Stroke Detection Using Different Machine Learning Models

CSE422 - Artificial Intelligence

CSE422 LAB PROJECT

BRAC University
Dhaka, Bangladesh

20 December 2022

Md Fuad Hasan - 20101345
Mustafiza Rahmani - 20101006
Ahnaf Tahmid - 20101555

Table of Contents:

Introduction:

According to the World Stroke Organization, stroke is still the third-leading cause of death, disability, and the second-leading cause of mortality globally (WSO). In accordance with the World Health Organization (WHO), 15 million people get a stroke annually. Five million pass away, and another five million suffer from lasting disabilities. These figures demonstrate that this is a significant issue that shouldn't be ignored. If there existed a predictor that could determine whether or not a person will get a stroke based on many characteristics, early intervention could save their life.

In this work, machine learning is used to create a model to forecast whether or not a person will have a stroke.

```
rawData = pd.read_csv('./healthcare-dataset-stroke-data.csv')
rawData.head(3)
```

✓ 0.9s

	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

Motivation:

Stroke care and diagnosis have been improved as a result of the detrimental effects it has on society. By methodically mining and storing the patient's medical records, it is feasible to develop prospects for better patient management with an increased synergy between technology and medical diagnosis. Therefore, it is crucial to investigate how these risk variables interact with one another in patient health records and comprehend how they each contribute differently to stroke prediction. For accurate stroke prediction, the numerous components in electronic health data are comprehensively analyzed in this study.

Dataset Description:

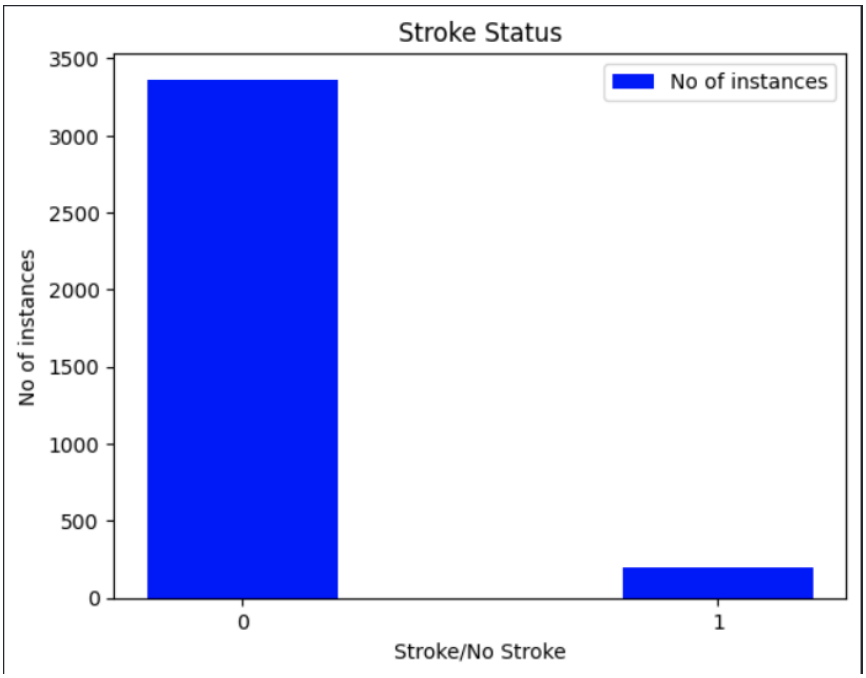
Source:
https://www.kaggle.com/datasets/fedesoriano/stroke-prediction-dataset?resource=download

Label, Features, No of instances

```
rawData.info()
✓ 0.7s
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3566 entries, 0 to 5108
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   id                 3566 non-null   int64
 1   gender             3566 non-null   object
 2   age                3566 non-null   float64
 3   hypertension       3566 non-null   int64
 4   heart_disease      3566 non-null   int64
 5   ever_married       3566 non-null   object
 6   work_type          3566 non-null   object
 7   Residence_type     3566 non-null   object
 8   avg_glucose_level  3566 non-null   float64
 9   bmi                3566 non-null   float64
 10  smoking_status     3566 non-null   object
 11  stroke             3566 non-null   int64
```

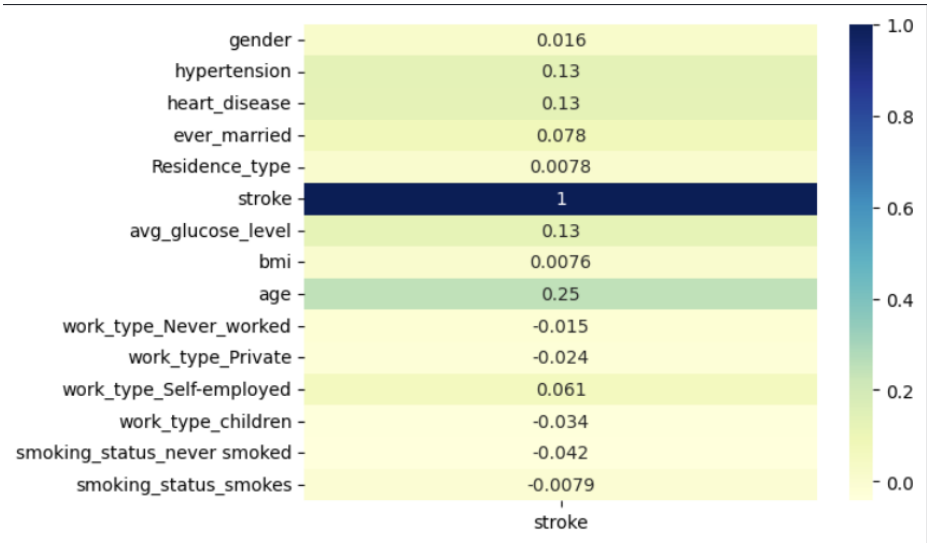
There are **12 column headers** in this dataset. Among them, *ID* isn't useful. We have **10 features** in our dataset. The last column (stroke) is our target column. Our target column/label has a **binary class** indicating if a stroke occurred or not. The dataset has **5111 unique data points**. The 10 types of features are as follows *gender*, *age*, *hypertension*, *heart_disease*, *ever_married*, *work_type*, *residence_type*, *average_glucose_level*, *bmi*, *smoking_status*.

The number of instances for stroke classes



The dataset doesn't have all equal instances for all classes. So the dataset we chose is extremely biased and will impact the result negatively.

Correlation of the features with stroke data



If the correlation is discovered, efficiency can be increased by keeping only one feature and getting rid of the other.
To examine for correlation, a heat map is generated using the Seaborn library.

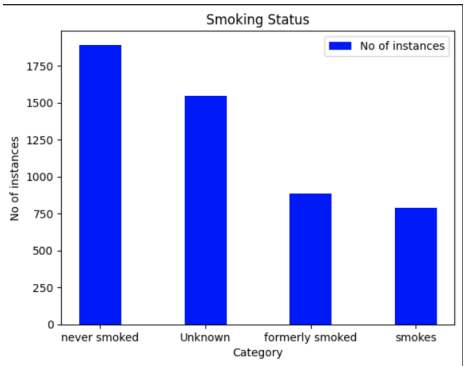
Dataset preprocessing:

Null values:
A null indicates that a variable doesn't point to any object and holds no value. It is commonly used to denote or verify the non-existence of something.

```
rawData.isnull().sum()
✓ 0.1s
id                   0
gender               0
age                  0
hypertension         0
heart_disease        0
ever_married         0
work_type            0
Residence_type       0
avg_glucose_level    0
bmi                201
smoking_status       0
stroke               0
dtype: int64
```

A feature such as BMI has 201 null values in our dataset.

Solution: we dealt with this problem using **SimpleImputer**, a **scikit-learn** class. It replaces the NaN values with a specified placeholder. In this case, we used the "mean" strategy.
And then we passed the BMI feature through a **fit()** method.



We also have **1892 rows** of 'unknown' data in the *smoking_status* column. As this is an important variable for stroke we need to handle this also.

Solution: Even though the number is quite high for unknown smoking_status we have **3566 entries** of data with a known value. As we can't assume unknown data we are dropping all the rows that have unknown smoking_status. As left-out data is still in a reasonable amount it should return good results.

Low records:

```
rawData["gender"].value_counts()
✓ 0.8s

Female    2994
Male      2115
Other        1
Name: gender, dtype: int64
```

We have only one record of the Other category for Gender. This will unnecessarily increase complexity and increase errors.

Solution: We will delete the row with this value.

Categorical features:

Categorical features

```
for i in rawData:
    #print(i)
    if rawData[i].dtypes == 'object':
        print(i,'=',rawData[i].unique())
✓ 0.4s

gender = ['Male' 'Female' 'Other']
ever_married = ['Yes' 'No']
work_type = ['Private' 'Self-employed' 'Govt_job' 'children' 'Never_worked']
Residence_type = ['Urban' 'Rural']
smoking_status = ['formerly smoked' 'never smoked' 'smokes']
```

In our dataset, we have *gender*, *ever_married*, *Residence_type*, *work_type* and *smoking_status* as our categorical values. Here *gender*, *ever_married* and *Residence_type is a binary categorical feature and work_type* and *smoking_status have more than 2 categories.*

Solution: We use the fit_transform() method to map the binary categorical values (string) to 0 and 1.

For the other ones, we used one hot encoding. Using a function called get_dummies() we converted categorical data into indicator variables.

Feature scaling:

```
rawData.describe()
```
✓ 0.7s

	id	age	hypertension	heart_disease	avg_glucose_level	bmi	stroke
count	3566.000000	3566.000000	3566.000000	3566.000000	3566.000000	3566.000000	3566.000000
mean	36785.750701	48.853898	0.125070	0.063937	108.982011	30.235208	0.056646
std	21239.999608	18.874375	0.330845	0.244675	48.292204	7.156410	0.231198
min	67.000000	10.000000	0.000000	0.000000	55.120000	11.500000	0.000000
25%	18042.750000	34.000000	0.000000	0.000000	77.460000	25.400000	0.000000
50%	37448.500000	50.000000	0.000000	0.000000	92.650000	28.893237	0.000000
75%	54984.250000	63.000000	0.000000	0.000000	116.677500	33.800000	0.000000
max	72915.000000	82.000000	1.000000	1.000000	271.740000	92.000000	1.000000

If the gap between data is too high AI model often makes mistakes. So we need to scale the data closer to get more accurate results. It is a technique for normalizing the variety of independent variables or features in data. It scales all the values in the range of -1 to 1. Without scaling features, the algorithm might favour the feature with larger magnitude values. The data table demonstrates that a feature like ID is not important and that maintaining this feature or column would be unnecessary. As a result, it is thrown off the table.
We used the **StandardScaler** library to scale age, BMI, and average_ glucose columns.

	avg_glucose_level	bmi	age
0	2.479202	0.889226	0.961273
1	1.930834	-0.187859	0.643316
2	-0.063211	0.316216	1.650179
3	1.289277	0.581757	0.007403
4	1.349126	-0.871730	1.597186
...	...	...	...
3560	-0.642849	-1.193174	0.431345
3561	-0.522324	-0.187859	1.650179
3562	0.336054	1.364404	1.703172
3563	-0.538062	0.050676	-0.734496
3564	1.186976	-0.648116	0.113388

3565 rows × 3 columns

Dataset splitting:

We split the data set by 80% of which was train data and the rest 20% was for testing using the stratified method. As a result, we will get the same ratio of positive and negative stroke data in test data as it was in the original dataset.

Model Training:

- **KNN:**
  One of the simplest machine learning algorithms, based on the supervised learning method, is K-Nearest Neighbour. The K-NN algorithm assumes that the new and existing cases are comparable, and it places the new instance in the category that is most like the existing categories. A new data point is classified using the K-NN algorithm based on similarity after storing all the existing data. This means that by utilizing the K-NN method, updated data can be quickly and accurately sorted into a suitable category. Using **n_neighbors=3** parameter, we determined the 3 closest neighbours by calculating the Euclidean distance.

	Precision	Recall	F1-Score	Support
0	0.94	0.99	0.96	673
1	0.00	0.00	0.00	40
Average	0.89	0.93	0.91	713

- **SVM:**
  The support vector machine (SVM), which has many kernel functions, is another paradigm for the binary classification problem. To categorize data points, an SVM model computes a hyperplane (or decision border) based on a feature set. Finding the hyperplane with the largest margin between the data points of two classes—which might assume many different shapes in an N-dimensional space—is the objective. The cost function for the SVM model is illustrated in a mathematical form:

$$J(\theta) = \frac{1}{2}\sum_{j=1}^{n} \theta^2 j$$

$$\theta^T x^{(i)} \geq 1,\ y^{(i)} = 1$$

$$\theta^T x^{(i)} \leq -1,\ y^{(i)} = 0$$

A linear kernel is used in the code above. Typically, kernels are used to fit multidimensional or challenging-to-separate data points.

	Precision	Recall	F1-Score	Support
0	0.94	1.00	0.97	673
1	0.00	0.00	0.00	40
Average	0.89	0.94	0.92	713

- **Decision Tree Classifier:**
  Classification and regression issues can be resolved using this supervised learning approach, although it is often preferred for doing so. It is a tree-structured classifier, where each leaf node represents the classification outcome and inside nodes represent the dataset's features. It is a great method for larger datasets. This method uses Entropy to choose the unique value from the features.

	Precision	Recall	F1-Score	Support
0	0.95	0.93	0.94	673
1	0.13	0.17	0.15	40
Avg	0.90	0.89	0.89	713

Our data set worked the best with this model. This model can predict the True values better despite the biases of the data set.

- **Logistic Regression:**
  The logistic regression function p(x) is the sigmoid function of f(x) : $p(x) = \frac{1}{1+\exp(f(x)}$. Thus, it's frequently either near 0 or 1. The adoption of a logistic regression (LR) model allows for the categorization of problems into binary or many classes using an intuitive equation. This is due to the fact that we are categorizing text based on a substantial feature set and producing a binary output (true/false or stroke/no stroke). We performed hyperparameter tuning to

produce the best results for each individual dataset, even though numerous parameters were examined before the LR model's maximum accuracies were obtained. The logistic regression hypothesis function can be defined mathematically as follows:

$$h_{\theta}(X) = \frac{1}{1+e^{-(\beta_{\theta}+\beta X)}}$$

A sigmoid function is used to convert the logistic regression's output to a probability value, with the aim of minimizing the cost function to get at

the best probability. As demonstrated in the cost function calculation:

	Precision	Recall	F1-Score	Support
0	0.94	1.00	0.97	673
1	0.00	0.00	0.00	40
Avg	0.89	0.94	0.92	713

This model works well when the input value is 0 or when the patient is not having a stroke then we get an F1-score of 0.97. But it cannot predict stroke well.

- **<u>Neural Network Classifier:</u>**
  This is an extra method we used to make our project better. Computers can make intelligent decisions with minimal human intervention thanks to neural networks. This is why they can learn and model complex, nonlinear relationships between input and output data. A basic neural network has interconnected artificial neurons in three layers. Neural networks can have a large number of hidden layers. Each hidden layer analyzes the output from the previous layer, processes it further, and passes it on to the next layer.

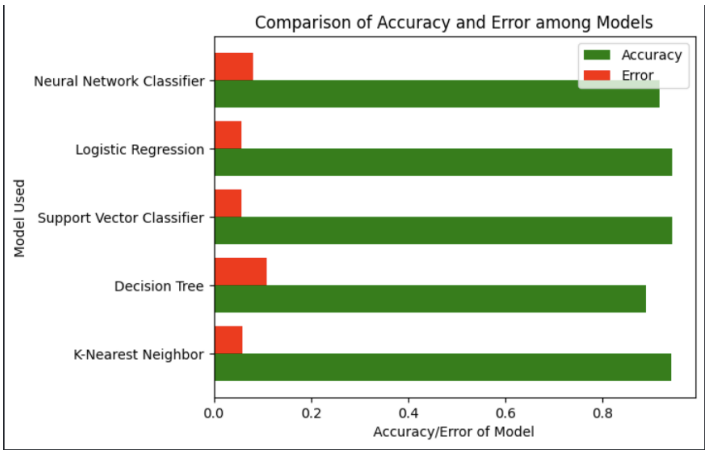
	Precision	Recall	F1-Score	Support
0	0.95	0.97	0.96	673

1	0.09	0.05	0.06	40
Avg	0.90	0.92	0.91	713

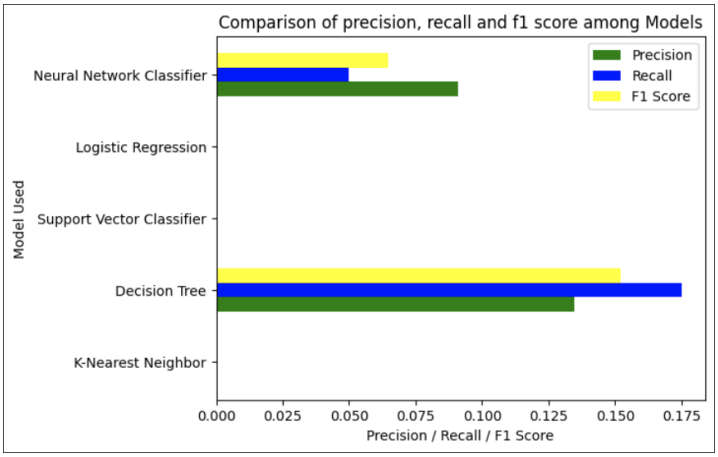
The hidden layer processes the data much better when the iteration is higher. The method learns by itself about the data during iteration. The more we iterate the better the result will be. From the result above we can see the F1 score is better than the rest of the methods.

Model Comparison analysis:

As we can see from the graph all the algorithm has pretty high accuracy while Logistic Regression, SVC, and KNN are slightly better. But that doesn't show the whole case. As we also need to check



As we can see from the graph all the algorithm has pretty high accuracy while Logistic Regression, SVC and KNN are slightly better. But that doesn't show the whole case. As we also need to check how the model is detecting positive results.



As we can see from here, while the accuracy was high. The precision, recall and F1 score of the algorithm are really bad. This is indicating our model is giving a lot of False Positive and False Negatives. For the Logistic regression, SVC and KNN algorithm there is 0 for precision, recall and F1 score. Because we have 0 True Positive.

Model testing:

We have tested the model using a set of unseen states. We used 20% data to train and then predicted the remaining 80% and checked it against our original data.

Conclusion:

The results show that, of the five models, the Decision Tree Classifier made the least error in directing the False Positives. It was more effective in figuring out the True and False positives and detecting the 1's better. The Neural Network model also managed to find some actual 1's. The rest of the models detected the 0's but failed to detect any 1's. The reason for that is because of the dataset being biased. Even for models that managed to find some True Positives, even for them, the F1 score is less than 0.2. So, overall all of the algorithms gave us very poor results. As detecting Stroke was our main goal. And making mistakes in identifying stroke most of the time can cost us dearly.

Future work:

In the future, we would like to collect more data and enrich our dataset so that the results are not as biased as it is now. Also, we need to try different preprocessing techniques and improve our algorithm so that we can get better results.

References:

1. Dataset-https://www.kaggle.com/code/ojoabimbola/stroke-prediction-accu-0-94-0-95
2. https://www.kaggle.com/code/casper6290/strokeprediction-99-acc
3. https://www.javatpoint.com/k-nearest-neighbor-algorithm-for-machine-learning
4. https://www.javatpoint.com/machine-learning-decision-tree-classification-algorithm
5. https://scikit-learn.org/stable/modules/neural_networks_supervised.html