BREAST CANCER CLASSIFICATION USING R TENSORFLOW

UNIVERSITY OF MINNESOTA MORRIS

Vipul Sharma

OVERVIEW

1. Project Description

The main purpose of the project is to predict whether cancer present is through Benign tumor or Malignant using deep learning neural networks. I am interested in this project because it helps me understand the depth on how oncology works and connecting it to artificial intelligence. This project will hopefully help me acquire skills on how to tackle various difficult situations that I might get stuck on while building this project.

2. Tools Used

Tools used were RStudio and R TensorFlow (Keras).

3. What is the dataset about?

The data set I used is called Breast Cancer Wisconsin (Diagnostic) Data Set. The features are computed from a digitized image of a fine needle aspirate (FNA) of a breast mass which describes characteristics of the cell nuclei present in the image. Data was obtained using Multisurface Method-Tree (MSM-T), a classification method which uses linear programming to construct a decision tree.

4. Who collected the data?

I got the data through this link .

Creators:

- Dr. William H. Wolberg, General Surgery Dept.

  University of Wisconsin, Clinical Sciences Center

  Madison, WI 53792

  wolberg '@' eagle.surgery.wisc.edu

- W. Nick Street, Computer Sciences Dept.

University of Wisconsin, 1210 West Dayton St., Madison, WI 53706

Street '@' cs.wisc.edu 608-262-6619

- Olvi L. Mangasarian, Computer Sciences Dept.

University of Wisconsin, 1210 West Dayton St., Madison, WI 53706

olvi '@' cs.wisc.edu

5. How to upload the data into RStudio?

After downloading the data from the above link go to RStudio. On the lower right pane look for Upload, clicking on which will ask the user to upload the data file.

6. Data Structure

```
str(dataset)
summary(dataset)
```

The summary metrics are mean, standard deviation, worst (or largest). Ten real-valued features are computed for each cell nucleus based on the three-summary metrics (except diagnosis the target variable). These ten variables are shown under the table below:

Variable Name	Variable Name
Radius	Perimeter
Area	Smoothness
Compactness	Concavity
Concave points	Symmetry
Fractal dimension	Benign or Malignant

In all, we have 569 instances with 32 attributes per instance and there are no NA’s.

7. Was I able to reach the goal?

The goal of this project concisely was to predict Benign or Malignant tumor using R TensorFlow. I was successful in using R TensorFlow using one of its API for R called Keras. More details on R TensorFlow can be found here. I got a good accuracy of 92.96 percent.

8. Packages Used

The packages that I used for the RStudio project served purposes both for the creation of graphs, to analysis of data, plotting the neural network and finally training and predicting the accuracy. The table below shows the different packages I used:

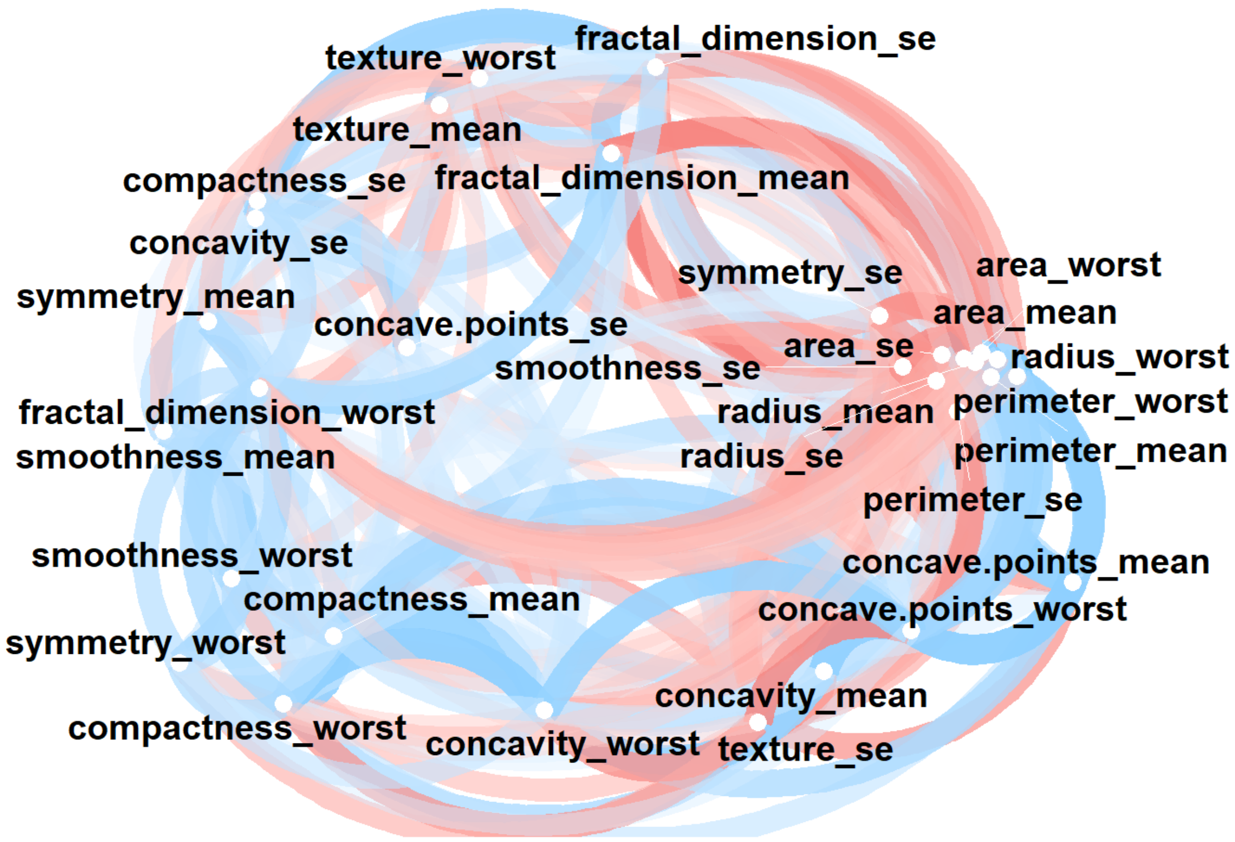
Package Name	Package Name
tensorflow	tfestimator
corrr	readr
plotly	caTools
caret	keras
corrplot	magrittr

9. Correlation Analysis

To check how each of these variables correlated I took out the summary and found that the mean factor for all these variables were giving me a strong correlation which I can group later to increase my accuracy of the model. This will reduce our feature space substantially, making our model more generalizable. The correlation plot uses clustering to make it easy to see which variables are closely correlated with each other. The closer each variable is to each other the higher the relationship while the opposite is true for widely spaced variables. The color of the line represents the direction of the correlation while the line shade and thickness represent the strength of the relationship.

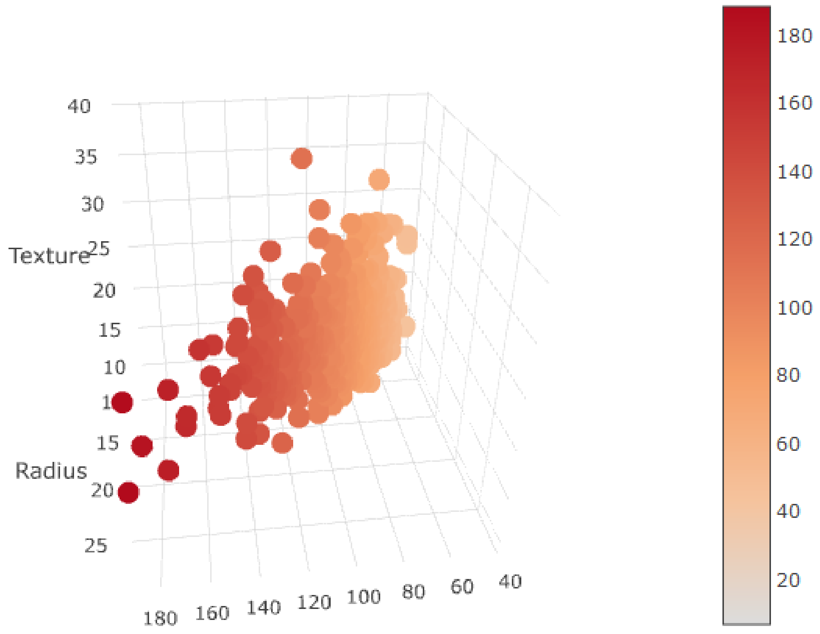
```
corr_mat <- cor(dataset[,3:ncol(dataset)])

network_plot(correlate(corr_mat))
```



10. Strength Graph

I plotted the strength map of which mean variable was the strongest which was texture, radius and perimeter. This showed us that texture had 70% data higher than 20 whereas perimeter had 70% data higher than 120 in each of their own domain.



11. One Hot Encoding

Our data has the label column as string. 62.74% of the data comprises of ‘B’ and the rest 37.26% is ‘B’. Therefore, For the neural network to process this data we convert the factor ‘M’ to 1 and ‘B’ to 0.

```
dataset$diagnosis[dataset$diagnosis=='M'] <-1

dataset$diagnosis[dataset$diagnosis=='B'] <-0

head(dataset$diagnosis, 20)

## [1] "1" "1" "1" "1" "1" "1" "1" "1" "1" "1" "1" "1" "1" "1" "1" "1" "1"
## [18] "1" "1" "0"
```

12. Train and Test Splitting

Separating data into training and testing sets is an important part of evaluating data mining models. R randomly samples the data to help ensure that the testing and training sets are similar. By using similar data for training and testing, one can minimize the effects of data discrepancies and better understand the characteristics of the model. After a model has been processed by using the training set, test the model by making predictions against the test set. Because the data in the testing set already contains known values for the attribute that you want to predict, it is easy to determine whether the model's guesses are correct. The package I used for train – test split was caTools and the split ratio I chose was 0.70. We are only going to check head for the training set and not the testing set because of the size of the output.

```
library(caTools)

X <- data.matrix(dataset[3:32])

y <- dataset$diagnosis

set.seed(1000)

split = sample.split(dataset$diagnosis, SplitRatio = 0.70)

X_train = subset(X, split==TRUE)

X_test = subset(X, split==FALSE)

split = sample.split(dataset$diagnosis, SplitRatio = 0.70)

y_train = subset(y, split==TRUE)

y_test = subset(y, split==FALSE)

head(X_train,5)
```

```
##      radius_mean texture_mean perimeter_mean area_mean smoothness_mean
## [1,]       17.99        10.38         122.80    1001.0          0.1184
## [2,]       19.69        21.25         130.00    1203.0          0.1096
## [3,]       11.42        20.38          77.58     386.1          0.1425
## [4,]       20.29        14.34         135.10    1297.0          0.1003
## [5,]       12.45        15.70          82.57     477.1          0.1278
##      compactness_mean concavity_mean concave.points_mean symmetry_mean
## [1,]           0.2776         0.3001             0.14710        0.2419
## [2,]           0.1599         0.1974             0.12790        0.2069
## [3,]           0.2839         0.2414             0.10520        0.2597
## [4,]           0.1328         0.1980             0.10430        0.1809
## [5,]           0.1700         0.1578             0.08089        0.2087
##      fractal_dimension_mean radius_se texture_se perimeter_se area_se
## [1,]                0.07871    1.0950     0.9053        8.589  153.40
## [2,]                0.05999    0.7456     0.7869        4.585   94.03
## [3,]                0.09744    0.4956     1.1560        3.445   27.23
## [4,]                0.05883    0.7572     0.7813        5.438   94.44
## [5,]                0.07613    0.3345     0.8902        2.217   27.19
##      smoothness_se compactness_se concavity_se concave.points_se
## [1,]      0.006399        0.04904      0.05373           0.01587
## [2,]      0.006150        0.04006      0.03832           0.02058
## [3,]      0.009110        0.07458      0.05661           0.01867
## [4,]      0.011490        0.02461      0.05688           0.01885
## [5,]      0.007510        0.03345      0.03672           0.01137
##      symmetry_se fractal_dimension_se radius_worst texture_worst
## [1,]     0.03003             0.006193        25.38         17.33
## [2,]     0.02250             0.004571        23.57         25.53
## [3,]     0.05963             0.009208        14.91         26.50
## [4,]     0.01756             0.005115        22.54         16.67
## [5,]     0.02165             0.005082        15.47         23.75
##      perimeter_worst area_worst smoothness_worst compactness_worst
## [1,]          184.60     2019.0           0.1622            0.6656
## [2,]          152.50     1709.0           0.1444            0.4245
## [3,]           98.87      567.7           0.2098            0.8663
## [4,]          152.20     1575.0           0.1374            0.2050
## [5,]          103.40      741.6           0.1791            0.5249
##       concavity_worst concave.points_worst symmetry_worst
## [1,]          0.7119               0.2654         0.4601
## [2,]          0.4504               0.2430         0.3613
## [3,]          0.6869               0.2575         0.6638
## [4,]          0.4000               0.1625         0.2364
## [5,]          0.5355               0.1741         0.3985
##       fractal_dimension_worst
## [1,]                 0.11890
## [2,]                 0.08758
## [3,]                 0.17300
## [4,]                 0.07678
## [5,]                 0.12440
```

13. Normalizing Features

The features above are now normalized by scaling them which basically is zero mean and unit variance on each of these features. The goal of normalization is to change the values of numeric columns in the dataset to use a common scale, without distorting differences in the ranges of values or losing information. We will use head again on the training set to see the normalized features.

```
X_train = scale(X_train)

X_test = scale(X_test)

head(X_train,3)
```

```
##       radius_mean texture_mean perimeter_mean  area_mean smoothness_mean
## [1,]   1.1445990   -2.0448775      1.3209129  1.0465217       1.5437124
## [2,]   1.6413301    0.4395844      1.6258477  1.6445948       0.9258259
## [3,]  -0.7751209    0.2407360     -0.5942473 -0.7740486       3.2358787
##       compactness_mean concavity_mean concave.points_mean symmetry_mean
## [1,]         3.204044       2.774844            2.578051     2.0926855
## [2,]         1.015543       1.435461            2.075634     0.8678387
## [3,]         3.321186       2.009296            1.481631     2.7156076
##       fractal_dimension_mean radius_se texture_se perimeter_se    area_se
## [1,]              2.1499505 2.5810565 -0.5821827    2.9096764  2.6630752
## [2,]              -0.3880538 1.2657773 -0.7985289    0.8580176  1.2679613
## [3,]               4.6893105 0.3246789 -0.1240917    0.2738790 -0.3017474
##      smoothness_se compactness_se concavity_se concave.points_se
## [1,]    -0.2459958      1.3425204    0.8307200          0.672795
## [2,]    -0.3305553      0.8201671    0.2543676          1.471998
## [3,]     0.6746500      2.8281445    0.9384354          1.147905
##      symmetry_se fractal_dimension_se radius_worst texture_worst
## [1,]    1.105890            1.0090735    1.9458265   -1.31286554
## [2,]    0.210544            0.3213961    1.5619188   -0.01550347
## [3,]    4.625444            2.2873395   -0.2748992    0.13796497
##      perimeter_worst area_worst smoothness_worst compactness_worst
## [1,]       2.3618701  2.0813792        1.3164215          2.582145
## [2,]       1.3869910  1.5200023        0.5420193          1.060893
## [3,]      -0.2417556 -0.5467701        3.3872949          3.848489
##      concavity_worst concave.points_worst symmetry_worst
## [1,]       2.2069538             2.340153       2.699877
## [2,]       0.9087849             1.994741       1.132278
## [3,]       2.0828458             2.218333       5.931860
##      fractal_dimension_worst
## [1,]               1.8762688
## [2,]               0.1976631
## [3,]               4.7757762
```

14. Neural Network Structure

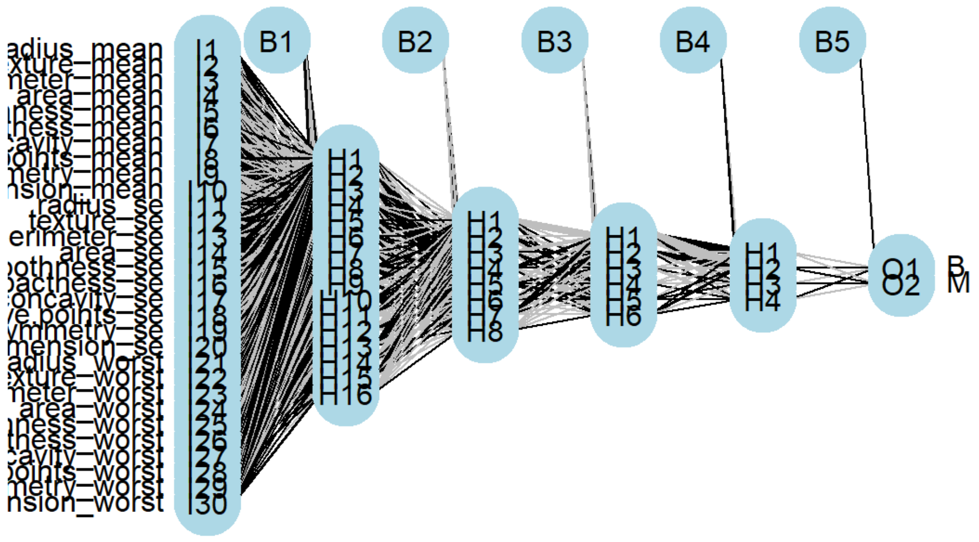
We are finally ready to work on the neural network. Based on the data we have 32 variables as input nodes out of which 10 will be grouped. The number of hidden layers I chose was 3, each of which has 16, 8 and 6 nodes. The output layer will have 2 nodes which are the labels itself. Overall the neural network will look like the following outputted diagram. The package I used is plotnet.

```
B <- c(rep(0, 397),1)

M <- c(0, rep(1, 397))

set.seed(2)

NN = neuralnet(B+M ~
radius_mean+texture_mean+perimeter_mean+area_mean+smoothness_mean+compactness
_mean+concavity_mean+concave.points_mean+symmetry_mean+fractal_dimension_mean
+radius_se+texture_se+perimeter_se+area_se+smoothness_se+compactness_se+conca
vity_se+concave.points_se+symmetry_se+fractal_dimension_se+radius_worst+textu
re_worst+perimeter_worst+area_worst+smoothness_worst+compactness_worst+concav
ity_worst+concave.points_worst+symmetry_worst+fractal_dimension_worst,
X_train, hidden = c(16,8,6,4) , linear.output = TRUE )
plotnet(NN)
```



15. Using R TensorFlow Keras API to add layers

We will use Keras, a package used in R Tensorflow core API. Now we will include all the hidden layers by adding specific weights and biases to each layer. We will use a sequential model with hidden layers having RELU function while the output is sigmoid. For compiling the model, the optimizer algorithm used is RMSPROP. The loss is done by binary cross entropy method whose formula given two distributions over discrete variable x, where q(x) is the estimate for true distribution p(x) and finally the validation metric I used is accuracy. The layers here are dense layers each of which runs on the rectified linear unit (ReLU) activation and the final layer has the sigmoid activation function for predicting the labels. The learning rate for training the model is 0.001 to prevent overfitting or underfitting of the model.

```
model <- keras_model_sequential() %>%
  layer_dense(units = 16, activation = "relu", input_shape = c(30)) %>%
  layer_dense(units = 8,  activation = "relu") %>%
  layer_dense(units = 6,  activation = "relu") %>%
  layer_dense(units = 1,  activation = "sigmoid")
model %>% compile(
  optimizer = optimizer_rmsprop(lr=0.001),
  loss = "binary_crossentropy",
  metrics = c("accuracy")
)
```

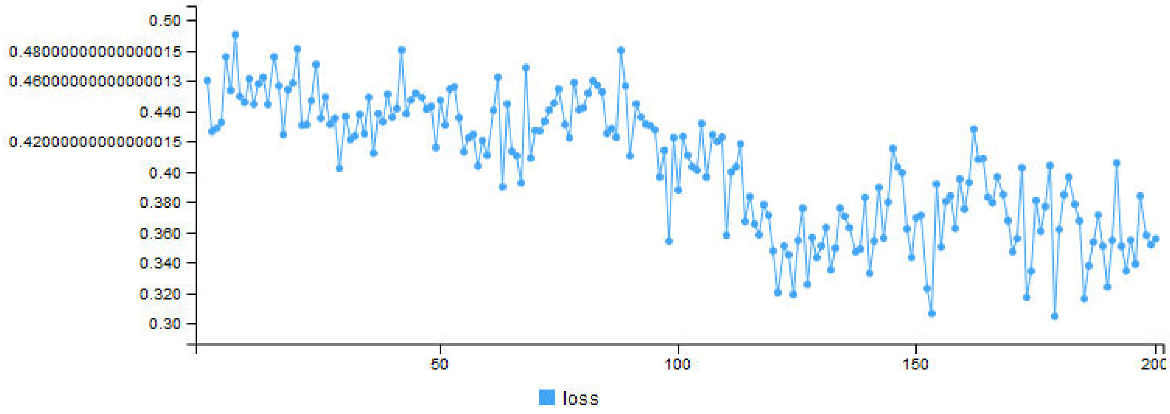
16. Explanation of the algorithm

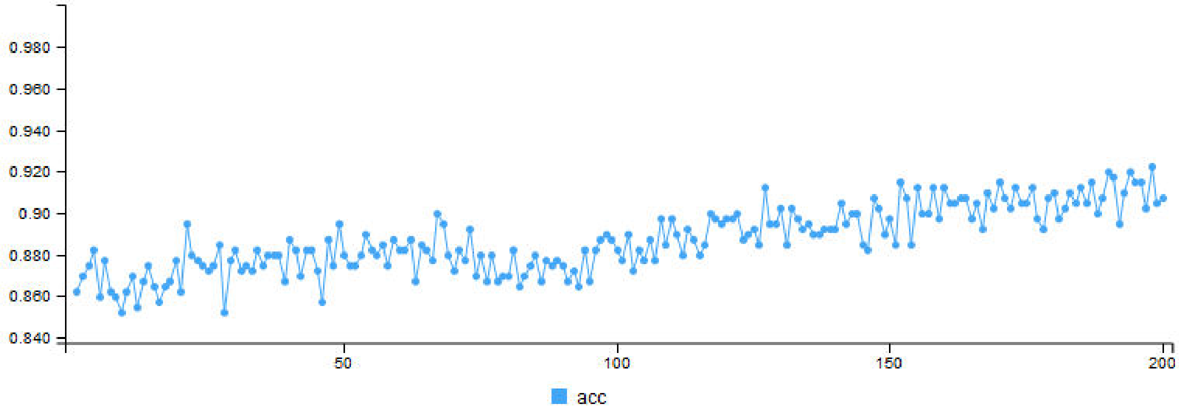
We will use forward propagation as the basis of the algorithm. For training, at each step of gradient descent in the hidden layers, we do forward propagation with the current model parameters to get an output. At each layer of the network, we compute a matrix multiplication of the output from the previous layer and the weights connecting the previous layer to the current layer. We then add the bias to improve the value at each step. After the summation, an activation funciton like Sigmoid is applied, which converts the weighted result into a value between 0 and 1, indicating the classification at that layer. We will then use backpropagation, an algorithm to adjust our parameters to minimize loss from that output. At the output layer we compare our prediction with the actual result to get the accuracy of the model.

17. Training and Compiling the model.

The hyperparameters used for fitting the model were learning Rate as 0.001, epochs which were 200 iterations and a batch size of 1. The model was trained on X_train and X_test specifically and evaluated on y_train and y_test which the model has not seen yet.

```
breast_model <- model %>% fit(
  X_train,
  y_train,
  epochs = 200,
  batch_size = 1)
results <- model %>% evaluate(X_test, y_test)
```





18. Results

As shown in the graph above I got of 93 percent and a loss of 0.28. This can further be removed if we group more variables together based on the correlation. Thus we can see that deep neural networks can help classify bigger problems like cancer itself without the use of a lot of human effort.