Midterm Exam, MATH 868

James DG

Fall 2022

Part (I) 70 points

If you can achieve an average validation MAD for the last 5 epochs less than $12000, you get 100 for the exam. You do not need do part (II)

Continue to play with the Boston Housing data by Kaggple.

1. Implement a deep learning model by Keras: Use a different architecture; it should be different from the architecture used in class examples or your homework 6.
2. Include an early stopping control in your training process with patience of 2; use a validation split of 0.2 and batch size of 32; report the mean validation MAE for the last 5 epochs of your training process.

```
train_data = read.csv('kaggle_house_pred_train.csv', header=T, stringsAsFactors=T)


# remove the first column, the id
train_data = train_data[,-1]
y_train =train_data[,80]
train_data = train_data[,-80]
```

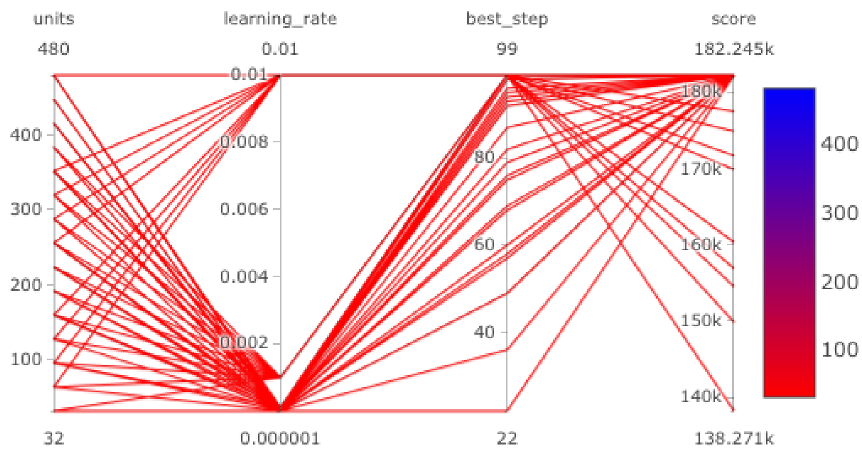
Discard columns that are mostly null values

```
train_data=train_data[-c(6,57,72,73,74,3)]
```

Preprocess with recipes

```
train_rec <- recipe( ~ ., data = train_data) %>%
             step_impute_knn(all_numeric_predictors()) %>%
             step_center(all_numeric_predictors()) %>%
             step_scale(all_numeric_predictors()) %>%
                        step_unknown(all_nominal_predictors(), new_level = "unknown") %>%
                        step_dummy(all_nominal_predictors(), one_hot=T, keep_original_cols =F)

train_rec <- prep(train_rec, training = train_data)
recipe_processed_train_data <- bake(train_rec, new_data = train_data)
recipe_processed_train_data = as.matrix(recipe_processed_train_data)
```

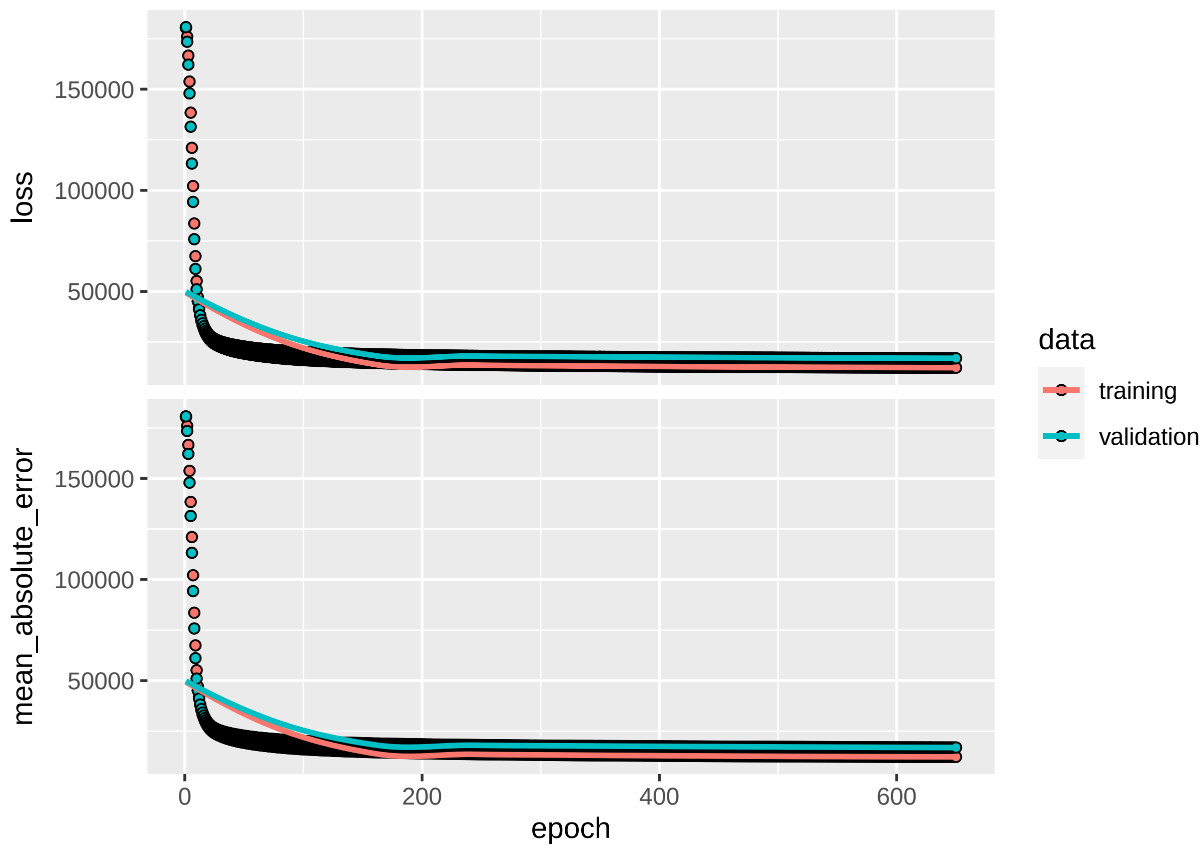
Using kerasTuneR, I found an optimal configuration of learning rate and units for my NN and the Adadgrad optimizer




```
model <- keras_model_sequential()
```

```
## Loaded Tensorflow version 2.9.2
```

```
model %>%
    layer_dense(units = 480, activation = "relu",
                input_shape= dim(recipe_processed_train_data)[[2]]) %>%
    layer_dense(units = 64, activation = "relu") %>%
    layer_dense(units = 1)


  model %>%
    compile(
      loss = "mae",
      optimizer = optimizer_adagrad(learning_rate= 1e-02, epsilon=1e-07),
      metrics =list("mean_absolute_error")
    )


history <- model %>% fit(
  as.matrix(recipe_processed_train_data),
  as.matrix(y_train),
    batch_size = 32,
  validation_split = 0.2,
    callbacks = callback_early_stopping(patience = 10, monitor = 'mean_absolute_error'),
  epochs = 650
)



plot(history)
```




```
cat('Last 5 validation MAE values:',
    tail(history[["metrics"]][["val_mean_absolute_error"]]),5)
```

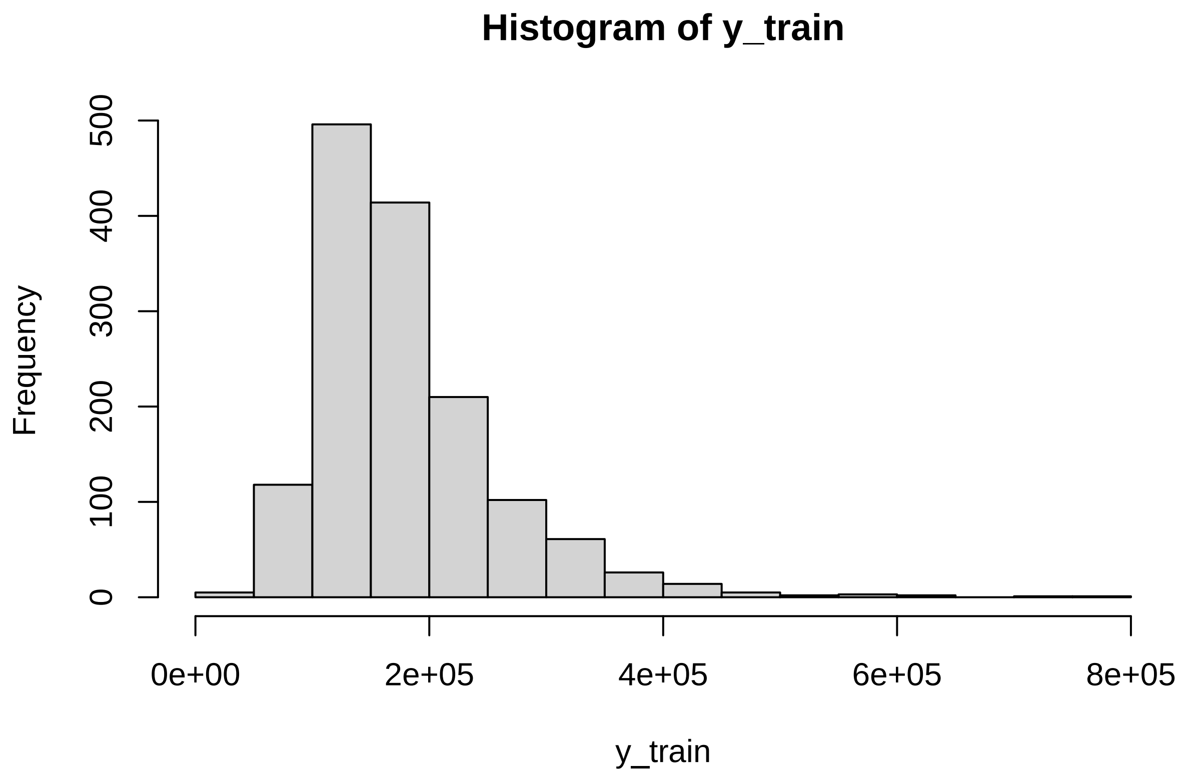
```
## Last 5 validation MAE values: 16940.18 16950.48 16957.67 16944.54 16934.94 16947.5 5
```

```
cat('Mean of last 5 validation MAE values:',
    mean(tail(history[["metrics"]][["val_mean_absolute_error"]]),5))
```

```
## Mean of last 5 validation MAE values: 16946.02
```

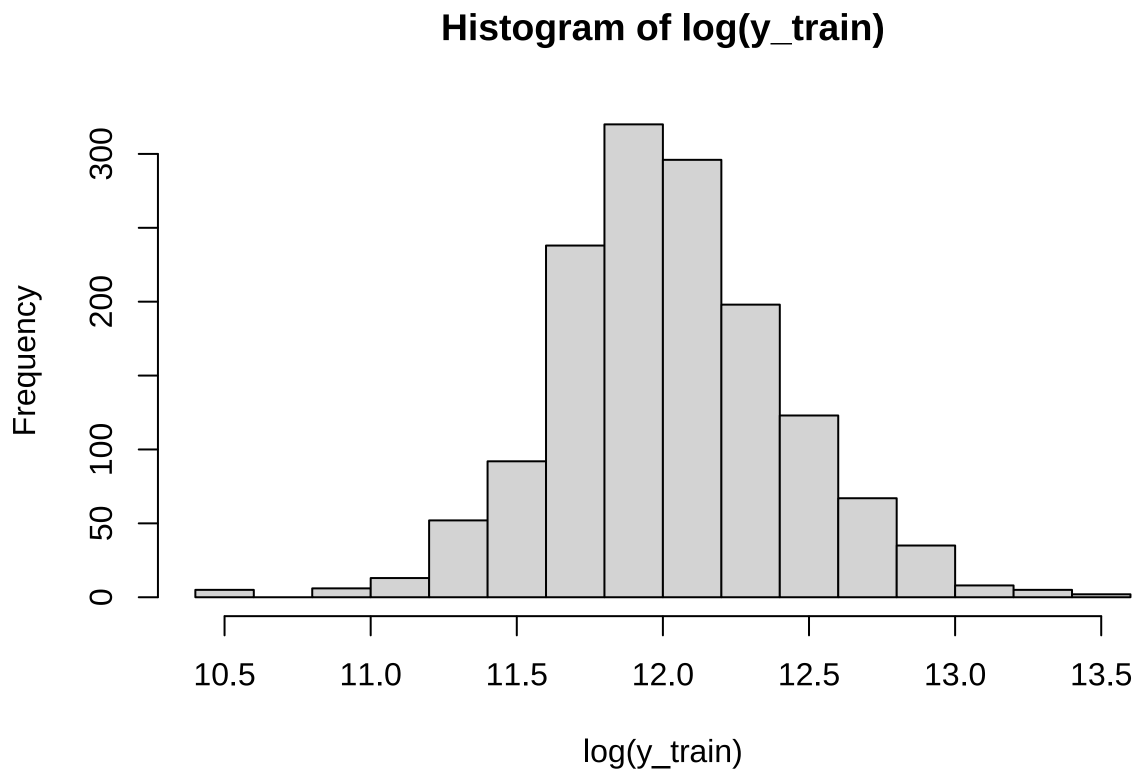
Since we are unable to achieve under $12000 for mae_val score, let's look at the target variable.

```
hist(y_train)
```



This looks skewed, what about a logarithmic transformation?

```
hist(log(y_train))
```



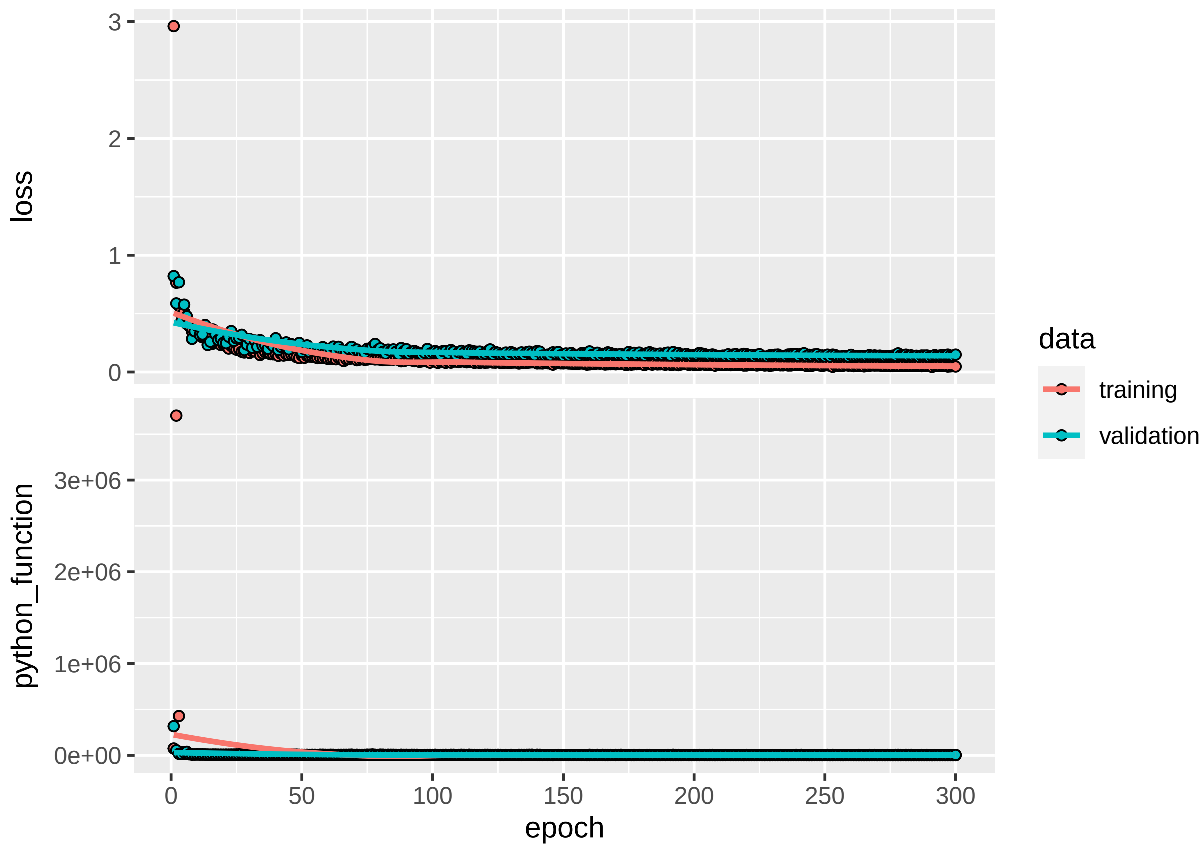
Since our MAE is currently $\frac{1}{n}\sum_{i=1}^{n}|ln(y)-\hat{y}|$, then we need to redefine our own metric $\frac{1}{n}\sum_{i=1}^{n}|e^{ln(y)}-e^{\hat{y}}| = \frac{1}{n}\sum_{i=1}^{n}|y-e^{\hat{y}}|$

```
my_mae <- function(y_true, y_pred){
    K        <- backend()
    # calculate the metric
    mae <- ((K$abs(exp(y_true)-exp(y_pred))))/35 # Based on chosen batch size and val.split
    val_mae <-(K$abs(exp(y_true)-exp(y_pred)))/7
}
```

```
model <- keras_model_sequential()
model %>%
    layer_dense(units = 480, activation = "relu", input_shape= dim(recipe_processed_train_data)[[2]]) %>%
    layer_dense(units = 64, activation = "relu") %>%
    layer_dense(units = 1)


  model %>%
    compile(
      loss = "mean_absolute_error",
      optimizer = optimizer_adagrad(learning_rate= 1e-02, epsilon=1e-07),
      metrics = my_mae
    )
```

```
history <- model %>% fit(
  as.matrix(recipe_processed_train_data),
  as.matrix(log(y_train)),
    batch_size = 35,
  validation_split = 0.2,
    epochs = 300
)
```

```
plot(history)
```




```
cat('Mean of last 5 validation MAE values:',
    mean(tail(history[["metrics"]][["val_python_function"]]),5))
```

```
## Mean of last 5 validation MAE values: 3996.089
```

Part (II) 30 points

The dataset contains 68 predictor variables and 20k records. The data was split into 2 parts:

- 10k records for training.

- 10k records for testing.

3. Clean and process your data. Fit a binary classifier to predict if a customer takes an offer (by the PURCHASE indicator, binary cross entropy loss)

```
# This data was not uniform between the training and testing set and so I
# needed to specifically find the purchase and unique ID columns for both sets
train_data = readRDS('train.rda')
dim(train_data)
```

```
## [1] 10000    69
```

```
test_data = readRDS('valid.rda')
which(colnames(train_data)=='PURCHASE')
```

```
## [1] 68
```

```
which(colnames(train_data)=='UNIQUE_ID')
```

```
## [1] 69
```

```
which(colnames(test_data)=='PURCHASE')
```

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

```
which(colnames(test_data)=='UNIQUE_ID')
```

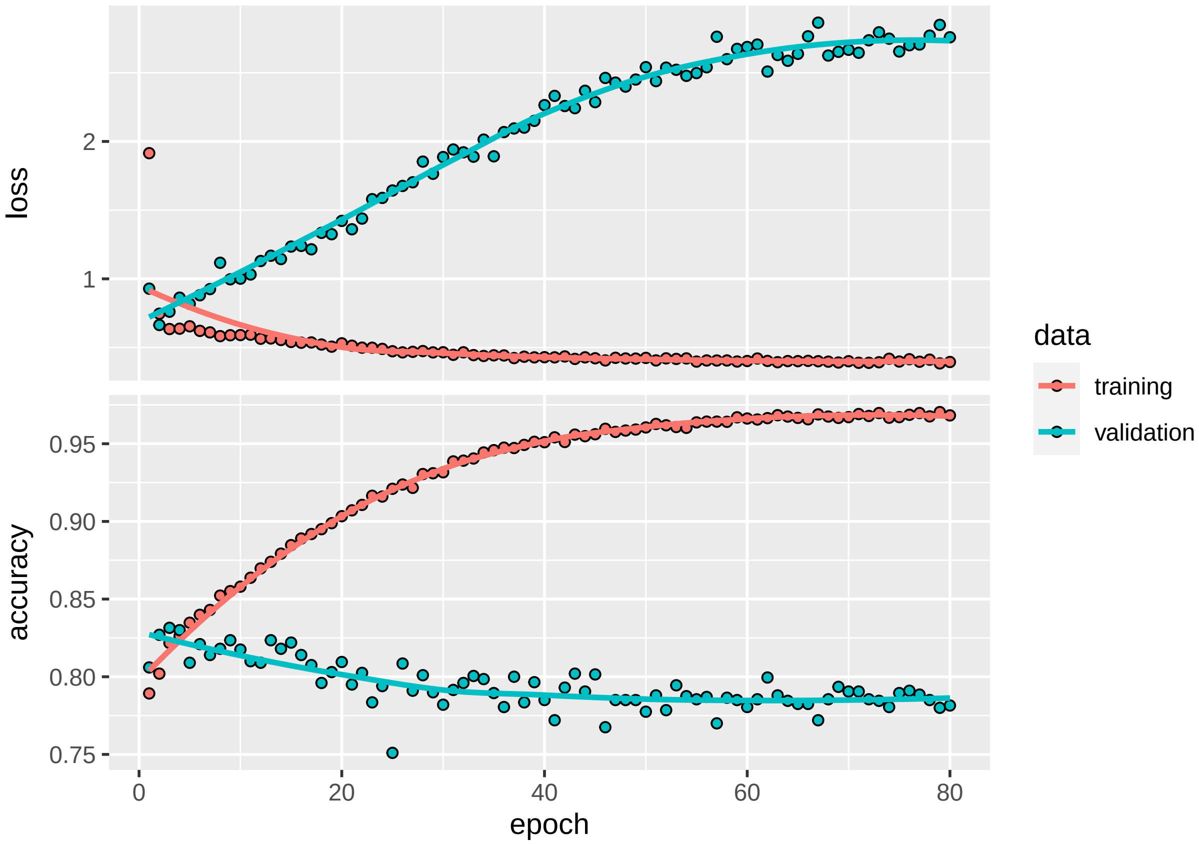
```
## [1] 58
```

```
y_train = train_data[,68]
y_test = test_data[,1]
train_data=train_data[-c(68,69)]
test_data =test_data[-c(58,1)]
```

Preprocess with recipes

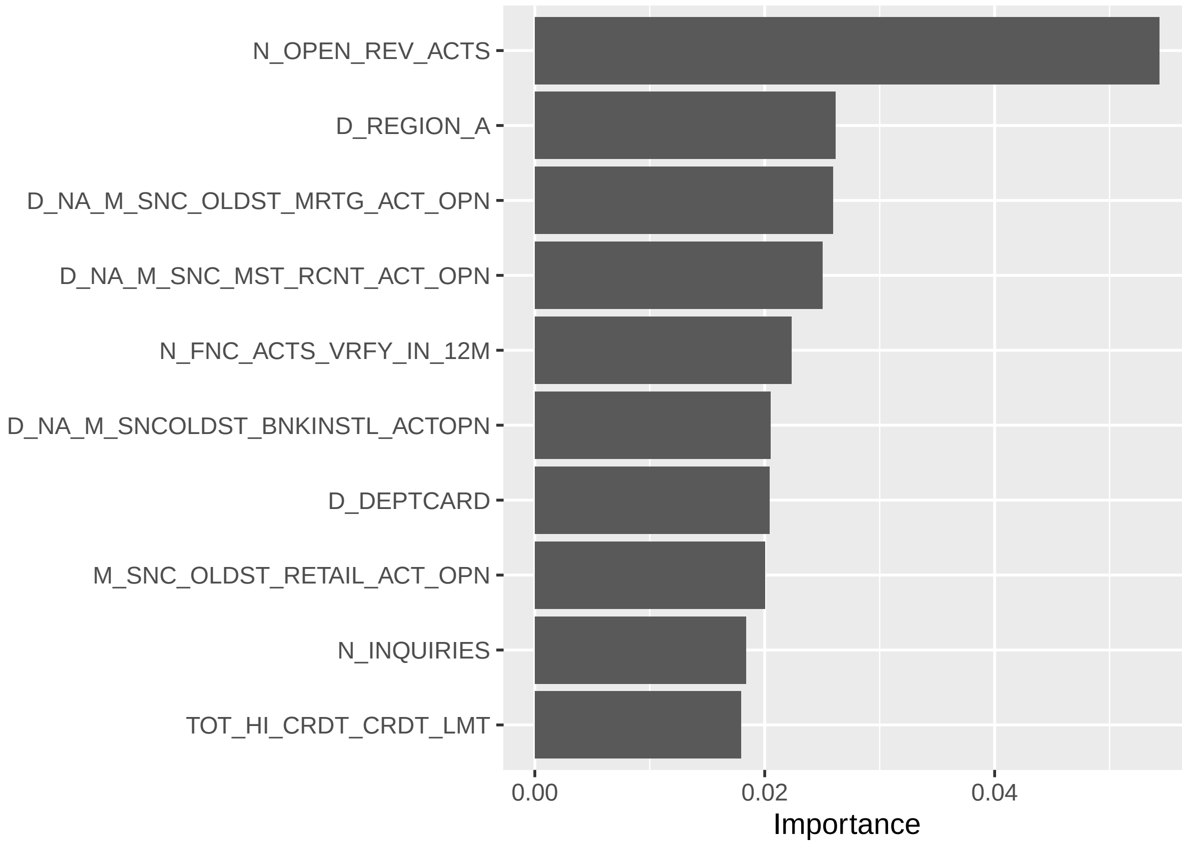
```
train_rec <- recipe(~ ., data = train_data) %>%
    step_center(all_numeric()) %>%
  step_scale(all_numeric())


train_rec <- prep(train_rec, training = train_data)
recipe_processed_train_data <- bake(train_rec, new_data = train_data)
recipe_processed_train_data = as.matrix(recipe_processed_train_data)
```

```
build_model <- function() {
  model <- keras_model_sequential() %>%
    layer_dense(units = 64, activation = "relu",
                input_shape = dim(recipe_processed_train_data)[[2]]) %>%
    layer_dense(units = 32, activation = "relu") %>%
    layer_dense(units = 16, activation = "relu") %>%
    layer_dense(units = 1)

    model %>% compile(
  optimizer = optimizer_rmsprop(learning_rate = 0.001,
                               rho = 0.9, epsilon = 1e-6, decay = 0,),
      loss = "binary_crossentropy",
    metrics = c("accuracy")
  )

}

model <- build_model()

history <- model %>% fit(recipe_processed_train_data,
                         y_train,
                         epochs = 80,
                         batch_size = 32,
                         verbose = 2,
                         callbacks = callback_early_stopping(patience = 10,
                                                            monitor = 'accuracy'),
                         validation_split=.2)
plot(history)
```



4. Use vip to find the most important ten predictors.

```
pred_wrapper <- function(object, newdata) {
  predict(object, x = as.matrix(newdata)) %>%
    as.vector()
}

set.seed(102)  # for reproducibility
p1 <- vip(
  object = model,                          # fitted model
  method = "permute",                      # request permutation-based VI scores
  num_features = 10,        # default only plots top 10 features
  pred_wrapper = pred_wrapper,             # user-defined prediction function
  target = y_train,              # name of the target variable column
  metric = "rsquared",                     # evaluation metric
  train = as.data.frame(recipe_processed_train_data),      # training data
)


print(p1)  # display plot
```




```
p1$data
```

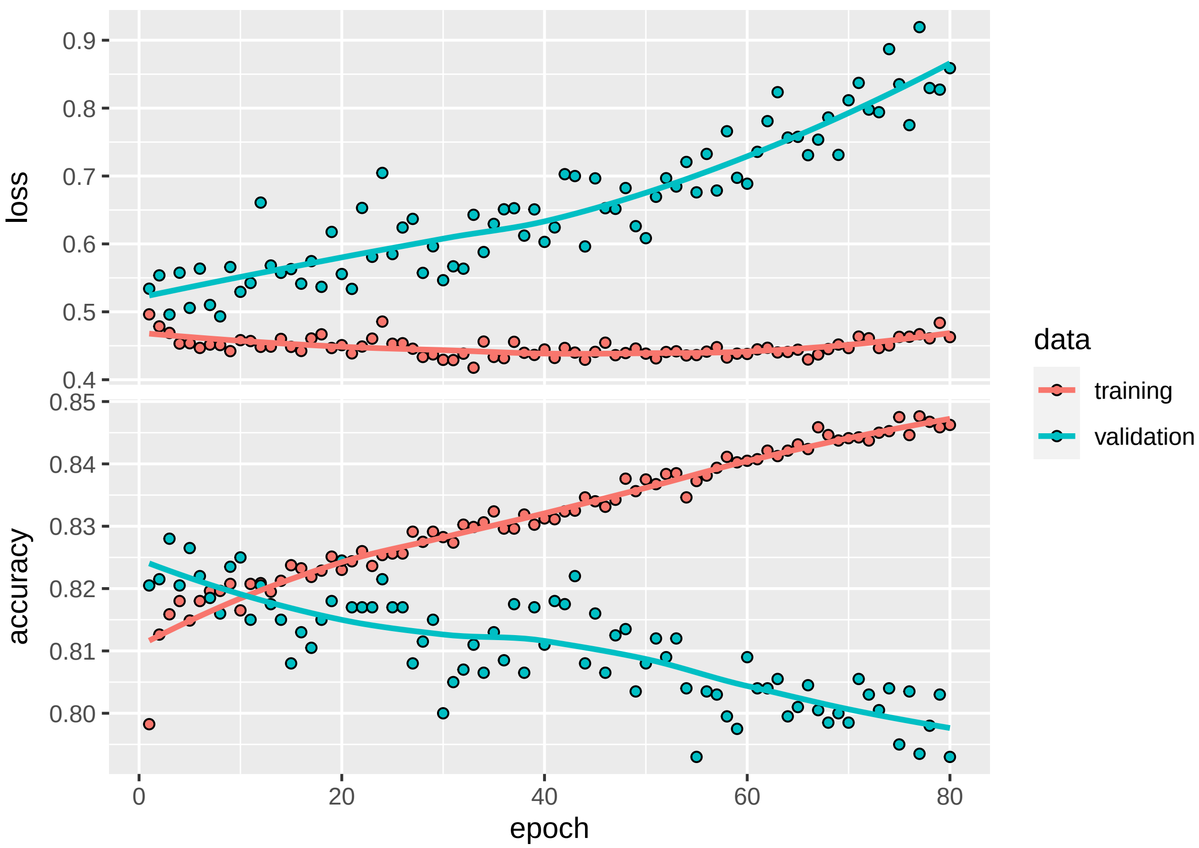
```
## # A tibble: 10 x 2
##    Variable                        Importance
##    <chr>                                <dbl>
##  1 N_OPEN_REV_ACTS                     0.0543
##  2 D_REGION_A                          0.0262
##  3 D_NA_M_SNC_OLDST_MRTG_ACT_OPN       0.0259
##  4 D_NA_M_SNC_MST_RCNT_ACT_OPN         0.0250
##  5 N_FNC_ACTS_VRFY_IN_12M              0.0223
##  6 D_NA_M_SNCOLDST_BNKINSTL_ACTOPN     0.0205
##  7 D_DEPTCARD                          0.0204
##  8 M_SNC_OLDST_RETAIL_ACT_OPN          0.0200
##  9 N_INQUIRIES                         0.0184
## 10 TOT_HI_CRDT_CRDT_LMT                0.0179
```

```
as.vector(p1$data[,1])
```

```
## # A tibble: 10 x 1
##    Variable
##    <chr>
##  1 N_OPEN_REV_ACTS
##  2 D_REGION_A
##  3 D_NA_M_SNC_OLDST_MRTG_ACT_OPN
```

```
##   4 D_NA_M_SNC_MST_RCNT_ACT_OPN
##   5 N_FNC_ACTS_VRFY_IN_12M
##   6 D_NA_M_SNCOLDST_BNKINSTL_ACTOPN
##   7 D_DEPTCARD
##   8 M_SNC_OLDST_RETAIL_ACT_OPN
##   9 N_INQUIRIES
## 10 TOT_HI_CRDT_CRDT_LMT
```

```
train_data10 <- subset(train_data, select=unlist(p1$data[,1]))

train_rec10 <- recipe( ~ ., data = train_data10) %>%
             step_impute_knn(all_numeric()) %>%
             step_center(all_numeric()) %>%
             step_scale(all_numeric())

train_rec10 <- prep(train_rec10, training = train_data10)
recipe_processed_train_data10 <- bake(train_rec10, new_data = train_data10)


recipe_processed_train_data10 = as.matrix(recipe_processed_train_data10)



build_model10 <- function() {
  model <- keras_model_sequential() %>%
    layer_dense(units = 64, activation = "relu",
                input_shape = dim(recipe_processed_train_data10)[[2]]) %>%
    layer_dense(units = 32, activation = "relu") %>%
    layer_dense(units = 16, activation = "relu") %>%
    layer_dense(units = 1)

    model %>% compile(
  optimizer = optimizer_rmsprop(learning_rate = 0.001,
                               rho = 0.9, epsilon = 1e-6, decay = 0,),
      loss = "binary_crossentropy",
    metrics = c("accuracy")
  )
}

model <- build_model10()
history <- model %>% fit(recipe_processed_train_data10,
                         y_train,
                         epochs = 80,
                         batch_size = 32,
                         verbose = 2,
                         callbacks = callback_early_stopping(patience = 10,
                                                              monitor = 'accuracy'),
                         validation_split=.2)
plot(history)
```



5. Find the accuracy for testing data (an acceptable model should generate an accuracy rate of around 80% for the testing data).

```
#Preprocess test_data like train_data
test_data10 <- subset(test_data, select=unlist(p1$data[,1]))
test_rec <- recipe(~ ., data = test_data10) %>%
    step_center(all_numeric()) %>%
  step_scale(all_numeric())

test_rec <- prep(test_rec, training = test_data10)

recipe_processed_test_data <- bake(test_rec, new_data = test_data10)
recipe_processed_test_data = as.matrix(recipe_processed_test_data)

score <- model %>% evaluate(recipe_processed_test_data, y_test, verbose = 0)
cat('Test accuracy:', score["accuracy"]*100, "% \n")
```

```
## Test accuracy: 79.93 %
```