Machine Learning with R

Mamta Mandan

5/3/2022

Introduction

What

For my final project I am exploring Machine Learning libraries for R. One that I have used before with Python is TensorFlow. Another option that I found in my research is called caret. Below you will find a run-through of an example of regression machine learning model done using the Keras add-on for TensorFlow. Going through each step of the process, I have explained what is happening and why it is important.

Why

It is my goal to become a Machine Learning Specialist as I continue down my Data Science career. As such, I feel it's best to be able to teach yourself a concept and be able demonstrate my understanding by showing others how to do the same. Starting at the baseline with an example provided in the documentation of a machine learning library will allow me to not only to be able to code the model, but also be able to process how it works. This will help me in the future when approaching various datasets to be able to identify which model/algorithm might be best to use for the inclined result.

How

While reading the documentation on TensorFlow and researching the caret library I found a few examples showing simple regression models. I felt it was best for me to follow what was shown, run each code block, and then explain the purpose behind that step. I chose stick to TensorFlow due to my familiarity. I also researched machine learning as broader topic as I feel it's a phrase that is brought up often, but few comprehend what it truly is. I paraphrased what I found to be easy to digest and given an example of each type of machine learning so one can quickly identify where they might come into play. This should help set the foundation for some of the concepts seen in the regression example walk-through.

Body

Machine Learning Research

Machine Learning is offset of Artificial Intelligence where models based on algorithms are create for the purpose of predicting future outcomes. It is able to do this learning from historical data, often referred to as training. There are 4 different types of machine learning: **Supervised**, **Unsupervised**, **Semi-Supervised**, and **Reinforcement**.

Supervised learning is when algorithms are trained on both input and desired output data. This training data must be labeled and the algorithm is provided the variables on which to determine correlation. This branch of machine learning is primarily used for classification and regression modeling. Think your email inbox, does it automatically filter for spam or junk emails? This is an example of supervised learning using classification.

Unsupervised learning only takes input data – most likely this data is unstructured, meaning it is not labeled or classified. The algorithms train on this data to create structure, this is generally done through clustering and/or finding anomalies. The main concept behind this type of modeling is something we learning in class: probability density function. Biologists may use this type of machine to analyze DNA clusters while studying evolution.

Semi-Supervised learning intuitively combines aspects of supervised and unsupervised learning. The algorithm is supplied with a labeled dataset to train on allowing it to use the model on a new, unlabeled dataset and apply the dimensions it learning. It has been found that this combo of using labeled and unlabled data creates the most accuracy. An example of when semi-supervised learning might be use is for fraud detection or language translation.

Reinforcement learning is where the algorithm is programmed to have a goal with positive consequences for achieving it or negative for not. Game theory implemented with code. I believe this where AI really comes into play, as this the type of machine learning used for robotics or video games. It's the computer chess player against you or the technology likely fueling Telsas in their driver-less functions.

Machine Learning in Action

Now that we have a base understand of machine learning, let's dive into an example. I will be running through a regression model using TensorFlow library and it's accompanying datasets. Through each step, I will explain what is going in that code block and why it is being done. Code is copied directly from https://tensorflow.rstudio.com/tutorials/beginners/basic-ml/tutorial_basic_regression/ with some additions of my own or slight changes.

For those who may want to follow along, I'll begin by showing the libraries needed to be installed:

```
install.packages("tensorflow")
install_tensorflow()
install.packages("tfdatasets")
install.packages("keras")
install_keras()
```

Now that we have the libraries installed, let's do a quick check to make sure:

```
library(tensorflow)
library(tfdatasets)
library(keras)
# The following libraries were not explicitly installed above as most likely you
#already have it from when we installed tidyverse in class
library(dplyr)
library(ggplot2)
```

Alright, Machine Learning, here we come!

```
# This dataset is available in the keras/tfdatasets packages:
boston_housing <- dataset_boston_housing()
```

```
## Loaded Tensorflow version 2.8.0
```

```
#boston_housing
# Wanted to show the data, but it is too big and takes up too much space in the
# pdf file
# Adjusting to show what's in this dataframe
dimnames.data.frame(boston_housing)
```

```
## [[1]]
## NULL
##
## [[2]]
## [1] "train" "test"
```

```
names(boston_housing$train)
```

```
## [1] "x" "y"
```

```
names(boston_housing$test)
```

```
## [1] "x" "y"
```

```
head(boston_housing$train$x)
```

```
##         [,1] [,2]  [,3] [,4]  [,5]  [,6]  [,7]   [,8] [,9] [,10] [,11]  [,12]
## [1,] 1.23247  0.0  8.14    0 0.538 6.142  91.7 3.9769    4   307  21.0 396.90
## [2,] 0.02177 82.5  2.03    0 0.415 7.610  15.7 6.2700    2   348  14.7 395.38
## [3,] 4.89822  0.0 18.10    0 0.631 4.970 100.0 1.3325   24   666  20.2 375.52
## [4,] 0.03961  0.0  5.19    0 0.515 6.037  34.5 5.9853    5   224  20.2 396.90
## [5,] 3.69311  0.0 18.10    0 0.713 6.376  88.4 2.5671   24   666  20.2 391.43
## [6,] 0.28392  0.0  7.38    0 0.493 5.708  74.3 4.7211    5   287  19.6 391.13
##      [,13]
## [1,] 18.72
## [2,]  3.11
## [3,]  3.26
## [4,]  8.01
## [5,] 14.65
## [6,] 11.74
```

```
head(boston_housing$train$y)
```

```
## [1] 15.2 42.3 50.0 21.1 17.7 18.5
```

```
head(boston_housing$test$x)
```

```
##          [,1] [,2]  [,3] [,4]  [,5]  [,6]  [,7]   [,8] [,9] [,10] [,11]  [,12]
## [1,] 18.08460    0 18.10    0 0.679 6.434 100.0 1.8347   24   666  20.2  27.25
## [2,]  0.12329    0 10.01    0 0.547 5.913  92.9 2.3534    6   432  17.8 394.95
## [3,]  0.05497    0  5.19    0 0.515 5.985  45.4 4.8122    5   224  20.2 396.90
## [4,]  1.27346    0 19.58    1 0.605 6.250  92.6 1.7984    5   403  14.7 338.92
## [5,]  0.07151    0  4.49    0 0.449 6.121  56.8 3.7476    3   247  18.5 395.15
## [6,]  0.27957    0  9.69    0 0.585 5.926  42.6 2.3817    6   391  19.2 396.90
##      [,13]
## [1,] 29.05
## [2,] 16.21
## [3,]  9.74
## [4,]  5.50
## [5,]  8.44
## [6,] 13.59
```

```
head(boston_housing$test$y)
```

```
## [1]  7.2 18.8 19.0 27.0 22.2 24.5
```

Looking at the dataset, we see it is already broken in to *train* and *test*, each with an x and y. he x dataframes have 13 columns, these columns will be described later on. The y data is the median-value of owned houses in the $1000's. This what we will run the final prediction as the last step of this build.

If you remember from the description of Supervised learning above, we train the algorithm on input data and the desired output, the *train* set will has both the input and the desired output. Then *test* data will be what we run the model on to see how it performs. So let's move on and assign their variables:

```
c(train_data, train_labels) %<-% boston_housing$train
c(test_data, test_labels) %<-% boston_housing$test
paste0("Training entries: ", length(train_data), ", labels: ", length(train_labels))
```

```
## [1] "Training entries: 5252, labels: 404"
```

```
paste0("Testing entries: ", length(test_data), ", labels: ", length(test_labels))
```

```
## [1] "Testing entries: 1326, labels: 102"
```

We assigned the variables to their own vectors where the x are the entries (train_data and test_data) and y are the labels (train_labels and test_labels). From there, using the concatenate function we are able to see how many examples we are training on (404) and how many we are testing on (102). When doing machine learning, you always want to have as much data as possible to train on, this allows for the model to be as accurate as possible, from there when running a supervised learning algorithm you also want provide a significant amount of data to test on, this will allow for enough points for the regression to be effective.

Now, let's start to understand this data. For this part, I'm copying and pasting from the TensorFlow for R Tutorial page (https://tensorflow.rstudio.com/tutorials/beginners/basic-ml/tutorial_basic_regression/)

"The dataset contains 13 different features: * Per capita crime rate. * The proportion of residential land zoned for lots over 25,000 square feet. * The proportion of non-retail business acres per town. * Charles River dummy variable (= 1 if tract bounds river; 0 otherwise). * Nitric oxides concentration (parts per 10 million). * The average number of rooms per dwelling. * The proportion of owner-occupied units built before 1940. * Weighted distances to five Boston employment centers. * Index of accessibility to radial highways. * Full-value property-tax rate per $10,000. * Pupil-teacher ratio by town. * 1000 * (Bk - 0.63) ** 2 where Bk is the proportion of Black people by town. * Percentage lower status of the population."

I'll start by first creating a table to match the description to the column names. I could just create a dataframe for this, however I wanted to explore the matrix() and as.table() functions. If I was doing this step in Python, I would create a dictionary as I feel it would be easier to read and look cleaner in the code chunk. In the future, I'd like to research if data dictionaries are possible with R and if so, how to build them and use them effectively.

```
table_column_names <- matrix(c('Per capita crime rate.','CRIM',
'The proportion of residential land zoned for lots over 25,000 square feet.','ZN',
'The proportion of non-retail business acres per town.','INDUS',
'Charles River dummy variable (= 1 if tract bounds river; 0 otherwise).','CHAS',
'Nitric oxides concentration (parts per 10 million).','NOX',
'The average number of rooms per dwelling.','RM',
'The proportion of owner-occupied units built before 1940.','AGE',
'Weighted distances to five Boston employment centers.','DIS',
'Index of accessibility to radial highways.','RAD',
'Full-value property-tax rate per $10,000.','TAX',
'Pupil-teacher ratio by town.','PTRATIO',
'1000 * (Bk - 0.63) ** 2 where Bk is the proportion of Black people by town.','B',
'Percentage lower status of the population.','LSTAT'), ncol = 2, byrow = TRUE)

colnames(table_column_names) <- c('Description', 'Column Name')
rownames(table_column_names) <- 1:13
table_column_names <- as.table(table_column_names)
table_column_names
```

```
##    Description
## 1  Per capita crime rate.
## 2  The proportion of residential land zoned for lots over 25,000 square feet.
## 3  The proportion of non-retail business acres per town.
## 4  Charles River dummy variable (= 1 if tract bounds river; 0 otherwise).
## 5  Nitric oxides concentration (parts per 10 million).
## 6  The average number of rooms per dwelling.
## 7  The proportion of owner-occupied units built before 1940.
## 8  Weighted distances to five Boston employment centers.
## 9  Index of accessibility to radial highways.
## 10 Full-value property-tax rate per $10,000.
## 11 Pupil-teacher ratio by town.
## 12 1000 * (Bk - 0.63) ** 2 where Bk is the proportion of Black people by town.
## 13 Percentage lower status of the population.
##    Column Name
## 1  CRIM
## 2  ZN
## 3  INDUS
## 4  CHAS
## 5  NOX
## 6  RM
## 7  AGE
## 8  DIS
## 9  RAD
## 10 TAX
## 11 PTRATIO
## 12 B
## 13 LSTAT
```

Now that we know what the columns are, let's add their names so examining the data makes a little more sense.

```
column_names <- c('CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE',
                  'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT')

train_df <- train_data %>%
  as_tibble(.name_repair = "minimal") %>%
  setNames(column_names) %>%
  mutate(label = train_labels)

test_df <- test_data %>%
  as_tibble(.name_repair = "minimal") %>%
  setNames(column_names) %>%
  mutate(label = test_labels)
```

Before we take a look at the top few rows of the dataframe, let's go over what we did in the above code chunk. First, we created a new variable and name it train_df and test_df, this variable is taking the train_data or test_data part of the combined vector we constructed earlier and converting it to a dataframe using the as_tibble() function. Within this function, we clarify to not check the names of the columns using .name_repair = "minimal". This is being done since the original data doesn't have column names and we are assigning them now using the setNames() function, here we are pulling the array for the column names built in the first line of the code block. Finally, we are adding a 14th column to this dataframe with the column name 'label' which is populated with y of the train/test data or what we called train_labels and test_labels. So now we have two new dataframes, let's peek at them:

```
head(train_df)
```

```
## # A tibble: 6 x 14
##     CRIM    ZN INDUS  CHAS   NOX    RM   AGE   DIS   RAD   TAX PTRATIO     B
##    <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>   <dbl> <dbl>
## 1 1.23     0    8.14     0 0.538  6.14  91.7  3.98     4   307    21    397.
## 2 0.0218  82.5  2.03     0 0.415  7.61  15.7  6.27     2   348    14.7  395.
## 3 4.90     0   18.1      0 0.631  4.97 100    1.33    24   666    20.2  376.
## 4 0.0396   0    5.19     0 0.515  6.04  34.5  5.99     5   224    20.2  397.
## 5 3.69     0   18.1      0 0.713  6.38  88.4  2.57    24   666    20.2  391.
## 6 0.284    0    7.38     0 0.493  5.71  74.3  4.72     5   287    19.6  391.
## # ... with 2 more variables: LSTAT <dbl>, label <dbl>
```

```
head(test_df)
```

```
## # A tibble: 6 x 14
##      CRIM    ZN INDUS  CHAS   NOX    RM   AGE   DIS   RAD   TAX PTRATIO     B
##     <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>   <dbl> <dbl>
## 1 18.1        0 18.1      0 0.679  6.43 100    1.83    24   666    20.2  27.2
## 2  0.123      0 10.0      0 0.547  5.91  92.9  2.35     6   432    17.8 395.
## 3  0.0550     0  5.19     0 0.515  5.98  45.4  4.81     5   224    20.2 397.
## 4  1.27       0 19.6      1 0.605  6.25  92.6  1.80     5   403    14.7 339.
## 5  0.0715     0  4.49     0 0.449  6.12  56.8  3.75     3   247    18.5 395.
## 6  0.280      0  9.69     0 0.585  5.93  42.6  2.38     6   391    19.2 397.
## # ... with 2 more variables: LSTAT <dbl>, label <dbl>
```

Alright, looking the data now is much easier on the eyes, plus since we are doing a regression model machine learning algorithm, we know it is considered supervised learning and therefore needed to be labeled. However while looking at this data as well, you will notice that all the columns have values in various scales. The

CHAS column is a binary indicator for the Charles River, the RAD and TAX columns also seem to be integers, and most other columns are doubles or floats. You can run the model with the data as is, but it is best practice to normalize the values so they are all on the same scale range.

```
spec <- feature_spec(train_df, label ~ . ) %>%
  step_numeric_column(all_numeric(), normalizer_fn = scaler_standard()) %>%
  fit()

spec
```

```
## -- Feature Spec ----------------------------------------------------------------
## A feature_spec with 13 steps.
## Fitted: TRUE
## -- Steps -----------------------------------------------------------------------
## The feature_spec has 1 dense features.
## StepNumericColumn: CRIM, ZN, INDUS, CHAS, NOX, RM, AGE, DIS, RAD, TAX, PTRATIO, B, LSTAT
## -- Dense features --------------------------------------------------------------
```

Amazing how far we've made it. It's finally time to build the model now that we have normalized the data.

```
input <- layer_input_from_dataset(train_df %>% select(-label))

output <- input %>%
  layer_dense_features(dense_features(spec)) %>%
  layer_dense(units = 64, activation = "relu") %>%
  layer_dense(units = 64, activation = "relu") %>%
  layer_dense(units = 1)

model <- keras_model(input, output)

summary(model)
```

```
## Model: "model"
## ________________________________________________________________________________
##  Layer (type)             Output Shape      Param #  Connected to
## ================================================================================
##  AGE (InputLayer)         [(None,)]         0        []
##
##  B (InputLayer)           [(None,)]         0        []
##
##  CHAS (InputLayer)        [(None,)]         0        []
##
##  CRIM (InputLayer)        [(None,)]         0        []
##
##  DIS (InputLayer)         [(None,)]         0        []
##
##  INDUS (InputLayer)       [(None,)]         0        []
##
##  LSTAT (InputLayer)       [(None,)]         0        []
##
##  NOX (InputLayer)         [(None,)]         0        []
##
##  PTRATIO (InputLayer)     [(None,)]         0        []
```

```
##
##  RAD (InputLayer)         [(None,)]         0        []
##
##  RM (InputLayer)          [(None,)]         0        []
##
##  TAX (InputLayer)         [(None,)]         0        []
##
##  ZN (InputLayer)          [(None,)]         0        []
##
##  dense_features (DenseFea  (None, 13)       0        [’AGE[0][0]’,
##  tures)                                               ’B[0][0]’,
##                                                       ’CHAS[0][0]’,
##                                                       ’CRIM[0][0]’,
##                                                       ’DIS[0][0]’,
##                                                       ’INDUS[0][0]’,
##                                                       ’LSTAT[0][0]’,
##                                                       ’NOX[0][0]’,
##                                                       ’PTRATIO[0][0]’,
##                                                       ’RAD[0][0]’,
##                                                       ’RM[0][0]’,
##                                                       ’TAX[0][0]’,
##                                                       ’ZN[0][0]’]
##
##  dense_2 (Dense)          (None, 64)        896      [’dense_features[0][0]’]
##
##  dense_1 (Dense)          (None, 64)        4160     [’dense_2[0][0]’]
##
##  dense (Dense)            (None, 1)         65       [’dense_1[0][0]’]
##
## ================================================================================
## Total params: 5,121
## Trainable params: 5,121
## Non-trainable params: 0
## ________________________________________________________________________________
```

Now that we have the model, let’s use the compile() function so it is configured for training. We are using the loss = “mse” which stands for mean squared error since we are running a regression model and not a classification one. In the Keras documenation, you can read about the details of this part of the function (https://keras.io/api/callbacks/). Our metric for evaluation is mean absolute error (mae). The optimizer chosen is the one recommended for recurrent neural networks and we have left the learning rate as default (https://tensorflow.rstudio.com/reference/keras/optimizer_rmsprop/).

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

From here, let’s combine the above two steps into a single function as we will need to use as we train the algorithm and play around with the iterations to ensure it’s as accurate as possible:

```r
build_model <- function() {
  input <- layer_input_from_dataset(train_df %>% select(-label))

  output <- input %>%
    layer_dense_features(dense_features(spec)) %>%
    layer_dense(units = 64, activation = "relu") %>%
    layer_dense(units = 64, activation = "relu") %>%
    layer_dense(units = 1)

  model <- keras_model(input, output)

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

  model
}
```

It's now time to train the model. Within Keras there is a Callbacks object the purose of which is to "perform actions at various stages of training" (https://keras.io/api/callbacks/). In our first attempt to train the model, we will create a custom callback setting a dot as the output for each iteration through the x and y data. Keras calls these iterations epochs and we will be running 500 of them for this training.

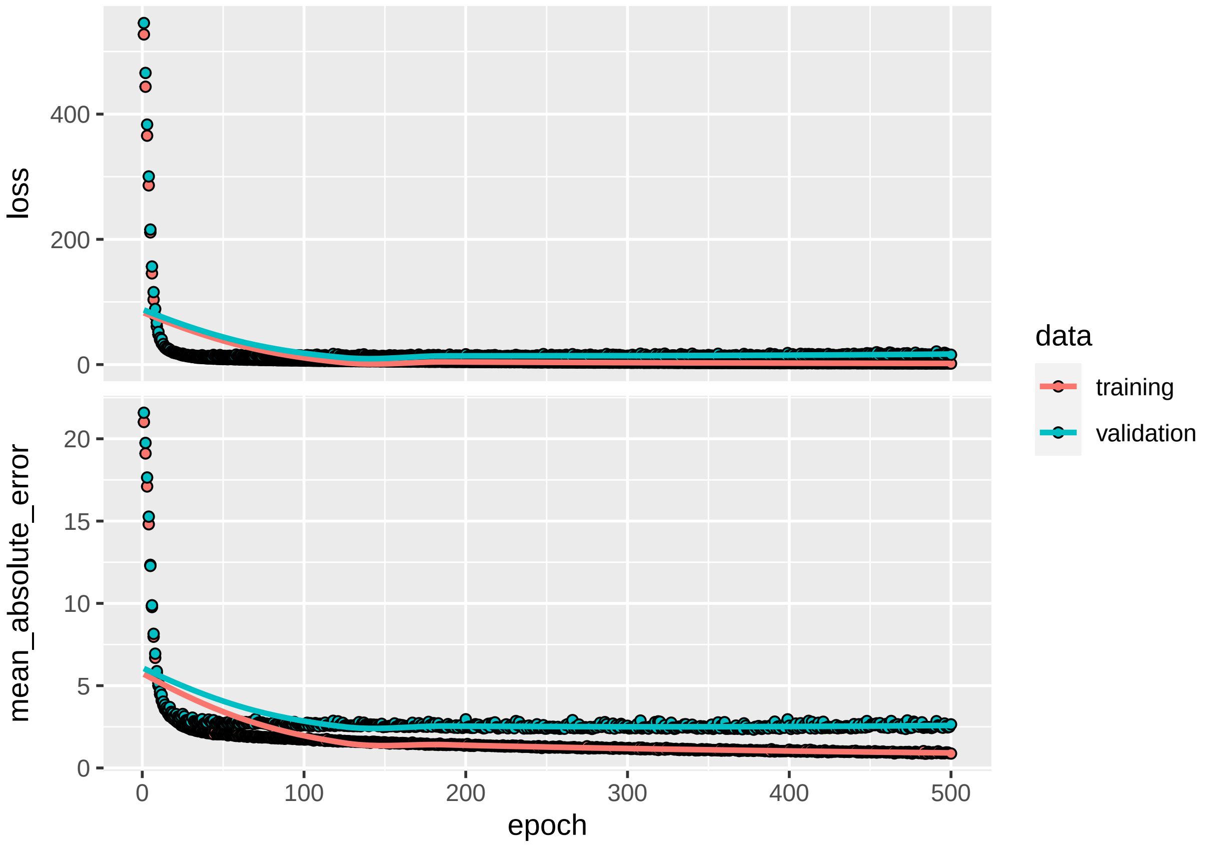
```r
# Display training progress by printing a single dot for each completed epoch.
print_dot_callback <- callback_lambda(
  on_epoch_end = function(epoch, logs) {
    if (epoch %% 80 == 0) cat("\n")
    cat(".")
  }
)

model <- build_model()

history <- model %>% fit(
  x = train_df %>% select(-label),
  y = train_df$label,
  epochs = 500,
  validation_split = 0.2,
  verbose = 0,
  callbacks = list(print_dot_callback)
)
```

```
##
## ................................................................................
## ................................................................................
## ................................................................................
## ................................................................................
## ................................................................................
## ................................................................................
## ....................
```

Just running the model and seeing the output of .'s isn't very informative. To understand what impact this training had, it would be best to see it plotted out.

```
plot(history)
```

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

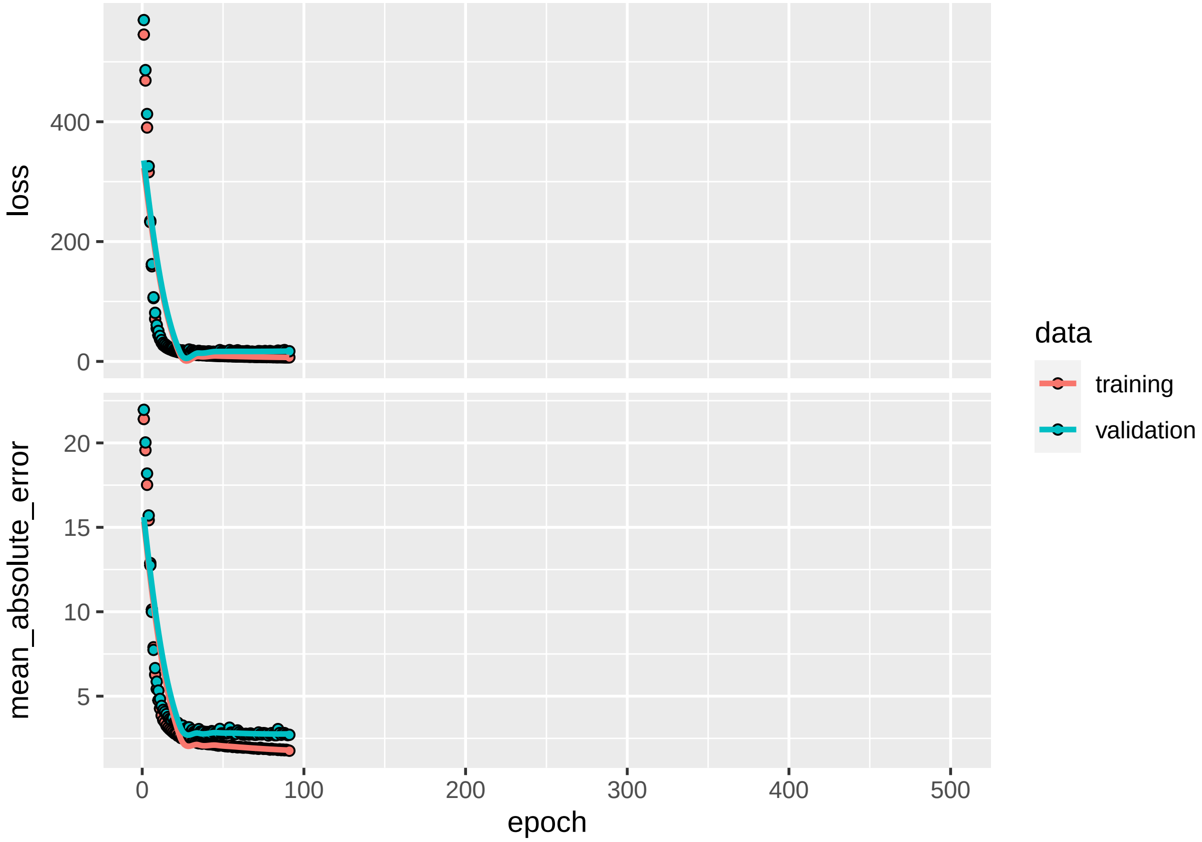


From these graphs we can see lines are flat after about 150/200 epochs, this indicates no improvement in the model training. This can be adjusted by using a different callback() function - callback_early_stopping(). In this we are setting it to monitor the loss and the patience = 20 tells the model to stop running after 20 epochs of no improvement. Let's re-train the model and plot the results:

```
# The patience parameter is the amount of epochs to check for improvement.
early_stop <- callback_early_stopping(monitor = "val_loss", patience = 20)

model <- build_model()

history <- model %>% fit(
  x = train_df %>% select(-label),
  y = train_df$label,
  epochs = 500,
  validation_split = 0.2,
  verbose = 0,
  callbacks = list(early_stop)
)
```

```
plot(history)
```

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



According to these graphs the average error is ~$2,500. Can we get a number to confirm?

```
c(loss, mae) %<-% (model %>% evaluate(train_df %>% select(-label), train_df$label, verbose = 0))

paste0("Mean absolute error on train set: $", sprintf("%.2f", mae * 1000))
```

```
## [1] "Mean absolute error on train set: $1979.17"
```

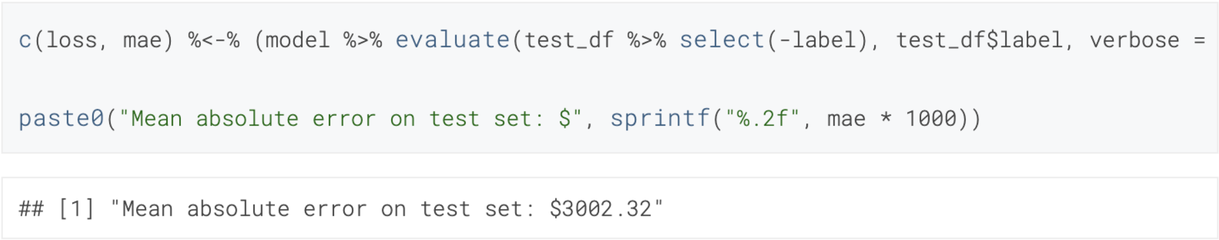
Let's compare this to what the model evaluates on the test dataset:

```
c(loss, mae) %<-% (model %>% evaluate(test_df %>% select(-label), test_df$label, verbose = 0))

paste0("Mean absolute error on test set: $", sprintf("%.2f", mae * 1000))
```

```
## [1] "Mean absolute error on test set: $3208.99"
```

Looking at the number I got for MAE is 3551.23, however on the TensorFlow tutorial page they show 3002.32:

I wonder if this is because I did not normalize the test data. Let's try doing that and then re-running the above code block to see what happens:



```
c(loss, mae) %<-% (model %>% evaluate(test_df %>% select(-label), test_df$label, verbose =

paste0("Mean absolute error on test set: $", sprintf("%.2f", mae * 1000))
```

```
## [1] "Mean absolute error on test set: $3002.32"
```

Figure 1: TensorFlow Tutorial MAE on test set

```
spec2 <- feature_spec(test_df, label ~ . ) %>%
  step_numeric_column(all_numeric(), normalizer_fn = scaler_standard()) %>%
  fit()

spec2
```

```
## -- Feature Spec --------------------------------------------------------------
## A feature_spec with 13 steps.
## Fitted: TRUE
## -- Steps ---------------------------------------------------------------------
## The feature_spec has 1 dense features.
## StepNumericColumn: CRIM, ZN, INDUS, CHAS, NOX, RM, AGE, DIS, RAD, TAX, PTRATIO, B, LSTAT
## -- Dense features -------------------------------------------------------------
```

```
c(loss, mae) %<-% (model %>% evaluate(test_df %>% select(-label), test_df$label, verbose = 0))

paste0("Mean absolute error on test set: $", sprintf("%.2f", mae * 1000))
```

```
## [1] "Mean absolute error on test set: $3208.99"
```

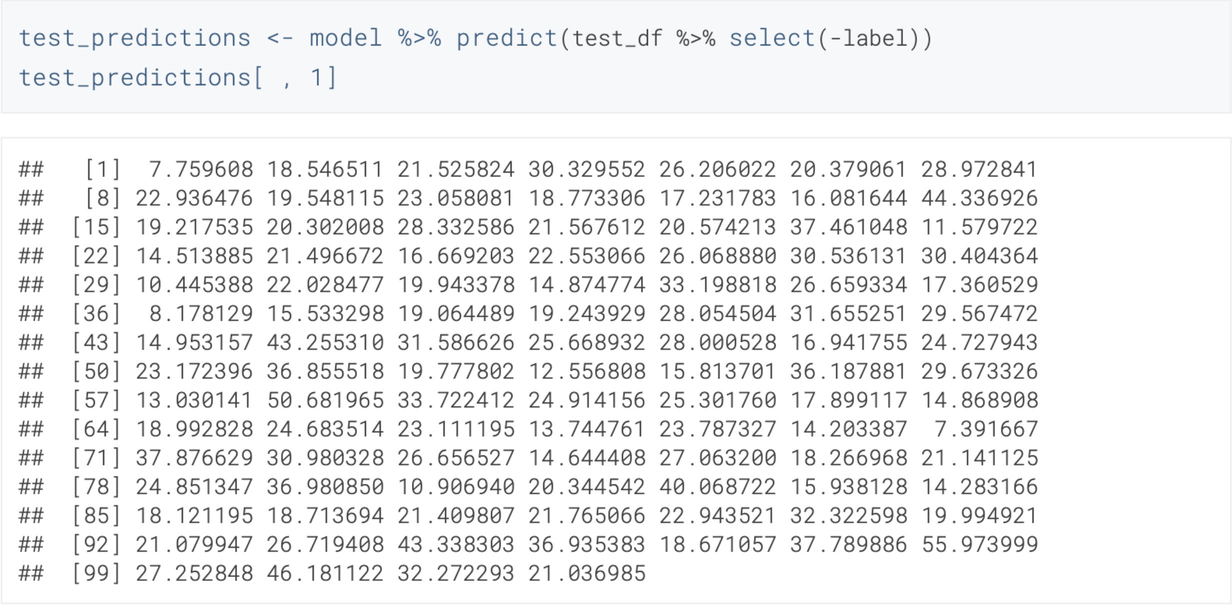
I am still getting the same result. This could be that data in the dataset has changed since they posted their tutorial. Or maybe I missed something else. This might be something I might need to dig into to solve the mystery. Either way, looking at these MAE values compared to some of the errors under the label column going up to $15,000, they are not as significant.

Since the purpose of most machine learning is predictive modeling, let's generate some predictions based on the model we have now trained. We are predicting on the *label* column of the test_df. This is providing us with new median-values of owned homes in the $1000's, however now with lowest errors. This is taking in all the columns from the train_df that the algorithm learned from and minimizing the MAE.

```
test_predictions <- model %>% predict(test_df %>% select(-label))
test_predictions[ , 1]
```

```
##   [1]  8.245428 18.055229 20.489489 30.754120 24.260664 20.193064 26.149389
##   [8] 21.082220 19.374914 22.555685 19.039837 15.979984 15.448673 41.217915
##  [15] 19.511984 19.295931 25.551441 19.937487 20.247995 40.186615 11.521178
##  [22] 15.384137 20.197397 14.102757 19.078789 25.633411 29.907469 26.287189
##  [29] 11.030869 20.146711 19.605700 14.610459 32.372753 24.049641 18.322388
##  [36]  8.593739 15.836400 17.656954 19.239395 24.490217 29.236774 27.275927
##  [43] 14.941142 39.635880 28.869858 23.079607 25.520535 16.149500 24.193928
##  [50] 21.702566 33.427559 19.778708 11.874095 15.347930 33.241310 27.343073
##  [57] 13.162894 46.585899 33.386398 22.182966 23.660604 17.291304 14.542415
##  [64] 18.508642 22.787167 19.622591 13.500798 21.134806 12.568538  7.280894
##  [71] 39.223015 29.278364 27.678175 13.782431 24.278994 17.251575 19.417875
##  [78] 22.781988 32.966194 10.935927 19.661896 37.476234 14.321540 14.890234
##  [85] 16.303583 17.701742 21.701857 19.801615 21.283588 30.074768 19.003878
##  [92] 18.468748 24.414557 41.648216 32.848801 19.870378 35.542732 47.718784
##  [99] 25.662081 41.380985 31.112265 19.363546
```

Again, my numbers are just slightly off from the tutorial:



```
test_predictions <- model %>% predict(test_df %>% select(-label))
test_predictions[ , 1]
```

```
##   [1]  7.759608 18.546511 21.525824 30.329552 26.206022 20.379061 28.972841
##   [8] 22.936476 19.548115 23.058081 18.773306 17.231783 16.081644 44.336926
##  [15] 19.217535 20.302008 28.332586 21.567612 20.574213 37.461048 11.579722
##  [22] 14.513885 21.496672 16.669203 22.553066 26.068880 30.536131 30.404364
##  [29] 10.445388 22.028477 19.943378 14.874774 33.198818 26.659334 17.360529
##  [36]  8.178129 15.533298 19.064489 19.243929 28.054504 31.655251 29.567472
##  [43] 14.953157 43.255310 31.586626 25.668932 28.000528 16.941755 24.727943
##  [50] 23.172396 36.855518 19.777802 12.556808 15.813701 36.187881 29.673326
##  [57] 13.030141 50.681965 33.722412 24.914156 25.301760 17.899117 14.868908
##  [64] 18.992828 24.683514 23.111195 13.744761 23.787327 14.203387  7.391667
##  [71] 37.876629 30.980328 26.656527 14.644408 27.063200 18.266968 21.141125
##  [78] 24.851347 36.980850 10.906940 20.344542 40.068722 15.938128 14.283166
##  [85] 18.121195 18.713694 21.409807 21.765066 22.943521 32.322598 19.994921
##  [92] 21.079947 26.719408 43.338303 36.935383 18.671057 37.789886 55.973999
##  [99] 27.252848 46.181122 32.272293 21.036985
```

Figure 2: TensorFlow Tutorial Predictions

However, we have successfully run through an example of a regression supervised machine learning model! From here, I would be curious to try this dataset using the caret package and see what the differences are and if one is easier or the numbers change as well.

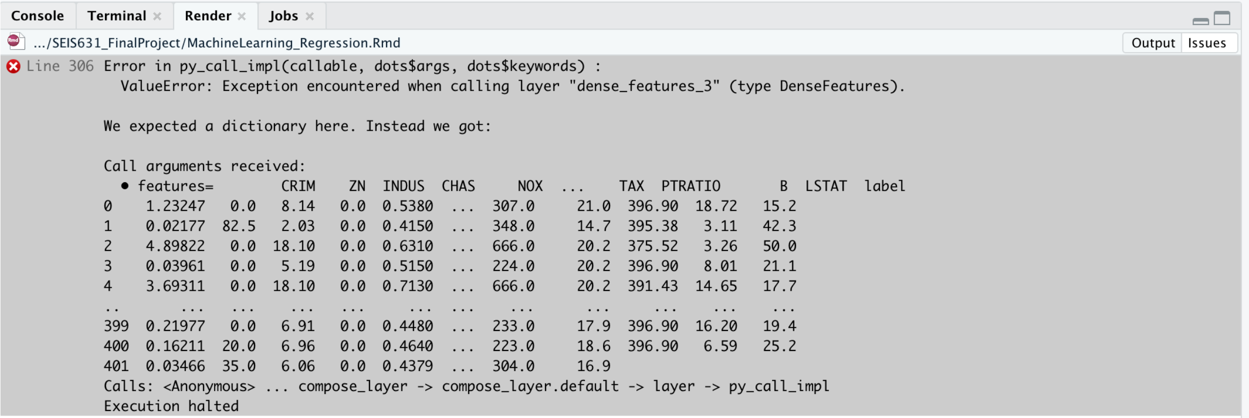
Topics from Class

R/R Markdown

During this project I learned a lot about R and R Markdown. While working through the code for the machine learning algorithm, I learned the following in R: %in% operator, paste0(), mutate(), as_tibble(), and how to write a function in R. With regards to R Markdown, I learned how to create bullet points, insert an image, and also dealt with new errors I hadn't seen before when attempting to knit my file. I was able to resolve the error by commenting out the code that was causing the error (see below), however, I was expecting the R Markdown file to print the error into the knitted PDF.

Github

I have worked with Github before both academically and professionally, so I felt comfortable using it for this project. I did learn how to clone a repo and add/commit/push using R Studio. While I think it's a unique



```
.../SEIS631_FinalProject/MachineLearning_Regression.Rmd
Line 306 Error in py_call_impl(callable, dots$args, dots$keywords) :
           ValueError: Exception encountered when calling layer "dense_features_3" (type DenseFeatures).

         We expected a dictionary here. Instead we got:

         Call arguments received:
           • features=        CRIM    ZN  INDUS  CHAS     NOX  ...    TAX  PTRATIO       B  LSTAT  label
         0    1.23247   0.0   8.14   0.0  0.5380  ...  307.0     21.0  396.90  18.72   15.2
         1    0.02177  82.5   2.03   0.0  0.4150  ...  348.0     14.7  395.38   3.11   42.3
         2    4.89822   0.0  18.10   0.0  0.6310  ...  666.0     20.2  375.52   3.26   50.0
         3    0.03961   0.0   5.19   0.0  0.5150  ...  224.0     20.2  396.90   8.01   21.1
         4    3.69311   0.0  18.10   0.0  0.7130  ...  666.0     20.2  391.43  14.65   17.7
         ..       ...   ...    ...   ...     ...  ...    ...      ...     ...    ...    ...
         399  0.21977   0.0   6.91   0.0  0.4480  ...  233.0     17.9  396.90  16.20   19.4
         400  0.16211  20.0   6.96   0.0  0.4640  ...  223.0     18.6  396.90   6.59   25.2
         401  0.03466  35.0   6.06   0.0  0.4379  ...  304.0     16.9
         Calls: <Anonymous> ... compose_layer -> compose_layer.default -> layer -> py_call_impl
         Execution halted
```

Figure 3: Render Error Print

feature, I prefer to use the terminal, so I did primarily use that rather than the inbuilt R Studio buttons. I also have my SSH set up for my Github profile, so when cloning the repo into R Studio and playing around with the Git capabilities in R Studio, those permissions were cleared on my computer and I had to reset the agent.

Regression

We've been studying regression during the last few weeks of class which is why I chose to do it as my machine learning example. I do feel it's important to have a base knowledge about regression and other statistical theories and equations prior to diving into machine learning as those are foundations for these algorithms, so it allows you know what is going on when the model is training on the data and where the predictive numbers come from. However, you don't get exposed to the actual equation, so in that sense, I felt a bit removed from the textbook and lectures we've done.

Machine Learning

I am very interested in machine learning and hope to center my career around it going forward so for me it was very important to get a feel for it in R. There have been Data Scientist positions that I haven't applied for as I didn't know R that well and therefore certainly didn't feel confident learning the existing machine learning techniques of a company. Now, I can comfortable code in R and through this project exhibited my ability to understand the models so I can apply them in future scenarios.

R Libraries/Packages

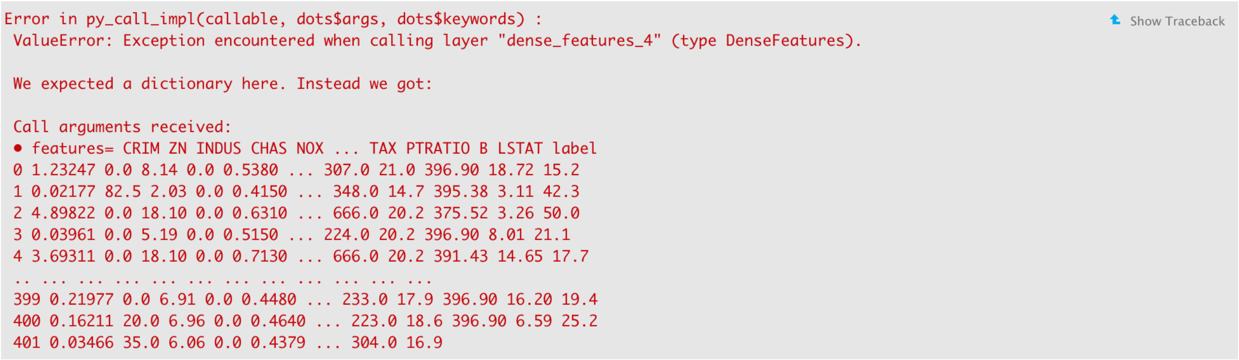
Throughout class we've been touching on some inbuilt R libraries and diving into others that needed to be installed such as LaTex and Tidyverse. I wanted to see what more was out there. In the future I might deep dive into an R versus Python comparison to see what the equivalent libraries are between the two. And as I found TensorFlow for R, I'm thinking there are more libraries that have been made compatible with R or libraries originally created for R that may have been adjusted to be imported by Python. Through running the code in this project, I also better understood the ones we've used in class such as ggplot2.

Conclusion

In my conclusion, I'd like to start by addressing the discrepancies I found at the end of the machine learning presentation where I stated that my numbers don't match what was in the TensorFlow tutorial. While knitting this file a few times, I realized that the numbers came out differently every time... took me a minute, but then I realized that these models adjust and learn as they go, they are not made to be exact and provide a fixed answer. Most likely if someone followed my project and ran their own model, they would find different results.

One thing I was not able to resolve was when I ran the following code I received the error in the image below it. I think this has something to do with how the dataframe is formatted in R versus Python as TensorFlow and Keras use Python even in R – you can see it's one of packages install when you install Tensorflow. I will have to do more reserach on this and try to find a solution. Luckily, this was not an integral part of the model, just something to show how the layer() function works.

```
#layer <- layer_dense_features(
 # feature_columns = dense_features(spec),
 # dtype = tf$float32
#)
#layer(train_df)
```



```
Error in py_call_impl(callable, dots$args, dots$keywords) :
 ValueError: Exception encountered when calling layer "dense_features_4" (type DenseFeatures).

 We expected a dictionary here. Instead we got:

 Call arguments received:
 • features= CRIM ZN INDUS CHAS NOX ... TAX PTRATIO B LSTAT label
 0 1.23247 0.0 8.14 0.0 0.5380 ... 307.0 21.0 396.90 18.72 15.2
 1 0.02177 82.5 2.03 0.0 0.4150 ... 348.0 14.7 395.38 3.11 42.3
 2 4.89822 0.0 18.10 0.0 0.6310 ... 666.0 20.2 375.52 3.26 50.0
 3 0.03961 0.0 5.19 0.0 0.5150 ... 224.0 20.2 396.90 8.01 21.1
 4 3.69311 0.0 18.10 0.0 0.7130 ... 666.0 20.2 391.43 14.65 17.7
 .. ... ... ... ... ... ... ... ... ... ... ...
 399 0.21977 0.0 6.91 0.0 0.4480 ... 233.0 17.9 396.90 16.20 19.4
 400 0.16211 20.0 6.96 0.0 0.4640 ... 223.0 18.6 396.90 6.59 25.2
 401 0.03466 35.0 6.06 0.0 0.4379 ... 304.0 16.9
```

This project absolutely advanced my knowledge, I'm glad I was able to pick a topic and aim high with it. Doing projects like these re-affirms to myself that I'm in the right industry and I'm excited to continue in this Master's program and learn more. I also enjoy the challenge of teaching myself. It's not always fun, in fact quite often frustrating, but it's an important life skill to have and once you accomplish something it's incredibly rewarding.

Bibliography

https://tensorflow.rstudio.com/tutorials/

https://tensorflow.rstudio.com/tutorials/beginners/basic-ml/tutorial_basic_regression/

https://keras.io/api/models/model_training_apis/

https://topepo.github.io/caret/

https://www.rdocumentation.org/packages/caret/versions/6.0-91

https://machinelearningmastery.com/machine-learning-in-r-step-by-step/

https://www.datacamp.com/community/tutorials/machine-learning-in-r

https://r4ds.had.co.nz/

https://www.techtarget.com/searchenterpriseai/definition/machine-learning-ML#:~:text=Machine%20learning%20(ML)%2
%20output%20values.

https://en.wikipedia.org/wiki/Machine_learning

https://blog.dataiku.com/unsupervised-machine-learning-use-cases-examples

https://www.rdocumentation.org/packages/tibble/versions/3.1.6/topics/tibble

https://www.marsja.se/how-to-use-in-in-r/

https://rmarkdown.rstudio.com/lesson-1.html

https://www.statology.org/create-table-in-r/

https://statisticsglobe.com/change-row-names-of-data-frame-or-matrix-in-r

https://imjbmkz.medium.com/analyzing-boston-housing-dataset-2c7928f2a87f