HW 6 MATH 686

James DG

Fall 2022

```
library(keras)
```

```
## Warning: package 'keras' was built under R version 4.1.2
```

```
library(recipes)
```

```
## Warning: package 'recipes' was built under R version 4.1.2

## Loading required package: dplyr

## Warning: package 'dplyr' was built under R version 4.1.2

##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union

##
## Attaching package: 'recipes'

## The following object is masked from 'package:stats':
##
##     step
```

```
library(mltools)

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

```
## [1] 1460   81
```

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

y_train =train_data[,80]
train_data = train_data[,-80]
```

#Let's preprocess the data before we remove the label variable

1. Build and train a linear regression model with multiple layers and print out the training history.

First we need to do some data cleaning, let's look at the types of each column

Let's try just removing all null values and see how large our training data set is

```
dataset <- na.omit(train_data)
dim(dataset)
```

```
## [1]  0 79
```

That leaves no data points available! Let's try to count which column has the most NA values

```
na_count <-sapply(train_data, function(y) sum(length(which(is.na(y)))))
na_count <- data.frame(na_count)
```

The columns that have the most NA values are Alley (1369), FireplaceQu (690), PoolQC (1453), Fence (1179), MiscFeature(1406). Since our data set is only 1460 data points, let's just drop these columns

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

```
## [1] 6
```

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

```
## [1] 57
```

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

```
## [1] 72
```

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

```
## [1] 73
```

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

```
## [1] 74
```

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

```
## [1] 3
```

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

One hot encoding with mltools library one_hot function

```
new_train_data <- sapply(train_data, unclass)
new_train_data <- one_hot(new_train_data)
dim(new_train_data)
```

```
## [1] 1460   73
```

find numeric features, since categorical features need to convert to proper code

```
i=0

numeric_features=c()

for (feature_name in names(train_data)) {


if ( class(train_data[, feature_name] )=='integer' |  class(train_data[, feature_name])=='numeric')
{
 cat(feature_name, '\n')
 numeric_features=c(numeric_features, feature_name)
 #show(train_data[1:5, feature_name])
 i=i+1
 show(i)
 }

}
```

```
## MSSubClass
## [1] 1
## LotArea
## [1] 2
## OverallQual
## [1] 3
## OverallCond
## [1] 4
## YearBuilt
## [1] 5
## YearRemodAdd
## [1] 6
## MasVnrArea
## [1] 7
## BsmtFinSF1
## [1] 8
## BsmtFinSF2
## [1] 9
## BsmtUnfSF
## [1] 10
## TotalBsmtSF
## [1] 11
## X1stFlrSF
## [1] 12
## X2ndFlrSF
## [1] 13
## LowQualFinSF
## [1] 14
## GrLivArea
## [1] 15
## BsmtFullBath
## [1] 16
## BsmtHalfBath
## [1] 17
## FullBath
## [1] 18
## HalfBath
## [1] 19
## BedroomAbvGr
## [1] 20
## KitchenAbvGr
## [1] 21
## TotRmsAbvGrd
## [1] 22
## Fireplaces
## [1] 23
## GarageYrBlt
## [1] 24
## GarageCars
## [1] 25
## GarageArea
## [1] 26
## WoodDeckSF
## [1] 27
```

```
## OpenPorchSF
## [1] 28
## EnclosedPorch
## [1] 29
## X3SsnPorch
## [1] 30
## ScreenPorch
## [1] 31
## PoolArea
## [1] 32
## MiscVal
## [1] 33
## MoSold
## [1] 34
## YrSold
## [1] 35
```

```
# find the numeric features with NA values
numeric_with_na=c()


for (feature_name in numeric_features) {

if (any(is.na(train_data[, feature_name]))) numeric_with_na=c(numeric_with_na, feature_name)

}



# view the variables with NA

#numeric_with_na

#which(is.na(train_data[, 'LotFrontage']))

# replace NA by the mean of the available data

for (feature_name in numeric_with_na) {
train_data[,feature_name][is.na(train_data[,feature_name])]  = mean(train_data[,feature_name], na.rm = T
}
train_data[, numeric_features] = apply(train_data[, numeric_features], MARGIN=2, function(x) (x-mean(x)
```

```
all_features=names(train_data)

non_numeric = all_features[!all_features%in% numeric_features]
for (feature_name in non_numeric) {

if (any(is.na(train_data[, feature_name]))) show(feature_name)

}
```

```
## [1] "MasVnrType"
## [1] "BsmtQual"
```

```
## [1] "BsmtCond"
## [1] "BsmtExposure"
## [1] "BsmtFinType1"
## [1] "BsmtFinType2"
## [1] "Electrical"
## [1] "GarageType"
## [1] "GarageFinish"
## [1] "GarageQual"
## [1] "GarageCond"
```

```
train_rec <- recipe( ~ ., data = train_data) %>%
             step_impute_knn(all_numeric()) %>%
             step_center(all_numeric()) %>%
             step_scale(all_numeric()) %>%
                         step_unknown(all_nominal(), new_level = "unknown") %>%
                         step_dummy(all_nominal(), 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)
```

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

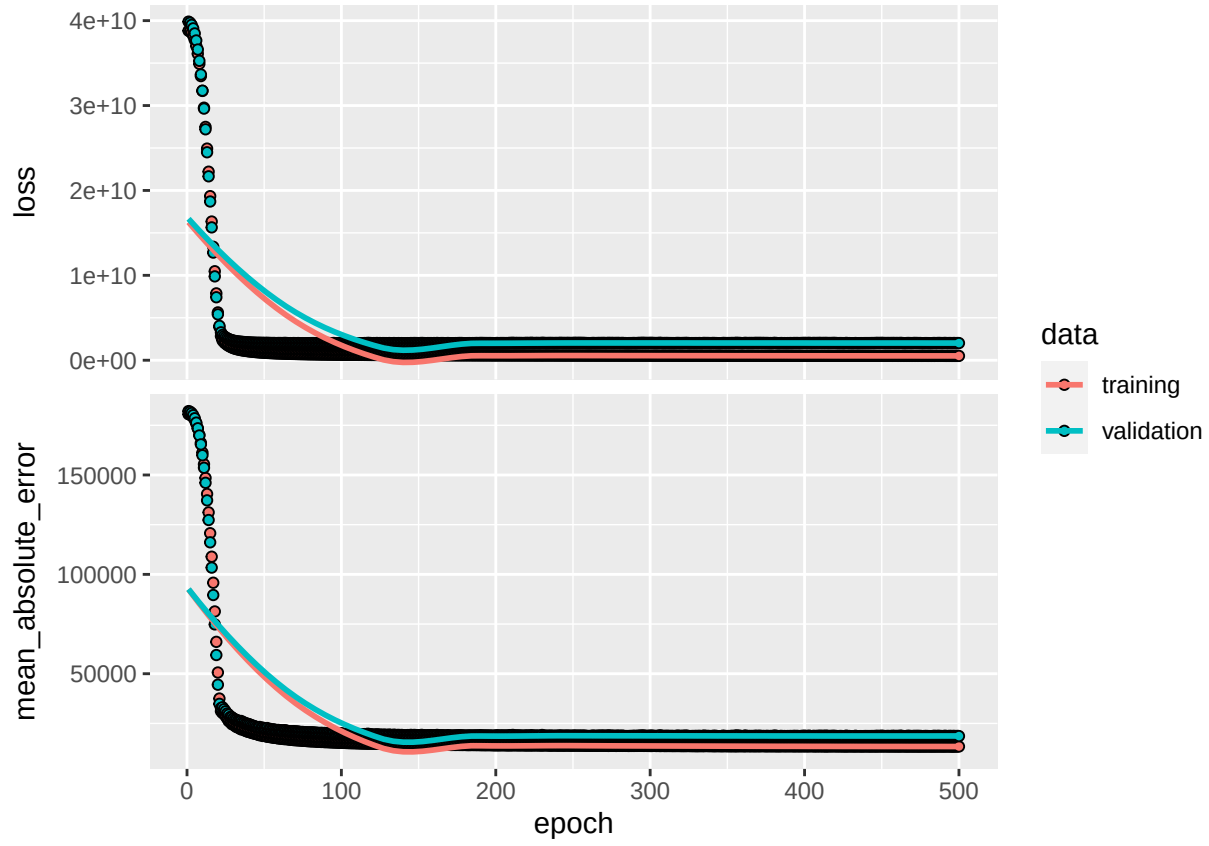
```
## Loaded Tensorflow version 2.9.2
```

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


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

```
history <- model %>% fit(
  as.matrix(recipe_processed_train_data),
  as.matrix(y_train),
    batch_size = 20,
  validation_split = 0.2,
    #callbacks = callback_early_stopping(patience = 2, monitor = 'mean_absolute_error'),
  verbose = 1,
  epochs = 500
)

plot(history)
```



2. Calculate the mean absolute error of your final model for the training data.

```
y_pred <- model %>% predict(recipe_processed_train_data)
mean(abs(as.numeric(y_pred)-y_train))
```

```
## [1] 14364.85
```

3. Calcualte the mean squared logarithmic error of your final model for the traning data.

For definition of the mean squared logarithmic error, see https://keras.io/api/losses/regression_losses/

```
logsquare=mean((log(y_train + 1) - log(y_pred + 1))^2, axis=-1)
print(logsquare)
```

```
## [1] 0.01571746
```