United International University

Project Report

Course Name: Pattern Recognition Laboratory

Course Code: CSI 416 (A)

Project Name:

Performance Comparison for Machine Learning Classification Algorithms on a trending dataset in Kaggle (Stroke Prediction Dataset)

Team Members:

Name	Student ID	Email Address
Md.Tarek Hasan	011181076	mhasan181076@bscse.uiu.ac.bd
Mohammad Nazmush Shamael	011181062	mshamael181062@bscse.uiu.ac.bd
H.M. Mutasim Billah	011181290	hbillah181290@bscse.uiu.ac.bd
Arifa Akter	011181254	aakter181254@bscse.uiu.ac.bd
Sumayra Islam	011181123	sislam181123@bscse.uiu.ac.bd

Problem Definition

According to the World Health Organization (WHO), stroke is the second leading cause of death globally, responsible for approximately 11% of total deaths. In our project we want to predict stroke from a trending dataset in Kaggle and compare the machine learning classification algorithms.

Dataset Description

The dataset is used to predict whether a patient is likely to get stroke based on the input parameters like gender, age, various disease, and smoking status. Each row in the data provides relevant information about the patient

Attribute Information:

i. id: unique identifier
ii. gender: “Male”, “Female” or “Other”
iii. age: age of the patient
iv. hypertension: 0 if the patient doesn’t have hypertension, 1 otherwise
v. heart_disease: 0 if the patient doesn’t have any heart disease, 1 otherwise
vi. ever_married: “No” or “Yes”
vii. work_type: "children", "Govt_jov", "Never_worked", "Private" or "Self-employed"
viii. Residence_type: “Rural” or “Urban”
ix. avg_glucose_level: average glucose level in blood
x. bmi: body mass index [contains null values “N/A”]
xi. smoking_status: "formerly smoked", "never smoked", "smokes" or "Unknown"*
xii. stroke: 1 if the patient had a stroke or 0 if not

*note: "Unknown" in smoking_status means that the information is unavailable for this patient

The data contains 5110 instances with these 12 attributes.

Dataset Credit: fedesoriano

Our Approach:

We've used all features except id and stroke as the feature vector and stroke as class label. We've used some machine learning classification algorithms on this dataset and observed their performances. Moreover, we've compared these results based on accuracy and F1 score.

Data Preprocessing:

The full dataset has been split by 80-20, where 80% for training and 20% for testing. But FCNNs we have split the dataset into three parts 70-10-20, where 70% for training, 10% for validation and 20% for testing. We've changed the class label by doing XOR by 1 operation.

As only one feature (bmi) has null values in the dataset and as it's a continuous type data we've replaced those null value by median.

As sklearn libraries can work only with numeric valued data, so we've converted the text features into numeric value using LabelEncoder from sklearn.preprocessing. Then we've scaled the dataset using StandardScaler from sklearn.preprocessing.

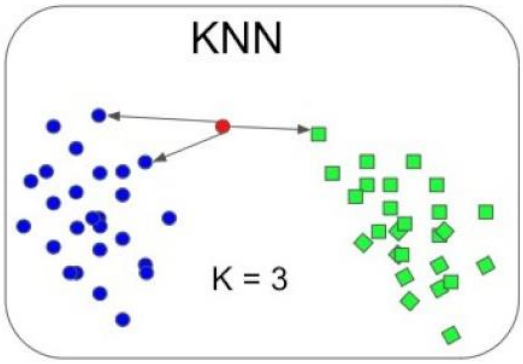
Algorithm Description:

K-Nearest Neighbor (KNN):

KNN is a supervised machine learning algorithm that means the dataset we use in this algorithm is labeled and also the algorithm is a lazy learner which means it'll use all the training samples each time while testing. Here K means how many closes/nearest samples to consider.

How it works:

To classify an unknown sample, at first KNN finds the closest k samples form training data then it finds which label has majority vote from those k closest sample then it classifies that unknown sample to the majority voted label. To find the k closest samples many ways can be applied such as Euclidian Distance, Hamming Distance, Cosine Similarity etc.



Results in our project:

In our project we used KNN classifier on 5110 samples with K=30 and we got Accuracy Score: 0.9510763209393346 and F1 Score: 0.9749247743229689

SVM:

The “Support Vector Machine” (SVM) is a supervised machine learning algorithm. SVM can predict an optimal decision boundary between the possible outputs. A new data point can be classified (by using the model) in the correct category. The decision boundary maximizes the distance to the nearest data points from different classes. Due to this property, SVM is referred to as a maximum-margin classifier.

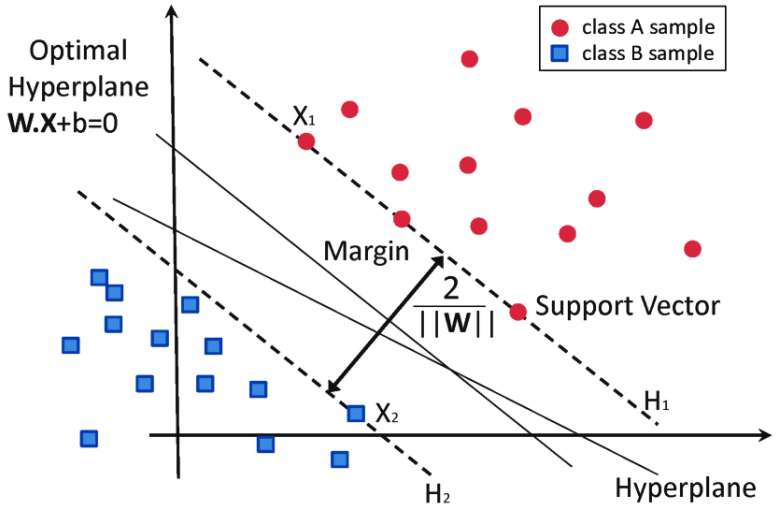


Figure: Classification of data by support vector machine (SVM)

Feature scaling for SVM:

Feature scaling is mapping the feature values of a dataset into the same range. Feature scaling is crucial for the SVM algorithm, as it considers distances between observations because the distance between two observations differs for non-scaled and scaled cases. The distance between data points affects the decision boundary SVM chooses**.** In other words, training an SVM over the scaled and non-scaled data leads to the generation of different models.

Standardization shifts the feature values to have a mean of zero, then maps them into a range such that they have a standard deviation of 1.

SVM implementation approach:

We've used Python sklearn library offered StandardScaler() function to standardize the data values into a standard format. Then we've performed the C-support vector classification from sklearn.svm.SVC.

Result on test data:

Accuracy: 0.9510763209393346, F1 Score: 0.9749247743229689

Naive Bayes classifiers:

Naive Bayes classifiers are a family of algorithms where all of them are based on the common principle of Bayes' theorem. Naive Bayes classifiers are Probabilistic classifiers. So, for each given input it provides probability for each class and selects the one with highest probability. Naive Bayes is an Eager learner as it will construct a generalize model using the training dataset before performing predictions on test dataset.

The reason Naive Bayes is naive because it makes the assumption that the features are independent of each other and carry equal weights. So in real world situations they are not usually effective as most features are not truly independent and doesn't always carry equal weight.

How it works:

According to Bayes' theorem,

$$P(\text{cause}|\text{effect}) = \frac{P(\text{effect}|\text{cause})P(\text{cause})}{P(\text{effect})}$$

If two events A and B are independent of each other then,

$$P(A, B) = P(A)P(B)$$

Let, Y be the class variable and X be the feature vector. Here, $X = \{x_1, x_2, x_3 \ldots\ldots\ldots, x_n\}$. Then,

$$P(Y|X) = \frac{P(Y,x1,x2,x3,\ldots\ldots\ldots,xn)}{P(x1,x2,x3,\ldots\ldots\ldots,xn)}$$

Let, $\alpha = 1/P(x_1, x_2, x_3, \ldots\ldots\ldots, x_n)$. So,

$P(Y|X) = P(Y, x_1, x_2, x_3, \ldots\ldots\ldots\ldots, x_n)$

$= \alpha P(Y) P(x_1|Y) P(x_2|x_1, Y) P(x_3|x_2,x_1,Y)\ldots\ldots\ldots P(x_n|x_{n-1},x_{n-1},x_{n-1},\ldots\ldots, x_2,x_1)$

$= \alpha P(Y) P(x_1|Y) P(x_2|Y) P(x_3|Y) \ldots\ldots\ldots P(x_n|Y)$

$= \alpha\ P(Y) \prod_{i=1}^{n} P(xi|Y)$

Using this formula, we will calculate probability for all possible values of Y and choose the one with the highest valued one as the output for X feature vector.

For continuous data we use Gaussian Naive Bayes. In GaussianNB, continuous values associated with each feature are assumed to be distributed according to a normal distribution. The conditional probability is given by:

$$P(x_i|Y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right)$$

Multinomial Naive Bayes uses Feature vectors represent the frequencies with which certain events have been generated by a multinomial distribution.

Naive Bayes Performance evaluation:

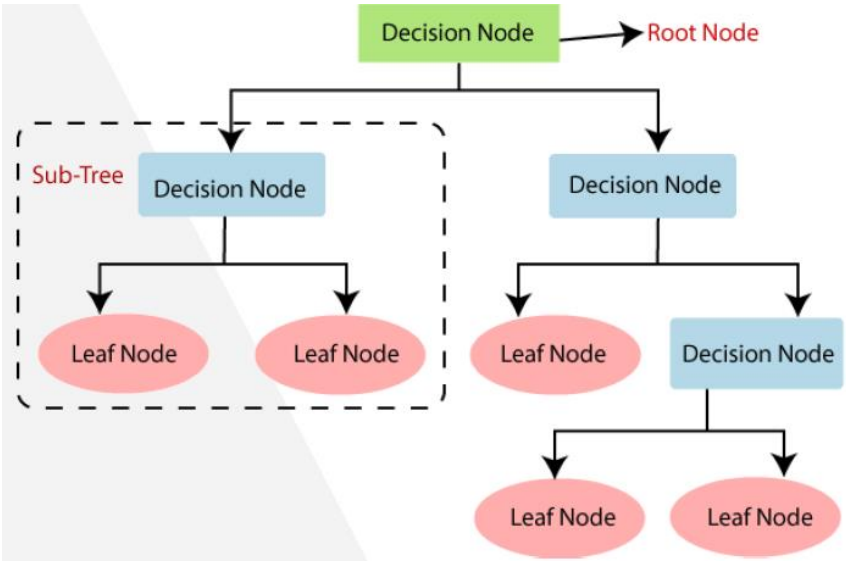
We've used sklearn library's sklearn.naive_bayes.GaussianNB and sklearn.naive_bayes.MultinomialNB to observe their performance. using GaussianNB we got accuracy score of 87.2798% and f1 score of 93.0332%. using Multinomial NB we got accuracy score of 81.2133% and f1 score of 89.3688%.

As mentioned before, Naive Bayes doesn't work well in real world situations because it assumes all features are equal and are independent. This is proven here with Naive Bayes classifiers having the worst accuracy and f1 score among the algorithms tested here

Decision Tree:

A Decision Tree is a supervised machine learning where the data is continuously split according to a certain parameter. It can be used for both classification and regression problems, but mostly preferred for solving classification problems.

It is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome.



Tree algorithms:

There are different tree algorithms such as-

ID3- The main goal of this algorithm is to find those categorical features, for every node, that will yield the largest information gain for categorical targets. ID3 uses information gain (entropy) measure for selecting attributes.

C4.5 (Gain Ratio) - It normalizes the information gain to reduce its bias. It tends to prefer unbalanced splits in which one partition is much smaller than the others. The attribute with the maximum gain ratio is selected as the splitting attribute.

CART - It is called Classification and Regression Trees algorithm. It basically generates binary splits by using the features and threshold yielding the largest information gain at each node (called the Gini index).

Classification of Decision tree:

The Scikit-learn library provides the module name **DecisionTreeClassifier** for performing multiclass classification on dataset.

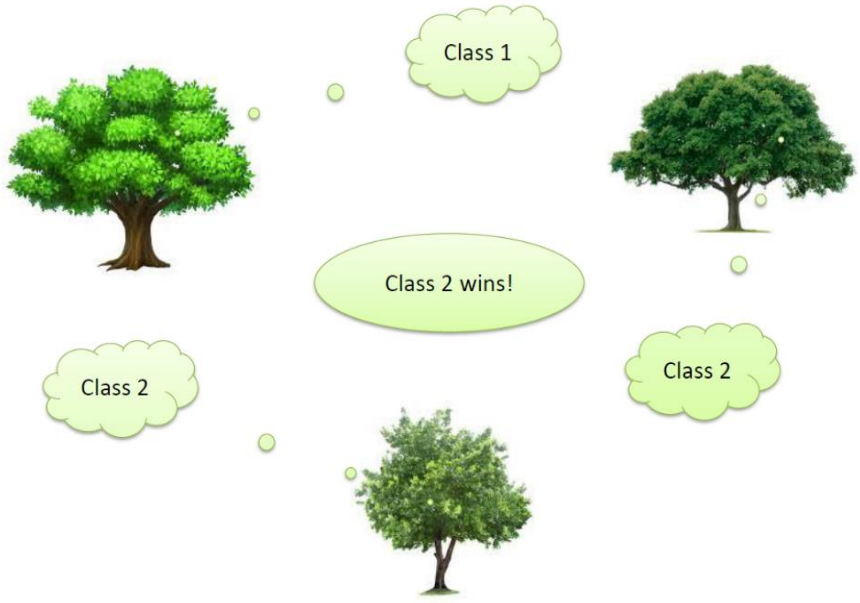
The function criterion is to measure the quality of a split. We use entropy which is for the information gain.

Results in our project:

Accuracy: 0.9060665362035225
F1 Score: 0.9503619441571872

Random Forest

The Random Forest also known as Random Decision Forest is an ensemble learning method for classification and regression that able to classify large amounts of data with accuracy.



The building block of Random Forest is Decision Tree. It builds multiple Decision Tree and takes the majority votes to provide the final decision. It builds the Decision Trees by using random subspaces technique which is also called attribute bagging. Efficiency of Random Forest can be improved by using Bagging/Bootstrap Aggregation and Adaptive Boosting.

We've used RandomForestClassfier from sklearn.ensemble, used parameter for n_estimators = 1000.

After using the Random Forest in our project, we've got accuracy score 0.9510763209393346 and F1 score 0.9748995983935742.

Logistic Regression:

Logistic Regression (or logit regression) is a binary-class classification method used in supervised learning to predict the dichotomous (binary) dependent variable (such that the dependent variable is categorical) on a given set of nominal, ordinal, interval or ratio-level independent variables. It uses a logistic function to model the probability of a binary dependent variable.

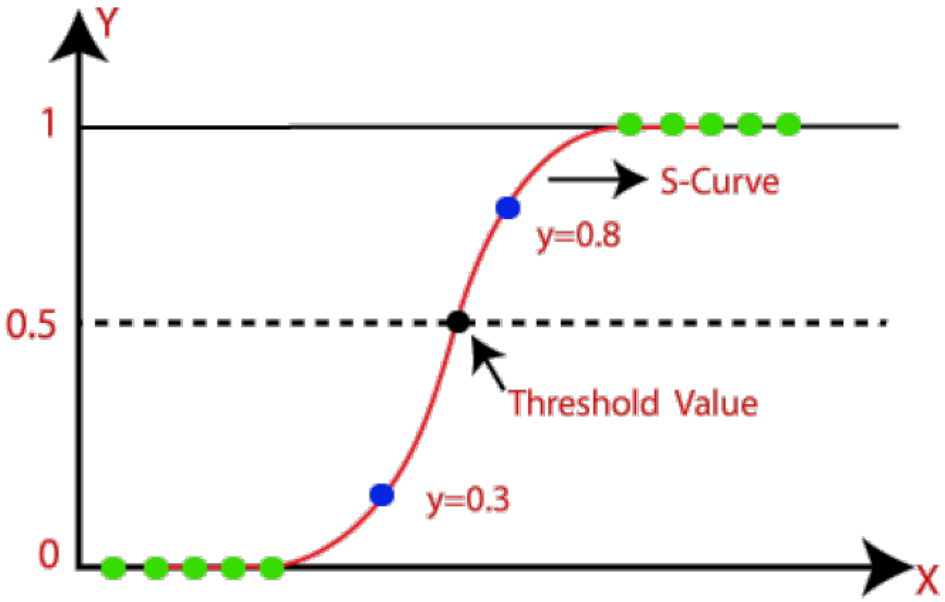


Figure: Logistic Function with threshold value 0.5

The hypothesis of Logistic Regression is:

$h(x) = \frac{e^{\theta_0+\theta_1 x_1+\theta_2 x_2+\cdots+\theta_n x_n}}{e^{\theta_0+\theta_1 x_1+\theta_2 x_2+\cdots+\theta_n x_{n+1}}}$ which is a sigmoid function or logistic function.

We've used LogisticRegression from sklearn.linear_model and found the accuracy score 0.9510763209393346 and F1 score 0.9749247743229689.

Fully connected neural networks:

Full connected neural networks (FCNNs) are one type of artificial neural network architecture where all neurons in one layer are connected to the neurons in the next layer. These networks are computationally expensive and may occur overfitting in some case, we can reduce this overfitting problem by using Dropout.

In our project we have used Sequential from keras.model and Dense from keral.layer. For activation function in the hidden layer, we've used Rectified Linear Unit Function (ReLU). For the output layer, the activation function that we've used is Sigmoid Function. For the backpropagation algorithm, we've used adam and for loss, we've used binary_crossentropy as we have only two classes.

We've used EarlyStopping with patience 200 and ModelCheckpoint which was monitoring validation loss. The epochs value was 2000 and batch size 512.

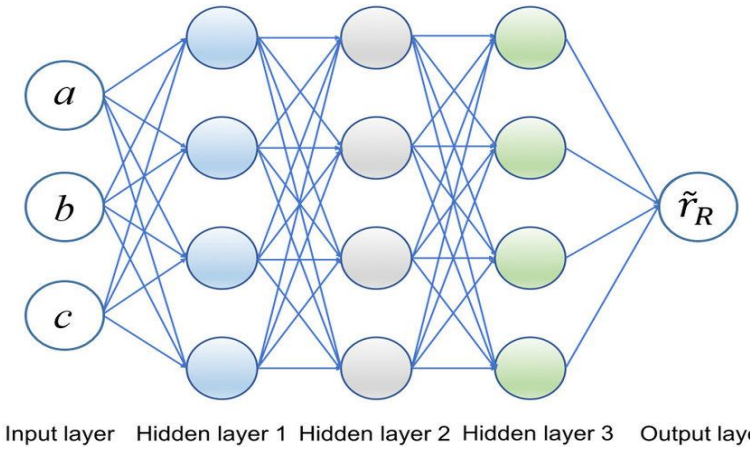


Figure: Example of a Fully Connected Neural Networks

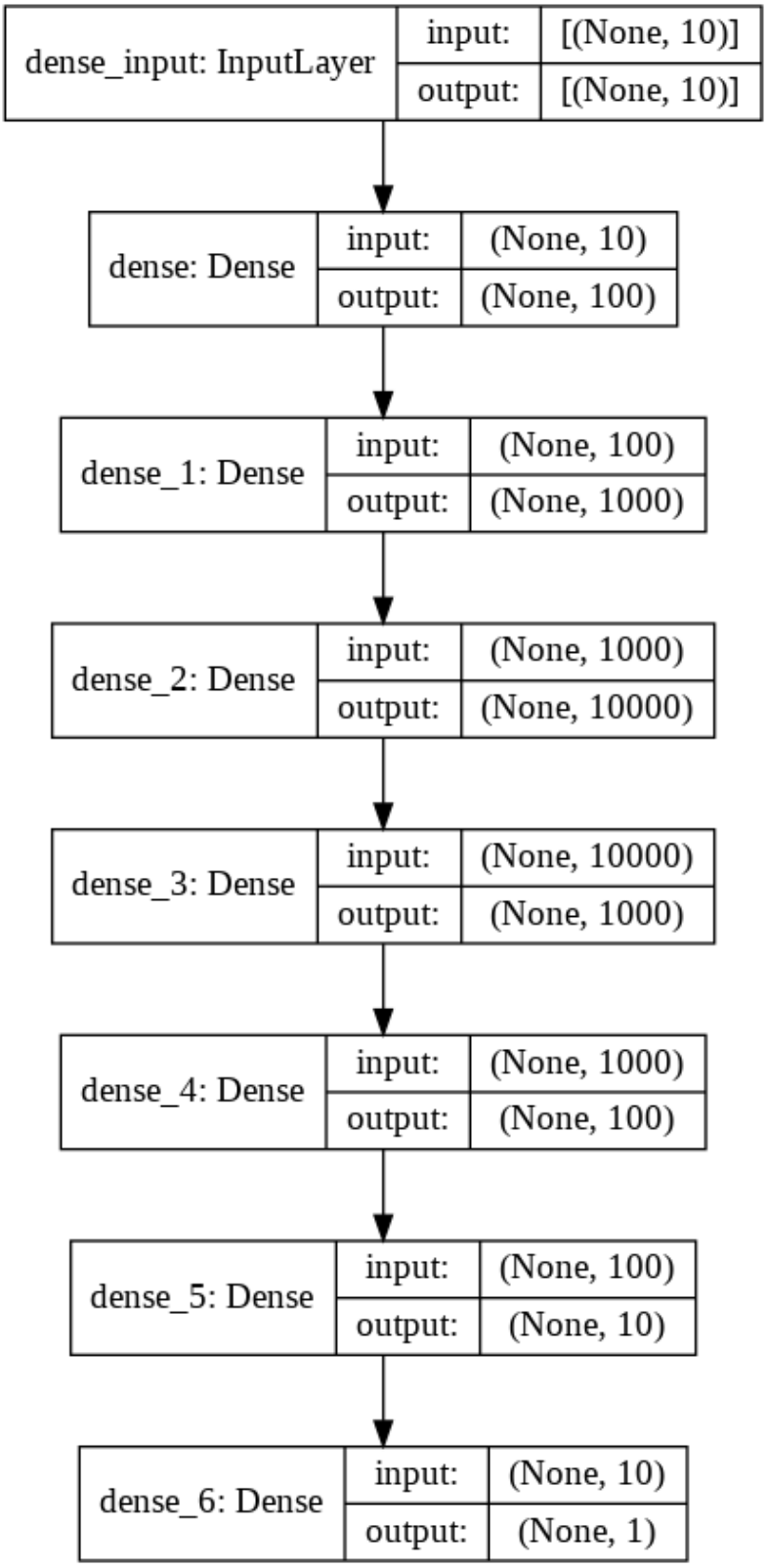


Figure: Fully Connected Neural Networks Architecture of our project

The accuracy we've got by using FCNNs 0.9510763209393346 and F1 score 0.9749247743229689.

Problem Faced:

The dataset we've chosen was not balanced. Class label 0 much higher occurrences than class label 1. As sklearn.metrics does the calculation based on the class label 1 so that we're getting a poor result at the beginning. That's why we've changed the class label 0 to 1 and 1 to 0 by performing XOR by 1 operation.

Result/Comparison Metrics

We've used accuracy score and F1 score to compare between the algorithms. As we know, there's a tradeoff between precision and recall. To get the result from this we need to calculate F1 score.

$$F1\ score = \frac{2 * Precision * Recall}{Precision * Recall}$$

$$Precision = \frac{Total\ True\ Positive}{Total\ Predicted\ Condition\ Positive}$$

$$Recall = \frac{\text{Total True Positive}}{\text{Total Condition Positive}}$$

$$Accuracy = \frac{True\ Positive + True\ Negative}{True\ Positive + True\ Negative + False\ Positive + False\ Negative}$$

Algorithm	Accuracy	F1 Score
SVM	95.11%	97.49%
KNN	95.11%	97.49%
Decision Tree	90.61%	95.04%
Random Forest	95.11%	97.49%
Gaussian NB	87.28%	93.03%
Multinomial NB	81.21%	89.37%
Logistic Regression	95.11%	97.49%
FCNNs	95.11%	97.49%