Low Birth weight classifier

Sharanya Venkat
PES1201700218
sharan.venkat@gmail.com

Anagha Ananth
PES1201700301
anagha.anan99@gmail.com

Aishwarya Ramanath
PES1201700872
aishwarya.ramanath@gmail.com

***Abstract*—Low Birth weight (LBW) acts as an indicator of sickness in newborn babies, and is closely associated with infant mortality. This project presents two machine learning techniques - K-Nearest Neighbours(KNN) and Artificial Neural Networks (ANN) with back propagation. It discusses the disadvantages of KNN and overcomes them via ANN by implementing transfer learning and adjusting the learning rate in each epoch.**

I. Problem Statement

Low Birth weight (LBW) acts as an indicator of sickness in newborn babies. LBW is closely associated with infant mortality as well as various health outcomes later in life. Various studies show strong correlation between maternal health during pregnancy and the child's birth weight.

We use health indicators of pregnant women such as age, height, weight, community etc:- in order to for early detection of potential LBW cases. This detection is treated as a classification problem between LBW and not-LBW classes.

II. Data Pre-processing

Boxplots, histograms were plotted for each of the features and outliers/noise was removed. Missing values are handled as follows:

- age, BP and HB is replaced with their medians
- History, res is replaced with mode
- weight1 is replaced with mean

Modules used:

- seaborn
- matplotlib

III. K Nearest Neighbours classifier

Our first approach was to use KNN, given that our dataset is small(102 rows). Furthermore, it is labelled and outliers had also been dropped. Euclidean distance was used as a distance measure

A. Splitting into test and train

A 90-10 train-test data split model is used to maximise the number of train instances. Usage of the word 'train' is unfounded. It implies that a random 90% of the dataset is correctly labelled. The test data will be classified wrt this 'train' set.

B. Initializing number of neighbours

Various values of k from 2-15 were taken and corresponding accuracy was checked. k = 5 yielded highest accuracy and was hence chosen.

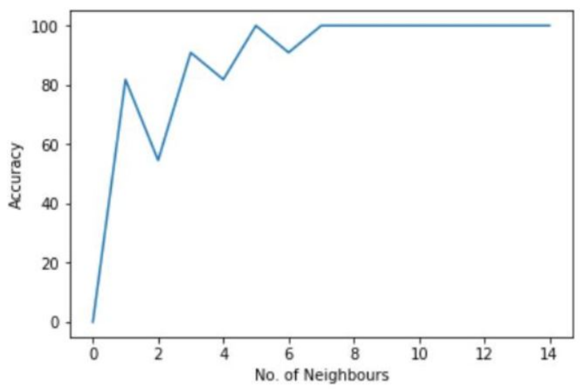

Fig. 1. Optimal 'k' Value

C. Testing phase

KNN has no training or validation phase since there is no model to build.

- **Finding nearest Neighbours**: Consider a test/query instance - $test_1$ = (3, 21, 50, 1, 9, 1, 1.375, 5, 1, 1) and train instance - $train_1$ = (1, 26, 37, 1, 5.9, 1, 1.44, 5, 1, 0) and x_i refers to Euclidean Distance given by:

$$d(\mathbf{test_1}, \mathbf{train_1}) = \sqrt{\sum_{i=1}^{9}(x_i - y_i)^2}$$

  where 9 is the total number of features in the dataset(last element is ignored since it is the output class) ,
  x_i refers to each value in $test_1$
  y_i refers to each value in $train_1$
  The distance of each query instance from all training instances is stored in a dictionary 'distances', along with the corresponding training instance.
- **Classifying the query instance** The dictionary 'distances' is sorted by key (= Euclidian distance) in ascending order. The top k (in our case, 5) distances and the corresponding training instances are picked.
  Let [$train_3$:1, $train_7$:0, $train_8$:1, $train_{92}$:1, $train_{36}$:0] be the 5 closest neighbours.
  The output class of each of these neighbours is noted and the following function is applied to obtain the output class of $test_1$:

$$\underset{i \epsilon neighbours}{argmax} \text{ (Output Class(i))}$$

where i refers to each neighbour.

Here the output will be max(1,0,1,1,0) = 1. Hence the training instance will be classified as LBW infant.

D. *Results and Inferences*

- Accuracy obtained = 90.9%
- Accuracy = Recall = 90.9%
- Confusion matrix (for 11 test instances):

	True	False
Positive	10	1
Negative	0	0

Lack of false negatives is a good sign, especially since this is a medical field of study. However, lack of true negatives is concerning.

IV. Drawbacks of KNN

- The model never undergoes a training phase, hence with different test-train splits the accuracy is highly inconsistent
- Only a single hyper parameter is used, i.e k

V. ANN with Back Propagation

In order to overcome the difficulties encountered with KNN, we created an ANN. Its results are more thorough because

- It uses back propagation to further improve the output
- It uses a form of **transfer learning** which means that once this neural network is trained on one task, its parameters can be used as a good initializer for another (similar) task.
- Training data can be discarded once the model has been built and trained.

A. *Initializing Hyper Parameters*

Following are the initialized hyper parameters for this model

- 9 Input nodes(each corresponding to a feature)
- 10 hidden layer node
- 1 output node
- Initial learning rate = 0.1
- Number of epochs for training = 20

This ANN uses 9 input nodes, s and . The hidden layer uses a Relu Activation function and output layer uses Sigmoid.

B. *Splitting Data into test and train*

A 90-10 train-test data split model is used to maximise the number of train instances. More the amount of training data, better will be the accuracy of the model

C. *Activation Functions*

- ReLU - Used from input to hidden layer. This avoids vanishing gradient.(Refer Fig. 2).
- Sigmoid - Used from hidden to output.
  Refer Fig. 3

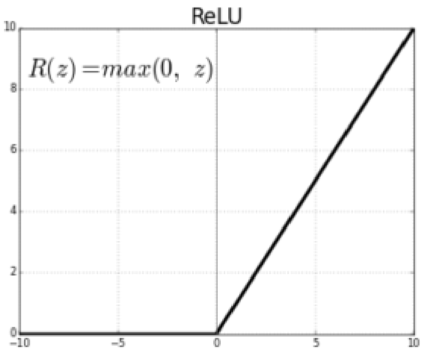

Fig. 2. ReLU

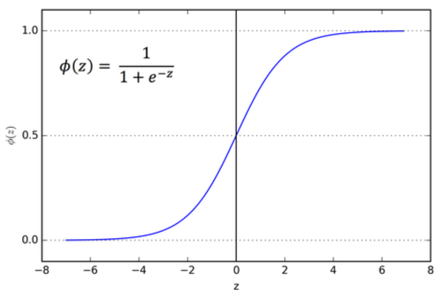

Fig. 3. Sigmoid

D. *Training Phase*

Consider a training instance- $train_{X1}$ = (1, 26, 37, 1, 5.9, 1, 1.44, 5, 1). This vector does not contain the output class.

The figure below summarises the entire ANN Architecture:

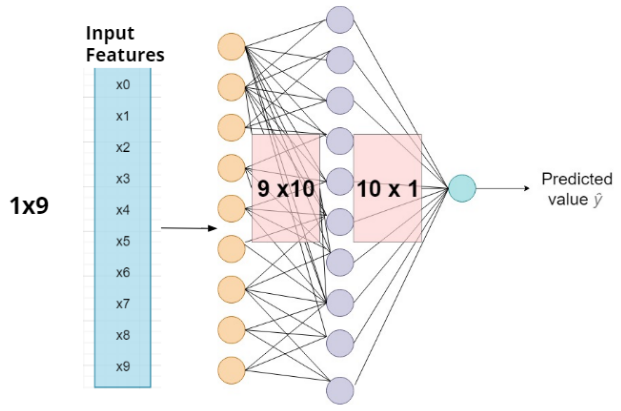

Fig. 4. ANN Architecture

1. **Forward Propagation**

- Activation from input to hidden uses ReLU. A weight matrix (in this case order = 9x10) is initialized.
- Activation from Hidden to Output uses Sigmoid. The output from Sigmoid gives us a predicted output class $\hat{Y}$

2. **Error Calculation** - Mean Square error E is calculated as:

$$\frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y_i})^2$$

$*\, n$ is the number of data points
$*\, Y_i$ represents observed values
$*\, \hat{Y_i}$ represents predicted values

3. **Back Propagation** - Here, our goal is to minimize the total error or cost. We do so by using chain rule to calculate the derivative of cost with respect to any weight in the network. Here are the equations we need to make a prediction and calculate total error, or cost:

Function	**Formula**	**Derivative**
Weighted input	$Z = XW$	$Z'(W) = X$
Sigmoid Activation	$S(Z) = \frac{1}{e^-z}$	$S'(Z)$=S(Z).(1-S(Z))
ReLU Activation	$R = max(0, Z)$	$R'(Z)$= $\begin{cases} 0 & \text{if } Z < 0 \\ 1 & \text{if } Z > 0 \end{cases}$
Total Error	$E = \frac{1}{n}.(\hat{y} - y)^2$	$E'(\hat{y}) = (\hat{y} - y)$

- For gradient descent from hidden to output layer the following formula is used:

$$\begin{aligned} E'(W_o) &= E'(S) \cdot S'(Z) \cdot Z'(W_o) \\ &= (\hat{y} - y) \cdot S'(Z) \cdot X \end{aligned}$$

  where W_o represents weights from hidden to output layers

- For gradient descent from output to input layer the following formula is used:

$$\begin{aligned} E'(W_h) &= E'(\hat{y}) \cdot O'(Z_o) \cdot Z'_o(H) \cdot H'(Z_h) \cdot Z'_h(W_h) \\ &= (\hat{y} - y) \cdot S'(Z_o) \cdot W_o \cdot R'(Z_h) \cdot X \end{aligned}$$

  where W_h represents weights from input to hidden layers

4. **Weight Updation** : The weights of the matrix are updated according to the formulae:

$$W_o = W_o + \alpha.E'(W_o)$$
$$W_h = W_h + \alpha.E'(W_h)$$

where α represents the learning rate

5. **Learning Rate Updation** - We update our learning rate in order to reach the local minima of the total error/cost function. The learning rate updation is given by the below function:

$$\alpha = \alpha * (1./(1. + decay * iterations))$$

Analysis of learning rate is done in the Fig. 5 and Fig. 6

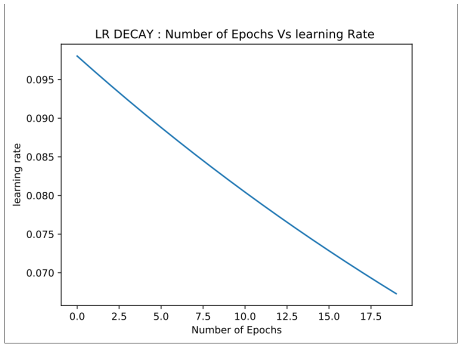

Fig. 5. Linear Reduction of learning rate as epochs increase

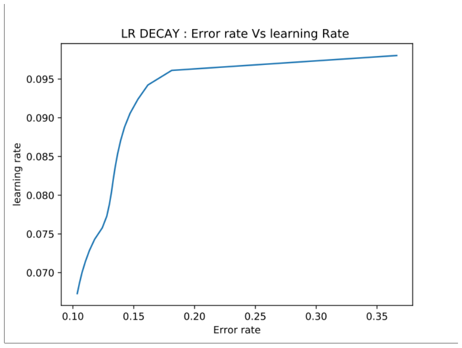

Fig. 6. As error increases, learning rate also increases

E. Testing Phase

A test instance is passed through the Forward Propagation channel only. The obtained output is the classification of the test instance.

F. Results and Inferences

- Due to implementation of adaptive learning we observe that Error Rate reduces with number of epochs

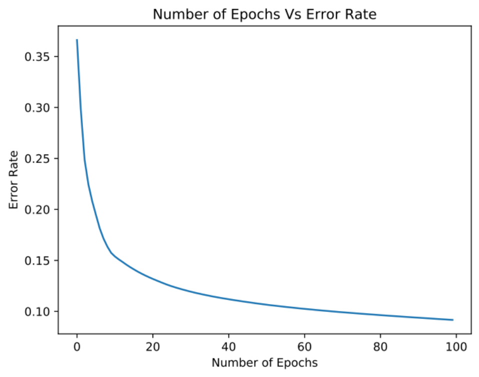

Fig. 7. Error Rate vs Epochs

- Accuracy = 92.3%
- Precision = 85.71%
- Recall = 100 %
- Confusion matrix (for 11 test instances):

	True	False
Positive	6	1
Negative	4	0

VI. Summary

Comparison of the two algorithms are as follows:

K-Nearest Neighbours	**Artificial Neural Network**
Requires No training Time	Training Time is intensive
Simple model with single hyperparameter - k	Model has many hyper parameters controlling size and structure of the network and optimization procedure
F1 Score = 95.23%	F1 Score = 92.3%

Once a neural network is trained on one task, its parameters can be used as a good initializer for another (similar) task. This is a form of transfer learning that cannot be achieved with k-NN. Moreover in a neural network, the training data is no longer needed to produce new predictions. This is obviously not the case with k-NN.
In conclusion - while k-NN seems theoretically correct, ANN yielded more realistic, desirable classifications.

VII. Acknowledgements

We would like to extend our gratitude to our teacher, Prof. K S Srinivas for guiding us through the project and helping us learn the necessary concepts. We would also like to thank our peers for their support and encouragement.