Lab_11T

36-290 – Statistical Research Methodology

Week 11 Tuesday – Fall 2021

Data, Part I

Below we read in 10000 training and 5000 test data from the Massive Black II dataset that you analyzed in the random forest lab.

```
rm(list=ls())
file.path = "https://raw.githubusercontent.com/pefreeman/36-290/master/EXAMPLE_DATASETS/DM_GALAXY/Massive_Black_II.
Rdata"
load(url(file.path))
rm(file.path)

resp.train = resp.train.df$prop.sfr
w = which(resp.train>0)
pred.train = pred.train[w,]
resp.train = log10(resp.train[w])

resp.test  = resp.test.df$prop.sfr
w = which(resp.test>0)
pred.test = pred.test[w,]
resp.test = log10(resp.test[w])

set.seed(202)
s = sample(length(resp.train),10000)
pred.train = pred.train[s,]
resp.train = resp.train[s]
s = sample(length(resp.test),5000)
pred.test = pred.test[s,]
resp.test = resp.test[s]

pred = rbind(pred.train,pred.test)
resp = c(resp.train,resp.test)
df = cbind(pred,resp)
names(df)[10] = "sfr"
s = 10001:15000
```

Question 1

Let's keep this simple: code a deep-learning model! (Utilize the code given in the notes. Looking above, we see that we have `df`, with response variable `sfr`, and we have set that the test data rows are stored as `s`. When looking at the code in the notes, ignore the `[,-8]` in the `model.matrix()` call, as that was there to remove the eighth column. We are not removing any columns here.)

Note that when I run my solution set for the random forest lab, the test-set MSE that I observe is $\approx$ 0.23. Keep that in mind as you play with your model below: can you tune the model so as to get relatively close to this figure (or beat it) in a reasonable amount of time?

If you want to play around with the model by, e.g., changing dropout regularization to lasso or ridge regression, search the internet for how to do so. Why? It is what I would do…I'm not a `keras` package expert. (If you look at the documentation, the number of functions is *huge*.)

Also note that for many of you, a substantial portion of the time spent on this question might involve installing software, as indicated in the notes.

```
library(keras)

tensorflow::install_tensorflow()
```

```
##
## Installation complete.
```

```
x = model.matrix(sfr~.,data=df)
y = df$sfr

set.seed(100)

model = keras_model_sequential() %>%
  layer_dense(units=10,activation="relu",input_shape=ncol(x)) %>%
  layer_dropout(rate=0.3) %>%
  layer_dense(units=1)
```

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

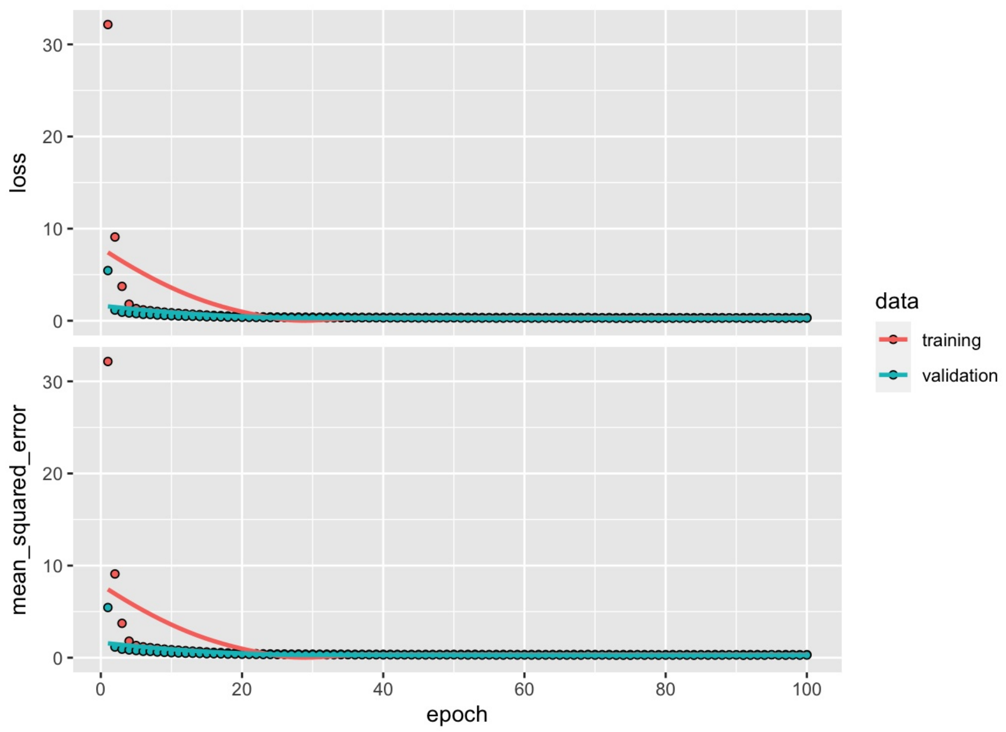
```
model %>% compile(loss="mse",optimizer=optimizer_rmsprop(),metrics=list("mean_squared_error"))

resp.pred = predict(model,x[s,])
mean((resp.pred-df[s,]$sfr)^2)
```

```
## [1] 59.40405
```

```
history = model %>%
  fit(x[-s,],y[-s],epochs=100,batch_size=128,validation_data=list(x[s,],y[s]),verbose=0)
plot(history)
```

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



```

```

Data, Part 2

And now we read in the contact binary dataset, also analyzed in the random forest lab.

```r
rm(list=ls())
file.path = "https://raw.githubusercontent.com/pefreeman/36-290/master/EXAMPLE_DATASETS/TD_CLASS/css_data.Rdata"
load(url(file.path))
rm(file.path)

# Eliminate the max.slope column (the 11th column), which has infinities.
predictors = predictors[,-11]

# Cut the CB and NON-CB class sizes to 5000 samples each.
set.seed(303)
w = which(response==1)
s = sample(length(w),5000)
predictors.cb = predictors[w[s],]
response.cb   = response[w[s]]
w = which(response!=1)
s = sample(length(w),5000)
predictors.noncb = predictors[w[s],]
response.noncb   = response[w[s]]
predictors = rbind(predictors.cb,predictors.noncb)
response   = c(response.cb,response.noncb)

w = which(response!=1)
response[w] = 0

df = data.frame(predictors,"class"=response)
s = sample(nrow(df),round(0.3*nrow(df)))
```

Question 2

Now code a classification model! At this point, this is easier said than done. Things to keep in mind:

1. Instead of `y = df$class`, you would do one-hot encoding: `y = to_categorical(df$class,num_classes=2)`.
2. Change the number of units in the last dense layer to `ncol(y)` and add an argument, either `activation="softmax"` or `activation="sigmoid"` (your choice).
3. In `compile()`, change the loss to `"categorical_crossentropy"`, and change the `metrics` to `"accuracy"`.
4. In `fit()`, change `y[-s]` to `y[-s,]`, etc., because `y` now has two columns.
5. The `predict()` function will now output a matrix of probabilities. As you've done many times, extract the second column to get Class 1 probabilities, then use `if` or `ifelse()` to map the Class 1 probabilities to predicted classes. (Assume a class-specification threshold of 0.5.)
6. Compute the test-set MCR, not the MSE.

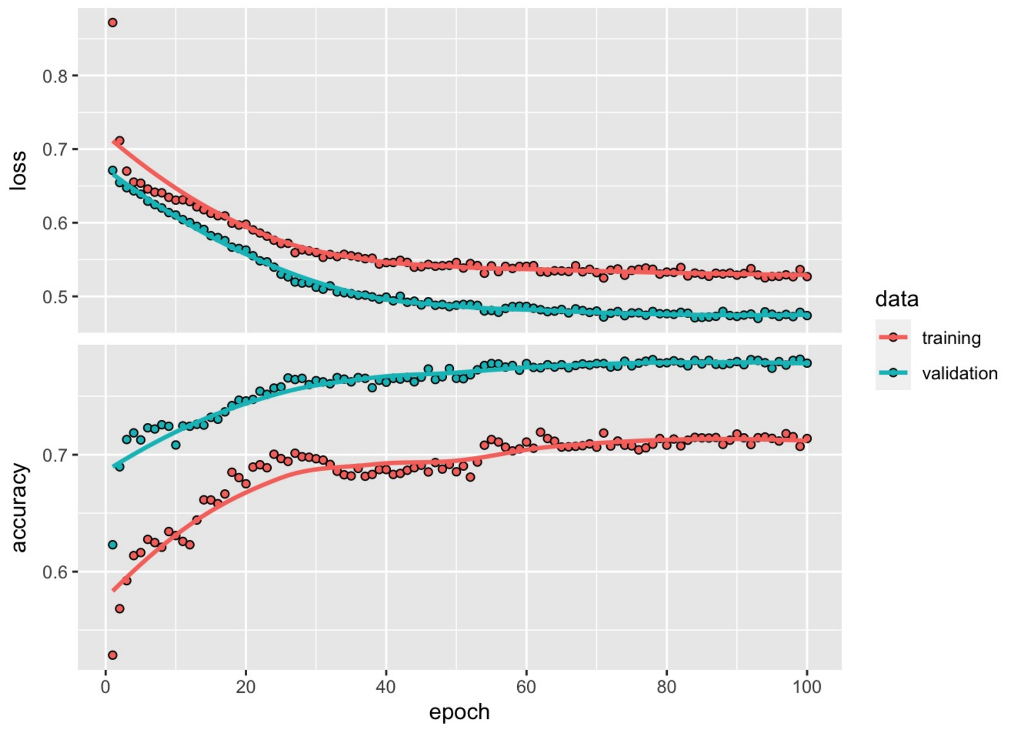
Note that my test-set MCR in the random forest lab was $\approx$ 0.2.

```r
x = model.matrix(class~.,data=df)
y = to_categorical(df$class,num_classes=2)

set.seed(100)
s = sample(nrow(df),0.3*nrow(df))
model = keras_model_sequential() %>%
  layer_dense(units=10,activation="relu",input_shape=ncol(x)) %>%
  layer_dropout(rate=0.3) %>%
  layer_dense(units=ncol(y),activation="softmax")
model %>% compile(loss="categorical_crossentropy",optimizer=optimizer_rmsprop(),metrics=list("accuracy"))

history = model %>%
  fit(x[-s,],y[-s,],epochs=100,batch_size=128,validation_data=list(x[s,],y[s,]),verbose=0)
plot(history)
```

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



```
resp.pred = predict(model,x[s,])[,2]
resp.prob = ifelse(resp.pred>0.5,1,0)

mean(resp.prob!=df[s,]$class)
```

```
## [1] 0.2216667
```