Deep Learning - Breast Cancer

Howard Nguyen

2023-01-23

Loading Breast Cancer dataset

There are many software packages that offer neural net implementations that may be applied directly. We will survey these as we proceed through the monograph. Our first example will be the use of the R programming language, in which there are many packages for neural networks.

Load library and dataset

Clean off rows with missing data

```
##        Id Cl.thickness Cell.size Cell.shape Marg.adhesion Epith.c.size
## 1 1000025            5         1          1             1            2
## 2 1002945            5         4          4             5            7
## 3 1015425            3         1          1             1            2
## 4 1016277            6         8          8             1            3
## 5 1017023            4         1          1             3            2
## 6 1017122            8        10         10             8            7
##   Bare.nuclei Bl.cromatin Normal.nucleoli Mitoses     Class
## 1           1           3               1       1    benign
## 2          10           3               2       1    benign
## 3           2           3               1       1    benign
## 4           4           3               7       1    benign
## 5           1           3               1       1    benign
## 6          10           9               7       1 malignant
```

```
## [1] 683  11
```

```
##  [1] "Id"              "Cl.thickness"    "Cell.size"       "Cell.shape"
##  [5] "Marg.adhesion"   "Epith.c.size"    "Bare.nuclei"     "Bl.cromatin"
##  [9] "Normal.nucleoli" "Mitoses"         "Class"
```

The last column in the data set is “Class” which is either bening or malignant. The goal of the analysis is to construct a model that learns to decide whether the tumor is malignant or not.

The Deepnet package

We apply the package to the cancer data set as follows. First, we create the dependent variable, and also the feature set of independent variables.

We then use the function nn.train from the deepnet package to model the neural network. As can be seen in the program code below, we have 5 nodes in the single hidden layer.

```
##            [,1]
## [1,] 0.2619092
## [2,] 0.4397302
## [3,] 0.2711848
## [4,] 0.4441957
## [5,] 0.2763118
## [6,] 0.4707129
```

We take the output of the network and convert it into classes, such that class "0" is benign and class "1" is malignant. We then construct the "confusion matrix" to see how well the model does in-sample. The table function here creates the confusion matrix, which is a tabulation of how many observations that were benign and malignant were correctly classified. This is a handy way of assessing how successful a machine learning model is at classification.

```
##    yhat
## y     0   1
##   0 423  21
##   1   4 235
```

We can see that the diagonal of the confusion matrix contains most of the entries, thereby suggesting that the neural net does a very good job of classification. The accuracy may be computed easily as the number of diagnal entries in the confusion matrix divided by the total count of values in the matrix.

```
## [1] 0.9633968
```

Now that we have seen the model work you can delve into the function nn.train in some more detail to examine the options it allows.

The neuralnet package

For comparison, we try the neuralnet package. The commands are mostly the same. The function in the package is also called neuralnet.

This package also performs very well on this data set. This package has an interesting function that allows plotting the neural network. Use the function plot() and pass the output object to it, in this case nn. This needs to be run interactively, but here is a sample outpt of the plot.

```
##    yhat
## y     0   1
##   0 439   5
##   1   0 239
```

Using H2O

The good folks at h2o, see http://www.h2o.ai/, have developed a Java-based version of R, in which they also provide a deep learning network application.

H2O is open source, in-memory, distributed, fast, and provides a scalable machine learning and predictive analytics platform for building machine learning models on big data. H2O's core code is written in Java. Inside H2O, a Distributed Key/Value store is used to access and reference data, models, objects, etc., across all nodes and machines. The algorithms are implemented in a Map/Reduce framework and utilizes multi-threading. The data is read in parallel and is distributed across the cluster and stored in memory in a columnar format in a compressed way. Therefore, even on a single machine, the deep learning algorithm in H2O will exploit all cores of the CPU in parallel.

Here we start up a server using all cores of the machine, and then use the H2O package's deep learning toolkit to fit a model.