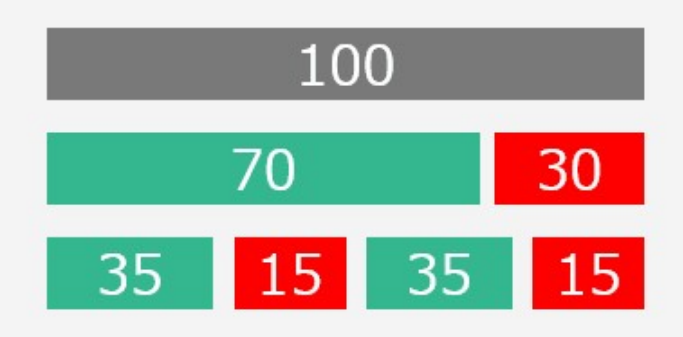
In this post, I will explain you why one should run simulations on their model training process so the models don't fail in production. Traditionally we are always used to training models at certain split ratio's of say, 70:30 or 80:20. The fundamental issue with this is that we don't always train models on different parts of data in those splits as shown in below image.



Hence, it becomes vital to make sure one trains their model with various scenarios to make sure the model is not biased. This also ensures that the model is reliable and robust enough to deploy it into production.

Below we will go over an example on how this all comes together. The steps are as follows:

1. load libraries
2. load mtcars data set
3. write a function to split data at different degrees
4. run simulation in a loop to get error rates
5. Finally visualizing the result

We will first begin with loading libraries and our data set as shown below

```
# load libraries
library(MASS)
library(e1071)
library(Metrics)
library(dplyr)

# load data
data(mtcars)
```

Next, we will write a function that includes

- set seed value. This is because we want to capture new data every time (Duh! that the whole point of this simulation)
- split the data in to train and test at various ratios
- build an SVM model using train data
- do predictions on test data
- calculate & return error value (MAE)

```
# function to run simulation
runsimulation = function(i, split){

  seedValue = i*rnorm(1)

  # change seed values
  set.seed(seedValue)

  # create samples
  samples = sample(1:nrow(mtcars), split*nrow(mtcars))
```

```
  # split data to test and train
  train = mtcars[samples, ]
  test = mtcars[-samples, ]

  # build a model
  model = svm(mpg ~ ., data  = train, scale = F, kernel = "radial")

  # do predictions
  prediction = predict(model, test %>% select(-mpg))

  # calculate error
  error = mae(actual = test$mpg, predicted = prediction)

  # return error values
  return(error)

}
```

We will create a sequence of split ratios and then run these ratios in the loop. For each split ratio, we will run around 300 runs.

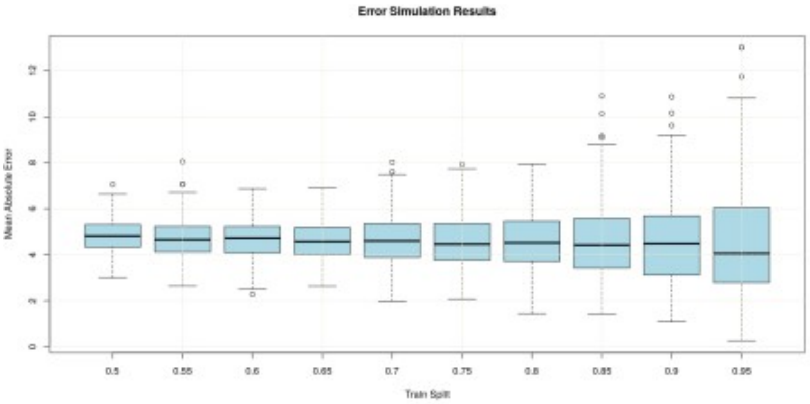
```
# create split ratios
split = seq(from = 0.5, to = 0.95, by = 0.05) %>% rep(300) %>% sort(decreasing =
FALSE)

# get the length of i for seed values
i = 1:length(split)

# get errors
errors = mapply(runsimulation, i = i, split = split)

# put data into a data frame
simResults = data.frame(split, errors)
```

Finally, we visualize the data and look at the results. In the below box plot we can see that the median decreases as the split ratio increases. This should be true as we are feeding in more data to the model. We also notice that the minimum error decreases as we add more data while training. This also increases the max errors. We can notice similar observation for quantile as well.



Next, we will look at the summary of mean and variance for each split ratios. We notice that the least average error is with 95% split and also comes with higher degree of SD. and vice versa.

```
# plot results
boxplot(errors~split,
data = simResults,
```

```
main = "Error Simulation Results",
xlab = "Train Split",
ylab = "Mean Absolute Error",
col = "light blue")
grid (NULL,NULL, lty = 6, col = "cornsilk2")

simResults %>%
  group_by(split) %>%
  summarise(mean = mean(errors), sd = sd(errors)) %>%
  data.frame
#     split   mean      sd
# 1   0.50 4.826838 0.7090876
# 2   0.55 4.701303 0.8178482
# 3   0.60 4.674690 0.8442144
# 4   0.65 4.645363 0.8727532
# 5   0.70 4.652534 1.0769249
# 6   0.75 4.555186 1.1046217
# 7   0.80 4.588761 1.3002216
# 8   0.85 4.572775 1.6021275
# 9   0.90 4.519118 1.7865828
# 10  0.95 4.443357 2.4188333
```

At this point, its up to the decision maker to decide what model one should go for. Can they afford significant variations in error rates or want to control the variance of error rate. If I was the decision maker, I would go with either 65% or 70% split and control that variance in error.

In conclusion, machine learning is hard. Its not as simple as fitting a model with data. You need to run simulations as above to analyze your models. The above is the most simplest case you could come across. Once you get to hyper parameters, it gets even more complicated. There is not one set of tools or flows that works for all. You sometimes need to get creative and come up with your own flows.