HW3_P13

IKM/KO/MC

6/4/2021

Code

```
# Setup ------------------------------------------------------------------

library(nnet)
library(dplyr)
library(ggplot2)

df_train = read.csv("~/Downloads/spam_stats315B_train.csv", col.names=paste0("X", 1:58))
df_test = read.csv("~/Downloads/spam_stats315B_test.csv", col.names=paste0("X", 1:58))

set.seed(2020)
NDEV = 1000
P = ncol(df_train) - 1; N = nrow(df_train)
slc = sample(1:nrow(df_train), NDEV)

# scale the data
scaler = scale(df_train[, 1:P])
df_train_sc = df_train; df_test_sc = df_test
df_train_sc[, 1:P] = scale(df_train_sc[, 1:P], center=attr(scaler, "scaled:center"), scale=attr(scaler,
df_test_sc[, 1:P] = scale(df_test_sc[, 1:P], center=attr(scaler, "scaled:center"), scale=attr(scaler, "

# apportion dev set(s)
df_tune_sc = df_train_sc[-slc, ]
df_dev_sc = df_train_sc[slc, ][1:(NDEV/2), ]
df_dev2_sc = df_train_sc[slc, ][(NDEV/2+1):NDEV, ]


# 13A --------------------------------------------------------------------

sapply(1:10, function(n_units){
  model = nnet.formula(X58 ~ ., data=df_train_sc, maxit=1e5, size=n_units, trace=F, rang=.5);
  yhat = round(predict(model, newdata=df_test_sc))
  acc = mean(yhat == df_test_sc$X58)
  acc
}) -> unit_tune

N_LAYERS = which.max(unit_tune)


# 13B --------------------------------------------------------------------
```

```r
sapply(seq(0, 1, .1), function(dec){
  acc_avg = c()
  for (i in 1:10){
    model = nnet.formula(X58 ~ ., data=df_train_sc, maxit=1e5, size=N_LAYERS, trace=F, decay=dec, rang=
    yhat = round(predict(model, newdata=df_test_sc))
    acc = mean(yhat == df_test_sc$X58)
    acc_avg = c(acc_avg, acc)
  }
  mean(acc_avg)
}) -> decay_tune

DECAY = seq(0, 1, .1)[which.max(decay_tune)]


# 13C ------------------------------------------------------------------------

# setup a full grid

grd = expand.grid(
  size=c(7, 10, 15),
  decay=seq(.05, .25, .05),
  skip=c(T)
)

lapply(1:nrow(grd), function(i){
  cat('Tune: ', i, '/', nrow(grd), '\n')
  tune_settings = grd[i, ];
  acc_good_avg = c()
  acc_avg = c()
  pi_matrix = list()
  for (j in 1:10){
    model = nnet.formula(X58 ~ .,
                         data=df_train_sc,
                         maxit=1e5,
                         size=tune_settings$size,
                         trace=F,
                         decay=tune_settings$decay,
                         skip=tune_settings$skip,
                         rang=.5
    );
    pi_matrix[[j]] = as.numeric(predict(model, newdata=df_test_sc))
    # yhat = round(predict(model, newdata=df_test_sc));
    # acc_good = mean(yhat[df_test_sc$X58 == 0] == 0)
    # acc = mean(yhat == df_test_sc$X58);
    # acc_avg = c(acc_avg, acc)
    # acc_good_avg = c(acc_good_avg, acc_good)
  }
  pi_matrix_ = do.call("rbind", pi_matrix) %>% colMeans()
  lapply(seq(.05, .95, .01), function(c){
    yhat = as.integer(pi_matrix_ >= c)
    acc_good = mean(yhat[df_test_sc$X58 == 0] == 0)
    acc = mean(yhat == df_test_sc$X58);
    c(acc, acc_good, c)
```

```
  }) %>%
    do.call("rbind", .) %>%
    data.frame() %>%
    `colnames<-`(c("acc", "acc_good", "cutoff")) -> cutoff_df
  cutoff_df %>%
    filter(acc_good >= .99) %>%
    arrange(acc) %>%
    tail(1) %>%
    mutate(i = i)

}) %>% do.call("rbind", .) -> tune_full
```

First, I want to emphasize that the hyperparameter procedure instructed by the problem differs from how I would estimate error. Whereas the problem instructs us to essentially mine the test set for the optimal settings (we're effectively letting the dev set double as the test set), the appropriate way, in my view, would be to tune these hyperparameters on a held-out dev set, and then make my estimates on a separate held out test set. Hence, the estimations of misclassification error here are likely generous, in light of the instructed validation procedure.

a.)

```
cat('Selected Number of Layers: ', N_LAYERS)
```

```
## Selected Number of Layers:  7
```

The procedure chooses 7 layers.

b.)

```
cat('Selected Decay (Avg. 10 runs): ', DECAY, "\n")
```

```
## Selected Decay (Avg. 10 runs):  0.2
```

```
cat('Corresonding Misclass estimate (Avg. 10 runs): ', 1 - max(decay_tune))
```

```
## Corresonding Misclass estimate (Avg. 10 runs):  0.04331376
```

Using the number of hidden layers as in a.), the procedure chooses a decay of .2. Misclassification is around .043.

c.)

The tuning grid is given by:

```
expand.grid(
  size=c(7, 10, 15),
  decay=seq(.05, .25, .05),
  skip=c(T)
)
```

```
##    size decay skip
## 1     7  0.05 TRUE
## 2    10  0.05 TRUE
```

```
## 3    15  0.05 TRUE
## 4     7  0.10 TRUE
## 5    10  0.10 TRUE
## 6    15  0.10 TRUE
## 7     7  0.15 TRUE
## 8    10  0.15 TRUE
## 9    15  0.15 TRUE
## 10    7  0.20 TRUE
## 11   10  0.20 TRUE
## 12   15  0.20 TRUE
## 13    7  0.25 TRUE
## 14   10  0.25 TRUE
## 15   15  0.25 TRUE
```

With the requirement that $< 1\%$ of good emails are misclassified, the selected settings are

```
best_tune_results = tune_full %>% arrange(desc(acc)) %>% head(1)
print(best_tune_results)
```

```
##          acc  acc_good cutoff i
## 20 0.9589041 0.9912664   0.79 6
```

```
print(grd[best_tune_results$i, ])
```

```
##   size decay skip
## 6   15   0.1 TRUE
```

The optimally chosen architecture under this constraint is a 15 unit model with decay .1 and a skip layer. It achieves the sufficient .9913 accuracy among good emails, and an overall .9589 accuracy on all emails – as averaged over the test set.