STAT7050: Assignment 3

Songze Yang, u7192786

2022-10-11

The SPAM data set has shown below. The data is divided into two parts based on the variables "testid". As various methods applied to our data, the standardization is utilized to result a better fit.

```
SPAM <- read_csv("E:/ANU Sem 3/STAT3050STAT4050STAT7050 - Advanced Statistical Learning - Sem 2 2022/A3/
```

```
## Rows: 4601 Columns: 59
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## dbl (57): make, address, all, 3d, our, over, remove, internet, order, mail, ...
## lgl  (2): spam, testid
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
preproc.param <- SPAM %>%
  preProcess(method = c("center", "scale"))


train_raw<-SPAM[which(SPAM$testid == 0),]
test_raw<-SPAM[which(SPAM$testid == 1),]

train <- preproc.param %>% predict(train_raw)
test <- preproc.param %>% predict(test_raw)
```

Question 1

Firstly, a neural network model is fitted followed by its train and test MSE.

```
n<-nrow(train)
x_train <- model.matrix (spam ~.-1, data = train[,-2])
y_train <- train$spam
y_tr <- to_categorical(y_train, 2)
```

```
## Loaded Tensorflow version 2.9.1
```

```
x_test <- model.matrix (spam ~.-1, data = test[,-2])
y_test <- test$spam
y_te <- to_categorical(y_test, 2)
```

The model is specified below.

```
set.seed (13)
model <- keras_model_sequential() %>%
  layer_dense(units = 256, activation = "relu", input_shape = ncol(x_train)) %>%
  layer_dropout(rate = 0.4) %>%
  layer_dense(units = 128, activation = "relu") %>%
  layer_dropout(rate = 0.3) %>%
  layer_dense(units = 2, activation = "softmax")

model %>% compile(
  loss = "binary_crossentropy",
  optimizer = "adam",
  metrics = c("accuracy")
)

model %>% fit(x_train, y_tr, epochs = 200, verbose = 0)
```

```
train.mse_nnet<-model %>% evaluate(x_train,  y_tr, verbose = 2)
nnet_train_error_rate<-1-train.mse_nnet[2]
nnet_train_error_rate
```

```
##    accuracy
## 0.004241407
```

```
test.mse_nnet<-model %>% evaluate(x_test,  y_te, verbose = 2)
nnet_test_error_rate<-1-test.mse_nnet[2]
nnet_test_error_rate
```

```
##   accuracy
## 0.05533856
```

The structure of the neural network is shown below.

```
nnet_train_var<-model$variables
summary(nnet_train_var)
```

```
##      Length Class             Mode
## [1,] 14592  tensorflow.tensor environment
## [2,]   256  tensorflow.tensor environment
## [3,] 32768  tensorflow.tensor environment
## [4,]   128  tensorflow.tensor environment
## [5,]   256  tensorflow.tensor environment
## [6,]     2  tensorflow.tensor environment
```

The weight variables of the neural network are shown below.

```
nnet_train_var
```

```
## [[1]]
## <tf.Variable 'dense_2/kernel:0' shape=(57, 256) dtype=float32, numpy=
```

```
## array([[ 0.11128589,  0.21910389, -0.17626987, ..., -0.18483128,
##          0.16595516,  0.01123055],
##        [-0.05417518, -0.03315337, -0.08862108, ..., -0.06326611,
##         -0.0710533 ,  0.23713058],
##        [ 0.05438044,  0.17446029,  0.20847727, ...,  0.04741945,
##          0.03817589,  0.21261206],
##        ...,
##        [-0.5205364 , -0.00980377,  0.1281394 , ...,  0.02449464,
##          0.06277702, -0.48352316],
##        [ 0.16682267,  0.15323696, -0.00391324, ..., -0.10111788,
##          0.2064759 , -0.35225102],
##        [-0.02288352,  0.13160393,  0.06224513, ..., -0.07306646,
##          0.16708666,  0.05863409]], dtype=float32)>
##
## [[2]]
## <tf.Variable 'dense_2/bias:0' shape=(256,) dtype=float32, numpy=
## array([-0.20019712, -0.19392747, -0.15285139, -0.17265666, -0.13725044,
##        -0.05511886, -0.06147473, -0.15445018, -0.14736997, -0.16502096,
##        -0.1503924 , -0.05701938, -0.13682644,  0.06032428, -0.04558419,
##        -0.05403459, -0.19629824, -0.17429386, -0.00328657, -0.22771527,
##        -0.13884152, -0.25478318, -0.08651102, -0.09643424, -0.08829464,
##        -0.25691822, -0.10503774, -0.10409251, -0.08836018, -0.05261109,
##        -0.14303125, -0.16465186, -0.11380793, -0.18368249, -0.07595981,
##        -0.11398473, -0.22346058, -0.09094662, -0.07805429, -0.1067689 ,
##        -0.16701601, -0.1111338 , -0.05640249, -0.11201821, -0.12155291,
##        -0.10377955, -0.23116542, -0.21467486, -0.05551519, -0.16830857,
##        -0.16031982, -0.1591078 , -0.08341433, -0.1064848 , -0.19747856,
##        -0.2632508 , -0.19583261, -0.24572591, -0.21368404,  0.1654126 ,
##        -0.20922963, -0.1769002 , -0.17923287, -0.17996979, -0.25357172,
##         0.00560284,  0.00664857, -0.0465028 , -0.00373282, -0.2131228 ,
##        -0.07085889, -0.13616243, -0.08954116, -0.08748348, -0.17826831,
##        -0.14711787, -0.04948953, -0.24096386, -0.11673927, -0.16944805,
##        -0.15994623, -0.11652395, -0.12198588, -0.21173127, -0.14915854,
##        -0.16023992, -0.09084339, -0.15706821, -0.18467012, -0.20613863,
##        -0.20210123, -0.10338674, -0.2297991 , -0.12037506, -0.16354878,
##        -0.15473926, -0.1823937 , -0.04861724, -0.03279863, -0.21091942,
##        -0.18803778, -0.14499865, -0.23817578, -0.27906972, -0.06914777,
##        -0.15604003, -0.15607965, -0.09779961, -0.12390517, -0.02206144,
##        -0.21985108, -0.1332619 , -0.16235182, -0.14521803, -0.07800347,
##        -0.0831535 , -0.10979801, -0.18339589, -0.19595225, -0.14401859,
##        -0.02870647, -0.28765994, -0.08480617, -0.21774969, -0.03322221,
##        -0.02660095, -0.09056216, -0.00802126, -0.07153853, -0.08333435,
##        -0.02909821, -0.10640047, -0.24812372, -0.15106107, -0.17051648,
##        -0.11530927, -0.02763698, -0.10064751, -0.06559504, -0.19471094,
##        -0.06691725, -0.07728336, -0.14200726, -0.25459823, -0.21417476,
##        -0.1040433 , -0.04876429, -0.06382058, -0.11990779, -0.15185057,
##        -0.10566663,  0.01020183, -0.1672076 , -0.11653228, -0.00317311,
##        -0.10325103, -0.0468669 , -0.21420138, -0.13016401, -0.03695211,
##        -0.05398255, -0.3017264 , -0.05539473, -0.21603623, -0.10962323,
##        -0.18031177, -0.14804663, -0.11015145, -0.16905588, -0.03783048,
##        -0.19419795, -0.13348863, -0.08483827, -0.16559966, -0.12880711,
##        -0.12643147, -0.21572682, -0.07971564, -0.30168387, -0.10958257,
##        -0.13619941, -0.13081093, -0.16027562, -0.33045194, -0.31033936,
##         0.01655337, -0.32725403, -0.06491533, -0.22605148, -0.06406229,
```

```
##        -0.16092728, -0.07365917, -0.2017964 , -0.2208273 , -0.0079677 ,
##        -0.24063572, -0.08727598, -0.07145774, -0.11713459, -0.24113438,
##        -0.06309927, -0.08845238, -0.09779859, -0.09896168,  0.01974527,
##        -0.10787525, -0.01263764, -0.05985105, -0.19833152, -0.03845068,
##        -0.05156108, -0.09702531, -0.19086432, -0.08292726, -0.03040706,
##        -0.10689259, -0.08106378, -0.23382182, -0.15772896, -0.12259933,
##        -0.03697505, -0.26274022, -0.07567446, -0.24134305, -0.15247239,
##        -0.24258891, -0.25418794, -0.12336093, -0.09534231, -0.21243773,
##        -0.12781277, -0.10499611, -0.15508926, -0.20960474, -0.10561212,
##        -0.22236028,  0.05203176, -0.05684914, -0.03036619, -0.07696126,
##        -0.11796622, -0.16216749, -0.0729284 , -0.18726018, -0.18598555,
##        -0.13339195, -0.19923234, -0.13083345, -0.1613439 , -0.08313423,
##        -0.10845497, -0.13361137, -0.00853475, -0.08834471, -0.11143468,
##         0.14445335], dtype=float32)>
##
## [[3]]
## <tf.Variable 'dense_1/kernel:0' shape=(256, 128) dtype=float32, numpy=
## array([[ 0.01206606,  0.00255125,  0.07685995, ...,  0.11304519,
##          0.02429586,  0.15979645],
##        [ 0.08287022,  0.08661482, -0.01374392, ..., -0.08608017,
##         -0.11509034,  0.16378212],
##        [-0.07557266,  0.11162257,  0.0366937 , ...,  0.04866334,
##          0.0317382 ,  0.06470314],
##        ...,
##        [-0.12018002,  0.05341503, -0.18273893, ..., -0.04398655,
##         -0.08645111,  0.0544485 ],
##        [-0.2371731 , -0.08887862,  0.02521941, ..., -0.2532629 ,
##          0.11243092, -0.31986916],
##        [ 0.10468944,  0.09276441, -0.07209782, ..., -0.03576062,
##          0.00527341,  0.06400017]], dtype=float32)>
##
## [[4]]
## <tf.Variable 'dense_1/bias:0' shape=(128,) dtype=float32, numpy=
## array([ 0.04034708,  0.29428065, -0.2960577 ,  0.18024617, -0.14335646,
##         0.2208719 ,  0.08949363,  0.2150105 ,  0.33066925,  0.15142453,
##         0.25582287,  0.18976857,  0.21060169,  0.05253075,  0.18191016,
##         0.20052414,  0.17768763,  0.15132047,  0.19062445,  0.19996065,
##         0.170777  ,  0.15421835, -0.01242512,  0.280193  ,  0.17829266,
##         0.04763686,  0.21699017,  0.17088027,  0.23932226,  0.21820739,
##         0.09032508, -0.05248412,  0.15315725,  0.1735491 ,  0.08108915,
##         0.30573496,  0.02492104,  0.09145691,  0.11071464,  0.10288052,
##         0.0958738 , -0.18649344,  0.26666117, -0.00379208,  0.15487386,
##         0.1495294 ,  0.21299936,  0.179542  ,  0.2376033 ,  0.11383663,
##         0.07054967, -0.00551012,  0.110356  ,  0.12040973, -0.19638717,
##         0.21997249,  0.13023765,  0.12096743,  0.05269625,  0.14529581,
##         0.09995156,  0.16175511, -0.00464328,  0.28275582,  0.19539574,
##         0.23334356,  0.07687224,  0.24956721, -0.02243799,  0.19394654,
##         0.11561003,  0.10908553,  0.19244997,  0.18775137,  0.08074734,
##        -0.21483506, -0.04550028,  0.2002575 ,  0.09013752,  0.1534542 ,
##         0.18132645, -0.01953831,  0.03237939,  0.04696153,  0.24588437,
##         0.22083627, -0.0445471 ,  0.26215875,  0.24269496,  0.14632724,
##         0.11261256,  0.13832948,  0.03184538,  0.22505146,  0.01006105,
##         0.11372288, -0.00074909,  0.2128342 ,  0.16904098,  0.16564713,
##         0.15064147,  0.27046916,  0.14251238,  0.13571385,  0.0993653 ,
```

```
##         0.17788471,  0.27812174,  0.1639461 ,  0.07486206,  0.08557497,
##         0.25668356, -0.02487177,  0.274076  ,  0.18554315,  0.16034147,
##         0.19913219,  0.2441991 ,  0.18435948,  0.17054515,  0.27289248,
##         0.09805833,  0.04602198,  0.09592944,  0.16255376, -0.11829536,
##         0.15291564,  0.22944653,  0.10663122], dtype=float32)>
##
## [[5]]
## <tf.Variable 'dense/kernel:0' shape=(128, 2) dtype=float32, numpy=
## array([[ 0.50643307, -0.46748975],
##        [-0.46933207,  0.47598082],
##        [ 0.20547284, -0.09286642],
##        [ 0.40338656, -0.45508698],
##        [-0.19898084,  0.25384992],
##        [ 0.5223088 , -0.5434589 ],
##        [ 0.27714407, -0.313756  ],
##        [-0.661915  ,  0.7216535 ],
##        [-0.6769277 ,  0.753483  ],
##        [-0.6066029 ,  0.619158  ],
##        [-0.42728326,  0.4780781 ],
##        [ 0.39041463, -0.37077546],
##        [-0.30693275,  0.25713864],
##        [ 0.41164666, -0.45587677],
##        [-0.49796242,  0.44296813],
##        [ 0.40967828, -0.40957606],
##        [-0.26523265,  0.2733027 ],
##        [ 0.3912642 , -0.378241  ],
##        [-0.10697162,  0.13869782],
##        [-0.37802565,  0.34758246],
##        [ 0.4785168 , -0.4431698 ],
##        [-0.3828857 ,  0.4128844 ],
##        [ 0.3029497 , -0.31454697],
##        [-0.4358315 ,  0.44331715],
##        [ 0.40635705, -0.30681655],
##        [-0.22574604,  0.25249353],
##        [-0.26002878,  0.26219887],
##        [ 0.5913746 , -0.50516105],
##        [-0.42162055,  0.44658202],
##        [ 0.5716836 , -0.5596791 ],
##        [ 0.39263174, -0.4486668 ],
##        [ 0.52962774, -0.42319763],
##        [ 0.23997392, -0.32047653],
##        [ 0.5191213 , -0.48985252],
##        [-0.18399273,  0.20149884],
##        [-0.6833621 ,  0.694443  ],
##        [ 0.11080191, -0.20845634],
##        [-0.36326903,  0.41100064],
##        [ 0.40113613, -0.38002786],
##        [ 0.27503988, -0.25956032],
##        [ 0.41112193, -0.39997467],
##        [ 0.32361034, -0.32043028],
##        [-0.39803806,  0.4279796 ],
##        [ 0.1374496 , -0.12423015],
##        [-0.32276323,  0.300566  ],
##        [ 0.17139363, -0.22582573],
```

```
##        [ 0.22375637, -0.22154772],
##        [-0.55978984,  0.5968415 ],
##        [-0.42154318,  0.4191304 ],
##        [ 0.13107023, -0.11819045],
##        [ 0.37964284, -0.3406043 ],
##        [ 0.27095214, -0.24242882],
##        [-0.23054615,  0.2749169 ],
##        [ 0.33828023, -0.29488054],
##        [ 0.18246406, -0.16818298],
##        [ 0.5060501 , -0.40245453],
##        [-0.43529812,  0.4568221 ],
##        [ 0.5310285 , -0.51356655],
##        [-0.192412  ,  0.01585156],
##        [ 0.28082016, -0.25754094],
##        [ 0.4350872 , -0.46969122],
##        [ 0.41800883, -0.3969663 ],
##        [ 0.28454432, -0.25640252],
##        [-0.5918685 ,  0.6249672 ],
##        [-0.18035896,  0.18004446],
##        [ 0.5763575 , -0.6474072 ],
##        [-0.12060057,  0.11514725],
##        [-0.43309963,  0.3629121 ],
##        [ 0.3762807 , -0.36858076],
##        [ 0.3963502 , -0.41437256],
##        [ 0.21715435, -0.17537074],
##        [ 0.2698077 , -0.15398867],
##        [-0.36696795,  0.34907126],
##        [-0.23879847,  0.21178931],
##        [-0.08888428,  0.09604006],
##        [-0.4923519 ,  0.4646614 ],
##        [-0.3291332 ,  0.33495376],
##        [-0.25758106,  0.24234664],
##        [ 0.07651398, -0.14694445],
##        [ 0.54695815, -0.5125773 ],
##        [-0.24120578,  0.24029481],
##        [ 0.32340002, -0.3873748 ],
##        [ 0.4192583 , -0.43323982],
##        [-0.14451514,  0.1695476 ],
##        [-0.48008263,  0.4469866 ],
##        [ 0.35229734, -0.3430462 ],
##        [ 0.25220868,  0.02056744],
##        [-0.5794302 ,  0.48355502],
##        [-0.3624172 ,  0.29038322],
##        [ 0.50344443, -0.46590468],
##        [-0.40369502,  0.27960405],
##        [ 0.364414  , -0.3286577 ],
##        [ 0.24419986, -0.2895429 ],
##        [-0.45123655,  0.46969303],
##        [ 0.35249305, -0.39757052],
##        [ 0.2789316 , -0.26492622],
##        [ 0.05966112, -0.21107331],
##        [ 0.391308  , -0.39805076],
##        [ 0.2677543 , -0.3217214 ],
##        [ 0.47139895, -0.5102115 ],
```

```
##        [-0.38644612,  0.39557746],
##        [-0.24494606,  0.27177474],
##        [ 0.36330667, -0.3762304 ],
##        [-0.5143993 ,  0.5710305 ],
##        [ 0.38576707, -0.38136926],
##        [-0.21137203,  0.24131398],
##        [-0.2780273 ,  0.32505792],
##        [ 0.3360325 , -0.38588208],
##        [ 0.37479872, -0.34544435],
##        [ 0.23461533, -0.25639075],
##        [-0.41714296,  0.40103802],
##        [ 0.22808   , -0.42047828],
##        [-0.7933626 ,  0.7189924 ],
##        [ 0.26028985, -0.24180324],
##        [ 0.31089363, -0.29393846],
##        [ 0.3274065 , -0.34758967],
##        [-0.5242242 ,  0.5552397 ],
##        [-0.2527964 ,  0.21877883],
##        [-0.70418906,  0.732314  ],
##        [-0.55383354,  0.5260442 ],
##        [ 0.30069396, -0.35422662],
##        [-0.22554582,  0.23078623],
##        [ 0.37173286, -0.3672718 ],
##        [ 0.43635857, -0.4164772 ],
##        [ 0.12970543,  0.17642055],
##        [ 0.49223602, -0.5234501 ],
##        [-0.34673953,  0.35139704],
##        [ 0.29420105, -0.33864895]], dtype=float32)>
##
## [[6]]
## <tf.Variable 'dense/bias:0' shape=(2,) dtype=float32, numpy=array([-0.08044273,  0.09050874], dtype=
```

Question 2

Alternatively, support vector machine is fitted.

```
set.seed(1)
tune.out=tune(svm, factor(y_train)~., data=data.frame(y_train,x_train), kernel="radial", ranges = list(
summary(tune.out)
```

```
bestmod=tune.out$best.model
summary(bestmod)
```

```
##
## Call:
## best.tune(method = svm, train.x = factor(y_train) ~ ., data = data.frame(y_train,
##     x_train), ranges = list(cost = c(5, 6, 7, 8, 9), gamma = c(0.01,
##     0.02, 0.03)), kernel = "radial")
##
##
## Parameters:
##    SVM-Type:  C-classification
```

```
##  SVM-Kernel:  radial
##        cost:  9
##
## Number of Support Vectors:  700
##
##  ( 336 364 )
##
##
## Number of Classes:  2
##
## Levels:
##  FALSE TRUE
```

The train and test MSE of SVM is similar to neural network.

```
svm_train_ypred=predict(bestmod, data.frame(y_train,x_train))
svm_train_error<-mean(svm_train_ypred != y_train)
svm_test_ypred=predict(bestmod, data.frame(y_test,x_test))
svm_test_error<-mean(svm_test_ypred != y_test)
svm_train_error
```

```
## [1] 0.03719413
```

```
svm_test_error
```

```
## [1] 0.06445312
```

Question 3

The flexible discriminant analysis is shown below with its train and test MSE.

```
fda_model <- fda(spam~., data = train[,-2], method = mars)
# Make predictions on train
fda_pred_train <- fda_model %>% predict(train[,-2])
# Model accuracy on train
fda_train_error<-mean(fda_pred_train != train$spam)
# Make predictions on test
fda_pred_test <- fda_model %>% predict(test[,-2])
# Model accuracy on test
fda_test_error<-mean(fda_pred_test != test$spam)
fda_train_error
```

```
## [1] 0.05709625
```

```
fda_test_error
```

```
## [1] 0.06901042
```

Question 4

The penalized discriminant analysis is shown below followed by its train and test MSE.

```
pda_y_train<-as.numeric(y_train)+1
pda_y_test<-as.numeric(y_test)+1
cv.out <-PenalizedLDA.cv(x_train,pda_y_train,lambdas=c(1e-4,1e-3,1e-2,.1,1),lambda2=.3)
cv.out$bestK
pda_out <- PenalizedLDA(x_train,pda_y_train,xte = x_test ,type="ordered",lambda=cv.out$bestlambda,K=cv.
pda_train = predict.penlda(pda_out, xte = x_train)
pda_train_error<-mean(unlist(pda_train)!=pda_y_train)
pda_test_error<-mean(pda_out$ypred!=pda_y_test)
```

```
pda_train_error
```

```
## [1] 0.1278956
```

```
pda_test_error
```

```
## [1] 0.1328125
```

Question 5

The information relates to the test result is shown below.

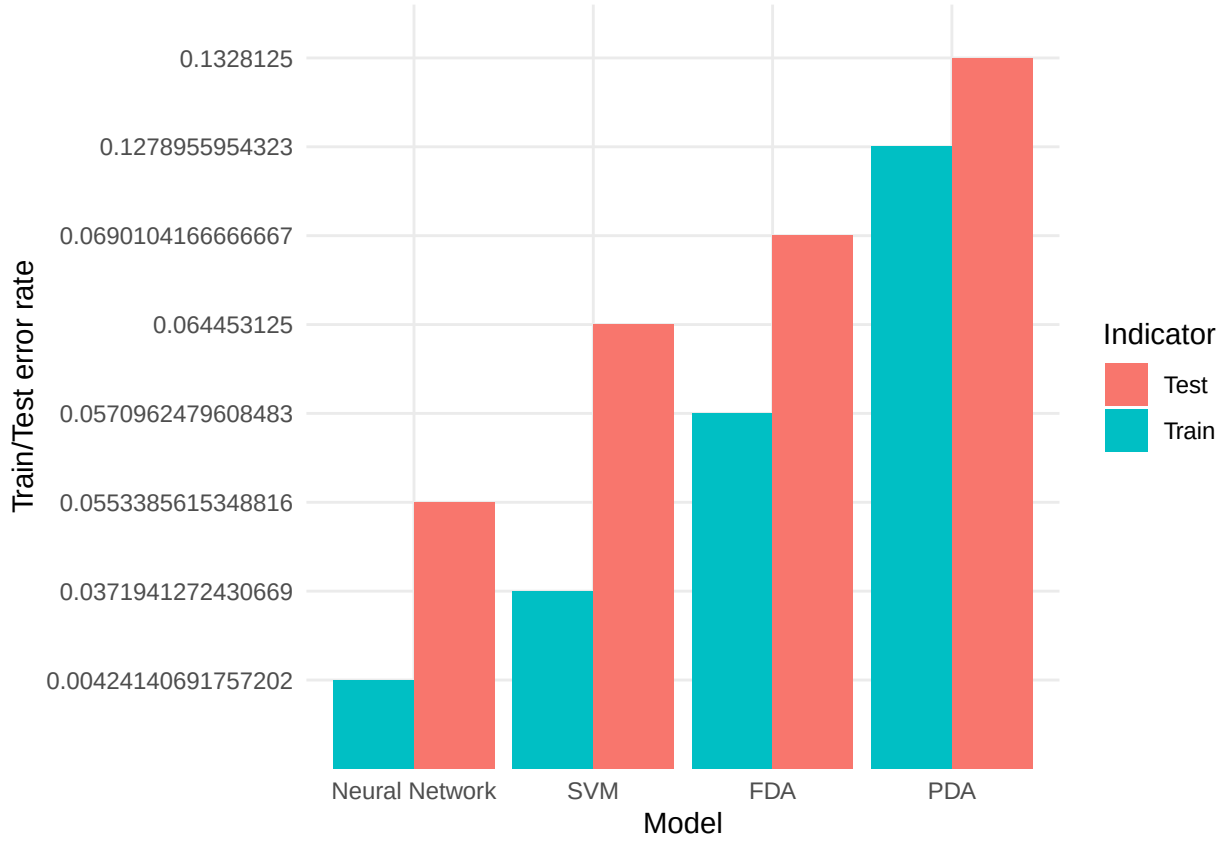
```
tes.res<-matrix(1:8,4,2)
tes.res[,1]<-c(nnet_train_error_rate,svm_train_error,fda_train_error,pda_train_error)
tes.res[,2]<-c(nnet_test_error_rate,svm_test_error,fda_test_error,pda_test_error)
df<-as.data.frame(tes.res)
colnames(df)<-c("Train error rate","Test error rate")
rownames(df)<-c("Neural Network","SVM","FDA","PDA")
df
```

```
##                Train error rate Test error rate
## Neural Network      0.004241407      0.05533856
## SVM                 0.037194127      0.06445312
## FDA                 0.057096248      0.06901042
## PDA                 0.127895595      0.13281250
```

Visualization is presented as follow.

```
modname<-c("Neural Network","SVM","FDA","PDA")
plot_df<-as.data.frame(t(rbind(c(modname,modname),c(df$`Train error rate`,df$`Test error rate`),c(rep("
colnames(plot_df)<-c("Models","Train/Test error rate","Indicator")

ggplot(data=plot_df, aes(x = factor(Models,levels =  modname), y=`Train/Test error rate`, fill=Indicato
  geom_bar(stat="identity", position=position_dodge())+
  xlab('Model')+
  theme_minimal()
```



First of all, all of the methods above are trying to find/approximate the true Bayesian posterior probability of spam given a bag of words. Therefore, the task is to find the true probability function.

Neural network has taken the crown regrading the performance. The two hidden layers neural is known for its universal approximator property. The single perceptron approximates any linear function, while the two layers neural network can approximate any function of any class. In the SPAM case, the boundary for separation is not linear, which can be seen from the performance of FDA. The neural network performs a great approximation to this non-linear boundary.

The support vector machine takes the second. It try to approximate the function by a kernel mapping (non-linear mapping). The kernel method in our context is the radial kernel. This kernel is very good in terms of non-linear mapping, however, in analogous to neural network, after a single projection, the class of spam is not necessary to be separable. Therefore, after tuning for the tolerance for error in the non-separable class, the performance is not better than the neural network.

This is not true for neural network. The 2-layers neural work can be written as:

$$P(spam = K|X = x) = O(g(l(x)))$$

The O() function is the output layers. The two layers of non-linear projection approximates the unknown function better in our case. Also, if the sigmoid kernel is selected, the performance of the support vector machine may be more similar to the neural network with sigmoid as activation function.

The basis expansion is also considered in the Flexible DA. Multivariate adaptive regression spline is used specifically. The assumption on the shape of the non-linear boundary adds additional information to our model. The true boundary may not be mars function. The assumption on a class of the function may lead to a exponential increase of the test error, which can be seen in our case.

The basis expansion in DA with penalty forms PDA. The penalized DA shrinks the coefficient in the basis expansion based on a l-2 penalty. The results show that the discriminant analysis in our case may be underfitting because the unpenalized version yields a better result (FDA), yet this is not ture in penalized cased.