = True

        d = pd.get_dummies(df[f"{f}"], prefix=f, drop_first=dropFirst)
        df.drop(columns=[f], inplace=True)
        df[d.columns] = d
    return df
```

```
[191]:
dataD = dummies(
    data,
    ["gender", "work_type", "ever_married", "Residence_type", "smoking_status"],
    dropFirstAll=True
)
dataD.head(n=3)
```

[191]:

	age	hypertension	heart_disease	avg_glucose_level	bmi	stroke	gender_Male	work_type_Never_worked	work_type_Private	work_type_Self-employed	work_type_children	ever_married_Yes	Residence_type_Urban	smoking_status_fo sr
0	67.0	0	1	228.69	36.6	1	1	0	1	0	0	1	1	
1	61.0	0	0	202.21	NaN	1	0	0	0	1	0	1	0	
2	80.0	0	1	105.92	32.5	1	1	0	1	0	0	1	0	

Figure-04: Categorical Features.

As demonstrated in the analysis above, specific BMI data are absent. The loss of corresponding observations may pose a more significant challenge than imputation, owing to the minority class problem. To prevent potential data leakage, the target variable "stroke" has been excluded during the imputation process of missing BMI values.

2.5 Classification Algorithm

This project uses five classification algorithms: Support *Vector Machine, K-Nearest Neighbor, Naïve Bayes Classification, Random Forest, and Logistic Regression.*

i. Support Vector Machine (SVM): SVM is a robust algorithm used for classification tasks. It works by finding the optimal hyperplane that maximally separates the different classes in the feature space.

ii. K-Nearest Neighbor (KNN): KNN is a simple and intuitive algorithm used for classification tasks. It works by finding the k nearest data points to a new data point and assigning it the most common class label among those k neighbors.

iii. Naïve Bayes Classification: Naïve Bayes is a probabilistic algorithm for classification tasks. It works by calculating the probability of a data point belonging to a particular class based on the feature values and class label probabilities.

iv. Random Forest: Random Forest is an ensemble algorithm for classification tasks. It creates multiple decision trees and combines their outputs to make a final classification decision.

v. Logistic Regression: Logistic Regression is a simple and widely used algorithm for classification tasks. It models the relationship between the features and the probability of a data point belonging to a particular class.

2.6 Data Analysis Techniques

Cross-validation involves dividing the dataset into training and testing sets and using the training set to train the model and the testing set to evaluate its performance. This can help avoid overfitting and ensure the model is generalizable to new data.

Ensemble methods involve combining multiple models to improve their performance. Examples include bagging, boosting, and random forests. Hyperparameter tuning involves adjusting the parameters of the machine learning model to optimize its performance. This can be done using methods such as grid search or random search.

But the study only utilizes some algorithms at last. It generates an ensemble that is used.

Achieving a balance between overfitting (i.e., high prediction variance) and underfitting (i.e., high prediction bias) is crucial for the model's effectiveness. A 2-fold cross-validation technique is employed for this objective. What is the reason for limiting the number of folds to only two? The primary motivation behind this is to optimize performance. The computational time required by the code would exceed its original duration by a factor of at least two. Nonetheless, it is sufficient to employ two folds, given the considerable size of the dataset. Utilizing a substantial validation dataset in this scenario confers the benefit of enhancing the credibility of the evaluation metrics. In cases with numerous folds, the variance of the evaluation metrics can increase significantly when comparing the individual folds. Empirically, it has been noted that in many cases, both the training and validation datasets exhibit overfitting. This phenomenon arises due to the fine-tuning of hyperparameters until the outcomes satisfy the specified criteria. The occurrence of overfitting on the leaderboard in Kaggle competitions is commonly referred to as "leaderboard overfitting." Two precautionary measures will be implemented to prevent the recurrence of the aforementioned issue.

- The seed is subject to frequent alteration.

- In addition to employing a training and validation dataset, reserving a separate test dataset is customary to evaluate a model’s performance without any hyperparameter tuning. The employment of this technique is restricted to the concluding stages of the ultimate model, with the purpose of evaluating the model's response to novel, unobserved data.

2.7 Block Diagram and Workflow Diagram of Proposed Model

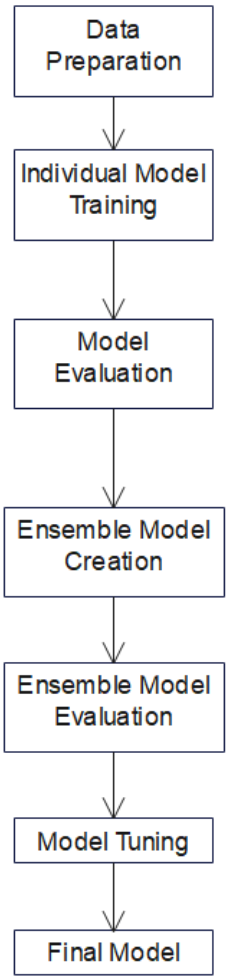

Figure-05: Block Diagram of Ensemble Model.

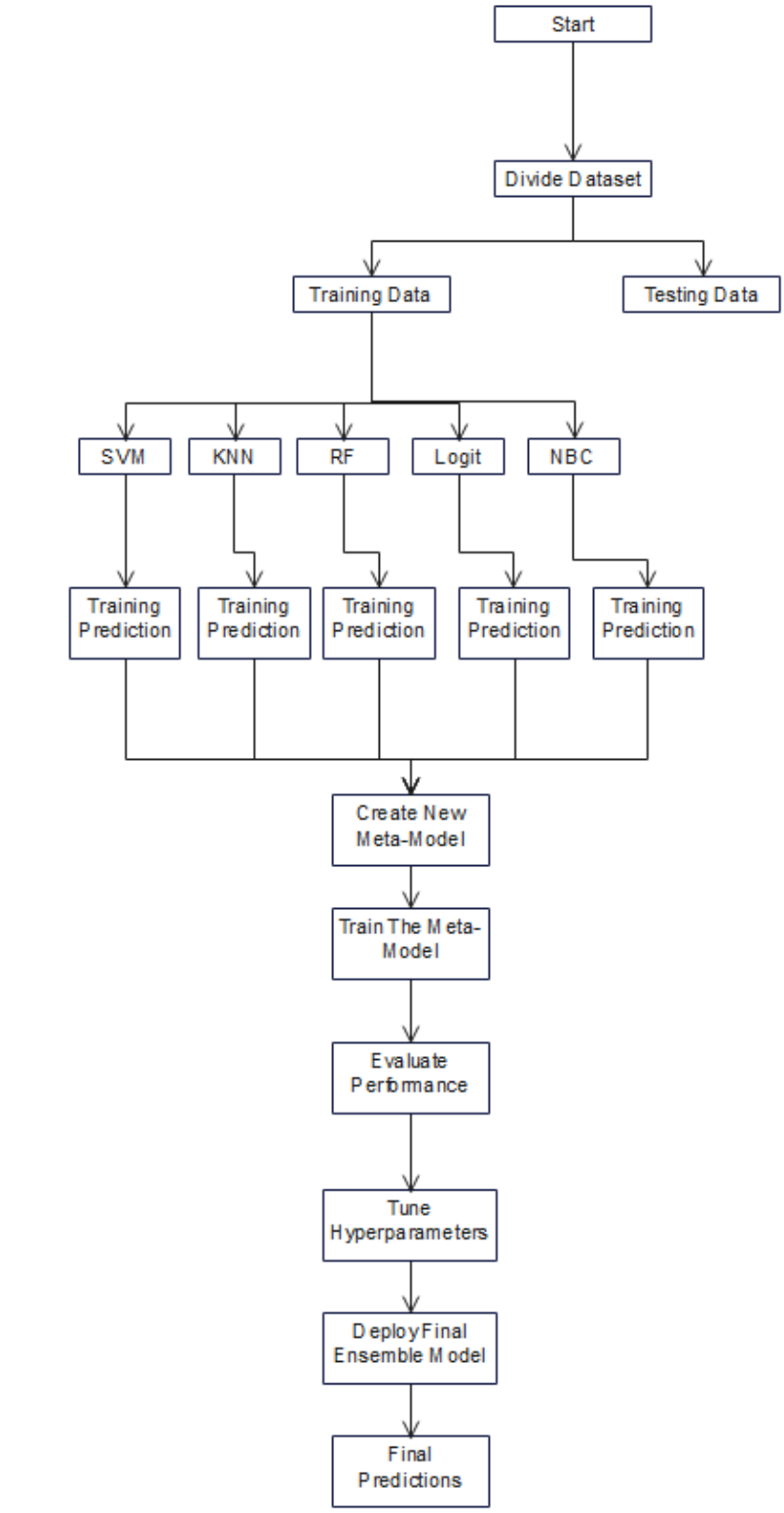

Figure-06: Workflow Diagram of Ensemble Model.

2.8 Experimental Setup and Implementations

The dataset was preprocessed by removing missing values, dropping unnecessary columns, and encoding categorical variables. The features were extracted using different methods such as dummy variables. The dataset was split into training and testing sets with a ratio of 80:20. Five classification algorithms, Support Vector Machine (SVM), K-Nearest Neighbor (KNN), Naïve Bayes Classification (NBC), Random Forest (RF), and Logistic Regression (LR) were used to train the models on the training set. The performance of the models was evaluated on the testing set using different evaluation metrics such as accuracy, precision, recall, and F1-score. Hyperparameter tuning: The hyperparameters of the best-performing models were tuned using GridSearchCV to improve their performance.

The provided training dataset is separated into k-folds and evaluates the performance for each combination of hyperparameters using k cross-validations. Once the best hyperparameter combination is found, the model is trained again with the entire training data set. Therefore, it has to be ensured that the option "refit" is not set to False. Subsequently, we will measure the performance of the best resulting hyperparameters of the corresponding model against the validation dataset. For this purpose, a confusion matrix is determined at a cut-off of 50 %. However, since the accuracy depends on this cut-off, the area under the receiver operating characteristic (ROC) curve AUC is also determined. In addition, we show where the cut-off of 50% is located on the ROC curve. If the ROC curve is flat at the beginning and steeper later, no strokes are predicted, even if the cut-off criterion is small. In this case, it would not be sufficient to look only at the AUC since it can be large regardless of the shape of the ROC.

The best-performing model is selected based on its evaluation metrics on the testing set.

3. RESULTS & DISCUSSION

An ensemble is assembled of the models with the best performance. The results are (weighted) averaged. The weighting is dependent based on the AUC performance. Higher performance should lead to a disproportionate weighting since the differences to the individual models would only be marginal with a more linear weighting.

The larger the exponent (exp), the stronger the weighting towards the methods with better-measured AUC. One could optimize this process by testing for many exponents per fold and taking the set with the best combination. But here, a greater value is considered generally (95). First, we apply the validation data set.

```
Weight Support Vector Machines fold 1 : 0.7
Weight Random Forest fold 1 : 0.01
Weight Logit fold 1 : 0.13
Weight Naive Bayes Classifier fold 1 : 0.15

Weight Support Vector Machines fold 2 : 0.15
Weight Random Forest fold 2 : 0.03
Weight Logit fold 2 : 0.77
Weight Naive Bayes Classifier fold 2 : 0.05
```

Figure-07: Ensemble Result.

Each model is included. Now let's look at the performance of the ensemble. Thereby we use a cut-off criterion, which at least keeps the accuracy on the share of the majority class and, at the same time, identifies not only the majority class.

	precision	recall	f1-score	support
0	0.96	0.99	0.97	2393
1	0.29	0.10	0.15	123
accuracy			0.94	2516
macro avg	0.62	0.54	0.56	2516
weighted avg	0.92	0.94	0.93	2516

	precision	recall	f1-score	support
0	0.95	0.99	0.97	2394
1	0.29	0.04	0.07	122
accuracy			0.95	2516
macro avg	0.62	0.52	0.52	2516
weighted avg	0.92	0.95	0.93	2516

Figure-08: Ensemble Evaluation.

3.1 Results Comparison

After using the Random Forest, SVM, Logistic Regression, KNN, and Naïve Bayes testing techniques, the results show that KNN has 0.95 of the mean f1 scores for both fold1 and fold2, then the Random Forest has the second highest value. After that, the SVM has 0.95 for fold1 and 0.87 for fold2. Logistic Regression has 0.88 for fold 1 and 0.87 for fold 2. As determined by this evaluation, the K-Nearest Neighbor model outperforms the other models. But the performance of KNN heavily depends on the value of K. So, the result varies. For the performed study, the prediction error is greater than the others. SVM, on the other hand, outscored them all and generated more precision and recall compared to the other two, earning it a higher F1 score. The research was then applied to the threshold Confusion Matrix, which exhibited enhanced precision in recognizing more strokes. For both stroke and non-stroke patients, the SVM model demonstrated greater accuracy. The SVM has the highest AUC score on the models for fold1 and fold2. The standard deviations are also the lowest.

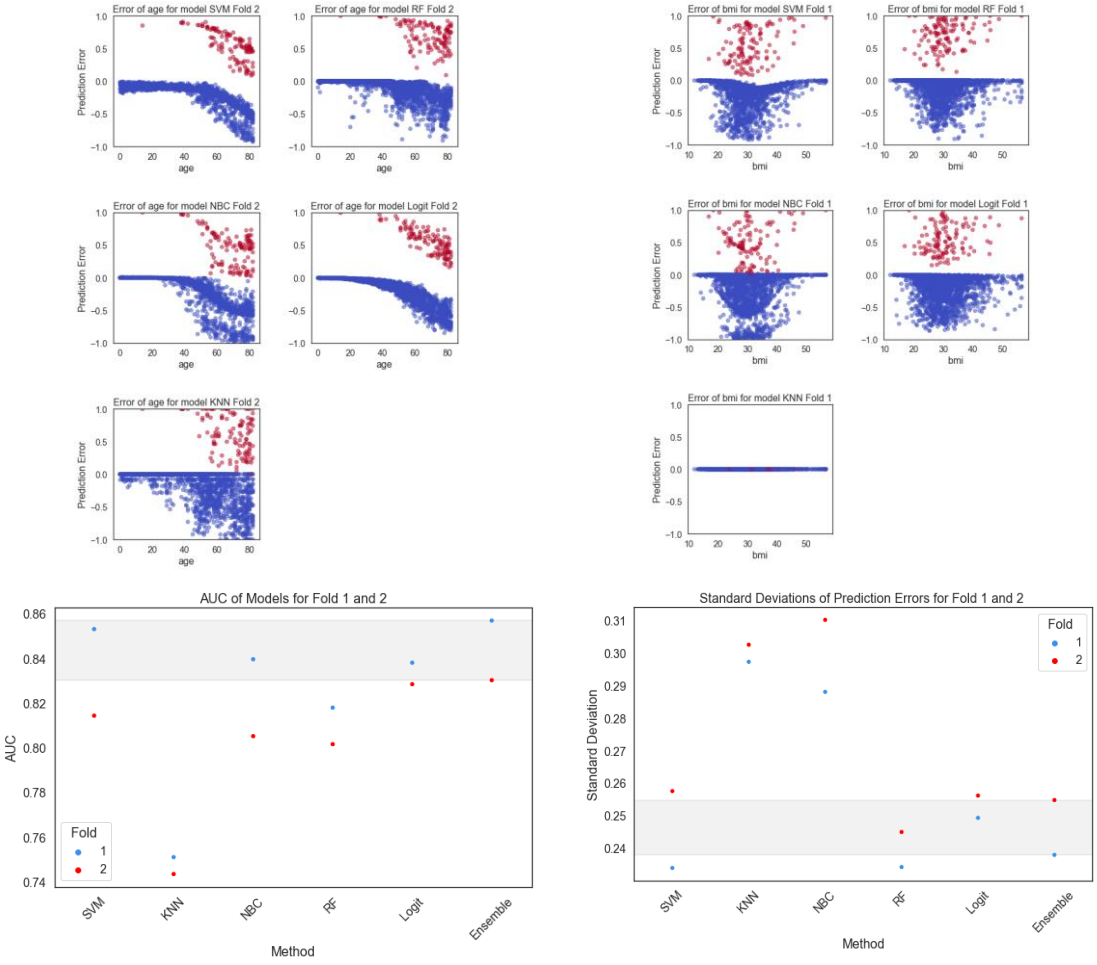

Figure-09: Comparisons Between Different Models With & Without SMOTE.

3.2 Confusion Matrix Analysis

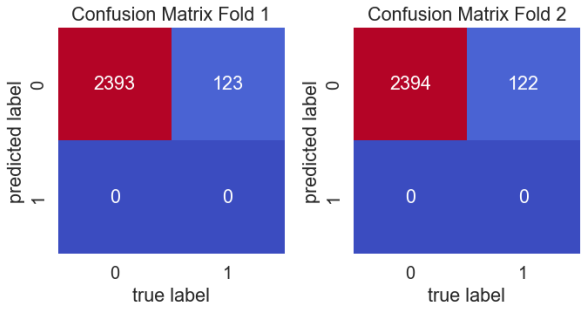

Fig-10: SVM Confusion Matrix without SMOTE

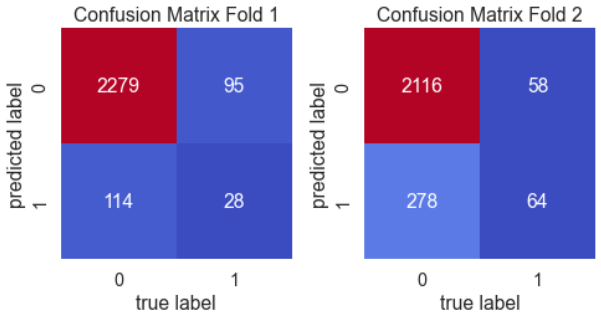

Fig-11: SVM Confusion Matrix using SMOTE

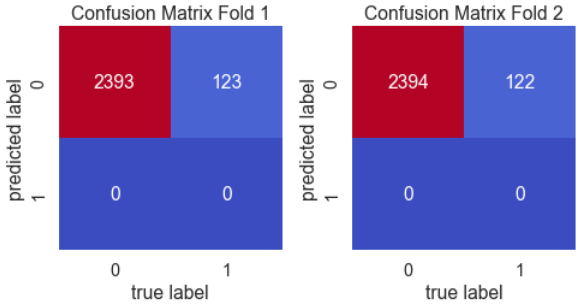

Fig-12: KNN Confusion Matrix without SMOTE

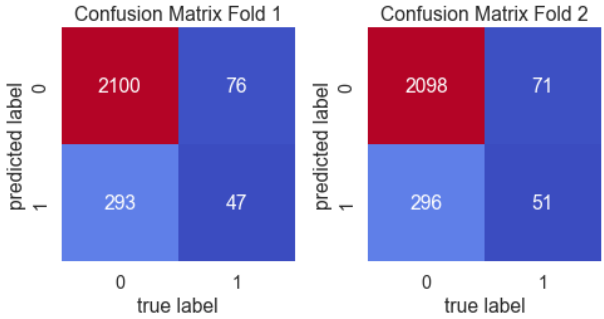

Fig-13: KNN Confusion Matrix using SMOTE

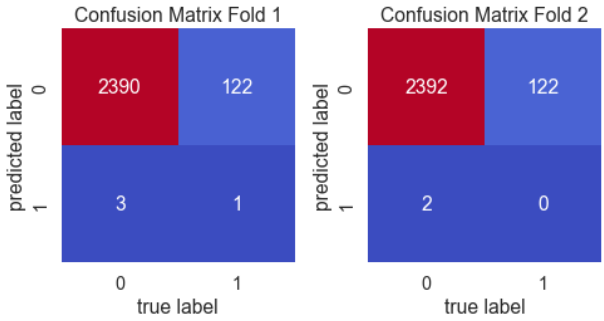

Fig-14: RF Confusion Matrix without SMOTE

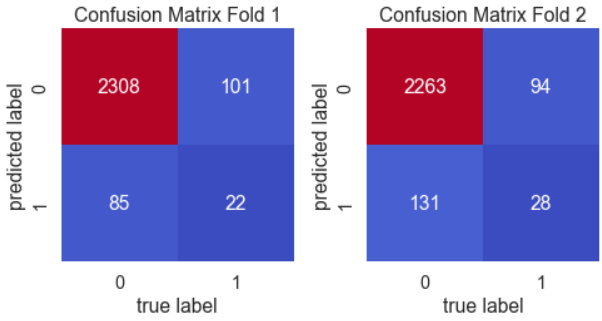

Fig-15: RF Confusion Matrix using SMOTE

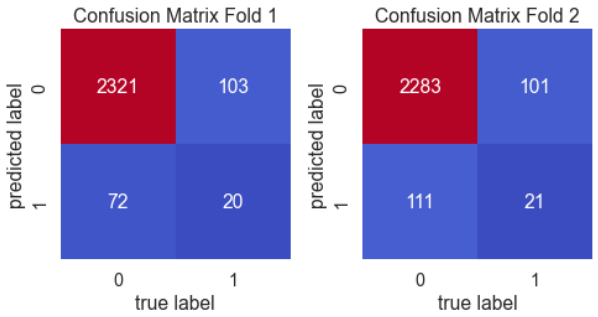

Fig-16: NBC Confusion Matrix without SMOTE

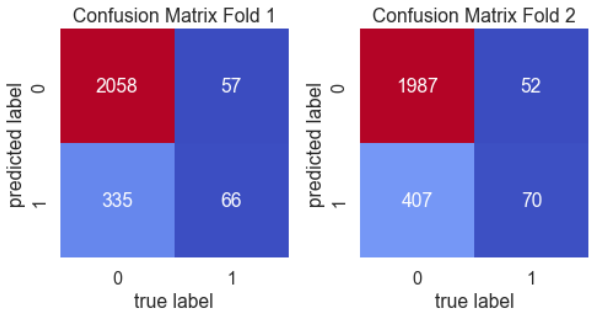

Fig-17: NBC Confusion Matrix using SMOTE

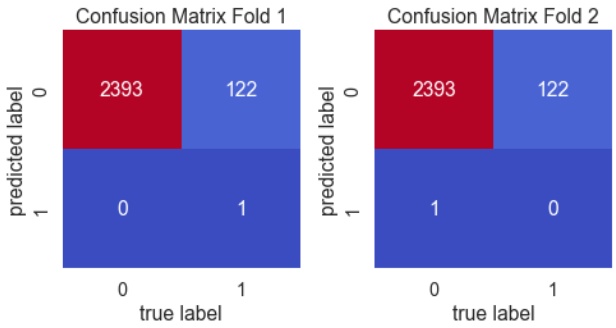

Fig-18: Logit Confusion Matrix without SMOTE

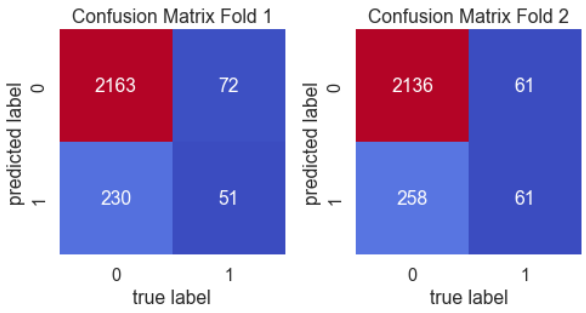

Fig-19: Logit Confusion Matrix using SMOTE

3.3 Graphical Representation of Results

The ensemble gives 94% accuracy for fold1 and 95% for fold2. If a classification occurs, the cut-off criterion should be examined more closely. Because of the fairly low sensitivity of the model at a cut-off criterion of 80%, no case of stroke is identified in the test dataset. However, for our purposes, the cut-off criterion is only used to better understand classification performance and evaluate the model's overall quality.

	precision	recall	f1-score	support
0	0.95	0.99	0.97	73
1	0.00	0.00	0.00	4
accuracy			0.94	77
macro avg	0.47	0.49	0.48	77
weighted avg	0.90	0.94	0.92	77

	precision	recall	f1-score	support
0	0.95	1.00	0.97	73
1	0.00	0.00	0.00	4
accuracy			0.95	77
macro avg	0.47	0.50	0.49	77
weighted avg	0.90	0.95	0.92	77

Figure-20: Result of Ensemble on Unseen Data.

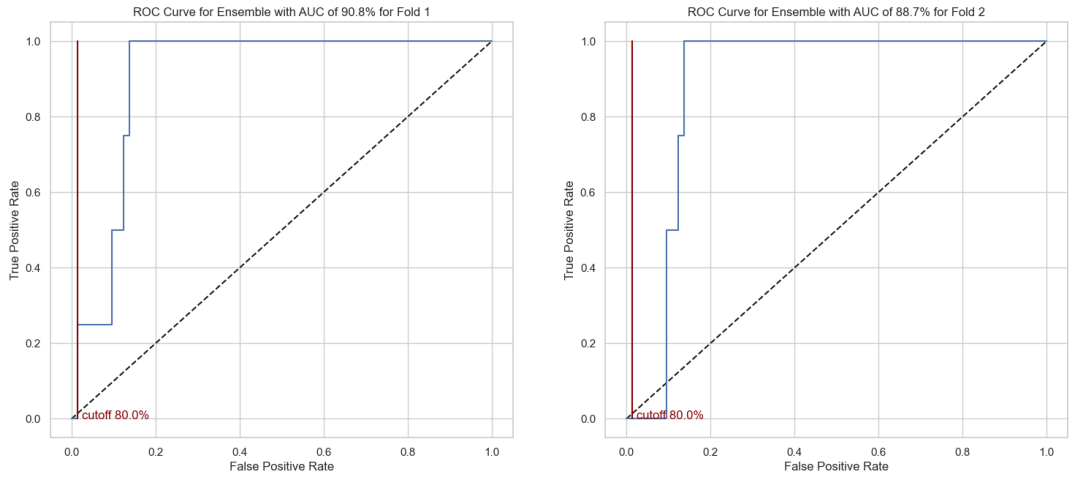

Figure-21: ROC Curve Represenation.

4. CONCLUSION & FUTURE RECOMMENDATIONS

The study started off by analyzing the data and discovering that some features, including age, appeared to be reliable predictors of stroke. Following considerable visualization, the study tried several models, testing Random Forest, SVM, and Logistic Regression. The models were then subjected to hyperparameter tuning to examine whether the outcomes might improve. While the Tuned Support Vector Machine offered the best recall and F1 score, Random Forest had the highest accuracy. As a result, the study's model of choice was tuned Support Vector Machine. The study carefully examined the most logical methodologies and algorithms, stating their advancements, to comprehend the accuracy and recall prediction of the selected model about their F1 measurement. According to the study, the Support Vector Machine model performed the best and was selected as the project's preferred model since it can improve the outcome using AdaBoost or XGBoost (depending on the selection of the model's data complexity) and might be utilized to achieve better outcomes for boosting a sensitive discrete dataset like the project. It could help clarify the discrepancies found in the study.

APPENDIXES

```
# -*- coding: utf-8 -*-
"""
Created on Fri Apr 28 14:10:57 2023

@author: 20-42915-1
"""

import pandas as pd
from pandas_profiling import ProfileReport as PR
import numpy as np
from zipfile import ZipFile
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
import seaborn as sns
import plotly.graph_objects as go
import gc

from sklearn.pipeline import Pipeline
from sklearn.tree import DecisionTreeRegressor,DecisionTreeClassifier
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import confusion_matrix, classification_report

from sklearn.metrics import roc_curve, auc, roc_auc_score

from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier as knn
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import GradientBoostingRegressor as GBR

from sklearn.impute import KNNImputer

import joblib

from imblearn.over_sampling import SMOTENC

from catboost import CatBoostClassifier, Pool

import shap

seeds = 11235813
np.random.seed(seeds)

import pywaffle
import re
import warnings
warnings.filterwarnings("ignore")

#Reading the dataset
data=pd.read_csv("healthcare-dataset-stroke-data.csv")
data.head().T

# Show Missing Data
df = pd.read_csv('healthcare-dataset-stroke-data.csv')
print('======\tMissing Data\t======')
df.isnull().sum()

# Using Decision Tree to fill up missing Data of BMI

DT_bmi_pipe = Pipeline( steps=[
('scale',StandardScaler()),
('lr',DecisionTreeRegressor(random_state=42))
])
X = df[['age','gender','bmi']].copy()
X.gender = X.gender.replace({'Male':0,'Female':1,'Other':-1}).astype(np.uint8) #
Classifying Gender
```

```
Missing = X[X.bmi.isna()]
X = X[~X.bmi.isna()]
Y = X.pop('bmi')
DT_bmi_pipe.fit(X,Y)
predicted_bmi =
pd.Series(DT_bmi_pipe.predict(Missing[['age','gender']]),index=Missing.index)
df.loc[Missing.index,'bmi'] = predicted_bmi
print('Present Missing Values:\t',sum(df.isnull().sum()))


from pywaffle import Waffle
background_color = "#FFFFFF"
fig = plt.figure(figsize=(3, 2),dpi=300,facecolor=background_color,
FigureClass=Waffle,
rows=2,
values=[1, 19],
colors=['#3a91e6', "#B0E2FF"],
characters='O',
font_size=18, vertical=True,
)

fig.text(0.035,0.78,'People Affected by a Stroke in our
dataset',fontfamily='Calibri',fontsize=12,fontweight='bold')
fig.text(0.035,0.70,'This is around 1 in 20 people [249 out of
5000]',fontfamily='Calibri',fontsize=6)

plt.show()


#pandas profiling
data.drop(columns=["id"], inplace=True)
profile = PR(
data,
title="Stroke Dataset Report",
dark_mode=True,
progress_bar=False,
explorative=True,
plot={"correlation": {"cmap": "coolwarm", "bad": "#000000"}}
)

profile.to_notebook_iframe()

data.loc[data.gender == "Other"]

data = data.loc[data.gender != "Other"]

#The continuous and categorical variables are designated
contVars = ["age", "avg_glucose_level", "bmi"]
catVars = [i for i in data.columns if i not in contVars and i != "stroke"]

#Target Separated Relationships 2D
def corPlot(df, color: str, title, bins=40):
sns.set_theme(style="white", font_scale=1.3)

pp=sns.pairplot(
df,
hue=color,
kind="hist",
diag_kind="kde",
corner=True,
plot_kws={"alpha": 0.9, 'bins':bins},
diag_kws = {'alpha':0.8, 'bw_adjust': 1, "fill": False, "cut": 0},
palette="coolwarm",
aspect=1.1,
height=3.2
)

pp.fig.suptitle(title, fontsize=15)
plt.show()
```

```
contVars.append("stroke")
corPlot(data[contVars], "stroke", "Continuous Variables of the Data Set")

contVars.pop(-1)

#Target Separated Relationships 3D
def plot3d(df, cls: list, c: str, X: str, Y: str, Z: str, title):

"""
Function to plot 3 dimensions, colored by category (c)

df=pd.DataFrame
cls=colors for no-stroke and stroke
c=category to separate by color
X=X dimension
Y=Y dimension
Z=Z dimension
title=title of the plot
"""

fig = go.Figure()

for i in range(len(df["%s" % (c)].unique())):
fig.add_trace(
go.Scatter3d(
x=df.loc[df["%s" % (c)] == i, X],
y=df.loc[df["%s" % (c)] == i, Y],
z=df.loc[df["%s" % (c)] == i, Z],
mode='markers',
marker=dict(
size=7,
color=cls[i],
opacity=0.6
), name = i,
hovertemplate =
f"<i>{X}</i>: " +"%{x} <br>"+
f"<i>{Y}</i>: " +"%{y} <br>"+
f"<i>{Z}</i>: " +"%{z}"
)
)
fig.update_layout(
hoverlabel=dict(font=dict(color='white'))
)
fig.update_layout(
title=title,
template="plotly_white",
margin=dict(l=65, r=20, b=0, t=10),
width=800,
height=800,
scene=dict(
xaxis_title=X,
yaxis_title=Y,
zaxis_title=Z,
camera={
"eye": {"x": 2, "y": 2, "z": 2}
}
),
title_y=0.95,
legend=dict(yanchor="top", y=0.9, xanchor="left", x=0.99, title="%s" % (c))
)

fig.show()

plot3d(data, ["#3a91e6", "red"], "stroke",
"age", "avg_glucose_level", "bmi",
"3D Plot of Age, Average Glucose Level, and BMI"
)

#Share of Stroke by Category
def barList(df, c, target="stroke"):
uni=df[f"{c}"].unique()
```

```
brs = []
for u in uni:
brs.append(df.loc[data[f"{c}"] == u, target].sum() / df[target].sum())
return brs, uni

fig, p = plt.subplots(nrows=4, ncols=2, figsize=(10,16))
r=0
c=0
for i in catVars:
bars, uniq = barList(data, f"{i}", target="stroke")
if i in ["hypertension", "heart_disease"]:
uniq = ["Yes" if j == 1 else "No" for j in uniq]

p[r,c].bar(
uniq,
bars,
color=["#3a91e6"]*len(bars)
)
p[r,c].set_title(f"Proportion of Stroke by {i}")
p[r,c].set_ylabel("", fontsize=14)

if len(uniq) > 2:
p[r,c].set_xticklabels(uniq, rotation=25)
if c == 1:
c=0
r+=1
else:
c+=1

plt.tight_layout(pad=1)
plt.delaxes(p[3,1])
plt.show()

#Target Separated Dispersion Continous vs. Categorical
def dispersions(cont: list, cat: list, target: str):
sns.set_palette(sns.color_palette(["#3a91e6", "red"]))
for co in cont:
for ct in cat:
l = (len(data["%s" % (ct)].unique()) - 2) * 2
pp=sns.catplot(
data=data,
x=ct,
y=co,
hue=target,
kind="box",
aspect=eval(f"1.{l}")
)

pp.fig.suptitle(
"{} by {} and {}".format(co, ct, target),
fontsize=15
)
plt.show()

dispersions(contVars, catVars, "stroke")

#Features
def dummies(df, feats: list, dropFirstAll=False) -> pd.DataFrame:
"""
This function creates dummies from categorical features of type str.
df=pd.DataFrame
feats=list of feature names
dropFirstAll=bool. If n-1 dummies for all features

output:
Data Frame with dummies
"""
df=df.copy()

for f in feats:
if len(df[f"{f}"].unique()) > 2:
```

```
dropFirst = False
else:
dropFirst = True

if dropFirstAll:
dropFirst = True

d = pd.get_dummies(df[f"{f}"], prefix=f, drop_first=dropFirst)
df.drop(columns=[f], inplace=True)
df[d.columns] = d
return df

dataD = dummies(
data,
["gender", "work_type", "ever_married", "Residence_type", "smoking_status"],
dropFirstAll=True
)
dataD.head(n=3)

#Cross Validation Strategy
test_holdout=0.985

CV_X, test_X, CV_y, test_y = train_test_split(
dataD.drop(columns=["stroke"]),
dataD.stroke,
train_size=test_holdout,
random_state=seeds,
shuffle=True,
stratify=dataD.stroke
)

dataD = pd.concat([CV_X, CV_y], axis=1)

X1, X2, y1, y2 = train_test_split(
dataD.drop(columns=["stroke"]),
dataD.stroke,
train_size=0.5,
random_state=seeds,
shuffle=True,
stratify=dataD.stroke
)

#folds for the crossvalidation grid search
folds=4

plt.figure(figsize=(3,3))
plt.title("Proportion of Class Stroke", fontsize = 14)
plt.bar(
["No Stroke", "Stroke"],
[1-y1.sum() / y1.shape[0], y1.sum() / y1.shape[0]],
color=["#3a91e6", "red"]
)
plt.show()

#To avoid data leakage, the target variable "strok" is omitted when imputing the missing
BMI values.
def bmiImputer(XOut, model):
"""
Function to train imputer on training dataset and apply to another or the same one.
"""
XOut=XOut.copy()

model.fit(XOut)
XOut[[i for i in XOut.columns if i != "stroke"]] = model.fit_transform(XOut)
XOut["bmi"] = np.round(XOut.bmi.values, 1)
XOut["bmi"] = XOut.bmi

return XOut

imputer = KNNImputer(n_neighbors=10, weights="distance")
```

```
X1 = bmiImputer(X1,imputer)
X2 = bmiImputer(X2, imputer)
test_X = bmiImputer(test_X, imputer)

bmi = pd.concat([X1.bmi, X2.bmi, test_X.bmi])
data["bmi"] = bmi

#SMOTE
SyntheticShare=1/3
neighbors=2

SMOTE = SMOTENC(
k_neighbors=neighbors,
random_state=seeds,
n_jobs=-1,
categorical_features=np.where([i not in contVars for i in X1.columns])[0],
sampling_strategy=SyntheticShare
)
Xr1, yr1 = SMOTE.fit_resample(X1, y1)
Xr2, yr2 = SMOTE.fit_resample(X2, y2)

plt.figure(figsize=(3,3))
plt.title("Proportion of Class Stroke after SMOTE", fontsize = 14)
plt.bar(
["No Stroke", "Stroke"],
[1-yr1.sum() / yr1.shape[0], yr1.sum() / yr1.shape[0]],
color=["#3a91e6", "red"]
)
plt.show()

contVars.append("stroke")
corPlot(pd.concat([Xr1, yr1.astype(int)], axis=1)[contVars], "stroke", "Minority Class
Oversampled")

contVars.pop(-1)

plot3d(
pd.concat([Xr1, yr1.astype(int)], axis=1),
["#3a91e6", "red"], "stroke",
"age", "avg_glucose_level", "bmi",
"3D Plot of Age, Average Glucose Level, and BMI (SMOTE)"
)

#SVM Modeling
SVM = make_pipeline(
StandardScaler(),
SVC(probability=True, random_state=seeds, kernel="rbf")
)

param_grid = {
'svc__C': [0.1 ,1, 10, 50],
'svc__gamma': [0.0001, 0.001, 0.01]
}

def CrossVal(model, grid, X: list, y: list, WisorBMI=56.6, **kwargs):
"""
Function to perform a cross validation wit a grid search.
We use winsorizing for bmi > 56.6. Default=56.6 is highest BMI stroke case.
"""
for i in [0,1]:
val = X[i]
val["bmi"] = val.bmi.map(lambda x: WisorBMI if x > WisorBMI else x)
globals()[f"grid{i+1}"] = GridSearchCV(model, grid, n_jobs=-1, cv=folds,
scoring="roc_auc")
eval(f"grid{i+1}").fit(val, y[i], **kwargs)


CrossVal(SVM, param_grid, [X1[contVars], X2[contVars]], [y1, y2])

def makeModelAndPred(g: list, m: str, X: list):
"""
```

```
This function uses the best estimator from grid search to predict on validation set.
"""
for i in [0,1]:
globals()[f"best_params_model{m}{i+1}"] = g[i].best_params_
globals()[f"model{m}{i+1}"] = g[i].best_estimator_
globals()[f"y_hat{i+1}"] = eval(f"model{m}{i+1}").predict(X[i])

return globals()[f"best_params_model{m}1"], globals()[f"best_params_model{m}2"]

makeModelAndPred([grid1, grid2], "SVM", [X2[contVars], X1[contVars]])

#SVM Evaluation without SMOTE
for i in [1, 2]:
print(classification_report(globals()[f"y{1 if i == 2 else 2}"], globals()[f"y_hat{i}"]))


def confMat(df, y: list, preds: list, target: str):
"""
Function to print confusion matrix.
The matrix corresponds to a cut-off criterion of 50%.
This statement must be relativised for methods such as support vector machines,
since a hyperplane is used here and not a probability.
"""
fig, p = plt.subplots(nrows=1, ncols=2, figsize=(7,8))
for i in [0,1]:
mat = confusion_matrix(y[i], preds[i])
sns.heatmap(
mat.T,
square=True,
annot=True,
fmt='d',
cbar=False,
xticklabels=np.sort(df[f"{target}"].unique()),
yticklabels=np.sort(df[f"{target}"].unique()),
cmap="coolwarm",
ax=p[i]
)
p[i].set_xlabel('true label')
p[i].set_ylabel('predicted label')
p[i].set_title(f'Confusion Matrix Fold {i+1}')
plt.tight_layout(pad=1)
plt.show()

confMat(data, [y2, y1], [y_hat1, y_hat2], "stroke")

def roc(yt: list, ph: list, title: str, di=False, cut=0.5):
"""
Function for calculating the receiver operating characteristic curve.
In addition, the point that lies at a cut-off criterion of 50% is marked to make
comparison easier.

Output: List of AUCs by fold, list of standard deviation of the prediction errors by fold
"""

fig, p = plt.subplots(nrows=1, ncols=2, figsize=(16,7))

auc=[]
s=[]

for i in [0,1]:
fpr, tpr, thresholds = roc_curve(yt[i], ph[i], drop_intermediate=di)
p[i].plot([0, 1], [0, 1], 'k--')
p[i].plot(fpr, tpr)
cutOff = np.where(np.min((thresholds - cut)**2) == (thresholds - cut)**2)[0][0]
p[i].plot([fpr[cutOff], fpr[cutOff]], [0,1], "darkred")
p[i].text(fpr[cutOff]+0.01, 0, s=f"cutoff {cut*100}%", c="darkred")
p[i].set_xlabel("False Positive Rate")
p[i].set_ylabel("True Positive Rate")

auc.append(roc_auc_score(yt[i], ph[i]))
p[i].set_title("ROC Curve for {} with AUC of {}% for Fold {}".format(
```

```
title,
round(auc[i]*100, 1),
i+1
)
)

s.append(np.std(yt[i] - ph[i]))

print('AUC value for Fold1 = ', auc[0])
print('AUC value for Fold2 = ', auc[1])
auc.append(f"{title}")
s.append(f"{title}")
plt.show()

return auc, s

aucSVM, std_SVM = roc(
[y2, y1],
[modelSVM1.predict_proba(X2[contVars])[:, 1], modelSVM2.predict_proba(X1[contVars])[:,
1]],
"SVM"
)

#SVM Evaluation using SMOTE
CrossVal(SVM, param_grid, [Xr1[contVars], Xr2[contVars]], [yr1, yr2])

makeModelAndPred([grid1, grid2], "SVM", [X2[contVars], X1[contVars]])

for i in [1, 2]:
print(classification_report(globals()[f"y{1 if i == 2 else 2}"], globals()[f"y_hat{i}"]))

confMat(data, [y2, y1], [y_hat1, y_hat2], "stroke")

aucSVM, std_SVM = roc(
[y2, y1],
[modelSVM1.predict_proba(X2[contVars])[:, 1], modelSVM2.predict_proba(X1[contVars])[:,
1]],
"SVM"
)

def save(m: list, name: str):
"""
Function to save the learned hypothesis in working directory for implementation purpose.
"""
for i, M in enumerate(m):
hyp = f"{name}{i+1}.pkl"
joblib.dump(M, hyp)

save([modelSVM1, modelSVM2], "svm")

#KNN Modeling and Evaluation without SMOTE
KNN = make_pipeline(
StandardScaler(),
knn()
)

param_grid = {
'kneighborsclassifier__n_neighbors': [5, 10, 50, 100, 200],
'kneighborsclassifier__weights': ['uniform', 'distance']
}

CrossVal(KNN, param_grid, [X1[contVars], X2[contVars]], [y1, y2])
makeModelAndPred([grid1, grid2], "KNN", [X2[contVars], X1[contVars]])
for i in [1, 2]:
print(classification_report(globals()[f"y{1 if i == 2 else 2}"], globals()[f"y_hat{i}"]))

confMat(data, [y2, y1], [y_hat1, y_hat2], "stroke")


aucKNN, std_KNN = roc(
[y2, y1],
```

```
[modelKNN1.predict_proba(X2[contVars])[:, 1], modelKNN2.predict_proba(X1[contVars])[:,
1]],
"KNN"
)

#KNN Evaluation using SMOTE
CrossVal(KNN, param_grid, [Xr1[contVars], Xr2[contVars]], [yr1, yr2])

makeModelAndPred([grid1, grid2], "KNN", [X2[contVars], X1[contVars]])

for i in [1, 2]:
print(classification_report(globals()[f"y{1 if i == 2 else 2}"], globals()[f"y_hat{i}"]))

confMat(data, [y2, y1], [y_hat1, y_hat2], "stroke")

aucKNN, std_KNN = roc(
[y2, y1],
[modelKNN1.predict_proba(X2[contVars])[:, 1], modelKNN2.predict_proba(X1[contVars])[:,
1]],
"KNN"
)

save([modelKNN1, modelKNN2], "knn")

#Random Forest Modeling and Evaluation without SMOTE
RF = RandomForestClassifier(random_state=seeds)

param_gridRF = {
'n_estimators': [150, 180],
'max_depth': [15, 18, 20],
'max_features': [0.6, 0.75, 0.9],
'max_samples': [0.6, 0.75, 0.9],
'min_samples_leaf': [1, 2],
'ccp_alpha': [0, 0.0001]#Pruning
}
CrossVal(RF, param_gridRF, [X1, X2], [y1, y2])
makeModelAndPred([grid1, grid2], "RF", [X2, X1])
for i in [1, 2]:
print(classification_report(globals()[f"y{1 if i == 2 else 2}"], globals()[f"y_hat{i}"]))

confMat(data, [y2, y1], [y_hat1, y_hat2], "stroke")

aucRF, std_RF = roc(
[y2, y1],
[modelRF1.predict_proba(X2)[:, 1], modelRF2.predict_proba(X1)[:, 1]],
"RF"
)


#Random Forest Evaluation using SMOTE
CrossVal(RF, param_gridRF, [Xr1, Xr2], [yr1, yr2])
makeModelAndPred([grid1, grid2], "RF", [X2, X1])
save([modelRF1, modelRF2], "rf")

for i in [1, 2]:
print(classification_report(globals()[f"y{1 if i == 2 else 2}"], globals()[f"y_hat{i}"]))

confMat(data, [y2, y1], [y_hat1, y_hat2], "stroke")

aucRF, std_RF = roc(
[y2, y1],
[modelRF1.predict_proba(X2)[:, 1], modelRF2.predict_proba(X1)[:, 1]],
"RF"
)

#Naive Bayes Classifier Modeling and Evaluation without SMOTE
NBC = make_pipeline(
StandardScaler(),
GaussianNB()
)
```

```
param_grid = {
'gaussiannb__var_smoothing': [1e-10, 1e-09, 1e-08, 1e-07]
}

CrossVal(NBC, param_grid, [X1[contVars], X2[contVars]], [y1, y2])

makeModelAndPred([grid1, grid2], "NBC", [X2[contVars], X1[contVars]])

for i in [1, 2]:
print(classification_report(globals()[f"y{1 if i == 2 else 2}"], globals()[f"y_hat{i}"]))

confMat(data, [y2, y1], [y_hat1, y_hat2], "stroke")

aucNBC, std_NBC = roc(
[y2, y1],
[modelNBC1.predict_proba(X2[contVars])[:, 1], modelNBC2.predict_proba(X1[contVars])[:,
1]],
"NBC"
)

#Naive Bayes Classifier Evaluation using SMOTE

CrossVal(NBC, param_grid, [Xr1[contVars], Xr2[contVars]], [yr1, yr2])
makeModelAndPred([grid1, grid2], "NBC", [X2[contVars], X1[contVars]])
save([modelNBC1, modelNBC2], "nbc")

for i in [1, 2]:
print(classification_report(globals()[f"y{1 if i == 2 else 2}"], globals()[f"y_hat{i}"]))

confMat(data, [y2, y1], [y_hat1, y_hat2], "stroke")

aucNBC, std_NBC = roc(
[y2, y1],
[modelNBC1.predict_proba(X2[contVars])[:, 1], modelNBC2.predict_proba(X1[contVars])[:,
1]],
"NBC"
)



#Logistic Regression Modeling and Evaluation without SMOTE
Logit = LogisticRegression(random_state=seeds, solver='liblinear')

param_grid = {
'penalty': ["l1", "l2"],
'C': np.linspace(0, 500, 26)
}
CrossVal(Logit, param_grid, [X1, X2], [y1, y2])
parmsLogit1, parmsLogit2 = makeModelAndPred([grid1, grid2], "Logit", [X2, X1])
print(parmsLogit1, parmsLogit2)
for i in [1, 2]:
print(classification_report(globals()[f"y{1 if i == 2 else 2}"], globals()[f"y_hat{i}"]))

confMat(data, [y2, y1], [y_hat1, y_hat2], "stroke")
aucLogit, std_Logit = roc(
[y2, y1],
[modelLogit1.predict_proba(X2)[:, 1], modelLogit2.predict_proba(X1)[:, 1]],
"Logit"
)


#Logistic RegressionEvaluation using SMOTE

CrossVal(Logit, param_grid, [Xr1, Xr2], [yr1, yr2])
parmsLogit1, parmsLogit2 = makeModelAndPred([grid1, grid2], "Logit", [X2, X1])
print(parmsLogit1, parmsLogit2)

save([modelLogit1, modelLogit2], "logit")
for i in [1, 2]:
print(classification_report(globals()[f"y{1 if i == 2 else 2}"], globals()[f"y_hat{i}"]))
confMat(data, [y2, y1], [y_hat1, y_hat2], "stroke")
```

```
aucLogit, std_Logit = roc(
[y2, y1],
[modelLogit1.predict_proba(X2)[:, 1], modelLogit2.predict_proba(X1)[:, 1]],
"Logit"
)

#Assessment of the Different Algorithms (Error Analysis)
def errors(x: list, y: list, m: list, titleSupplements: list, var: str):

x=x.copy()

fig, p = plt.subplots(nrows=3, ncols=2, figsize=(10,14))

r=0
c=0
for i, M in enumerate(m):
X=x[i].copy()
X["preds"] = M.predict_proba(X)[:,1]
X["err"] = y[i] - X.preds

p[r,c].scatter(x=X[var], y=X["err"], c=y[i].astype("int"), cmap="coolwarm", alpha=0.5)
p[r,c].set_title("Error of %s for model %s" % (var, titleSupplements[i]))
p[r,c].set_ylim([-1, 1])
p[r,c].set_xlabel(var)

if c == 0:
p[r,c].set_ylabel("Prediction Error")

if c == 1:
c=0
r+=1
else:
c+=1

plt.tight_layout(pad=3)
plt.delaxes(p[2,1])
plt.show()


errors(
[X2[contVars], X2, X2[contVars], X2, X2[contVars]], [y2, y2, y2, y2, y2],
[modelSVM1, modelRF1, modelNBC1, modelLogit1, modelKNN1],
[
"SVM Fold 2",
"RF Fold 2",
"NBC Fold 2",
"Logit Fold 2",
"KNN Fold 2"
],
"age"
)

errors(
[X1[contVars], X1, X1[contVars], X1, X1[contVars]], [y1, y1, y1, y1, y1],
[modelSVM2, modelRF2, modelNBC2, modelLogit2, modelKNN1],
[
"SVM Fold 1",
"RF Fold 1",
"NBC Fold 1",
"Logit Fold 1",
"KNN Fold 1"
],
"bmi"
)


testX2 = pd.concat([y2, X2], axis=1)
# We use SVM1 becaus of the good performance
testX2["preds"] = modelRF1.predict_proba(X2)[:,1]
testX2["err"] = np.abs(testX2.stroke - testX2.preds)
errGBR = GBR(loss="absolute_error", max_depth=6, n_estimators=100)
```

```
errGBR.fit(testX2.drop(columns=["preds", "err", "stroke"]), testX2["err"])
from sklearn.metrics import mean_absolute_error
print("Mean absolute Error for Random Forest:", round(mean_absolute_error(y1,
errGBR.predict(X1)), 4))
#And we save the learned model hypothesis
joblib.dump(errGBR, "errGBR.pkl");


testX2 = pd.concat([y2, X2], axis=1)
# We use SVM1 becaus of the good performance
testX2["preds"] = modelKNN1.predict_proba(X2[contVars])[:,1]
testX2["err"] = np.abs(testX2.stroke - testX2.preds)
errGBR = GBR(loss="absolute_error", max_depth=6, n_estimators=100)
errGBR.fit(testX2.drop(columns=["preds", "err", "stroke"]), testX2["err"])
from sklearn.metrics import mean_absolute_error
print("Mean absolute Error for KNN:", round(mean_absolute_error(y1, errGBR.predict(X1)),
4))
#And we save the learned model hypothesis
joblib.dump(errGBR, "errGBR.pkl");

testX2 = pd.concat([y2, X2], axis=1)
# We use SVM1 becaus of the good performance
testX2["preds"] = modelLogit1.predict_proba(X2)[:,1]
testX2["err"] = np.abs(testX2.stroke - testX2.preds)
errGBR = GBR(loss="absolute_error", max_depth=6, n_estimators=100)
errGBR.fit(testX2.drop(columns=["preds", "err", "stroke"]), testX2["err"])
from sklearn.metrics import mean_absolute_error
print("Mean absolute Error for Logit:", round(mean_absolute_error(y1,
errGBR.predict(X1)), 4))
#And we save the learned model hypothesis
joblib.dump(errGBR, "errGBR.pkl");


testX2 = pd.concat([y2, X2], axis=1)
# We use SVM1 becaus of the good performance
testX2["preds"] = modelSVM1.predict_proba(X2[contVars])[:,1]
testX2["err"] = np.abs(testX2.stroke - testX2.preds)
errGBR = GBR(loss="absolute_error", max_depth=6, n_estimators=100)
errGBR.fit(testX2.drop(columns=["preds", "err", "stroke"]), testX2["err"])
from sklearn.metrics import mean_absolute_error
print("Mean absolute Error for SVM:", round(mean_absolute_error(y1, errGBR.predict(X1)),
4))
#And we save the learned model hypothesis
joblib.dump(errGBR, "errGBR.pkl");

testX2 = pd.concat([y2, X2], axis=1)
# We use SVM1 becaus of the good performance
testX2["preds"] = modelNBC1.predict_proba(X2[contVars])[:,1]
testX2["err"] = np.abs(testX2.stroke - testX2.preds)
errGBR = GBR(loss="absolute_error", max_depth=6, n_estimators=100)
errGBR.fit(testX2.drop(columns=["preds", "err", "stroke"]), testX2["err"])
from sklearn.metrics import mean_absolute_error
print("Mean absolute Error for Naive Bayes:", round(mean_absolute_error(y1,
errGBR.predict(X1)), 4))
#And we save the learned model hypothesis
joblib.dump(errGBR, "errGBR.pkl");


exp=95
for i in [1,2]:
ssa=aucSVM[i-1]**exp+aucRF[i-1]**exp+aucLogit[i-1]**exp+aucKNN[i-1]**exp+aucNBC[i-1]**exp

svmp = eval(f"modelSVM{i}").predict_proba(eval(f"X{2 if i == 1 else 1}[contVars]"))[:,
1]\
* aucSVM[i-1]**exp/ssa
print("Weight Support Vector Machines fold %i" % i, ":", round(aucSVM[i-1]**exp/ssa, 2))
rfp = eval(f"modelRF{i}").predict_proba(eval(f"X{2 if i == 1 else 1}"))[:, 1]\
* aucRF[i-1]**exp/ssa
print("Weight Random Forest fold %i" % i, ":", round(aucRF[i-1]**exp/ssa, 2))

# knn = eval(f"modelKNN{i}").predict_proba(eval(f"X{2 if i == 1 else 1}"))[:, 1]\
```

```
#    * aucKNN[i-1]**exp/ssa
#print("Weight KNN fold %i" % i, ":", round(aucKNN[i-1]**exp/ssa, 2))

logp = eval(f"modelLogit{i}").predict_proba(eval(f"X{2 if i == 1 else 1}"))[:, 1]\
* aucLogit[i-1]**exp/ssa
print("Weight Logit fold %i" % i, ":", round(aucLogit[i-1]**exp/ssa, 2))

nbcp = eval(f"modelNBC{i}").predict_proba(eval(f"X{2 if i == 1 else 1}[contVars]"))[:,
1]\
* aucNBC[i-1]**exp/ssa
print("Weight Naive Bayes Classifier fold %i" % i, ":", round(aucNBC[i-1]**exp/ssa, 2))
print("")

#globals()[f"p{i}"] = svmp + rfp + logp + knn + nbcp
globals()[f"p{i}"] = svmp + rfp + logp + nbcp


for i, k in zip([1, 2], [0.80, 0.80]):
print(classification_report(
globals()[f"y{1 if i == 2 else 2}"],
[1 if i >= k else 0 for i in globals()[f"p{i}"]]
))

aucEnsemble, std_Ensemble = roc(
[y2, y1],
[
p1,
p2
],
"Ensemble",
cut=0.80
)

#Cross Comparison

def tempDf(inp: list, colnames: list, by: str, i: str, j: str, name:str):
"""
Function to create a dataframe to compare model metrics.
"""
tmp = pd.DataFrame(
inp,
columns=colnames,
)
tmp["id"] = tmp.index

tmp = pd.wide_to_long(tmp, by, i=i, j=j)
tmp.reset_index(inplace=True)
tmp.rename(columns={by: name, j: by}, inplace=True)

return tmp

Std_dev = tempDf([
std_SVM,
std_KNN,
std_NBC,
std_RF,
std_Logit,
std_Ensemble
], ["Fold 1", "Fold 2", "Method"], "Fold ", "id", "partition", "Standard Deviation")

model_AUC = tempDf([
aucSVM,
aucKNN,
aucNBC,
aucRF,
aucLogit,
aucEnsemble
], ["Fold 1", "Fold 2", "Method"], "Fold ", "id", "partition", "AUC")

plt.figure(figsize=(10,6))
plt.title("AUC of Models for Fold 1 and 2")
```

```
p=sns.swarmplot(
data=model_AUC[["AUC", "Method", "Fold "]],
x="Method",
y="AUC",
hue="Fold ",
palette=["#3a91e6", "red"]
)
rect=mpatches.Rectangle(
(-1,aucEnsemble[1]),8,aucEnsemble[0]-aucEnsemble[1],
fill = True,
color = "grey",
alpha=0.1,
linewidth = 2
)
plt.gca().add_patch(rect)
p.set_ylabel("AUC")
p.tick_params(axis='x', rotation=45)
plt.show()

plt.figure(figsize=(10,6))
plt.title("Standard Deviations of Prediction Errors for Fold 1 and 2 ")

p=sns.swarmplot(
data=Std_dev[["Standard Deviation", "Method", "Fold "]],
x="Method",
y="Standard Deviation",
hue="Fold ",
palette=["#3a91e6", "red"]
)
rect=mpatches.Rectangle(
(-1,std_Ensemble[1]),8,std_Ensemble[0]-std_Ensemble[1],
fill = True,
color = "grey",
alpha=0.1,
linewidth = 2
)
plt.gca().add_patch(rect)
p.set_ylabel("Standard Deviation")
p.tick_params(axis='x', rotation=45)
plt.show()


#Unseen Test Data
for i in [1,2]:
ssa=aucSVM[i-1]**exp+aucRF[i-1]**exp+aucLogit[i-1]**exp+aucNBC[i-1]**exp

svmp = eval(f"modelSVM{i}").predict_proba(test_X[contVars])[:, 1]\
* aucSVM[i-1]**exp/ssa
rfp = eval(f"modelRF{i}").predict_proba(test_X)[:, 1]\
* aucRF[i-1]**exp/ssa
logp = eval(f"modelLogit{i}").predict_proba(test_X)[:, 1]\
* aucLogit[i-1]**exp/ssa

nbcp = eval(f"modelNBC{i}").predict_proba(test_X[contVars])[:, 1]\
* aucNBC[i-1]**exp/ssa

globals()[f"p{i}"] = svmp + rfp + logp + nbcp

for i, k in zip([1, 2], [0.79, 0.79]):
print(classification_report(
test_y,
[1 if i >= k else 0 for i in globals()[f"p{i}"]]
))

_, _ = roc(
[test_y, test_y],
[
p1,
p2
],
```

```
"Ensemble",
cut=0.80
)
```