Data 609 - Module8

Amit Kapoor

11/28/2021

Contents


```
library(nnet)
library(caret)
```

```
## Loading required package: lattice
```

```
## Loading required package: ggplot2
```

Ex.1

Use the nnet package to analyze the iris data set. Use 80% of the 150 samples as the training data and the rest for validation. Discuss the results.

Solution

```
# iris dataset
data(iris)

set.seed(111)

partition <- sample(nrow(iris), nrow(iris)*0.80)
train <- iris[partition,]
test <- iris[-partition,]

# apply model
iris_nnet <- nnet(Species ~ ., size=2, data = train)
```

```
## # weights:  19
## initial  value 143.031042
## iter  10 value 55.654176
## iter  20 value 24.628118
## iter  30 value 3.135566
## iter  40 value 0.026453
## final  value 0.000087
## converged
```

```
# model summary
summary(iris_nnet)
```

```
## a 4-2-3 network with 19 weights
## options were - softmax modelling
##   b->h1  i1->h1  i2->h1  i3->h1  i4->h1
##   -1.44   -0.50   -1.29    0.88    2.93
##   b->h2  i1->h2  i2->h2  i3->h2  i4->h2
##   -1.04   -2.67   -1.44   -2.14   -1.26
##   b->o1  h1->o1  h2->o1
##  177.15 -468.68    2.70
##   b->o2  h1->o2  h2->o2
##  119.57  -44.96   -1.50
##   b->o3  h1->o3  h2->o3
## -297.87  513.30    0.38
```

```
# confusionMatrix
confusionMatrix(as.factor(predict(iris_nnet, test, type = 'class')), test$Species)
```

```
## Confusion Matrix and Statistics
##
##             Reference
## Prediction   setosa versicolor virginica
##   setosa         11          0         0
##   versicolor      0          6         2
##   virginica       0          1        10
##
## Overall Statistics
##
##                Accuracy : 0.9
##                  95% CI : (0.7347, 0.9789)
##     No Information Rate : 0.4
##     P-Value [Acc > NIR] : 1.698e-08
##
##                   Kappa : 0.8477
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: setosa Class: versicolor Class: virginica
## Sensitivity                 1.0000            0.8571           0.8333
## Specificity                 1.0000            0.9130           0.9444
## Pos Pred Value              1.0000            0.7500           0.9091
## Neg Pred Value              1.0000            0.9545           0.8947
## Prevalence                  0.3667            0.2333           0.4000
## Detection Rate              0.3667            0.2000           0.3333
## Detection Prevalence        0.3667            0.2667           0.3667
## Balanced Accuracy           1.0000            0.8851           0.8889
```

We can see here using nnet package we see 90% accuracy in prediction. There are 19 weights used with final value as 0.000087.

Ex.2

As a mini project, install the keras package and learn how to use it. Then, carry out various tasks that may be useful to your project and studies.

Solution

We will first install the keras package using tensorflow.

```
#install.packages("devtools")
#devtools::install_github("rstudio/keras")
#install_tensorflow()
```

```
library(keras)
library(tensorflow)
```

```
##
## Attaching package: 'tensorflow'
```

```
## The following object is masked from 'package:caret':
##
##     train
```

We will use here MNIST dataset from keras recognizing handwritten digits. MNIST dataset includes 28 x 28 grayscale images of handwritten digits. This dataset includes images labels as well to recognize the digit.

```
mnist <- dataset_mnist()
```

```
## Loaded Tensorflow version 2.7.0
```

```
# train and test data
img_training <- mnist$train$x
label_training <- mnist$train$y
img_testing <- mnist$test$x
label_testing <- mnist$test$y
```

Seeing the dimension of training and testing images, there are 60K images for training and 10k images for testing.

```
dim(img_training)
```

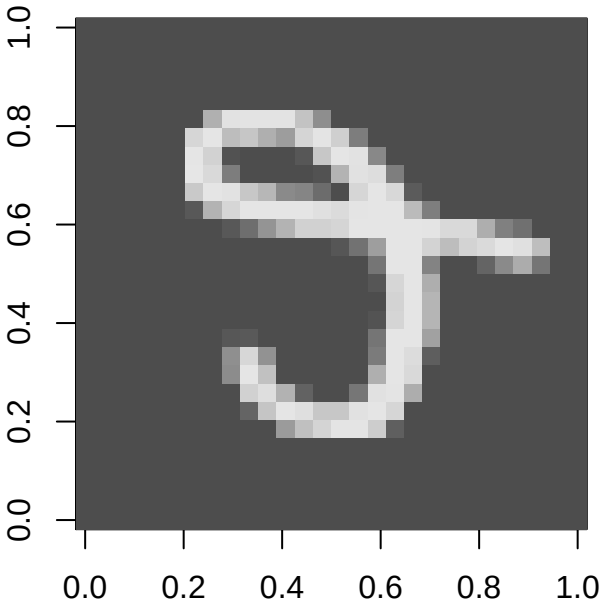
```
## [1] 60000    28    28
```

```
dim(img_testing)
```

```
## [1] 10000    28    28
```

Here is an example of 17th image from training set.

```
# 17th image
digit <- img_training[17, 1:28, 1:28]
# set the parameter- "s" for square plotting region
par(pty="s")
image(t(digit), col=gray.colors(256))
```



As we seen above, the x data is a three dim array (images, width, height) that would be reshaped in single dimension (28x28 into a single object of length 784). Next we will convert grayscale values from integers (between 0 and 255) into floating values between 0 and 1. Next is y data that is integer values between 0 and 9 and it will be one hot encoded for training.

```
# reshaping
img_training <- array_reshape(img_training, c(nrow(img_training), 784))
img_testing <- array_reshape(img_testing, c(nrow(img_testing), 784))
# reacaling
img_training <- img_training / 255
img_testing <- img_testing / 255

# one-hot encoded
label_training <- to_categorical(label_training, 10)
label_testing <- to_categorical(label_testing, 10)
```

Next we will create the sequential model and add layers into it. We have 3 layers here by providing input shape on the first one. we have used activation functions in first 2 layers as relu followed by softmax in final one.

```
model <- keras_model_sequential()
model %>%
  layer_dense(units = 256, activation = "relu", input_shape = c(784)) %>%
  layer_dense(units = 128, activation = "relu") %>%
  layer_dense(units = 10, activation = "softmax")

summary(model)
```

```
## Model: "sequential"
## ________________________________________________________________________________
##  Layer (type)                       Output Shape                    Param #
## ================================================================================
##  dense_2 (Dense)                    (None, 256)                     200960
##
##  dense_1 (Dense)                    (None, 128)                     32896
##
```

```
##  dense (Dense)                      (None, 10)                        1290
##
## ================================================================================
## Total params: 235,146
## Trainable params: 235,146
## Non-trainable params: 0
## ________________________________________________________________________________
```

Next we will compile the model.

```
# compile
model %>% compile(loss="categorical_crossentropy", optimizer=optimizer_rmsprop(), metrics=c("accuracy"))
```

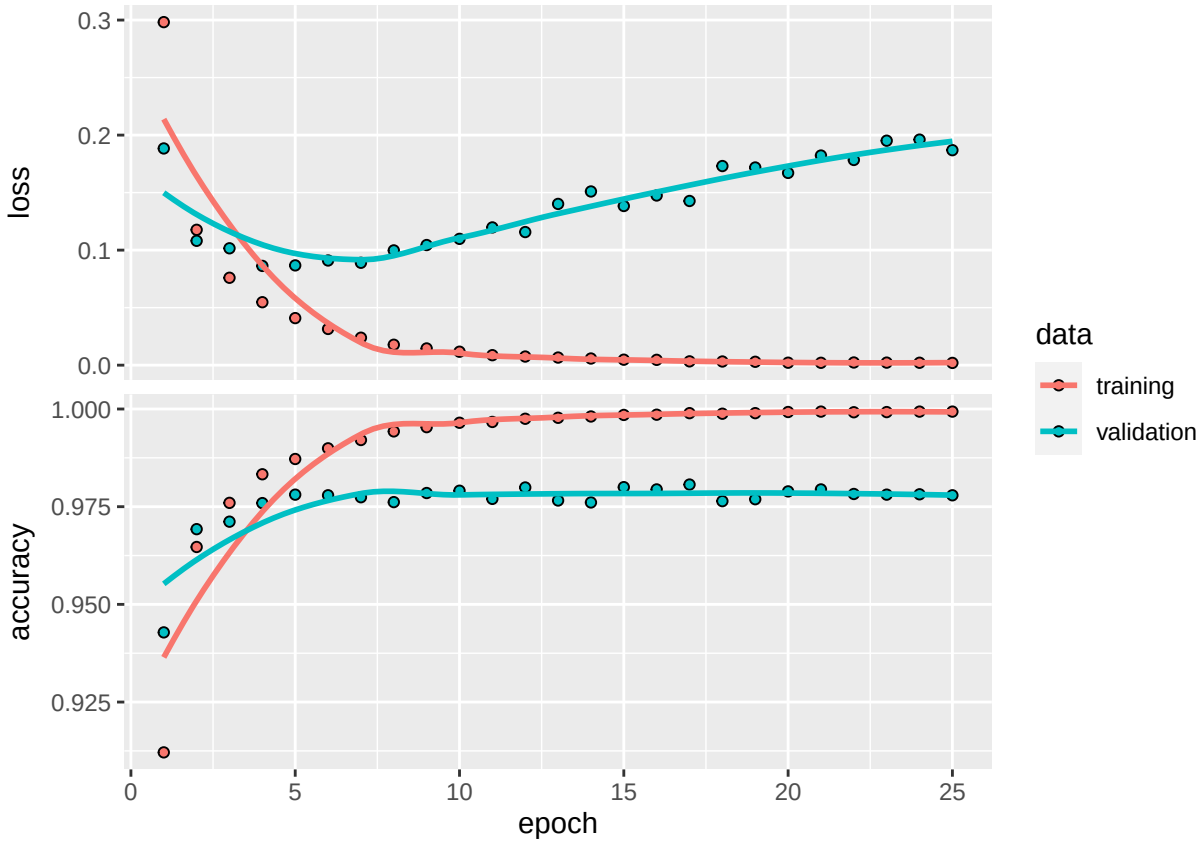
Next we will use fit() function to train the model with 25 epochs and batches of 128 images.

```
set.seed(609)

hist <- model %>% fit(
  img_training,
  label_training,
  epochs=25,
  batch_size=128,
  validation_split=0.2
)
```

```
plot(hist)
```

```
## `geom_smooth()` using formula 'y ~ x'
```



Here is loss and accuracy of model trained above.

```
# eval loss and accuracy
model %>% evaluate(img_testing, label_testing)
```

```
##      loss  accuracy
## 0.1577761 0.9796000
```

Now we will do final predictions from trained model.

```
# predicted classes
predictions <- model %>% predict(img_testing)
head(predictions)
```

```
##              [,1]         [,2]         [,3]         [,4]         [,5]
## [1,] 5.216219e-20 6.331575e-18 5.938360e-15 2.225248e-10 6.840071e-30
## [2,] 1.684273e-26 4.498968e-13 1.000000e+00 1.115118e-27 0.000000e+00
## [3,] 2.311298e-17 1.000000e+00 1.054413e-11 2.356686e-18 3.663629e-15
## [4,] 1.000000e+00 5.475427e-25 3.796372e-14 1.573599e-22 1.293738e-27
## [5,] 8.480671e-17 1.377107e-21 1.091259e-18 2.030316e-21 1.000000e+00
## [6,] 1.687905e-20 1.000000e+00 4.859610e-17 8.819515e-24 8.225830e-14
##              [,6]         [,7]         [,8]         [,9]        [,10]
## [1,] 2.374924e-19 6.095964e-26 1.000000e+00 3.835530e-21 2.193630e-13
## [2,] 1.650445e-30 3.891976e-23 0.000000e+00 3.347279e-25 0.000000e+00
## [3,] 4.321702e-16 4.630135e-13 1.662335e-10 5.615734e-12 1.687546e-19
## [4,] 1.729467e-23 9.377982e-16 2.286436e-18 8.322800e-24 1.385317e-12
## [5,] 2.931027e-20 1.549729e-14 1.154737e-12 5.713521e-16 3.164323e-08
## [6,] 2.027219e-23 9.485509e-19 1.782775e-10 4.514305e-16 2.241644e-22
```

During this study we have learned how to create keras sequential model, split the mnist dataset in train and test, reshape the data, fit the data into model and then do predictions. I see keras as a very good tool to used in future projects.

References

https://www.datacamp.com/community/tutorials/keras-r-deep-learning https://www.analyticsvidhya.com/blog/2017/06/getting-started-with-deep-learning-using-keras-in-r/