Comparative Analysis of Two Neural Network Architectures on Dataset for stroke prediction

Siddharth Madan
Department of Computer Science
School of Science and Technology
City, University of London

Abstract

This work analyses the intricacies of two neural network architectures over the stroke occurrences dataset. A Feedforward Multi-Layer Perceptron (MLP) and Support Vector Machines (SVM) are the two algorithms being reviewed. This is a supervised learning problem and these two approaches are the most standard neural network algorithms used for a binary classification prediction task. In a grid search approach, numerous models are tested while having their hyperparameters varied, and they are then validated via stratified cross-validation. Confusion matrices are used to compare the tested outcomes from the best-performing models. In most research problems where the interconnections between features and predicted variables are not that well defined, it has been evident that MLP trumps SVM and as the outcome of our analysis, the MLP algorithm has again emerged to be more effective for such classification problems.

Keywords: Neural Network; Multi-Layer Perceptron; Support Vector Machine; Stroke Prediction.

1 Introduction

A stroke is a life-threatening medical condition that develops when there is a blockage in the blood supply to a part of the brain. A stroke is a serious medical issue that has to be treated immediately. A stroke occurs in the United States every 40 seconds and every 3.5 minutes, a stroke victim passes away [(1)]. The advance in the medical science industry with the help of technology has now enabled us to accurately predict medical conditions based on different parameters and analyse the major symptoms of any health condition. In our study, we try to use the Stroke Prediction Dataset [(2)] to predict whether a stroke may occur or not depending upon the factors like gender, age, marital status, and many more.

In this study, we objectively assess two models intended to forecast stroke prognosis by identifying the pattern of 12 distinct features. A Feedforward Multilayer Perceptron (MLP) and a Support Vector Machine (SVM) are the two neural network techniques considered in this exploratory analysis. To address the issue of stroke prediction, we look into several model configurations and data distributions.

We give a brief overview of the dataset that was used to train and test the models in Section 2. Comparing the methods and approaches employed during the implementation stage is covered in Section 3. In Section 4, the models are critically compared while the implementation outcomes are evaluated. Finally, Section 5 is where we conclude and present prospects for further explorations.

1.1 Multi-Layer Perceptron

A form of neural network called a feedforward multi-layer perceptron (MLP) is frequently employed for classification and regression problems. The MLP is made up of many layers of neurons, each of which is completely interconnected to every other neuron in the layer above it [(3)].

Neurons that represent the input features are found in the input layer, the first layer of the MLP. The output layer, which is the final layer of the MLP, contains neurons that stand in for the model's output. The layers between the input and output layers are referred to as hidden layers, and these levels contain neurons that modify the input data nonlinearly. In order to minimise a loss function, which gauges the discrepancy between the model's anticipated output and the actual output, the MLP learns to modify the weights of the connections between the neurons in each layer over the course of training. By iteratively adjusting the weights to minimise the loss function, an optimisation procedure like stochastic gradient descent is used to update the weights.

Activation functions are used by the neurons in the MLP's hidden layers to add nonlinearity to the model. Sigmoid, hyperbolic tangent, and rectified linear unit (ReLU) functions are examples of typical activation functions. The effectiveness of the model can be significantly impacted by the selection of the activation function. Following the learning of the MLP's weights, predictions may be made on new input data by propagating it through the network and computing the output of the output layer. By examining the neural activation in the hidden layers, the MLP may also be used to extract features from the input data.

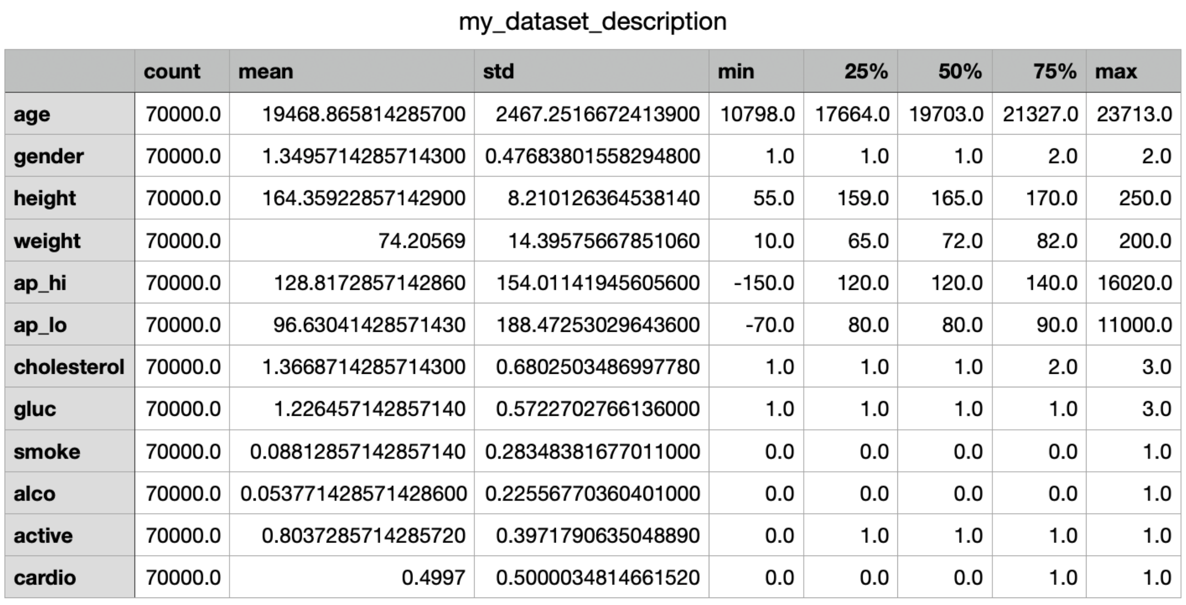
1.2 Support Vector Machine

A machine learning approach known as a Support Vector Machine (SVM) can be utilised for both classification and regression tasks. The optimal hyperplane to divide the input data into distinct classes is found using the SVM. The objective of the SVM is to identify the set of weights and bias term that maximises the margin between the hyperplane and the closest data points for each class. The hyperplane is defined by a set of weights and a bias term [(3)].

The SVM can locate a hyperplane that correctly divides the data into multiple classes when it comes to linearly separable data. In the case of non-linearly separable data, the SVM can use a technique called the kernel trick to map the input data into a higher-dimensional feature space where the data is linearly separable. The performance of the model can be significantly impacted by the optimisation strategy used to train the SVM. Examples of optimisation strategies include quadratic programming and gradient descent. In order to prevent overfitting, the SVM can also be regularised by including a penalty term in the optimisation target. All things considered, the SVM is a potent machine-learning algorithm that can be applied to a variety of classification and regression tasks. The performance of the SVM can be improved by choosing the right hyperparameters and regularisation methods. The main parameters to be considered while application of SVM are the kernel, regularization parameter, and gamma parameter. For SVM to perform well, the kernel, regularisation parameter, and gamma parameter must be set appropriately, which frequently necessitates trial and tweaking.

2 Dataset description

As mentioned above, the "Cardiovascular disease dataset" has been obtained from Kaggle and the dataset consists of 13 columns and 70,000 rows. The dataset is clean and doesn't contain any null values or outliers which can be seen in the box plots. The distribution of the target variable is also normal with 35021 False values and 34979 True values in column cardio. These 13 columns represent characteristics such as Age, gender, height, weight, blood pressure levels (hi-lo), etc. The basic statistics for the dataset can be seen in Figure 1:


my_dataset_description

	count	mean	std	min	25%	50%	75%	max
age	70000.0	19468.865814285700	2467.2516672413900	10798.0	17664.0	19703.0	21327.0	23713.0
gender	70000.0	1.3495714285714300	0.47683801558294800	1.0	1.0	1.0	2.0	2.0
height	70000.0	164.35922857142900	8.210126364538140	55.0	159.0	165.0	170.0	250.0
weight	70000.0	74.20569	14.39575667851060	10.0	65.0	72.0	82.0	200.0
ap_hi	70000.0	128.8172857142860	154.01141945605600	-150.0	120.0	120.0	140.0	16020.0
ap_lo	70000.0	96.63041428571430	188.47253029643600	-70.0	80.0	80.0	90.0	11000.0
cholesterol	70000.0	1.3668714285714300	0.6802503486997780	1.0	1.0	1.0	2.0	3.0
gluc	70000.0	1.226457142857140	0.5722702766136000	1.0	1.0	1.0	1.0	3.0
smoke	70000.0	0.08812857142857140	0.28348381677011000	0.0	0.0	0.0	0.0	1.0
alco	70000.0	0.053771428571428600	0.22556770360401000	0.0	0.0	0.0	0.0	1.0
active	70000.0	0.8037285714285720	0.3971790635048890	0.0	1.0	1.0	1.0	1.0
cardio	70000.0	0.4997	0.5000034814661520	0.0	0.0	0.0	1.0	1.0

Figure 1: Basis statistics of dataset.

2.1 Initial Analysis

In order to begin our first exploration, we plotted the pair plots and correlation plots in order to discern any egregiously apparent correlation between any of the features. It was found that no such evident correlation exists in the dataset.

As a part of the initial analysis of the dataset, we trained and tested a single hidden layer neural network model on a total of 9 combinations of learning rates and the number of neurons. We have taken learning rate values of 0.1, 0.4, 0.8, and the number of neurons in the hidden layer as 2, 20, and 50, pairing each value of the learning rate against every value of the hidden layer neurons number once. We plotted the loss calculated over test predictions and test labels for each combination of learning rate and the number of neurons scaled on a logarithmic scale for better visualization. We have used Binary Cross-Entropy with Logit Loss as our loss function for calculating the loss. As the conclusion of our initial analysis we found that the loss is minimum at a learning rate of 0.1 and 50 neurons in the hidden layer and its maximum at a learning rate of 0.8 and 50 neurons which can be attributed to a high learning rate value. The most optimal solution is at a learning rate of 0.1 and 50 neurons, but the model is performing overall well for lower learning rates. There is a particularly high loss at 20 neurons which get lower when the number of neurons is increased or decreased which suggests overfitting at 20 neurons.

3 Methods

We describe the architecture and hyperparameters used to construct the MLP and SVM models in this section, along with information on how the training, validation, and testing processes were carried out. In order to compare the best MLP and SVM models' algorithms, the methodology holds back 20 percent (i.e. 14000 rows) of the original dataset for testing purposes. 80 percent (i.e. 56000 rows) of the remaining data is utilised for training, validating, and comparing the two algorithms while 20 percent is used for model selection.

The performance of the SVM and MLP models with various hyperparameter combinations is assessed using cross-validation. In particular, the GridSearchCV function is used to do a grid search over a predetermined range of hyperparameters, and cross-validation is used to assess the model's performance. Using the training data, the GridSearchCV function fits both of our models with various combinations of hyperparameters, and then assesses the model's performance on a validation set created by folding the training data into k-folds. The model's generalisation performance is estimated using the validation set. Each of the k folds serves as a validation set exactly once during the k times this operation is carried out. The model's performance is estimated using the mean performance over all k-folds. The incorporation of cross-validation in the code above has the benefit of enabling the model's performance to be assessed using all available data, as opposed to simply a single train-test split. As a result, estimations of the model's performance are more trustworthy, and the hyperparameters are better chosen. Also, the model is tested on many subsets of the data using cross-validation, which can assist prevent overfitting to a certain portion of the data [(4)].

3.1 Architecture and Parameters used for the MLP

The weights of the neurons are updated using a training algorithm called "Bayesian Regulation backpropagation." We have employed two popular activation functions for classification issues: the hyperbolic tan activation function and the rectified linear activation function. The neural network structure must have two outputs in order to apply these nonlinear activation functions, even if it is possible to distinguish between both classes with just one output (with or 0 or 1).

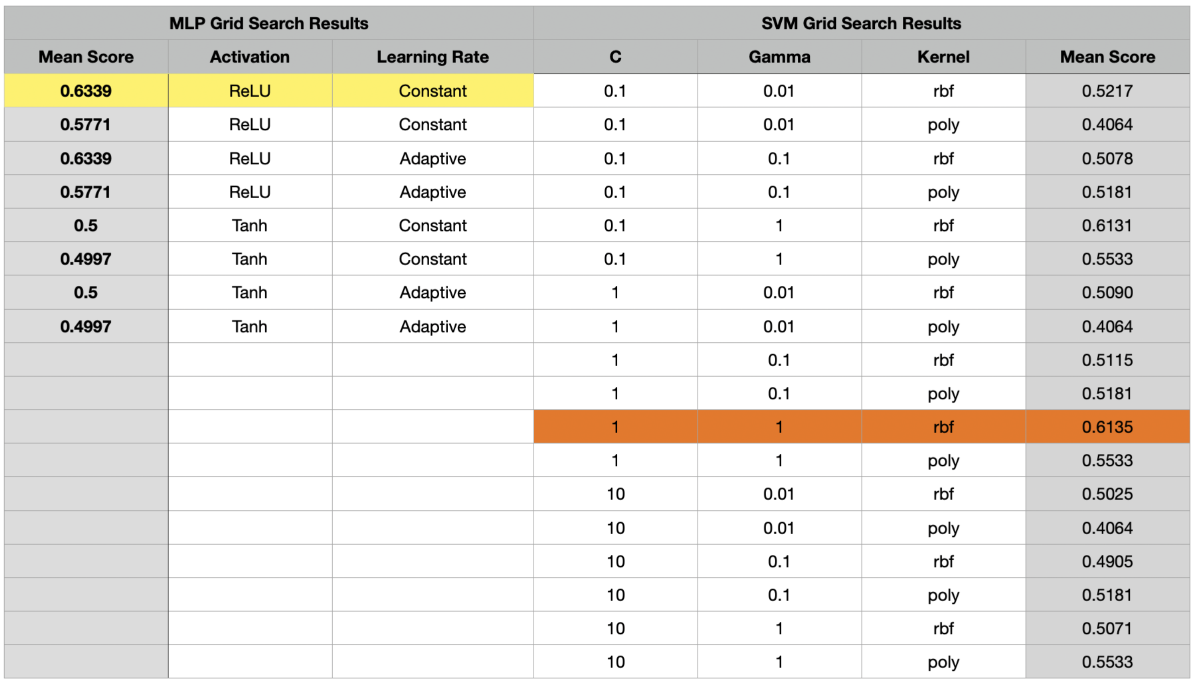
To prevent neural networks from being trapped in the same local minimum each time they are trained, the initial neural network procedure often starts with random weight values. The network will be initialised differently during each training phase due to the random weights in place, resulting in varied ultimate results and accuracies. The accuracy will also be impacted by the samples used for training, verifying, and testing. By performing a k-fold cross-validation for the validating stage but not the testing one, this impact may be mitigated. The three hyperparameters to be examined are learning rate, random state, and activation function. By setting random state to a fixed value, in this case 42, the same initial weights will be used each time the model is trained, resulting in a more reproducible model. If random state is not set, the weights will be initialized randomly each time the model is trained, potentially resulting in different performance on the same data. For both the training and the testing stages, we used 5 validation checks.

3.2 Architecture and Parameters used for the SVM

In a high-dimensional space, the SVM method generates a hyperplane or set of hyperplanes that can be utilised for classification. It can only divide data into two classes because it is a binary classifier. By maximising the margin between the two classes, the SVM algorithm determines the best hyperplane. The margin is the separation between the nearest data points from each class and the hyperplane. Finding the hyperplane that optimises the margin is indeed the objective of the SVM algorithm. Several kernel functions, including linear, polynomial, and radial basis function (RBF) kernels, can be employed with the SVM algorithm. The input data is transformed by the kernel functions into a higher-dimensional space, making it simpler to locate a separating hyperplane. Since SVMs do not start with random weights, they do not have the issue of tumbling into random local minima. SVM is still needed to handle the problem of training with various data sets, which enables a more accurate validation of the model. We use 5-fold cross validation for the testing stage in SVM in our final algorithm. The results from 5-fold CV were slight better from 2-fold and 10-fold CV and hence we present the results from 5-fold in the next section.

4 Results and discussion

The best models chosen for MLP and SVM are shown in Table 2 along with the hyperparameters grid search, which are highlighted in yellow and orange, respectively. Each best model's prediction was compared to the cross-validation process's class label targets in order to assess how broadly applicable it was. Rerunning the same grid search considerably changes the accuracy results for each MLP model, however this does not happen for SVMs, which indicates that MLP are strongly influenced by the weights of the initial neurons. SVMs, on the other hand, are more sensitive once hyperparameters are integrated. Depending on it, accuracy can range from 65 to 40 percent, but for MLP models, this doesn't seem to be a significant factor. To make the combination of hyperparameters that contributed to increasing or reducing the averaged validation accuracy easier to see, the accuracy columns are highlighted in grey.



MLP Grid Search Results			SVM Grid Search Results			
Mean Score	**Activation**	**Learning Rate**	**C**	**Gamma**	**Kernel**	**Mean Score**
0.6339	ReLU	Constant	0.1	0.01	rbf	0.5217
0.5771	ReLU	Constant	0.1	0.01	poly	0.4064
0.6339	ReLU	Adaptive	0.1	0.1	rbf	0.5078
0.5771	ReLU	Adaptive	0.1	0.1	poly	0.5181
0.5	Tanh	Constant	0.1	1	rbf	0.6131
0.4997	Tanh	Constant	0.1	1	poly	0.5533
0.5	Tanh	Adaptive	1	0.01	rbf	0.5090
0.4997	Tanh	Adaptive	1	0.01	poly	0.4064
			1	0.1	rbf	0.5115
			1	0.1	poly	0.5181
			1	1	rbf	0.6135
			1	1	poly	0.5533
			10	0.01	rbf	0.5025
			10	0.01	poly	0.4064
			10	0.1	rbf	0.4905
			10	0.1	poly	0.5181
			10	1	rbf	0.5071
			10	1	poly	0.5533

Figure 2: Grid Search for both algorithms

4.1 Result Comparison

The results of the testing phase comparing the top MLP and SVM models in a confusion matrix arrangement are shown in Fig. 3 and Fig. 4. We found that the best MLP model could reach 63.39 percent accuracy and the best SVM model 62.35 percent during the model selection procedure.

Also, we can refer to the grid search table to discuss the hyperparameters and their impact on the mean score. In MLP, we iterated ReLU and TanH activation fuctions. We have preferred considering TanH activation over softmax in our final testing as it showed better performance. ReLU activation clearly shows better results than TanH and the improvement in accuracy in ReLU from TanH is about 13 percent. ReLU coupled with the learning rate to be constant in every epoch and even with the adaptive learning rate gives the best test accuracy of 63.39 percent each. This also shows us the redundancy of the learning rate being constant or adaptive in this dataset and hence we have highlighted the constant learning rate and also considered it as one of the best parameters along with ReLU for MLP.

In the SVM grid search table we can identify three columns representing three hyperparameters first: C - the regularization parameter which is iterated over three values 0.1, 1, and 10; second: Gamma which is iterated over 0.001, 0.1, and 1; and third: Kernel which is iterated over two different types of kernels i.e. rbf (radial basis function) and polynomial (poly) kernel. A brief description about both the kernels is provided in the Glossary. As highlighted in orange we can see the best combination of parameter resulting in highest accuracy for SVM i.e. 61.35 percent. The best parameter values in SVM came out to be C: 1, Gamma: 1, and rbf kernel.

In the confusion matrix for MLP we can see that we have true negative to be 6519 as opposed to actually higher as false negative are 4710. And false positives are 469 where actually true positives are 2302. Similarly we can anlayse the confusion matrix for SVM in which false positives are on the higher side with 4371 counts as opposed to actually true positives being 5331.

In the cardiovascular stroke prognosis, it is more crucial to label someone who does not currently have a stroke risk as having one than vice versa. If the diagnostics check reveals that a person is not sick, they will certainly be in a precarious situation or perhaps even face death. It is therefore preferable to have a True Negative Rate i.e. chance of identifying a sample as not prone to a stroke equals 100. Nonetheless, some False Negatives must occur because our classifiers are not flawless. So, the objective is to minimise False Positives by setting the classifier threshold to classify all True Negatives. As a result, MLP is a slightly better algorithm to utilise for this kind of problem and the dataset under consideration.

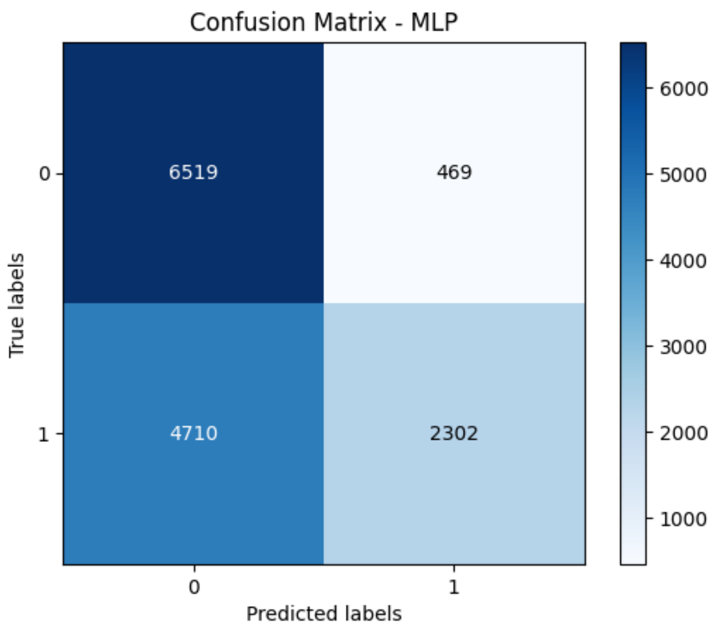


Figure 3: Confusion matrix for MLP

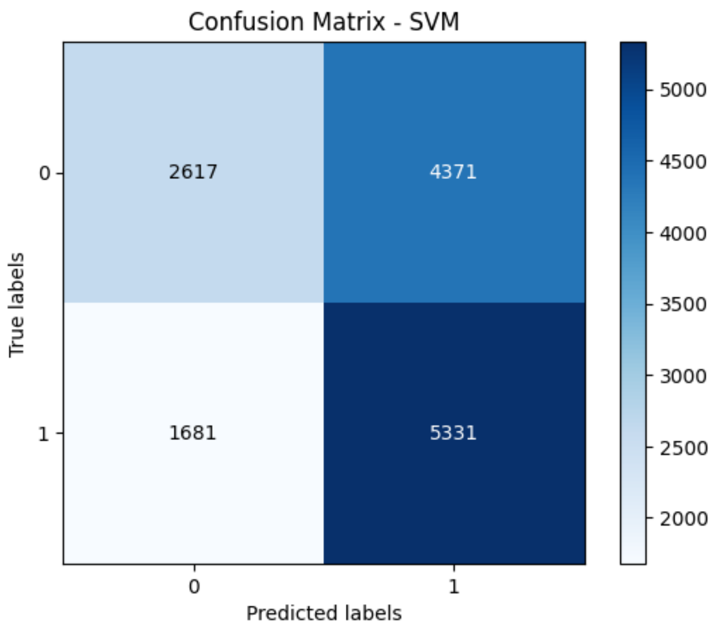


Figure 4: Confusion matrix for SVM

5 Conclusion and Further Explorations

Our study investigated the specificity with which two trained models—a Multilayer Perceptron and a Support Vector Machine—can predict a person's propensity for having a stroke based on variables like age, weight, blood pressure, glucose levels, and so on. This is widely dynamic research field which encompasses human health longevity in the light of cutting-edge technological and computational forefront. In the advent of smart watches and nano-chips acting as monitoring probes this research area holds great value in current time.

The conclusion is that MLP and SVM are comparable, but it is important to understand that MLPs have considerable accuracies variations because of their dependence on the neuron's weights. In the training, validation, and testing phases, both models produced results that were comparable, with better accuracy in the training than in the testing phase, as would be anticipating. Though the MLP minimises false positives while maintaining a 100 percent true negatives rate, it proved to be a superior solution for this particular problem.

To better understand what features are being extracted from the data, we think it would be highly interesting to evaluate the performance of both the MLP and SVM algorithms using other training strategies, such as bagging [(5)]. Researchers in [(6)], consider principal component analysis to be a significant step in order to identify important features and eventually get rid of redundant dimensions in the data. The objective is to attempt to minimise this effect by reducing the dependence of data allocation and random initial neuron weights, which was not successfully achieved by the employed cross-validation in this work.

References

1. Tsao CW, Aday AW, Almarzooq ZI, Alonso A, Beaton AZ, Bittencourt MS, et al. Heart Disease and Stroke Statistics—2022 Update: A Report From the American Heart Association. Circulation. 2022;145(8): e153–e639.
2. "Cardiovascular Dataset" : kaggle.com
3. "Artificial Intelligence for Humans, Vol 3: Neural Networks and Deep Learning": heatonreseach.com
4. Arlot, S., Celisse, A. (2010). A survey of cross-validation procedures for model selection. Statistics surveys, 4, 40-79. https://doi.org/10.1214/09-SS054
5. Pasha Syed, A.R., Anbalagan, R., Setlur, A.S. et al. Implementation of ensemble machine learning algorithms on exome datasets for predicting early diagnosis of cancers. BMC Bioinformatics 23, 496 (2022). https://doi.org/10.1186/s12859-022-05050-w
6. Soumyabrata Dev, Hewei Wang, Chidozie Shamrock Nwosu, Nishtha Jain, Bharadwaj Veeravalli, Deepu John, "A predictive analytics approach for stroke prediction using machine learning and neural networks, Healthcare Analytics" https://www.sciencedirect.com/science/article/pii/S2772442522000090

6 Glossary

Here, we elaborate on some of the key underlying concepts not featured in our report.

6.1 Neural Networks

A form of machine learning technique known as an artificial neural network (ANN) is based on the structure and operation of the human brain. Artificial neurons, which are layers of interconnected nodes in ANNs, accept input signals, carry out calculations, and output signals [(3)]
Data from the outside world is first received by the input layer of an ANN, after which it is transferred through one or more hidden layers, where the computation is done. A weighted total of the inputs is computed by each neuron in the hidden layer after it receives inputs from several neurons in the preceding layer. The activation function, which adds nonlinearity to the model and aids the network in learning intricate correlations between inputs and outputs, is then applied to the weighted sum. The output layer receives the output of the hidden layer and uses it to create the network's final output. Depending on the kind of problem being solved, the output layer may have one or many neurons. One neuron in the output layer, for instance, will provide a probability between 0 and 1, indicating the chance that the input belongs to a certain class in a binary classification problem. One neuron that generates a continuous output value will be present in the output layer of a regression problem.
In order to reduce the discrepancy between the predicted output and the actual output, the network modifies the weights of the connections between the neurons during training.

6.2 Activation function in MLP

Multilayer Perceptrons (MLPs) and other neural network architectures must have activation functions. The main benefit of activation functions is to add non-linearity to a neural network's output, which enables it to recognise intricate patterns in the input data. An activation function creates an output by applying a non-linear transformation on the weighted sum of the inputs and a bias term. The neural network's next layer receives the output of the activation function after that [(3)].
A neural network would be nothing more than a linear regression model without activation functions, unable to understand complicated relationships in the data. The network would produce a linear combination of the input variables, for instance, if we were to apply a linear activation function, which would prevent it from simulating non-linear interactions. The activation functions sigmoid, ReLU, tanh, and softmax are some of the ones utilised in MLPs; each has special characteristics and applications. The problem being solved and the properties of the data determine the activation function to be used.

6.2.1 Rectified Linear Unit activation function

A typical activation function in MLPs (Multi-Layer Perceptrons) for neural networks is ReLU (Rectified Linear Unit). Any input value that is less than zero is mapped to zero by the ReLU function, and any value that is equal to or greater than zero is mapped to itself. The ReLU function can be defined mathematically as follows:

$$f(x) = max(0, x)$$

where x is the input value to the neuron and f(x) is the output [(3)]. ReLU has grown to become an increasingly popular activation function for MLPs due to its many benefits. Firstly, because ReLU calculation just requires simple element-wise operations, it is computationally efficient. Second, ReLU aids in preventing the vanishing gradient problem, which happens when gradients shrink dramatically during backpropagation. Deep neural network training may be challenging as a result of this issue. Last but not least, ReLU permits network sparsity since some neurons will be deactivated (i.e., output zero) for specific inputs. This may lessen the model's overfitting. ReLU does have some restrictions, though. One significant drawback is that it may produce ”dead neurons” that output 0 for all inputs, which can result in data loss and diminish the network's capacity [(3)].

6.2.2 Hyperbolic Tangent activation function

An additional frequently used activation function in MLPs is tanh (Hyperbolic Tangent). Due to its symmetry, it produces values between -1 and 1, as opposed to the ReLU function, which produces values between 0 and infinity [(3)].
Following is a definition of the tanh function:

$$tanh(x) = \frac{(e^x - e^{-x})}{(e^x + e^{-x})}$$

Although it is zero-centered and has a greater gradient around the origin, it has a form that is comparable to the sigmoid function. As a result, it works better when applied to MLPs that are intended to deal with data that has been standardised or normalised to have a zero mean and unit variance. The vanishing gradient issue that might arise during backpropagation is one benefit of employing the tanh function over the sigmoid function. This is due to the fact that, particularly at the origin, the derivative of tanh is larger than the derivative of the sigmoid function. While it can help to prevent the issue of ”stuck” neurons that can arise with the sigmoid function when inputs are far from the decision boundary, the tanh function can be a suitable option for MLPs that are used for classification problems, especially those that have numerous classes [(3)].

6.3 Kernel function in SVM

The kernel is a key element of Support Vector Machines (SVM), which turns the input data into a higher dimensional space to facilitate classification. A supervised learning method known as SVM looks for the optimum hyperplane (also known as a decision boundary) to divide data into two or more groups. The data may occasionally not be linearly separable in the original input space, though. In these circumstances, the data is transformed into a higher dimensional space using the kernel method, where it is more likely to be linearly separable [(3)].

6.4 Regularization parameter (C) in SVM

The C parameter is a hyperparameter in Support Vector Machines (SVM) that regulates the trade-off between increasing the margin of the decision boundary and reducing the classification error. It is a regularisation parameter that, by including a penalty term in the optimisation objective, aids in preventing overfitting.
The cost of misclassifying training examples is controlled by the C parameter. Whereas a bigger C value will provide a narrower margin and fewer misclassifications, a lesser C value will produce a broader margin and more misclassifications. When C is very big, overfitting may occur because the optimisation target prioritises correctly classifying all training data. In other words, the C parameter establishes the trade-off between enforcing stricter classification accuracy at the expense of a narrower margin that may be more susceptible to overfitting and allowing some misclassifications in the training set to obtain a wider margin and potentially more generalizable decision boundary [(3)].

6.5 Gamma parameter in SVM

The gamma parameter in Support Vector Machines (SVM) is a hyperparameter that establishes the weight of a single training example. It regulates the flexibility of the model as well as the decision boundary's shape. A single training example's influence is determined by the gamma parameter, with low values denoting ”far” and large values denoting ”near.” A decision boundary that is broader and less prone to overfit the training set is produced by a low gamma value. A high gamma value, on the other hand, leads to a tighter decision boundary that is more likely to overfit the training set of data.
When using kernel functions, such as the radial basis function (RBF) kernel, which is frequently used in SVM, the gamma parameter is essential. The RBF kernel calculates a weighted total of the distances between all training examples and compares the similarity between two examples in a feature space.
A decision border with a high gamma value is more complicated and irregular and can fit closely to the training data since only neighbouring training examples have a significant influence on it. On the other hand, a low gamma value indicates that the influence of training examples is more widespread, leading to a smoother and easier to understand decision boundary that may be less prone to overfitting [(3)].