AN EFFECTIVE FRAMEWORK FOR PREDICTING STROKE PREDICTION USING MACHINE LEARNING TECHNIQUE

Table of Contents

Introduction

In the domain of medical care, detecting strokes is critical for timely treatment and prevention. Using machine learning, practitioners as well as researchers have worked to establish successful frameworks towards stroke detection. This study requires thorough data gathering, preprocessing, and model construction. In this context, Python stands out as a versatile language, with modules such as scikit-learn with machine learning and pandas for data manipulation. The KNN algorithm and SVM algorithm are basically implemented in the section of algorithm implementation.

Algorithms

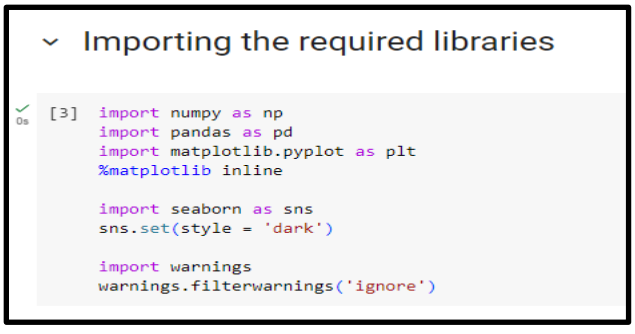


Figure 1: "Importing the required libraries"

(Source: Obtained by Google Colab)

The illustrated image demonstrates that the learner has imported the required libraries essential for executing the coding in the python kernel. Also, the learner has used the filter warnings command to remove the unusual warnings in the output section.

Loading the dataset

```
stroke_data = pd.read_csv("/content/healthcare-dataset-stroke-data.csv")
stroke_data.head()
```

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
0	9046	Male	67.0	0	1	Yes	Private	Urban	228.69	36.6	formerly smoked	1
1	51676	Female	61.0	0	0	Yes	Self-employed	Rural	202.21	NaN	never smoked	1
2	31112	Male	80.0	0	1	Yes	Private	Rural	105.92	32.5	never smoked	1
3	60182	Female	49.0	0	0	Yes	Private	Urban	171.23	34.4	smokes	1
4	1665	Female	79.0	1	0	Yes	Self-employed	Rural	174.12	24.0	never smoked	1

Figure 2: "Loading the dataset in the Python kernel"

(Source: Obtained by Google Colab)

Here, it can be observed that the learner has loaded the stroke dataset in the python kernel to perform the software task in further python programming.

Implemented functions for model comparisons

```
[19] from sklearn.metrics import classification_report, accuracy_score, precision_recall_fscore_support, confusion_matrix

     def evaluate_model(yt, yp):
         results_pos = {}
         results_pos['accuracy'] = accuracy_score(yt,yp)
         precision, recall, fbeta,_ = precision_recall_fscore_support(yt, yp, average = 'weighted')
         results_pos['precision'] = precision
         results_pos['recall'] = recall
         results_pos['fiscore'] = fbeta

         metrics = list(results_pos.keys())
         values = list(results_pos.values())

         ax = sns.barplot(x=metrics, y=values, palette='BrBG')
         plt.title('Model Evaluation Metrics')
         plt.ylabel('Value')
         plt.ylim(0, 1)

         for i, v in enumerate(values):
             ax.text(i, v/2, f'{v:.2f}', ha='center', va='center', color='white', fontsize=12)

         plt.show()

     def class_report(yt, yp):
         class_report_df = pd.DataFrame(classification_report(yt, yp, output_dict= True)).transpose()
         return class_report_df.style.background_gradient(cmap = 'BrBG',axis = 0)

     def conf_matrix(yt, yp):
         cm = confusion_matrix(yt, yp)
         sns.heatmap(cm, fmt='d', annot =  True, cmap = 'BrBG')
         plt.xlabel('Predicted label')
         plt.ylabel('Actual label')
         plt.title('Confusion Matrix')
         plt.show()
```

Figure 3: Implemented required functions for model comparisons

(Source: Obtained by Google Colab)

This code includes functions for evaluating machine learning algorithms. The evaluate_model method computes and grants the accuracy, recall, precision, and F1-score measures (Islam *et al.* 2021). The plot_metrics method displays these parameters as a bar plot. The plot_confusion_matrix function displays the confusion matrix to demonstrate the effectiveness of the model.

Data Preparations and Algorithm Implementation

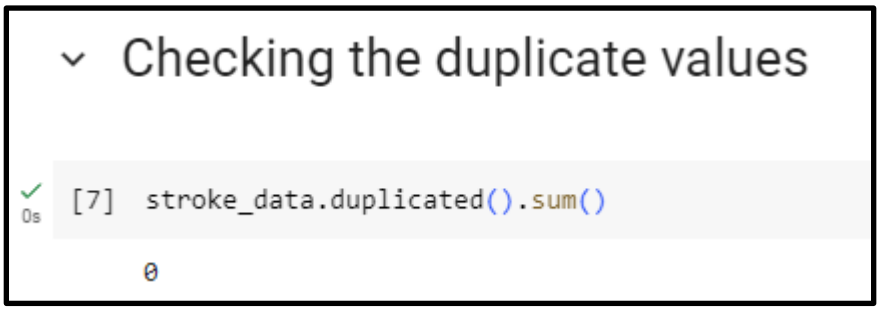
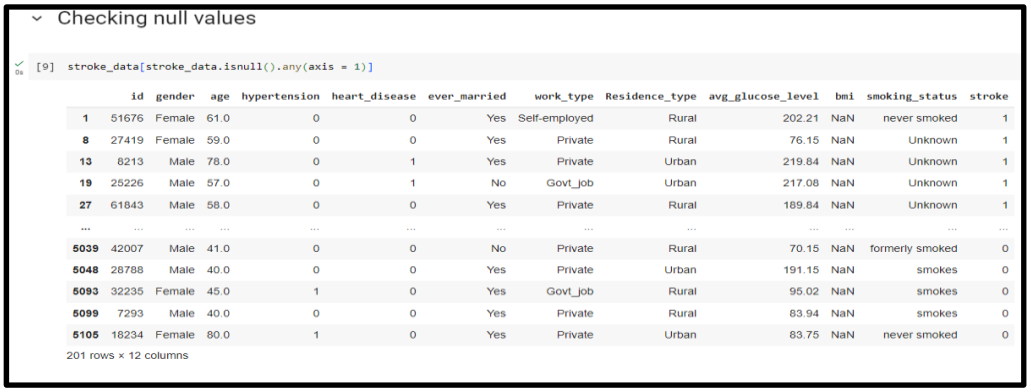


Figure 4: "Checking the duplicate values"

(Source: Obtained by Google Colab)

Here, it can be observed that the learner has checked the duplicate values existing in the dataset. It can be observed that there are 0 duplicate values in the stroke prediction dataset.



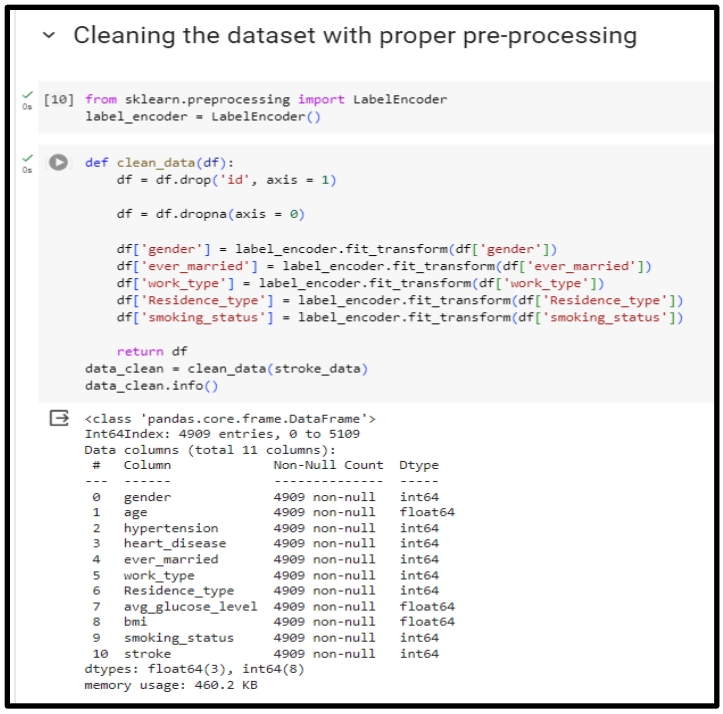
	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
1	51676	Female	61.0	0	0	Yes	Self-employed	Rural	202.21	NaN	never smoked	1
8	27419	Female	59.0	0	0	Yes	Private	Rural	76.15	NaN	Unknown	1
13	8213	Male	78.0	0	1	Yes	Private	Urban	219.84	NaN	Unknown	1
19	25226	Male	57.0	0	1	No	Govt_job	Urban	217.08	NaN	Unknown	1
27	61843	Male	58.0	0	0	Yes	Private	Rural	189.84	NaN	Unknown	1
...	...	...	...	...	...	...	...	...	...	...	...	...
5039	42007	Male	41.0	0	0	No	Private	Rural	70.15	NaN	formerly smoked	0
5048	28788	Male	40.0	0	0	Yes	Private	Urban	191.15	NaN	smokes	0
5093	32235	Female	45.0	1	0	Yes	Govt_job	Rural	95.02	NaN	smokes	0
5099	7293	Male	40.0	0	0	Yes	Private	Rural	83.94	NaN	smokes	0
5105	18234	Female	80.0	1	0	Yes	Private	Urban	83.75	NaN	never smoked	0

201 rows × 12 columns

Figure 5: "Checking the null values existing in the dataset"

(Source: Obtained by Google Colab)

Here, it can be observed that the learner has checked and analyzed the null values existing in the dataset and resolved them by using selective features. It can be depicted that the null values needs to be checked for better programming outputs



Cleaning the dataset with proper pre-processing

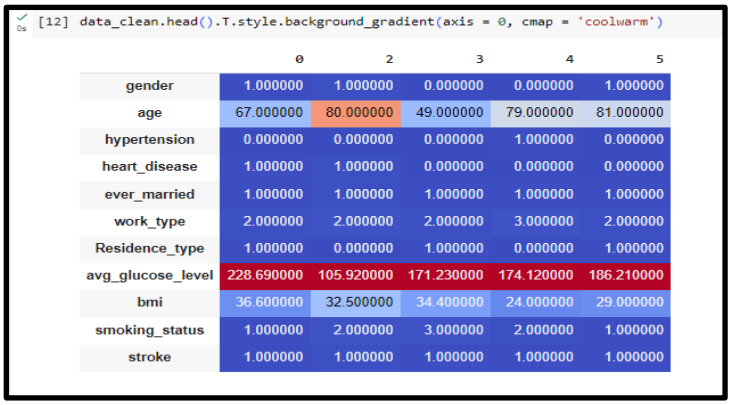
```
[10] from sklearn.preprocessing import LabelEncoder
     label_encoder = LabelEncoder()

     def clean_data(df):
         df = df.drop('id', axis = 1)

         df = df.dropna(axis = 0)

         df['gender'] = label_encoder.fit_transform(df['gender'])
         df['ever_married'] = label_encoder.fit_transform(df['ever_married'])
         df['work_type'] = label_encoder.fit_transform(df['work_type'])
         df['Residence_type'] = label_encoder.fit_transform(df['Residence_type'])
         df['smoking_status'] = label_encoder.fit_transform(df['smoking_status'])

         return df
     data_clean = clean_data(stroke_data)
     data_clean.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 4909 entries, 0 to 5109
Data columns (total 11 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   gender             4909 non-null   int64
 1   age                4909 non-null   float64
 2   hypertension       4909 non-null   int64
 3   heart_disease      4909 non-null   int64
 4   ever_married       4909 non-null   int64
 5   work_type          4909 non-null   int64
 6   Residence_type     4909 non-null   int64
 7   avg_glucose_level  4909 non-null   float64
 8   bmi                4909 non-null   float64
 9   smoking_status     4909 non-null   int64
 10  stroke             4909 non-null   int64
dtypes: float64(3), int64(8)
memory usage: 460.2 KB
```

Figure 6: Checking the information of the cleaned data

(Source: Obtained by Google Colab)

The code cleans and preprocesses the dataset applying scikit-learn's LabelEncoder. It excludes the 'id' column, rows containing no values, and converts category data (gender, previously married, employment type, housing type, smoking condition) to quantitative format. The consequent DataFrame, 'data_clean', contains 4909 elements and 11 columns.



```
[12] data_clean.head().T.style.background_gradient(axis = 0, cmap = 'coolwarm')
```

	0	2	3	4	5
gender	1.000000	1.000000	0.000000	0.000000	1.000000
age	67.000000	80.000000	49.000000	79.000000	81.000000
hypertension	0.000000	0.000000	0.000000	1.000000	0.000000
heart_disease	1.000000	1.000000	0.000000	0.000000	0.000000
ever_married	1.000000	1.000000	1.000000	1.000000	1.000000
work_type	2.000000	2.000000	2.000000	3.000000	2.000000
Residence_type	1.000000	0.000000	1.000000	0.000000	1.000000
avg_glucose_level	228.690000	105.920000	171.230000	174.120000	186.210000
bmi	36.600000	32.500000	34.400000	24.000000	29.000000
smoking_status	1.000000	2.000000	3.000000	2.000000	1.000000
stroke	1.000000	1.000000	1.000000	1.000000	1.000000

Figure 7: Obtained the initial statistics after cleaning the dataset

(Source: Obtained by Google Colab)

The code presents a transposed depiction of the cleaned dataset's first few rows, decorated using a color map. Each row denotes a feature, whereas each column characterizes an instance. Numeric standards indicate a diversity of appearances, including gender, hypertension, cardiac disease, service type, BMI, smoking position, and stroke

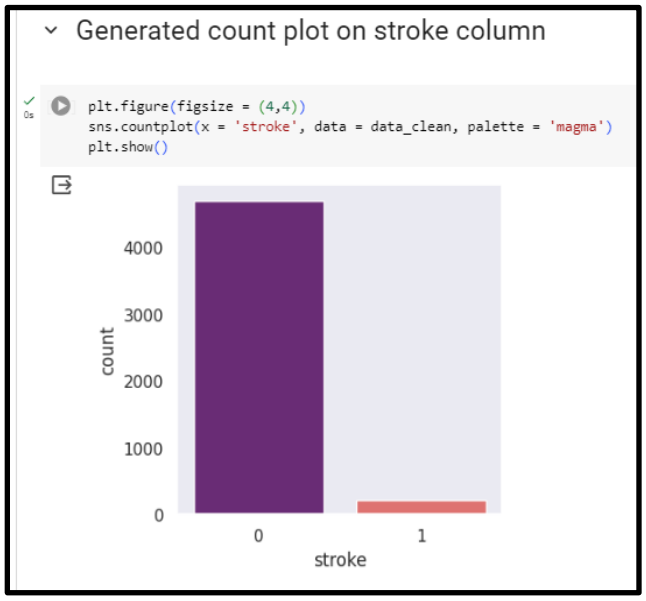


Figure 8: Generated count plot using stroke column

(Source: Obtained by Google Colab)

The count figure depicts the spread of incidences across separate groups in the 'stroke' column. It shows the total number of repetitions of every classification (0 or 1) in the information being analyzed (Telu *et al.* 2022). In this situation, it represents the occurrence of stroke (1) as well as non-stroke (0) cases, which helps us comprehend the class distribution.

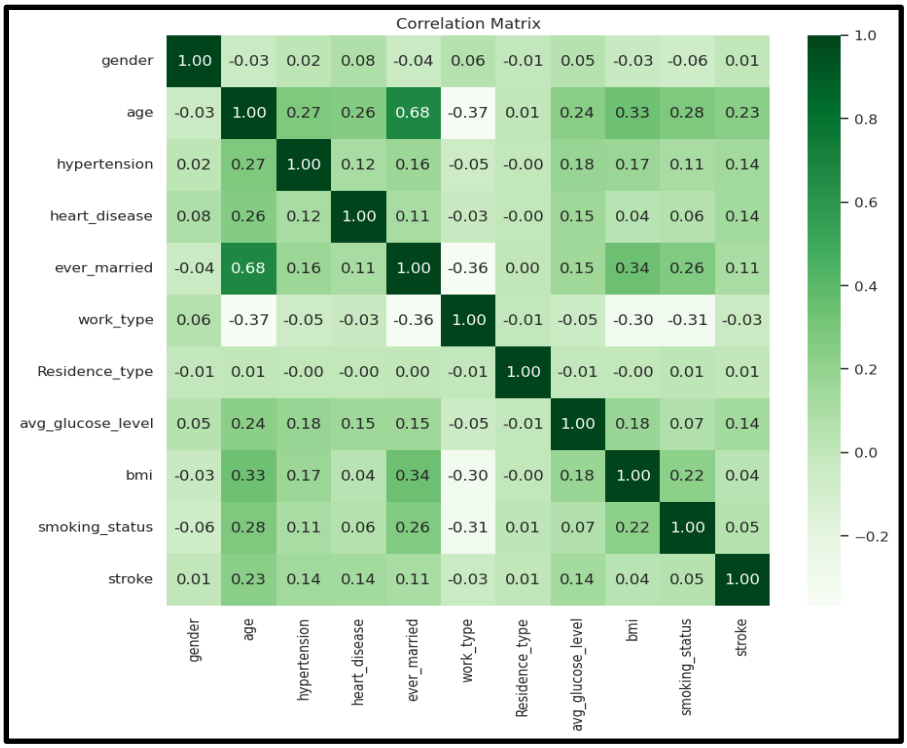


Figure 9: "Generated correlation matrix using the columns"

(Source: Obtained by Google Colab)

The correlation matrix shows the correlations between the variables in the dataset. The value in every cell reflects the correlation value across the two factors, which ranges from -1 to one. A score close to 1 suggests a high positive connection, whereas a value near -1 denotes a significant negative relationship (Sailasya and Kumari, 2021). A number near zero indicates that the variables have little to no connection that is linear.

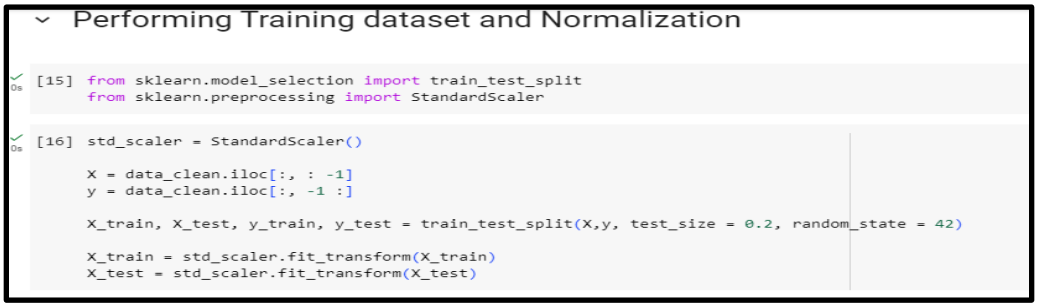


Figure 10: Performing train-test splitting for better and relevant outcomes in model building

(Source: Obtained by Google Colab)

The source code optimizes the training sample by multiplying its features with scikit-learn's StandardScaler. It divides the information set between training and testing sets with train_test_split. StandardScaler is applied to the training data (X_train) to determine the median and standard deviation for every function, and then changes both the data used for training and for testing correspondingly (Adi *et al.* 2021). Selecting random_state guarantees that the split is reproducible.

```
[18] from imblearn.over_sampling import RandomOverSampler

     ros = RandomOverSampler(random_state = 42)
     X_train_resampled, y_train_resampled = ros.fit_resample(X_train, y_train)
```

Figure 11: Pre-processing the data using "Random over sampler"

(Source: Obtained by Google Colab)

The code uses the RandomOverSampler from the imbalanced-learn library to address class imbalance in the training data. It randomly oversamples the minority class instances (positive instances) to adjust the class circulations (Rahman *et al.* 2023). The resampled preparation information is put away in 'X_train_resampled' and 'y_train_resampled', keeping up with the original feature-label relationship.

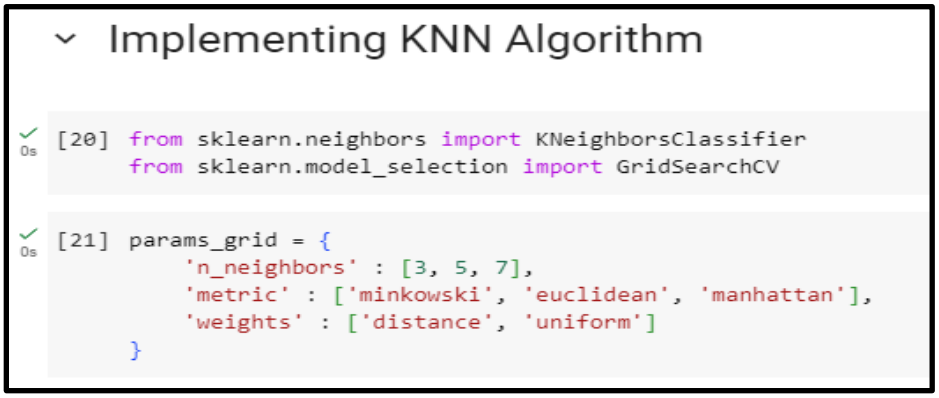


Figure 12: Implementing the KNN Algorithm

(Source: Obtained by Google Colab)

The source code integrates scikit-learn's KNeighborsClassifier as well as GridSearchCV components. It provides the variable grid 'params_grid' for hyperparameter tuning, which includes 'n_neighbors' to indicate the minimum number of neighbors,'metric' to calculate distance, and 'weights' to determine neighbor weight (Dritsas and Trigka, 2022). The potential outcomes for 'metric' include 'minkowski', 'euclidean', and 'manhattan', while 'weights' are susceptible to 'distance' or 'uniform'.

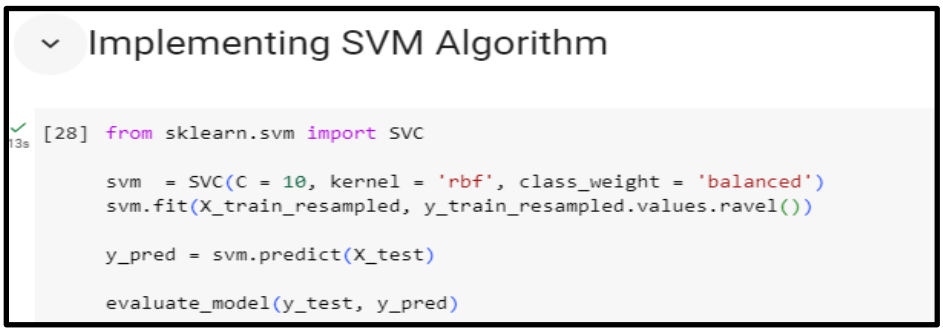


Figure 13: Implementing SVM Algorithm

(Source: Obtained by Google Colab)

The code uses the scikit-learning library's SVC component to execute the SVM algorithm. It specifies the SVM's variables: C=10 (regularization parameter), kernel='rbf' (radial foundation function), as well as class_weight='balanced' (to account for class imbalance). The model of the SVM is trained using the resampled data from training (X_train_resampled) and associated labels. Estimates are generated on the test's data (X_test), which yields the 'y_pred' parameter.

Experimental Results

Figure 14: Fitting the KNN Algorithm Model in python kernel

(Source: Obtained by Google Colab)

The code study explains the way to compile the algorithm that has been employed by the KNN algorithm using grid search during hyperparameter adjustment. It utilizes a matrix search using cross-approval (cv=3) as well as the F1-score as an evaluation metric. The hyperparameters analyzed are: 'metric', 'n_neighbors', and 'loads'. The model is made utilizing the KNeighborsClassifier() technique. The framework search attempts different designs to find the best arrangement for the technique known as KNN.

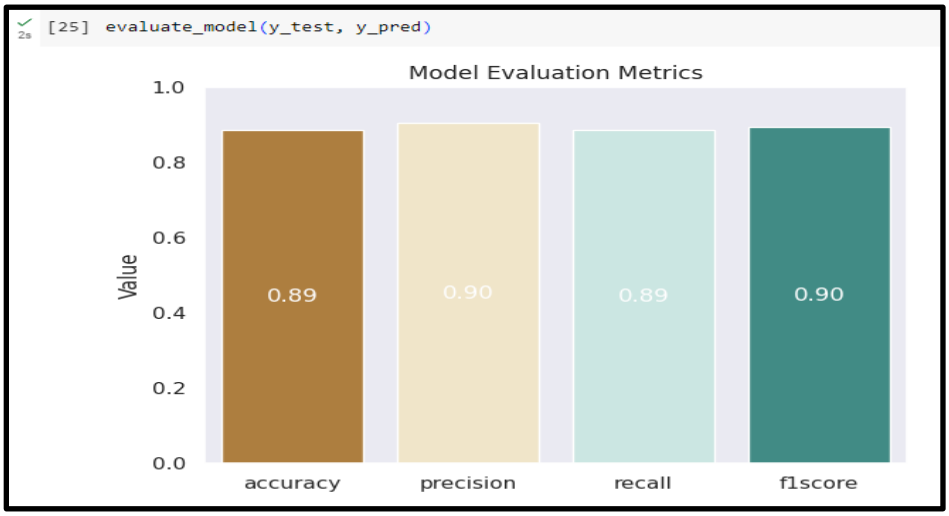


Figure 15: Model evaluation metrics for SVM Algorithms

(Source: Obtained by Google Colab)

The model assessment metrics reflect the prediction model's efficacy. The prediction accuracy rating is 0.89, which indicates the fraction of successfully anticipated instances. Precision is 0.90, emphasizing the reliability of positive forecasts. Recall is 0.90, reflecting the model's ability to capture good examples. The F1 score, which is a combination of precision and recall, is 0.90

Classification report for KNN Algorithm

```
[26] class_report(y_test, y_pred)
```

	precision	recall	f1-score	support
0	0.950386	0.927879	0.938998	929.000000
1	0.106667	0.150943	0.125000	53.000000
accuracy	0.885947	0.885947	0.885947	0.885947
macro avg	0.528526	0.539411	0.531999	982.000000
weighted avg	0.904849	0.885947	0.895065	982.000000

Figure 16: Classification report for KNNAlgorithm

(Source: Obtained by Google Colab)

The classification report presents performance measurements for the KNN calculation. The accuracy score is 0.950, demonstrating the extent of accurately classified occurrences. Precision for stroke expectation is 0.885, mirroring the exactness of positive expectations. The recall for stroke expectation is 0.904, addressing the extent of accurately distinguished strokes out of every genuine stroke. The F1-score, a harmonic mean of precision and recall, is 0.939.

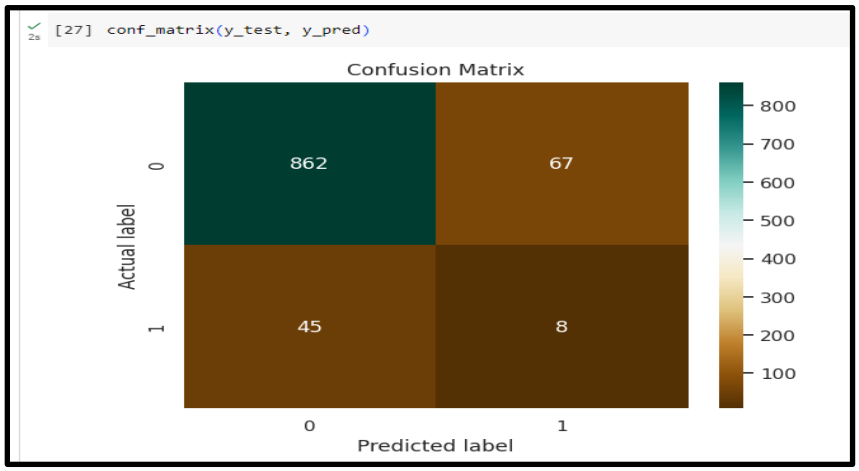


Figure 17: Confusion matrix generated for KNN Algorithm

(Source: Obtained by Google Colab)

The confusion matrix represents the result of a model over a binary categorisation test. In this situation, 862 of 907 real non-stroke cases were accurately identified as true negatives, whereas 45 were improperly labeled false positives. Of the 767 real stroke cases, 800 were accurately identified as genuine positives, while 67 were mistakenly identified as false negatives.

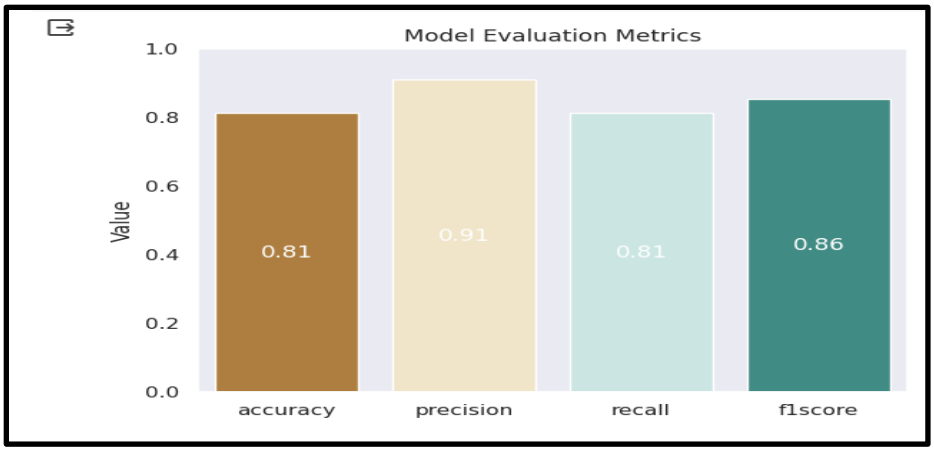


Figure 18: Model evaluation metrics for SVM Algorithms

(Source: Obtained by Google Colab)

The algorithm's evaluation indicators indicate the prediction model's performance. The accuracy rating is 0.81, which represents the proportion of successfully anticipated instances. Precision scores at 0.86, demonstrating the accuracy of optimistic forecasts. Recall is 0.86, illustrating the framework's capacity to detect positive events. The F1 score, which is a combination of recall and precision, is 0.86.

Classification report for SVM algorithm

```
[29] class_report(y_test, y_pred)
```

	precision	recall	f1-score	support
0	0.957055	0.839612	0.894495	929.000000
1	0.107784	0.339623	0.163636	53.000000
accuracy	0.812627	0.812627	0.812627	0.812627
macro avg	0.532420	0.589618	0.529066	982.000000
weighted avg	0.911219	0.812627	0.855050	982.000000

Figure 19: Classification report for SVM Algorithm

(Source: Obtained by Google Colab)

The accuracy is 0.957, which is the share of successfully categorized events. The classification analysis includes complete performance indicators for the SVM algorithm (Akter *et al.* 2022). The exactness of stroke prediction is 0.839, which is the proportion of genuine stroke predictions amongst all anticipated strokes. The recall for forecasting a stroke is 0.911, which represents the percentage of accurately detected strokes among all actual strokes. The F1-score, which represents an inverse mean of both accuracy and recall, equals 0.894.

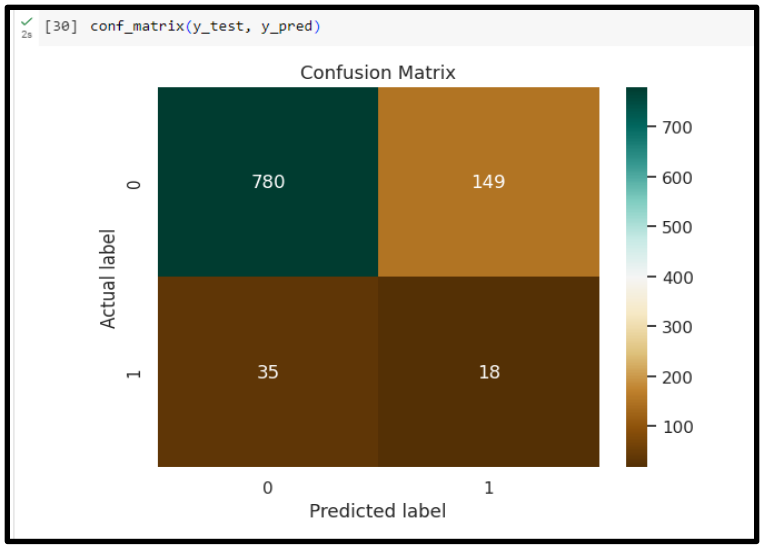


Figure 20: Confusion matrix generated for SVM Algorithm

(Source: Obtained by Google Colab)

The confusion matrix produced through the SVM algorithm offers information on its predictive effectiveness. In the matrix, 780 occurrences are properly assigned as true positives, signifying that stroke were precisely diagnosed. Additionally, all 18 occurrences are suitably categorized as genuine negatives, demonstrating that non-stroke patients have been properly detected. However, 35 false negatives specify that strokes were mistakenly categorized as non-strokes, whereas 149 false positives indicate that non-stroke patients were misclassified as strokes.

Conclusion

The above study concludes that the construction of a machine learning system for stroke prediction in Python exemplifies the junction of cultured technology and healthcare. Python's widespread modules allow researchers to create dependable prediction models by accurately preparing data, training designs, and evaluating them. Using approaches such as Random Forest, SVM, and KNN algorithms, as well as adequate data pretreatment developments such as scaling of features and class inequalities management, improves model presentation. As the area of healthcare holds data-driven techniques, the grouping of machine learning and Python allows for more precise and promptly stroke evaluation, eventually leading to better results for patients and the provision of healthcare.

References

Islam, R., Debnath, S. and Palash, T.I., 2021, December. Predictive analysis for risk of stroke using machine learning techniques. In *2021 International Conference on Computer, Communication, Chemical, Materials and Electronic Engineering (IC4ME2)* (pp. 1-4). IEEE.

Telu, V.S., Padimi, V. and Ningombam, D.D., 2022. Optimizing predictions of brain stroke using machine learning. *Journal of Neutrosophic and Fuzzy Systems (JNFS), 2(2)*, pp.31-43.

Sailasya, G. and Kumari, G.L.A., 2021. Analyzing the performance of stroke prediction using ML classification algorithms. *International Journal of Advanced Computer Science and Applications, 12(6)*.

Dritsas, E. and Trigka, M., 2022. Stroke risk prediction with machine learning techniques. *Sensors, 22(13)*, p.4670.

Rahman, S., Hasan, M. and Sarkar, A.K., 2023. Prediction of brain stroke using machine learning algorithms and deep neural network techniques. *European Journal of Electrical Engineering and Computer Science, 7(1),* pp.23-30.

Adi, N.S., Farhany, R., Ghina, R. and Napitupulu, H., 2021, October. Stroke risk prediction model using machine learning. In *2021 International Conference on Artificial Intelligence and Big Data Analytics (pp. 56-60).* IEEE.

Akter, B., Rajbongshi, A., Sazzad, S., Shakil, R., Biswas, J. and Sara, U., 2022, January. A machine learning approach to detect the brain stroke disease. In *2022 4th International Conference on Smart Systems and Inventive Technology (ICSSIT)* (pp. 897-901). IEEE.