




NOVEMBER 21, 2023

END-TO-END MACHINE LEARNING PROJECT

TEAM 7: AIZA BAJWA, PRAKRITI BISWAS, PRANAV KULKARNI

EECS 3401: FALL 2023, SECTION A
PROFESSOR RUBA AL OMARI
York University



FRAMING THE PROBLEM AND LOOKING AT THE BIG PICTURE

The overall task is to build a model to predict whether a patient is likely to get a stroke. The label “stroke” has two possible values: 0 to indicate the patient did not have a stroke and 1 to indicate the patient did have a stroke. Since the dataset has *labelled* training examples, this problem involves supervised learning. And since the label is categorical, a category is to be predicted, and hence, this is a classification task. In addition, a small data set that is unchanging has been provided with no continuous flow of data coming into the system, resulting in batch learning techniques to be used.

The model’s output (a prediction of a patient’s likelihood of getting a stroke) will be used to help inform and alert physicians of patients at risk of stroke. Physicians can use the information to perform more tests and more importantly, can discuss future lifestyle changes with the patient to decrease the risk, such as quitting smoking, reducing weight, taking preventive medications.

DESCRIPTION OF THE DATASET

The original dataset contains 5110 examples and 12 features, all revolving around a patient’s health and lifestyle information. However, as described in the limitations section (below), another dataset was slightly modified and merged with the original one to add more data (while keeping the features consistent), resulting in a total of 20110 examples.

With regards to the features, there are four numerical features (*id*, *age*, *avg_glucose_level*, *bmi*) and eight categorical features. Three of the eight categorical features, specifically *hypertension*, *heart_disease* and *stroke*, are encoded numerically, with 0 and 1 representing the different categories. These numerically encoded categorical features are classified as integers by the system. The numerical and the categorical features not numerically encoded are represented as integers/floats and objects, respectively.

There are missing values in two features: 201 in the *bmi* column and 1544 in the *smoking_status* column. *Bmi*’s missing values are represented as NaN values, while the author made a note that unavailable information for *smoking_status* is represented as “Unknown”. Both missing values must be properly dealt with before training the machine learning algorithms, which will be discussed in the “Data Cleaning and Preprocessing” section below.

The data scales for the four numerical features vary greatly. *Id* ranges from roughly 0 to 100000, *age* ranges from roughly 0 to 90, *avg_glucose_level* ranges from roughly 50 to 275 and *bmi* ranges from roughly 10 to 100. These scales must be adjusted to balance the impact of the different variables, which will be discussed in the “Data Cleaning and Preprocessing” section.

3 GRAPHS OF EDA

Using the Pearson correlation coefficients with regards to the target, stroke, depicted in **Figure 1**, it can be concluded that *heart_disease* has the strongest positive correlation with it, then *avg_glucose_level* and then *age*. To look deeper into these top

stroke	1.000000
heart_disease	0.169728
avg_glucose_level	0.121842
age	0.112974
id	0.084921
hypertension	0.066128
bmi	-0.036354
Name: stroke, dtype: float64	

Figure 1. Pearson’s R with regards to target

three correlations, a *Heart_Disease* vs *Stroke* categorical plot (**Figure 2**), *Avg_Glucose_Level* vs *Stroke* boxplot (**Figure 3**) and an *Age* vs *Stroke* boxplot (**Figure 4**) were created.

To the right, **Figure 2** clearly depicts that a larger proportion of people *with* heart disease have a stroke. The plot can be understood by finding the proportions of people with and without stroke in both categories of *heart_disease*. That is, in category 0 of *heart_disease*, which represents not having heart disease, about 8500 people do not have stroke and about 4000 people do have stroke. So, from the total of 12500 people without heart disease, about 32% have a stroke. On the other hand, for category 1 of heart-disease, which represents people who have heart disease, about 4000 people do not have and have stroke. So, out of 8000 people who have heart disease, about 50% have a stroke. Hence, this demonstrates that it is more likely for people that have heart disease to have a stroke.

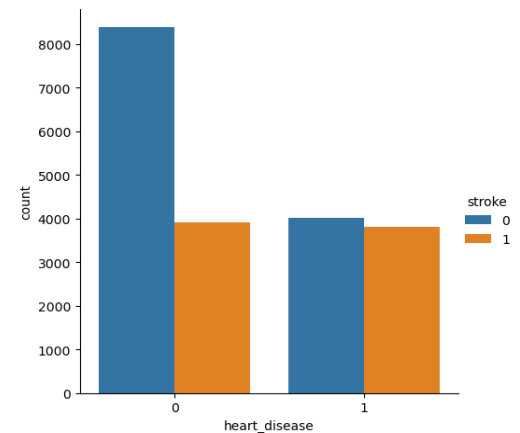


Figure 2. Heart_Disease vs Stroke categorical plot

In addition, in **Figure 3** and **Figure 4** below, because the orange boxes are higher than the blue boxes in both box plots, this also confirms the fact that people who experienced a stroke have, on average, higher average glucose levels and are older.

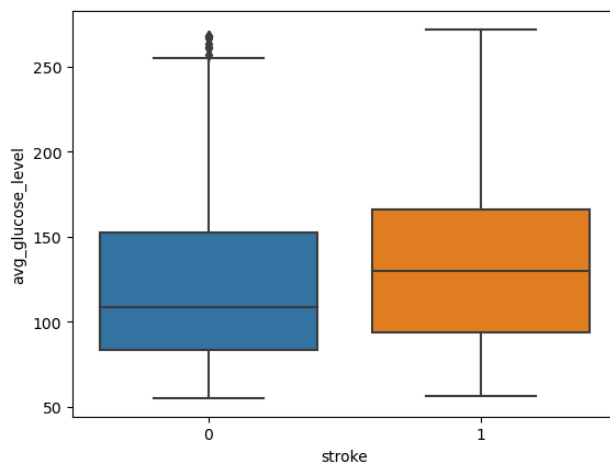


Figure 3. Avg_Glucose_Level vs Stroke boxplot

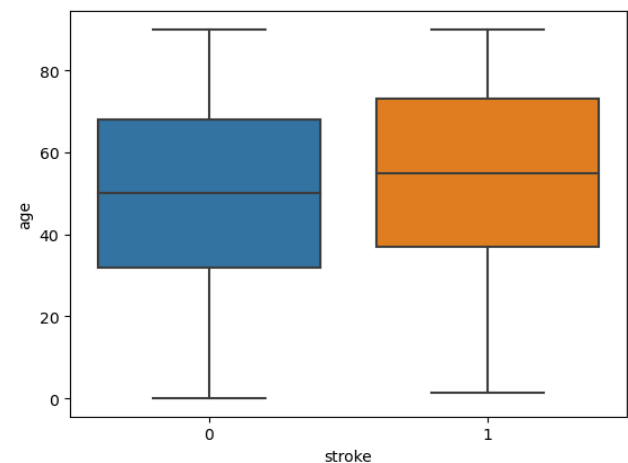


Figure 4. Age vs Stroke boxplot

DATA CLEANING AND PREPROCESSING

There are two main steps required in data cleaning: deleting duplicates and handling the missing values, if there are any for both cases.

In terms of deleting duplicates, there were no duplicates found in the dataset, so there was nothing to delete in this step. However, in terms of handling the missing values, there were two things to be taken care of. Even though there were no missing values represented with a question mark (“?”), there were missing values represented with NaN and with “Unknown”. In order to handle the “Unknown” values, they first had to be converted to NaN, so that the computer could

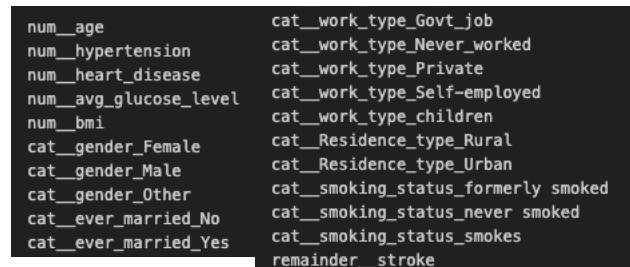
decipher them as missing values and not a category. Regardless, since 1544 out of the 20110 instances of the *smoking_status* feature is “Unknown”, this only represents 8% of the feature missing and so it seemed best to perform imputation. Similarly, the NaN values only accounted for 201 missing values of the *bmi* feature, which is an even smaller percentage, and so it seemed best to perform imputation for this feature as well. In addition, the *id* column was completely deleted since it is simply a unique identifier for each patient and should not bear any logical connection with the *stroke* column.

For the data preprocessing, a pipeline was created to automate four steps for all the features:

1. Fill in the missing numerical values with the mean using a SimpleImputer.
 - This handles the missing values of the *bmi* feature, replacing them with the mean, 27.824171.
2. Scale the numerical columns using StandardScaler
3. Fill in the missing categorical values with the most_frequent value using SimpleImputer
 - This handles the missing values of the *smoking_status* feature, replacing them with the most frequent value.
4. Encode the categorical columns using OneHotEncoder.
 - This creates a binary column for each category of all categorical features to convert categorical values into numerical values in the system.
 - For example, it converts the *gender* feature into *gender_Female*, *gender_Male* and *gender_Other*.

Applying the pipeline on the dataset, resulted in *dataset_prepared* (which can be seen in **Figure 5** to the right), with 21 columns, all of which have 0 missing values now.

Once the pipeline was applied on the dataset, it was ensured that the target only had two possible values/categories using *value_counts*. Since this was the case, no further modifications had to be made and the dataset was ready for training and evaluation of machine learning algorithms.



num_age	cat_work_type_Govt_job
num_hypertension	cat_work_type_Never_worked
num_heart_disease	cat_work_type_Private
num_avg_glucose_level	cat_work_type_Self-employed
num_bmi	cat_work_type_children
cat_gender_Female	cat_Residence_type_Rural
cat_gender_Male	cat_Residence_type_Urban
cat_gender_Other	cat_smoking_status_formerly_smoked
cat_ever_married_No	cat_smoking_status_never_smoked
cat_ever_married_Yes	cat_smoking_status_smokes
	remainder_stroke

Figure 5. Columns of *dataset_prepared*

TRAINING AND EVALUATION OF THREE MACHINE LEARNING ALGORITHMS, ANALYZE FINDINGS, AND COMPARE RESULTS

The three models trained on the dataset are as follows:

1. An SVM model with kernel='rbf', C=0.01, gamma=0.01.
2. A Logistic Regression Model with C=0.01, penalty='l2', solver='newton-cg'
3. A Decision Tree Model with max_depth=6

GridSearchCV was first used for each model separately, in order to find the best values for the hyperparameters for all three of the models. Please note that for the SVM model, GridSearchCV was used separately for kernel values *linear* and *rbf*. Refer to the “Limitations” section below to see why a kernel value of *poly* was not used.

The outputs of GridSearchCV were used as follows: For the SVM model, the *linear* winner and *rbf* winner were tested behind the scenes separately, and the *rbf* one was better, so its parameters were used for training the SVM model. For the Logistic Regression model, the winner's parameters were used as is. Lastly, for the Decision Tree model, the winner's parameters were played around with until 'max_depth = 6' on its own proved to give the strongest result.

To evaluate which machine learning algorithm out of the three is the best, it is important to first identify whether precision or recall is more important for this dataset. Similar to the coronavirus model, a false positive means that a healthy person is sent for more tests (because algorithm outputs they have a stroke risk), while a false negative means that a person who has a stroke will be classified as healthy. Due to this, a false negative is worse/more dangerous and hence, recall is more important than precision (this is a high recall model).

To start evaluation of the three models, it was first confirmed that the models were not overfitting or underfitting. This was tested by applying techniques to fix overfitting and underfitting and see if it resulted in any change. One such technique used was Grid Search to select values for regularization-like hyperparameters. Another technique used (not shown in the notebook) was adding more polynomial features to prevent underfitting by using The Kernel Method. In addition, to test that the models were not overfitting or underfitting, the accuracy of the models when tested on the training set was compared to the accuracy of the models when tested on the test set. If the models were underfitting, both accuracies would be low, and if they were overfitting, the accuracy would be high in the training set but low in the testing set. For the SVM and Logistic Regression models, the accuracy on X_{train} was 0.62, while it was 0.63 for Decision Tree, and the accuracy was 0.61 on X_{test} for all three models. Since these values are not severely low and are relatively consistent with each other, the three models are not underfitting and are not overfitting.

Next, all three models were evaluated using their classification reports, cross validation with $cv=5$, and F-Beta scores with $Beta = 2$ (i.e. F_2 scores, since more weight on recall). Two performance comparison tables from the results can be seen below:

Table 1. Performance Comparison Table obtained from the Three Classification Reports of the Three Models

Model Type	Precision		Recall		F1-Score		Accuracy
	0.0	1.0	0.0	1.0	0.0	1.0	
SVM	0.61	0.41	0.99	0.01	0.75	0.02	0.61
Logistic Regression	0.63	0.50	0.85	0.23	0.72	0.32	0.61
Decision Tree	0.64	0.51	0.84	0.26	0.72	0.35	0.61

Table 2. Performance Comparison Table from Cross Validations (Mean Accuracy) and F_2 Scores

Model Type	Mean Accuracy	F_2 Scores
SVM	0.616484732966742	0.4458030299073793
Logistic Regression	0.6167338755249313	0.5284867293866086
Decision Tree	0.6192200655583404	0.539233472799332

As can be seen from both tables above, although the Mean Accuracy for the Decision Tree model is the highest, since all three models have very similar accuracies, it is impossible to distinguish which one is the best based on just looking at this column of values. So, it is necessary to look at the Precision, Recall and F1-score values carefully.

To start, it was easy to rule out one of the models: SVM, because it has the largest gap between the 0.0 and 1.0 values for both Recall and F1-Score. For the Recall, it is to the point where it is practically 1 and 0, which is extremely poor performance for a model. Then, for the remaining two models, it was difficult to choose between them since both the SVM and Logistic Regression models have extremely similar values across the entire table. However, to break the tie, it can be seen that the Decision Tree model has higher values for both Precision-0.0 and Precision-1.0, and it has a higher Recall-1.0 and F1Score-1.0, reducing the gap between the two target categories. It can also be seen that the Decision Tree has the highest F-Beta score, $\beta=2$. Therefore, it can be concluded that Decision Tree is the best model.

It is important to note that even though 0.61 accuracy seems low, based on research, the benchmark for accuracies for the dataset is around 0.5. Hence, this model is going above this benchmark, an indication of our model being good enough.

3 GRAPHS FOR THE BEST PERFORMING ALGORITHM

The three graphs chosen to depict the best model and its performance are as follows: a Confusion Matrix, an AUC-ROC Curve, and a Feature Importance Graph.

The confusion matrix, shown in **Figure 6** below, depicts the exact number of true positives, true negatives, false positives and false negatives the model guesses on the test set. It is clear that the best model was able to predict the “no stroke” category better than the “stroke” category.

Next, the AUC-ROC Curve, shown in **Figure 7**, depicts how good the model is overall, with better models having larger areas under the curve. The best model produced an area of 0.55, depicting that it is not purely randomly guessing, but rather has at least slight predictive power.

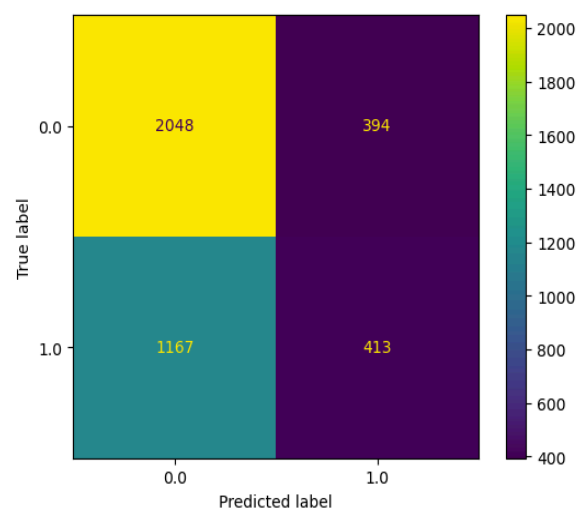


Figure 6. Confusion Matrix of Best Model

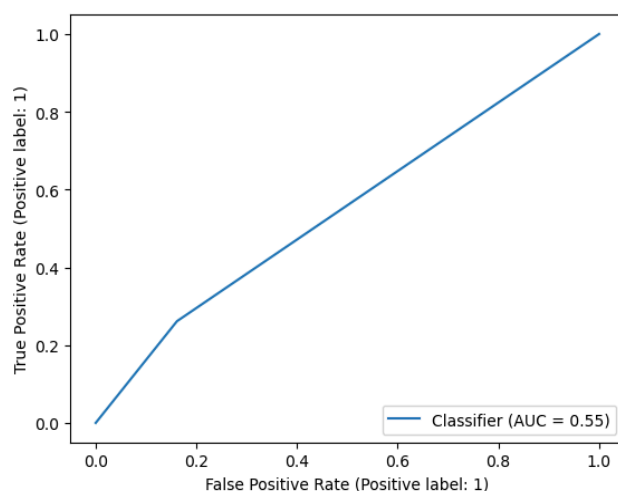


Figure 7. AUC-ROC Curve of Best Model

Finally, the feature importance graph depicts which features had the most impact on predicting the stroke output. According to the graph, the *age*, *avg_glucose_level* and *heart_disease* features are the top three most important features for the best model. This matches the top 3 correlations to the stroke feature shown in **Figure 1** (Pearson's correlations), but just in a different order.

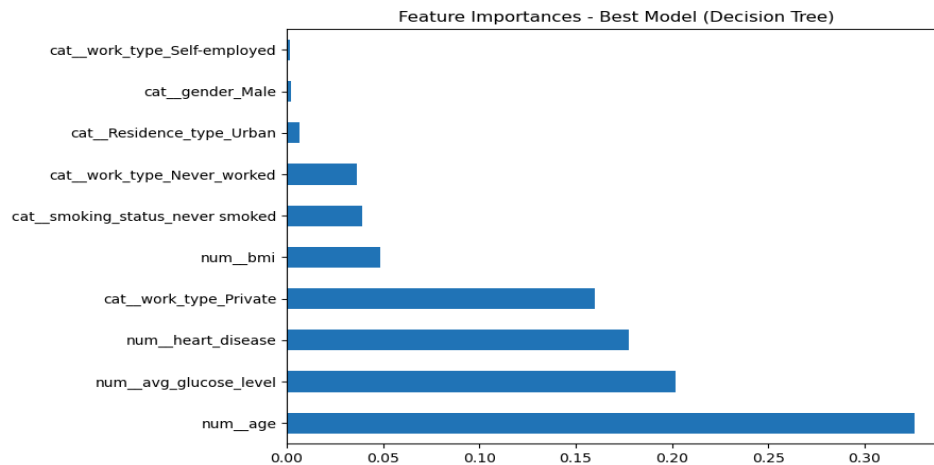


Figure 8. Feature Importance Graph of Best Model

LIMITATIONS

One limitation we experienced is that the number of instances in the original dataset were limited and were very imbalanced for the target variable. This was causing some values of the classification report for all models to be 0.0 because the results were inconclusive and resulting in models not predicting the *stroke* category at all. To work around this, we merged another dataset (<https://www.kaggle.com/datasets/teamincrito/stroke-prediction>) with the original dataset to add 15000 more instances and to make the target variable more balanced. This fixed the problem of inconclusive results in most classification reports.

Another limitation we ran into was GridSearchCV taking too long. We tried running it many times with supplied kernel values *rbf*, *linear*, and *poly*, however, even though we waited 1000 minutes, it would never finish running. To work around this, we ran it with only *linear*, and then again with only *rbf* and then finally the last time with only *poly*. The *rbf* and *linear* ones produced outputs fast, but the *poly* one got stuck. This was a problem because we had to forget about the *poly* kernel and hence, we are uncertain if a *poly* kernel could have provided an even better set of hyperparameters. So, our best model has been chosen based on the assumption that the *poly* kernel did *not* have the best hyperparameters for the SVM model.

CONCLUSION

In short, we have learned that a person's age is the number one predictor of whether or not they will have a stroke. This was surprising to us because we all initially assumed after looking at the correlation coefficients (**Figure 1**) that heart disease would be the number one predictor. We have also learned that a high accuracy (0.8 and above) cannot always be possible, and it depends entirely on the dataset (since we were hoping for a higher accuracy than 0.61). It is also impossible to figure out the highest accuracy until a wide range of models, hyperparameters and underfitting/overfitting techniques are experimented with.

APPENDIX 1

1 – Look at the big picture and frame the problem

Frame the problem

1. Supervised learning – training examples are labeled
2. A classification task – predict a category (0/1)
3. Batch learning
 - Small data set
 - No continuous flow of data coming into the system
 - No need to adjust to changing data rapidly

Look at the big picture

Predictions will be used to help inform physicians of patients at risk. Physicians can use the information to perform more tests and more importantly, can discuss future lifestyle changes with the patient to decrease the risk, such as quitting smoking, reducing weight, taking preventive medications.

```
# Import libraries

import sklearn
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

2 – Load the dataset

```
# Load the database from our Github repo

url = "https://github.com/PranavKulkarni33/EECS3401-Group-
Project/raw/main/Dataset.csv"
dataset = pd.read_csv(url)
```

Create a backup copy of the dataset

```
dataset_backup = dataset
```

2.1 Take a quick look at the data structure using head, info and describe

```
dataset.head()
```

```
dataset.info()
```

```
dataset.describe()
```



```
dataset.shape
```

3 – Explore and visualize the data to gain insights

3.1 Plot a histogram of the data

```
dataset.hist(figsize=(24, 16))  
plt.show()
```

3.2 Look for correlations between the features

```
# Check for correlation between attributes using sns.pairplot.  
sns.pairplot(dataset)
```

Look for correlations using Pearson correlation coefficient. Plot some graphs

```
#corr method has pearson standard correlation coefficient as the default  
  
corr_matrix = dataset.corr(numeric_only=True)  
corr_matrix
```

Look at correlations with regard to our target

```
corr_matrix["stroke"].sort_values(ascending=False)
```

Plot Heart_Disease vs Stroke in a Categorical Plot Using Count

```
sns.catplot(data=dataset, x= 'heart_disease', kind='count', hue = 'stroke')
```

Plot Avg_Glucose_Level vs Stroke in a Box Plot

```
sns.boxplot(data = dataset, x = "stroke", y ="avg_glucose_level")
```

Plot Age vs Stroke in a Box Plot

```
sns.boxplot(data = dataset, x = "stroke", y ="age")
```

4. Prepare the data for Machine Learning Algorithms

4.1 Check for duplicate rows and remove them if any

```
# Check for number of duplicate rows  
  
dataset.duplicated().sum()
```

There are zero duplicated rows and so nothing to delete

4.2 Handle the missing values

First check if there are any missing values represented with a '?'

```
# Check how many '?' there are in each column
dataset.isin(['?']).sum()
```

There are zero missing values represented with a '?', so there is nothing to replace with null (nan)

Next, check how many "Unknown" values there are

```
# Check how many 'Unknown' there are in each column
dataset.isin(['Unknown']).sum(axis=0)
```

For the smoking_status feature, we will replace all "Unknown" values to NaN

```
# replace all "Unknown" with NaN
dataset = dataset.replace('Unknown', np.nan)
```

Next, handle the missing (nan) values.

```
# count how many NaN values in each column
dataset.isna().sum()
```

_There are only 201 missing values in the "BMI" column and 1544 missing values in the "Smoking_Status" column. We will fill these missing values with the average/mean and most frequent value later._

For the id feature, we will delete the whole feature

```
dataset.drop(labels=['id'], axis=1, inplace=True)
```

```
dataset.info()
```

4.3. Create a pipeline that will:

1. Fill in the missing numerical values with the mean using a SimpleImputer
2. Scale the numerical columns using StandardScaler. Do not scale the target
3. Fill in the missing categorical values with the most_frequent value using SimpleImputer
4. Encode the categorical columns using OneHotEncoder

```
num_cols = dataset.select_dtypes(include='number').columns.to_list()
cat_cols = dataset.select_dtypes(exclude='number').columns.to_list()

# Exclude the target from numerical columns
num_cols.remove("stroke")

# Create pipelines for numeric and categorical columns
num_pipeline = make_pipeline(SimpleImputer(strategy='mean'), StandardScaler())
cat_pipeline = make_pipeline(SimpleImputer(strategy='most_frequent'), OneHotEncoder())

# Use ColumnTransformer to set the estimators and transformations

preprocessing = ColumnTransformer([('num', num_pipeline, num_cols),
                                   ('cat', cat_pipeline, cat_cols)],
                                  remainder='passthrough'
                                  )
```

```
# show numerical columns
num_cols
```

```
# show categorical columns
cat_cols
```

```
# Show the pipeline
preprocessing
```

```
# Apply the preprocessing pipeline on the dataset

dataset_prepared = preprocessing.fit_transform(dataset)

# Scikit-learn strips the column headers, so just add them back on afterward.
feature_names=preprocessing.get_feature_names_out()
dataset_prepared = pd.DataFrame(data=dataset_prepared, columns=feature_names)

dataset_prepared.shape # column sizes have increased due to ONEHOTENCODER (shows
pipeline worked correctly)
```

```
# display what dataset looks like after pipeline
dataset_prepared
```

Below confirms that there are no null values anymore, so our pipeline worked correctly

```
#Check how many NaN values there are
dataset_prepared.isna().sum()
```

Next, find the value count of the target column, Stroke, to make sure it only has two categories (yes or no)

```
# Number of different categories for stroke column, and how many values are in each one
dataset["stroke"].value_counts()
```

This confirms it only has two categories, as required, so we don't have to do any modifications

5 - Select a model and train it

5.1 Use Grid Search to find the best hyperparameter values for SVM, Logistic Regression and Decision Tree

```
from sklearn.model_selection import train_test_split

X = dataset_prepared.drop(["remainder__stroke"], axis=1)
y = dataset_prepared["remainder__stroke"]
```

```
X_train, X_validation_test, y_train, y_validation_test = train_test_split(X, y,
test_size=0.4, random_state=42)

X_validation, X_test, y_validation, y_test = train_test_split(X_validation_test,
y_validation_test, test_size=0.5, random_state=42)

print(X_train.shape, y_train.shape, X_validation.shape, y_validation.shape,
X_test.shape, y_test.shape)
```

```
from sklearn.model_selection import GridSearchCV
```

Grid Search for SVM, for Linear and RBF Kernels separately

```
# GridSearchCV for SVM model with linear kernel only

# code author luisguiserrano
from sklearn.svm import SVC

svm_parameters = {'kernel': ['linear'],
                  'C': [0.01, 0.1, 1, 10],
                  'gamma': [0.01, 1, 10]
                  }

svm = SVC()
svm_gs = GridSearchCV(estimator = svm,
                      param_grid = svm_parameters)
svm_gs.fit(X_train.iloc[:10000], y_train.iloc[:10000])
```

```
linear_svm_winner = svm_gs.best_estimator_  
linear_svm_winner.score(X_validation, y_validation)
```

```
# show best parameter values for svm model with a linear kernel  
linear_svm_winner.get_params()
```

```
# GridSearchCV for SVM model with rbf kernel only
```

```
# code author luisguiserrano  
from sklearn.svm import SVC
```

```
svm_parameters = {'kernel': ['rbf'],  
                  'C': [0.01, 0.1, 1, 10],  
                  'gamma': [0.01, 1, 10]  
                  }
```

```
svm = SVC()  
svm_gs = GridSearchCV(estimator = svm,  
                      param_grid = svm_parameters)  
svm_gs.fit(X_train.iloc[:10000], y_train.iloc[:10000])
```

```
rbf_svm_winner = svm_gs.best_estimator_  
rbf_svm_winner.score(X_validation, y_validation)
```

```
# show best parameter values for svm model with a rbf kernel  
rbf_svm_winner.get_params()
```

Grid Search for Logistic Regression

```
# GridSearchCV for Logistic Regression
```

```
from sklearn.linear_model import LogisticRegression
```

```
logreg_parameters = {'penalty': ['l1', 'l2', 'elasticnet', 'None'],  
                     'C': [0.01, 0.1, 1, 10, 100],  
                     'solver': ['lbfgs', 'liblinear', 'newton-cg', 'newton-cholesky',  
                                'sag', 'saga']  
                     }
```

```
logreg = LogisticRegression()  
logreg_gs = GridSearchCV(estimator = logreg,  
                         param_grid = logreg_parameters)  
logreg_gs.fit(X_train.iloc[:10000], y_train.iloc[:10000])
```

```
logreg_winner = logreg_gs.best_estimator_  
logreg_winner.score(X_validation, y_validation)
```

```
# show best parameter values for logistic regression model
logreg_winner.get_params()
```

Grid Search for Decision Tree

```
# GridSearchCV for Decision Tree

from sklearn.tree import DecisionTreeClassifier

dectree_parameters = {'criterion': ['gini', 'entropy', 'log_loss'],
                      'max_depth': [6, 10, 12],
                      'max_leaf_nodes': [2, 4, 6, 10, 50],
                      'min_samples_leaf': [2, 4, 6, 10, 50],
                      }

dectree = DecisionTreeClassifier()
dectree_gs = GridSearchCV(estimator = dectree,
                          param_grid = dectree_parameters)
dectree_gs.fit(X_train.iloc[:10000], y_train.iloc[:10000])

dectree_winner = dectree_gs.best_estimator_
dectree_winner.score(X_validation, y_validation)
```

```
# show best parameter values for decision tree model
dectree_winner.get_params()
```

5.2 Train Models using Best Hyperparameter Values from Grid Search Results

Split the dataset into a training dataset (80%) and testing dataset.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
print(X_train.shape, y_train.shape, X_test.shape, y_test.shape)
```

5.2.1 Train model #1: SVM

```
from sklearn.svm import SVC

model_svm = SVC(kernel='rbf', C=10, gamma=0.01)
model_svm.fit(X_train, y_train)
```

5.2.2 Train model #2: Logistic Regression

```
# code from DataCamp

from sklearn.linear_model import LogisticRegression
```

```
model_logreg = LogisticRegression(C=0.01, penalty='l2', solver='newton-cg')
model_logreg.fit(X_train, y_train)
```

5.2.3 Train Model #3: Decision Tree

```
# code from DataCamp

from sklearn.tree import DecisionTreeClassifier

model_dectree = DecisionTreeClassifier(max_depth=6)
model_dectree.fit(X_train, y_train)
```

6 – Test all 3 models on the test set, and report the classification reports

```
from sklearn.metrics import classification_report
```

```
svm_y_predict = model_svm.predict(X_test)
print(classification_report(y_test, svm_y_predict))
```

```
logreg_y_predict = model_logreg.predict(X_test)
print(classification_report(y_test, logreg_y_predict))
```

```
dectree_y_predict = model_dectree.predict(X_test)
print(classification_report(y_test, dectree_y_predict))
```

7 – Evaluate all 3 models using cross validation with cv = 5. Report on the cross_val_score, and the mean of the accuracy scores.

```
from sklearn.model_selection import cross_val_score
```

```
svm_scores = cross_val_score(model_svm, X_train, y_train, cv=5)
svm_scores
```

```
logreg_scores = cross_val_score(model_logreg, X_train, y_train, cv=5)
logreg_scores
```

```
dectree_scores = cross_val_score(model_dectree, X_train, y_train, cv=5)
dectree_scores
```

```
print(f'SVM Model Cross-Validation Mean Accuracy: {svm_scores.mean()}')
print(f'Logistic Regression Model Cross-Validation Mean Accuracy: {logreg_scores.mean()}')
print(f'Decision Tree Cross-Validation Mean Accuracy: {dectree_scores.mean()}')
```

8 - Evaluate all 3 models using F-Beta Score, with Beta = 2

```
from sklearn.metrics import fbeta_score
```

```
print("SVM Model F2-Score:")  
fbeta_score(y_test, svm_y_predict, average='macro', beta=2)
```

```
print("Logistic Regression Model F2-Score:")  
fbeta_score(y_test, logreg_y_predict, average='macro', beta=2)
```

```
print("Decision Tree Model F2-Score:")  
fbeta_score(y_test, dectree_y_predict, average='macro', beta=2)
```

9 - Best performing model

```
model_dectree
```

Print Classification Report

```
bestmodel_y_predict = model_dectree.predict(X_test)  
print(classification_report(y_test, bestmodel_y_predict))
```

Print Confusion Matrix Display

```
from sklearn.metrics import ConfusionMatrixDisplay  
ConfusionMatrixDisplay.from_predictions(y_test, bestmodel_y_predict)
```

AUC-ROC

```
from sklearn.metrics import RocCurveDisplay  
RocCurveDisplay.from_predictions(y_test, bestmodel_y_predict)
```

Feature Importance Graph

```
# code referenced from https://stackoverflow.com/questions/44511636/plot-feature-importance-with-feature-names  
  
plt.figure(figsize=(8,6))  
feat_importance = pd.Series(model_dectree.feature_importances_, index=X.columns)  
feat_importance.nlargest(10).plot(kind='barh')  
plt.title("Feature Importances - Best Model (Decision Tree)")  
plt.show()
```


APPENDIX 2

Video Link: https://drive.google.com/file/d/1dfXsvJSLjBRBUNZTwHayKpxGb_n98Yag/view

Dataset Link: <https://www.kaggle.com/datasets/fedesoriano/stroke-prediction-dataset>

- Merged Dataset Link: <https://www.kaggle.com/datasets/teaminciribo/stroke-prediction>

Executed Jupyter Notebook Link: <https://github.com/PranavKulkarni33/EECS3401-Group-Project/blob/main/project.ipynb>