AdvanceML_Discussion_2

Ali Alghaithi

12/20/2020

```r
library(ggplot2)
library(cowplot)
library(dplyr)
library(tidyverse)
library(MASS)
library(xgboost)
library(caret)
#library(h2o)
library(e1071 )
library(tensorflow)
library(keras)
```

The data we used fro this disscusion is classic dataset of handwritten images. The goal is to take an image of a handwritten single digit, and determine what that digit is. The data files train.csv from zero through nine.Each image is 28 pixels in height and 28 pixels in width, for a total of 784 pixels in total. Each pixel has a single pixel-value associated with it, indicating the lightness or darkness of that pixel, with higher numbers meaning darker. This pixel-value is an integer between 0 and 255, inclusive.

The training data set, (train.csv), has 785 columns. The first column, called "label", is the digit that was drawn by the user. The rest of the columns contain the pixel-values of the associated image.

```r
library(readr)

train <-  read_csv("trainMNIST.csv")
test = read_csv("testMNIST.csv")

train$label<- as.integer(train$label)
#first 10 columns
head(train[,1:10])
```

```
## # A tibble: 6 x 10
##   label pixel0 pixel1 pixel2 pixel3 pixel4 pixel5 pixel6 pixel7 pixel8
##   <int>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>
## 1     1      0      0      0      0      0      0      0      0      0
## 2     0      0      0      0      0      0      0      0      0      0
## 3     1      0      0      0      0      0      0      0      0      0
## 4     4      0      0      0      0      0      0      0      0      0
## 5     0      0      0      0      0      0      0      0      0      0
## 6     0      0      0      0      0      0      0      0      0      0
```

```r
train<- na.omit(train)
```

```r
train.data = train[,-1]
train.label = train[,1]
```

```r
# Notmailzing the images array to be in the range of 0-1 by dividing them by the max possible value (wh
train.data = train.data/255
test = test/255.0
train.label<- as.numeric(as.character(unlist(train.label[[1]])))
```

# XGBoost

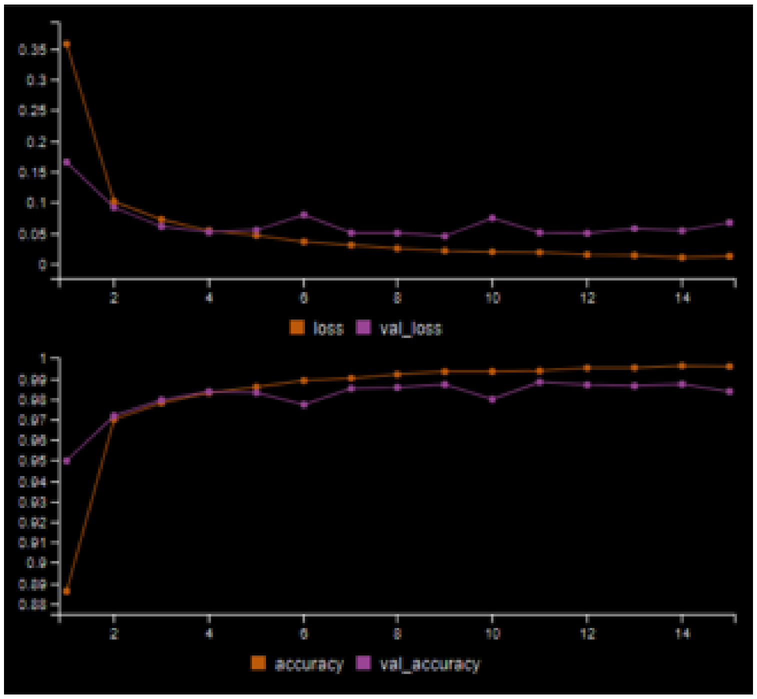
```r
set.seed(123)
trainIndex <- sample(1:length(train.label),0.8*length(train.label))
cv.data.x = train.data[-trainIndex,]
train.data.x = train.data[trainIndex,]
cv.label.y = train.label[-trainIndex]
train.label.y = train.label[trainIndex]

# convert every variable to numeric, even the integer variables
train.data.x <- as.data.frame(lapply(train.data.x, as.numeric))
cv.data.x <- as.data.frame(lapply(cv.data.x, as.numeric))

# convert data to xgboost format
train.xgb <- xgb.DMatrix(data = data.matrix(train.data.x), label = train.label.y)
cv.xgb <- xgb.DMatrix(data = data.matrix(cv.data.x), label = cv.label.y)



params <- list (
                eta = 0.3,
                max_depth = 5,
                min_child_weight = 1,
                subsample = 1,
                colsample_bytree = 1,
                objective = "multi:softmax",
                eval_metric = "merror",
                num_class = 11
)


xgb.model <- xgb.train(params, train.xgb, nrounds = 10)

xgb.predict <- predict(xgb.model, cv.xgb)
xgb.cm = confusionMatrix(as.factor(xgb.predict), as.factor(cv.label.y))
print(xgb.cm)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1   2   3   4   5   6   7   8   9
##          0 791   0  11   5   1   9  10   5   2   7
##          1   0 920   7   9   2   7   4   2  19   2
##          2   2   3 741  17   3   7   1  17   9   5
##          3   2   7  11 793   1  18   0   4  11  10
##          4   3   1  10   2 765   5  10   6   7  19
##          5   2   4   0  17   2 668  19   0   7   7
##          6   1   3   6   6   4  13 798   0   8   0
##          7   1   3  13  12   2   3   4 781   2  20
##          8   7   3   7  19   5  10  12   4 726   7
##          9   1   0   3  10  36   4   0  27  13 797
##
## Overall Statistics
##
##                Accuracy : 0.9262
##                  95% CI : (0.9204, 0.9317)
##     No Information Rate : 0.1124
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.918
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity           0.97654   0.9746  0.91595   0.8910  0.93179  0.89785
## Specificity           0.99341   0.9930  0.99157   0.9915  0.99169  0.99242
## Pos Pred Value        0.94055   0.9465  0.92050   0.9253  0.92391  0.92011
## Neg Pred Value        0.99749   0.9968  0.99105   0.9871  0.99260  0.99010
## Prevalence            0.09643   0.1124  0.09631   0.1060  0.09774  0.08857
## Detection Rate        0.09417   0.1095  0.08821   0.0944  0.09107  0.07952
## Detection Prevalence  0.10012   0.1157  0.09583   0.1020  0.09857  0.08643
## Balanced Accuracy     0.98498   0.9838  0.95376   0.9412  0.96174  0.94514
##                      Class: 6 Class: 7 Class: 8 Class: 9
## Sensitivity           0.93007  0.92317  0.90299  0.91190
## Specificity           0.99456  0.99206  0.99026  0.98751
## Pos Pred Value        0.95113  0.92866  0.90750  0.89450
## Neg Pred Value        0.99206  0.99140  0.98974  0.98975
## Prevalence            0.10214  0.10071  0.09571  0.10405
## Detection Rate        0.09500  0.09298  0.08643  0.09488
## Detection Prevalence  0.09988  0.10012  0.09524  0.10607
## Balanced Accuracy     0.96232  0.95761  0.94662  0.94970
```

CNN

```
train.x.keras <- array_reshape(data.matrix(train.data),
                               dim = c(nrow(train.data), 28, 28, 1))

test.x.keras <- array_reshape(data.matrix(test),
                               dim = c(nrow(test), 28, 28, 1))
#install_tensorflow()

train.y.cn = to_categorical( as.numeric(as.character(unlist(train.label[[1]])))

, num_classes = 10)



cn.model <- keras_model_sequential() %>%
```

```r
  layer_conv_2d(filters = 32,
                kernel_size = c(4,4),
                padding = "same", activation = "relu",
                input_shape = c(28, 28, 1)
                ) %>%

  layer_max_pooling_2d(pool_size = c(3,3)) %>%

  layer_conv_2d(filters = 32,
                kernel_size = c(4,4),
                padding = "same", activation = "relu",
                input_shape = c(28, 28, 1)
                ) %>%

  layer_max_pooling_2d(pool_size = c(3,3)) %>%

  layer_conv_2d(filters = 32,
                kernel_size = c(4,4),
                padding = "same", activation = "relu",
                input_shape = c(28, 28, 1)
                ) %>%

  layer_max_pooling_2d(pool_size = c(3,3)) %>%

  layer_flatten() %>%

  layer_dense(units = 16,
              activation = "relu") %>%

  layer_dense(units = 10,
              activation = "softmax",
              name = "Output"
              )

cn.model %>%
      compile(
        optimizer = optimizer_adam(lr=0.001),
        loss = "categorical_crossentropy",
        metrics = c("accuracy"))
```



Comments:

We see that CNN perform better than XGBoot in this data. Both methods were very chalanching to apply for multilabe classification problem and it was difficult; to see which one is performing better. However, the minst data is a great data to see what model performs better in term of accuracy. It is also saves a lot of time comparing to the MOAs data set"project data".

we only applied these method to onle validate from the train data set, and it might be different conculation if we test them on a test data have have a new behavior were images are not simmiler or including many colors.

In summary, both models are a great for image classfication problems.