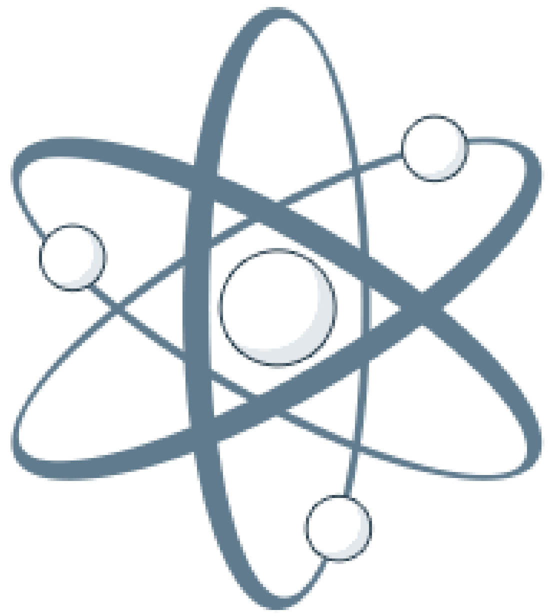
Core API: linear regression

INTRODUCTION TO TENSORFLOW IN R

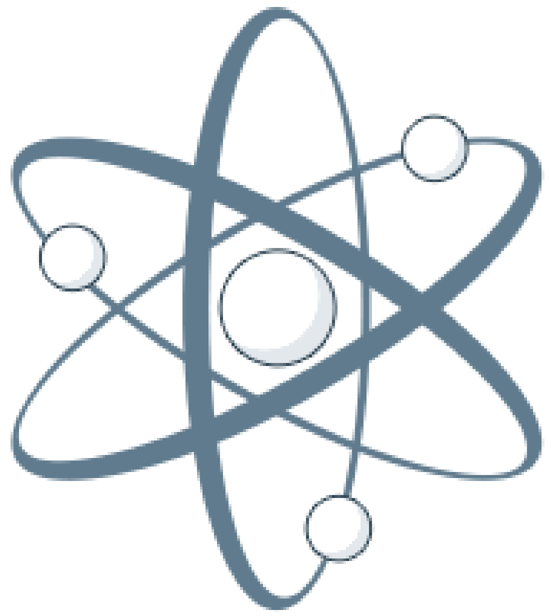
Colleen Bobbie
Instructor

TensorFlow APIs

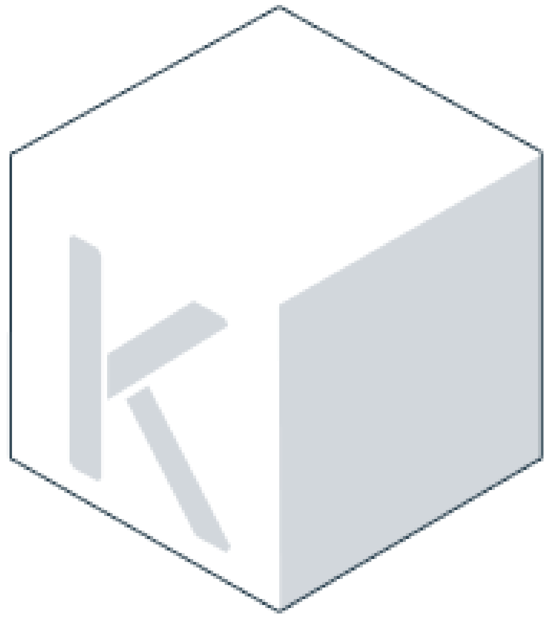


Core API

TensorFlow APIs

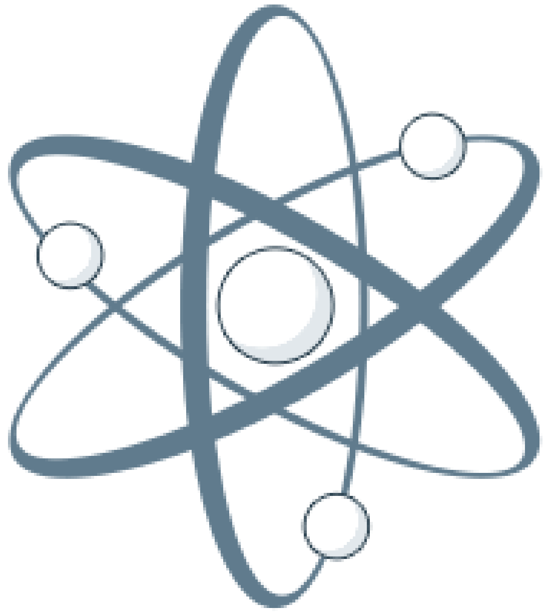


Core API

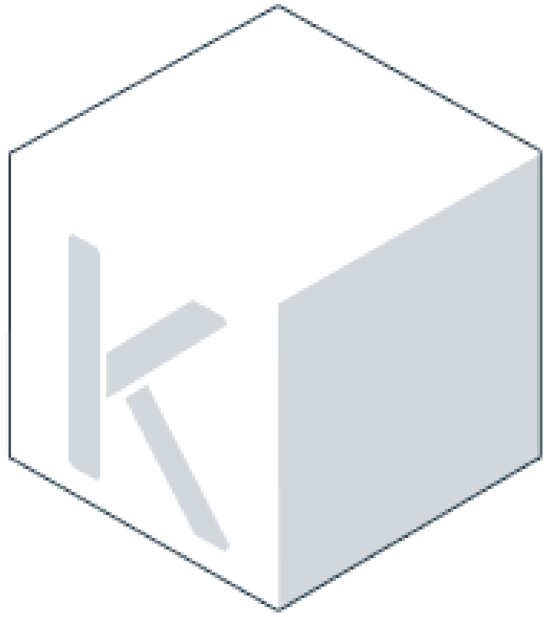


Keras API

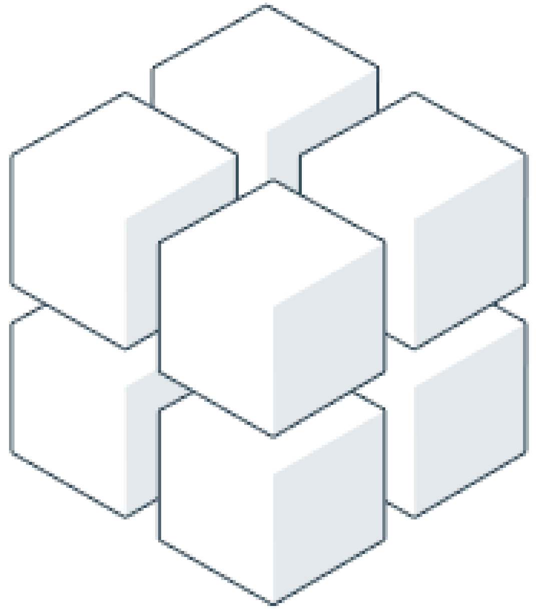
TensorFlow APIs



Core API



Keras API



Estimators API

[1] https://tensorflow.rstudio.com/

Introduction to our first dataset

For this simple linear regression example:

Research question: Can we predict how much beer is consumed in a university town on a given day, using

- Amount of Daily Precipitation (mm)

Our dependent variable, amount of beer consumed, will be measured in liters (L).

Note: we'll be adding on more variables when we conduct a multiple linear regression later in this chapter!

Creating testing and training datasets

We first must separate our dataset into training and testing datasets. For the purposes of this lesson, we'll use an 80/20 split.

First, select randomly 80% of the tuples in our dataset.

```
beerrows <- sample(1:nrow(beer_consumption), size = 0.8 * nrow(beer_consumption))
```

Then, use the selected rows to create a training and testing dataset.

```
beer_consumption_train <- beer_consumption[beerrows,]
beer_consumption_test <-beer_consumption[-beerrows,]
```

Consider pausing to complete Datacamp's Introduction to Machine Learning if you need a refresh.

Parsing out dependent and independent variables

Next step:

Define x-variable:

```
x_actual <- beer_consumption_train$precip
```

Define y-variable:

```
y_actual <- beer_consumption_train$beer_consumed
```

Defining w, b, and y variables

Remember the equation for a straight line?

`y = wx+b` where `w` is the slope (aka 'Weights') and `b` is the intercept (aka 'Bias').

Our next step is to define our `w` , `b` , and `y` variables.

```
w <- tf$Variable(tf$random_uniform(shape(1L), -1, 1))
```

```
b <- tf$Variable(tf$zeros(shape(1L)))
```

```
y_predict <- w * x_data + b
```

Let's practice!

INTRODUCTION TO TENSORFLOW IN R

Core API: linear regression part II

INTRODUCTION TO TENSORFLOW IN R

Colleen Bobbie
Instructor

Defining the cost function

Cost function:

- measure of how *wrong* a model is
- also known as **loss** or **error**
- typically compare predicted and known values of Y

Here, loss = Mean Squared Error (MSE):

```
loss <- tf$reduce_mean((y_predict-y_actual)^2)
```

mean squared differences between predicted and actual Y values

Defining the optimizer function

Next, we'll define our optimizer.

Gradient Descent Optimizer:

- model learns direction (gradient) should take to reduce errors

Use `tf$train$GradientDescentOptimizer()` with the learning rate in brackets:

```
optimizer <- tf$train$GradientDescentOptimizer(0.001)
```

Minimizing MSE loss

Our final step is to minimize the MSE loss, which we'll define in a variable called `train` .

```
train <- optimizer$minimize(loss)
```

and launch the session and initialize our variables

```
sess <- tf$Session()
sess$run(tf$global_variables_initializer())
```

Let's practice!

INTRODUCTION TO TENSORFLOW IN R

Core API: linear regression part III

INTRODUCTION TO TENSORFLOW IN R

Colleen Bobbie
Yes, still your instructor

Training your linear regression model

Remember in our last lesson, we defined:

```
train <- optimizer$minimize(loss)
```

which will minimize our MSE loss. Now we'll train our model using this strategy and 2000 epochs.

```
for (step in 1:2000) {
    sess$run(train)
    if (step %% 500 == 0)
        cat("Step = ", step, "Estimate w = ", sess$run(w),
        "Estimate b =", sess$run(b))
}
```

Evaluating your linear regression model

Your final weight and bias from your training step can be accessed using:

`sess$run(w)` and `sess$run(b)`

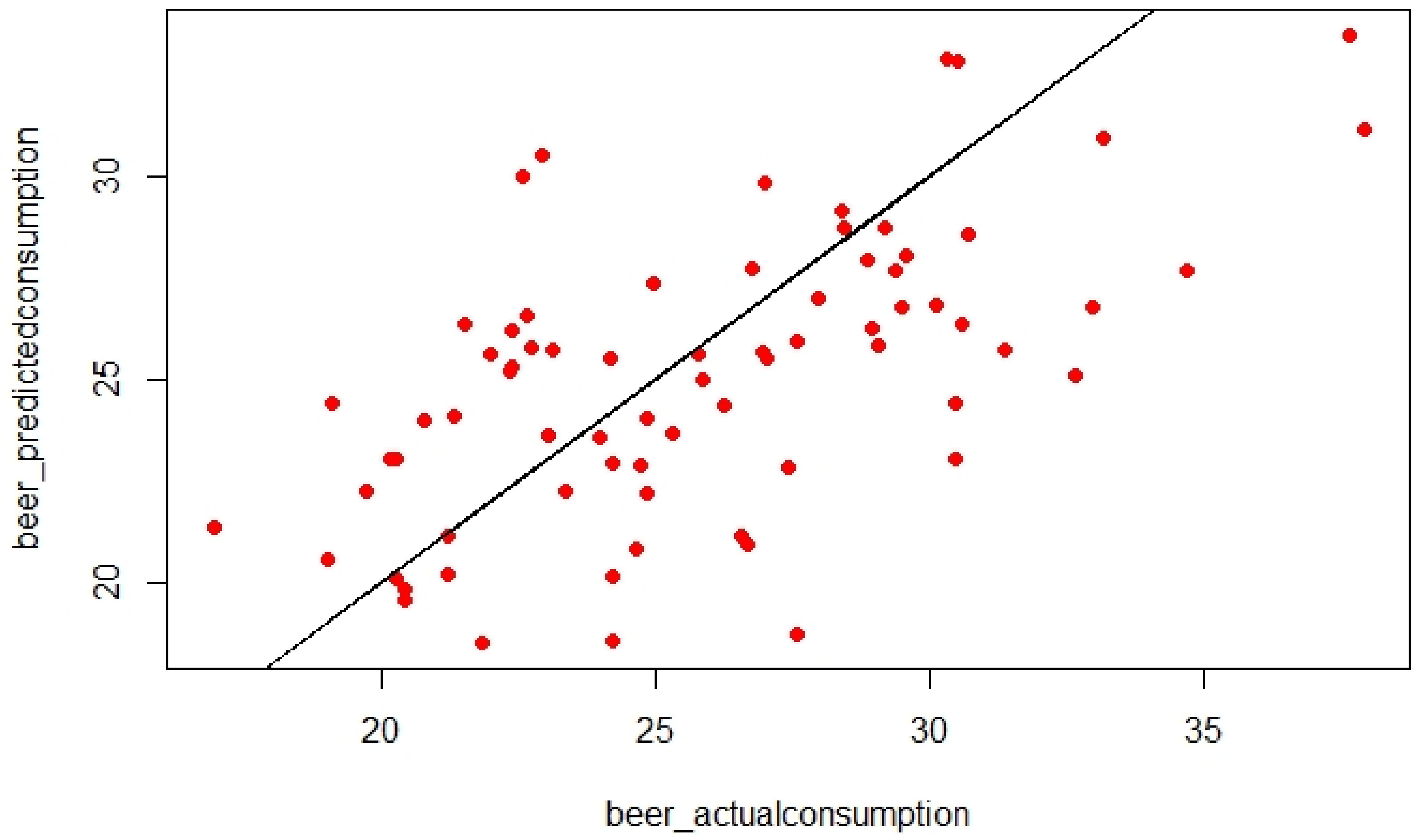
From here, we can plot the difference between the actual values and our predicted values.

```
beer_actualconsumption <- beer_consumption_test$beer_consumed
beer_predictedconsumption <- sess$run(w)*beer_consumption_test$precip+sess$run(b)
```

```
plot(beer_actualconsumption, beer_predictedconsumption)
lines(beer_actualconsumption, beer_actualconsumption)
```

Evaluating your linear regression model

...which will give you an output of something like this:



Calculate the prediction accuracy of your model

We can calculate the prediction accuracy dependent on a correlation between our actual and predicted values:

```
meandiff <- data.frame(cbind(beer_actualconsumption, beer_predictedconsumption))
```

```
correlation_accuracy <-cor(meandiff)
correlation_accuracy
```

```
                          beer_actualconsumption  beer_predictedconsumption
beer_actualconsumption                 1.0000000                  0.6018578
beer_predictedconsumption              0.6018578                  1.0000000
```

Let's practice!

INTRODUCTION TO TENSORFLOW IN R

Estimators API: multiple linear regression

INTRODUCTION TO TENSORFLOW IN R

Colleen Bobbie
Instructor

Adding a few more variables to our dataset

In our last lesson, we concluded that one variable likely isn't enough to predict beer consumption in our university town.

Let's add a few more predictor variables, including:

- Minimum Daily Temperature
- Maximum Daily Temperature
- Is day a weekend (Y/N)

Defining feature columns

The first step in creating a canned model with the Estimators API is to define feature columns.

These are your independent attributes that will be used to predict your dependent variable.

There are 10 canned types including:

`numeric_column` (for integers)

`categorical_column_with_identity` (for categorical columns, such as our `weekend` variable)

Defining feature columns

We can define feature columns using:

```
ftr_colns <- feature_columns(
    tf$feature_column$numeric_column("numericcolumnname"),
    tf$feature_column$categorical_column_with_identity(
      "categoricalcolumnname",
      NumCategories)
  )
```

Choosing your model

There are six canned models to choose from in Estimators:

- `linear_regressor` or `linear_classifier`
- `dnn_regressor` or `dnn_classifier`
- `dnn_linear_combined_regressor` or `dnn_linear_combined_classifier`

To choose a model:

```
nameofyourmodel <- linear_regressor(feature_columns = ftr_colns)
```

Defining input function

Estimators also needs an input function, which defines your dataset, your features variables, and your response variable.

This looks like:

```
nameofyourinputfunction = function(data){
    input_fn(data,
        features = c("feature1", "feature2"...),
        response = "responsevariable")
}
```

Note that `data` is a constant and will be input later when we run the function.

Training and evaluating your model

To train your model, simply input your model name, the input function you defined earlier, and the name of your training dataset.

```
train(nameofyourmodel,
    nameofyourinputfunction(trainingdatasetname))
```

To evaluate your model, replace the `train` above, with `evaluate`, make sure to specify your testing dataset and save as a new variable:

```
modeleval <- evaluate(nameofyourmodel,
    nameofyourinputfunction(testingdatasetname))
modeleval
```

Let's practice!

INTRODUCTION TO TENSORFLOW IN R