PROBLEM STATEMENT

Many classification problems have asymmetric costs associated with errors of different kinds. Explore how misclassification costs can be taken into account in training a backprop network, and apply on the UCI human activity recognition dataset (based on smart phone sensor measurements)
http://archive.ics.uci.edu/ml/datasets/Human+Activity+Recognition+Using+Smartphones
Set up a reasonable cost matrix, e.g., the cost of misclassifying "walking" as "sitting" is higher than the cost of misclassifying "walking" as "walking upstairs".

ANALYSIS

1. DATASET

Using three data sets from the UCI Machine Learning Dataset Repository for this experiment and below are the links and brief description of the Datasets.
Set1: http://archive.ics.uci.edu/ml/datasets/Cardiotocography
Set2: http://archive.ics.uci.edu/ml/datasets/Page+Blocks+Classification
Set3: http://archive.ics.uci.edu/ml/datasets/Human+Activity+Recognition+Using+Smartphones

	Number of instances	**Number of attributes**	**Number of Classes**
Set1 (Cardio)	2126	23	3
Set2 (Page Block)	5473	10	5
Set3 (HAR)	10299	561	6

2. CODE

Misclassifications are visible completely at the output nodes because output nodes are where the final values are compared with the expected values. The neurons which are connected to these nodes are the ones who should bear the major amount of the cost, hence the errors propagated back to these nodes are carry the major chunk of the misclassification costs which are further back propagated to internal hidden nodes.

Tried modifying the algorithm by only introducing cost of one order as a product to the error as below. y_j is desired output and o_j is actual output and K the misclassification cost matrix.

$$\delta = (y_j - o_j) * o_j *(1-o_j) * K[c,j]$$

However, the improvements were not found to be significant mostly due to low learning rate which is why introduced cost of order two (i.e. square of the misclassification cost) as stated in equation 16 of [6]. Due to high order of cost the errors propagated had significant impact hence the final accuracies are found to improve. Code: backprop_costmatrix.py:139

$$\delta = (y_j - o_j) * o_j *(1-o_j) * K^2[c,j]$$

From the links [2,3] mentioned in the references the code at [1] has been developed. This code includes scripts for preprocessing and implementation of back propagation with the misclassification cost matrices as defined in next section.

3. EXPERIMENT AND RESULTS

Defined and Used below cost matrices for each of the datasets. For all datasets and for all classes if "i" represents actual class and "j" represents predicted class then the cost of misclassification is retrieved by using Cost[i][j] from the cost matrix. The data is divided 50% train and 50% test. We use two folds method where we divide the dataset randomly twice, train and test the network for each of the fold. Since folds=2 and data is divided into half for training and half for testing the final number of data points we have both for testing and training because of two folding is equal to the total number of original data points.

Set 1 being Cardiotocography has three classes – Normal, Suspect, Pathologic. Misclassifying a normal case as suspect and pathologic can lead to more medical tests where the cost of tests is a financial cost only hence the cost of such misclassifications are assumes to be low (1-2). However, classifying suspect and pathologic cases as normal would cost the most as the cases get ignored and the patient doesn't receive any medical treatment and continues to suffer health wise. Hence, cost (health detoriation+financial costs+reputation+.. etc.,) of medical institute is generally high (4-8). Similar intuition has been draw while deciding the cost for other two datasets (page-blocks and human activity recognition).

Set1	Normal	Suspect	Pathologic
Normal	0	1	2
Suspect	4	0	4
Pathologic	8	6	0

Misclassification Cost Matrix: Cardiotocography(set1) data set

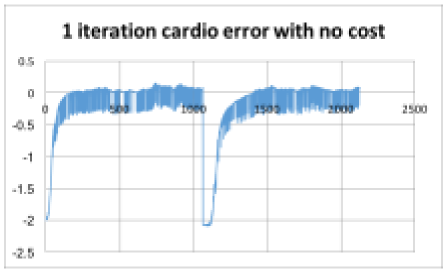
Set2	Text	Horizontal line	Picture	Vertical line	Graphic
Text	0	2	5	6	18
Horizontal line	1	0	3	2	6
Picture	1	3	0	1	4
Vertical line	1	4	2	0	3
Graphic	1	1	2	2	0

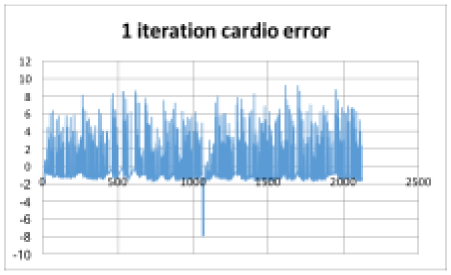
Misclassification Cost Matrix: Page blocks classification(set2) data set

Set2	Walking	Walk Upstairs	Walk Downstairs	Sitting	Standing	Laying
Walking	0	1	1	3	2	3
Walk Upstairs	1	0	2	2	3	4
Walk Downstairs	1	2	0	2	3	4
Sitting	2	3	3	0	1	1
Standing	1	2	2	1	0	2
Laying	4	5	5	1	2	0

Misclassification Cost Matrix: Human activity recognition (set3) data set

Since the costs are asymmetric and does not start costing equally with 1 as mentioned in [4] we are not considering the probability of the classes from the trained data points to avoid over overfitting of the data on the network model. Due to Computational limitations the number of hidden layers was limited to just one and the number of neurons or nodes on the hidden layer were fixed to n=5 for Dataset 1, n=3 for Dataset 2 and n=10 for Dataset 3 and the following observations were made.





The introduction of the cost matrix for misclassification helped in improving the error values as seen in above plots (right image with cost). Similar changes in error has been observed for other datasets resulting in improving of the accuracies (though only for selected features in case of human activity recognition dataset).

Iterations	Set 1 (Cardio) – 5 neurons	With cost matrix?	Misclassifications out of 2126 (folds=2, 50%train)	Overall final cost
1	77.846 %	Yes	471	33920
10	87.159 %	Yes	273	347558
100	**88.899 %**	**Yes**	**236**	**3412752**
1	77.846 %	No	471	NA
10	79.821 %	No	429	NA
100	88.335 %	No	248	NA

We tested the model after training them for 1,10 and 100 iterations on datasets 1(cardio) and 2(page) and found below results. For Dataset 1 and 2 the accuracies improved with increase in iterations and also when a cost matrix was used indicating the improvement in the network model.

Iterations	Set 2 (pages) – 3 neurons	With cost matrix?	Misclassifications out of 5473 (folds=2, 50%train)	Overall final cost
1	89.784 %	Yes	559	5814
10	89.784 %	Yes	559	55062
100	**93.183 %**	**Yes**	**373**	**3279981**
1	89.784 %	No	559	NA
10	89.784 %	No	559	NA
100	92.873 %	No	390	NA

However, for the dataset 3 (Human activity recognition) the accuracies didn't affect by using cost matrix. This is mainly due to usage of all 561 features to train the model. Hence, we selected few features as suggested in paper [5] where the perceptron neural network gave nearly 40% accuracies where as our backpropagation with cost matrix gave improved results as shown in below table. The misclassification counts decrease considerably with the introduction of cost matrix as shown in the fourth column below. The Overall final cost is the quantity in terms of cost (actually square of the cost for each misclassification) which is propagated as a product to the error (hence increasing error).

Iterations	Set 3 (HAR) – 10 neurons (with feature selection)	With Cost Matrix ?	Misclassifications out of 10299 (folds=2, 50% train)	Overall Final cost which impacted the training (error backpropagation)
1	48.194	Yes	5335	201438
10	79.443	Yes	2117	2770844
1	17.664	No	8479	NA
10	64.925	No	3612	NA

4. CONCLUSION

For datasets with less number of features and classes it has been observed that the accuracy improves by using a misclassification cost matrix indicating that the cost matrix has played a positive role in training the neural network.

The accuracy for the larger dataset with many features with no feature selection step the is observed to go down despite using cost matrix, so we used the feature selection step as mentioned in the paper [4] and then trained the neural network to get better accuracies

One important result concluded from this experiment is that the accuracy is found to improve with same magnitude even if the cost from the misclassification cost matrix is used directly, i.e., without squaring the cost as mentioned in the paper [4].

5. REFERENCES

[1] Code:
https://github.com/sheshappanavar/MisclassificationCostBasedNeuralNetworkTraining.git

[2] Code that helped: https://machinelearningmastery.com/implement-backpropagation-algorithm-scratch-python/

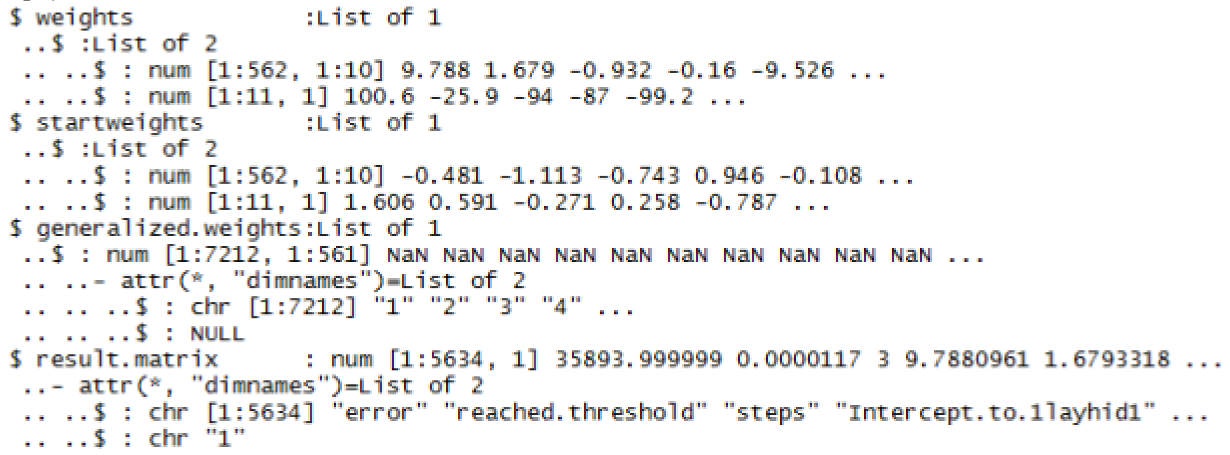
[3] Preprocessing code for Human Activity Recognition Data:
https://github.com/UserIdentificationBTAS/btas-2016/blob/master/code/preprocess.R

[4] Useful documentation for updating errors using misclassification costs
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.13.8285&rep=rep1&type=pdf

[5] Documentation to compare accuracies for HAR dataset using Neural Network and other algorithms http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7791159

APPENDIX

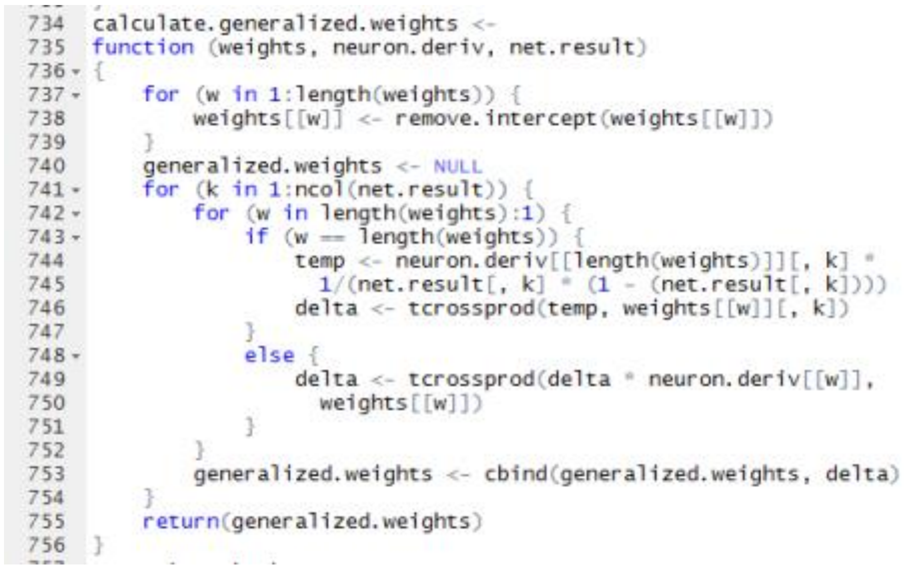
The R code library turned out to be buggy during some of the iterations where it used to fail to handle divide by zero issues leading to NaN (not a number) values as shown below for the generalized weight vectors leading to premature neural network model (within couple of training steps).



```
 $ weights            :List of 1
  ..$ :List of 2
  .. ..$ : num [1:562, 1:10] 9.788 1.679 -0.932 -0.16 -9.526 ...
  .. ..$ : num [1:11, 1] 100.6 -25.9 -94 -87 -99.2 ...
 $ startweights       :List of 1
  ..$ :List of 2
  .. ..$ : num [1:562, 1:10] -0.481 -1.113 -0.743 0.946 -0.108 ...
  .. ..$ : num [1:11, 1] 1.606 0.591 -0.271 0.258 -0.787 ...
 $ generalized.weights:List of 1
  ..$ : num [1:7212, 1:561] NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN ...
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : chr [1:7212] "1" "2" "3" "4" ...
  .. .. ..$ : NULL
 $ result.matrix      : num [1:5634, 1] 35893.999999 0.0000117 3 9.7880961 1.6793318 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : chr [1:5634] "error" "reached.threshold" "steps" "Intercept.to.1layhid1" ...
  .. ..$ : chr "1"
```

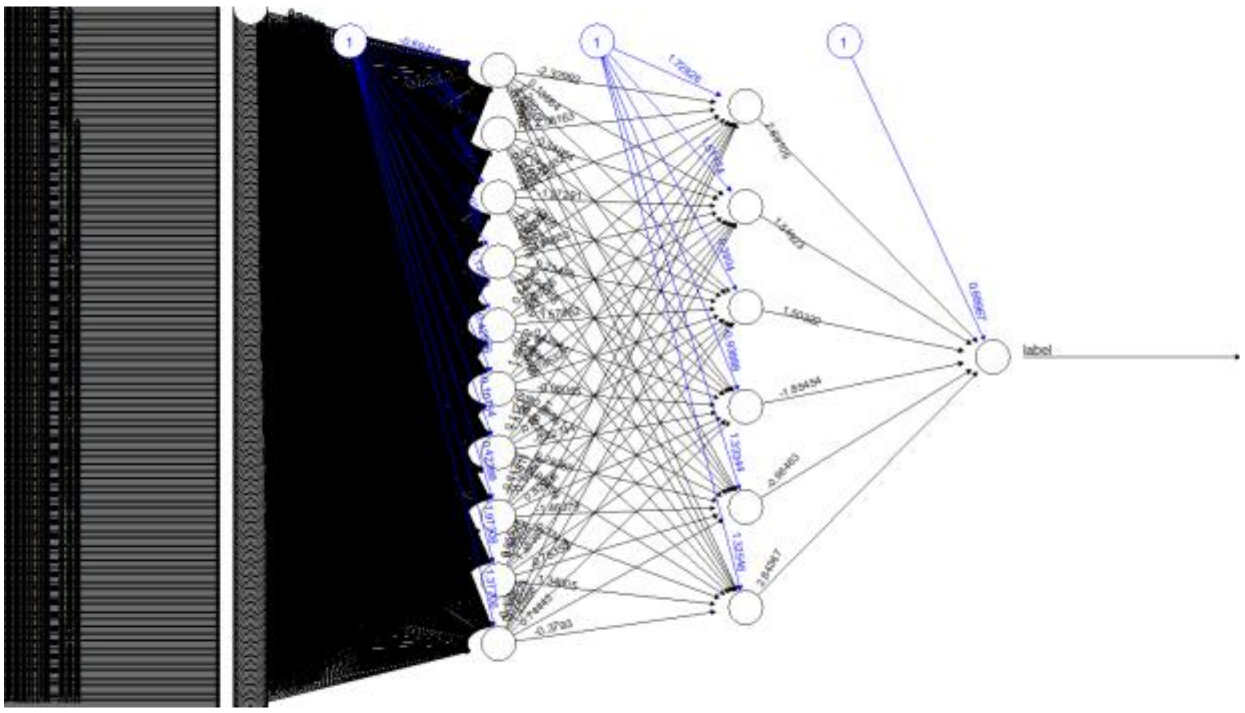
library(neuralnet): https://cran.r-project.org/web/packages/neuralnet/neuralnet.pdf

Reviewed neuralnet library documentation and the code with guidelines provided in link (http://russellsstewart.com/notes/0.html) to trace the buggy nature of the library code as shown below where if net.result[,k] if at all took a value of 0 or 1 then the denominator of temp evaluating expression turned out to be a 0 hence leading to a NaN value for temp and hence to the generalized weights. However, a work around didn't help in this case as also for some data points the entire weight vectors turned to 0 whenever the numerator was 0.

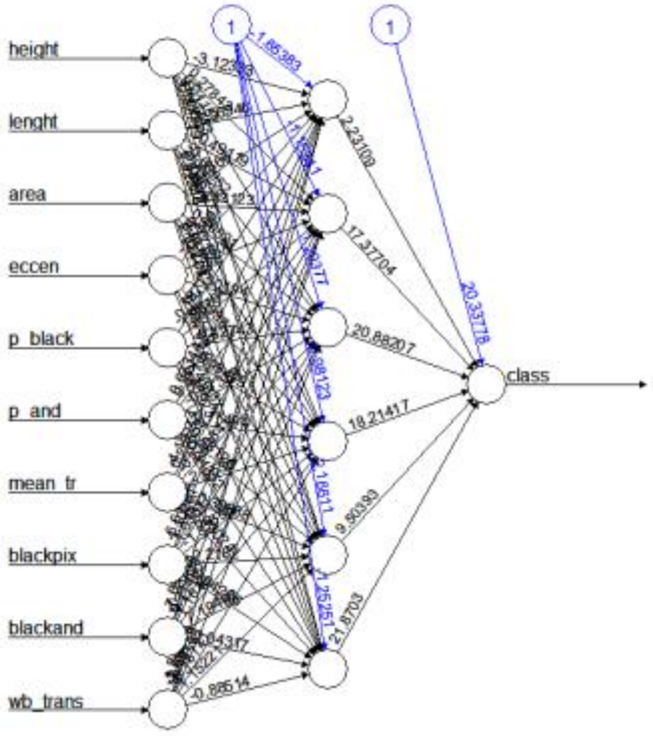


```
calculate.generalized.weights <-
function (weights, neuron.deriv, net.result)
{
    for (w in 1:length(weights)) {
        weights[[w]] <- remove.intercept(weights[[w]])
    }
    generalized.weights <- NULL
    for (k in 1:ncol(net.result)) {
        for (w in length(weights):1) {
            if (w == length(weights)) {
                temp <- neuron.deriv[[length(weights)]][, k] *
                  1/(net.result[, k] * (1 - (net.result[, k])))
                delta <- tcrossprod(temp, weights[[w]][, k])
            }
            else {
                delta <- tcrossprod(delta * neuron.deriv[[w]],
                  weights[[w]])
            }
        }
        generalized.weights <- cbind(generalized.weights, delta)
    }
    return(generalized.weights)
}
```

Below two are the visualization of the network models for Human activity recognition and pages-block datasets where the model stopped training due to NaN or 0 values of the generalized weights. Weights not visible as the model has 561 input features.



Human Activity Recognition Network Model



Page Block