Most of us have come across situations where, we do not have enough data for building reliable models due to various reasons such as, it’s expensive to collect data (human studies), limited resources, lack of historical data availability (earth quakes). Even before we begin talking about how to overcome the challenge, let’s first talk about why we need minimum samples even before we consider building model. First of all, we can build a model with low samples. It is definitely possible! But, the as the number of samples decreases, the margin of error increases and vice versa. If you want to build a model with the highest accuracy you would need to have as many samples as possible. If the model is for a real world application, then you need to have data across multiple days to account for any changes in the system. There is a formula that can be used to calculate the sample size and is as follows:

Where, n = sample size

Z = Z-score value

σ = populated standard deviation

MOE = acceptable margin of error

Now we know that why minimum samples are required for achieving required accuracy, say in some case we do not have an opportunity to collect more samples or available. Then we have an option to do the following

1. K-fold cross validation
2. Leave-P-out cross validation
3. Leave-one-out cross validation
4. New data creation through estimation

In K-fold method, the data is split into k partitions and then is trained with each partition and tested with the left out kth partition. In k-hold method, not all combinations are considered. Only user specified partitions are considered. While in leave-one/p-out, all combinations or partitions are considered. This is more exhaustive technique in validating the results. The following above two techniques are the most popular techniques that is used both in machine learning and deep learning.

When it comes to handling NA’s in a data set we have always imputed it through mean, median, zero’s and random numbers. But, this would probably not make sense when we want to create new data.

In new data creation through estimation technique, rows of missing data is created in the data set and a separate data imputation model is used to impute missing data in the rows. Multivariate Imputation by Chained Equations (MICE) is one of the most popular algorithms that are available to insert missing data irrespective of data types such as mixes of continuous, binary, unordered categorical and ordered categorical data.

There are various tutorials available for k-fold and leave one out models. This tutorial will focus on the fourth model where new data will be created to handle less sample size. In the and a simple classification model with be trained to see if there was a significant improvement. Also, distribution of imputed and non-imputed data will be compared to see any significant difference.

**Load libraries**

Let’s load all the libraries needed for now.

options(warn=-1)

# load libraies

library(mice)

library(dplyr)

**Load data into a data frame**

The data available and attached in my repository as used for the analysis.

setwd("C:/OpenSourceWork/Experiment")

#read csv files

file1 = read.csv("dry run.csv", sep=",", header =T)

file2 = read.csv("base.csv", sep=",", header =T)

file3 = read.csv("imbalance 1.csv", sep=",", header =T)

file4 = read.csv("imbalance 2.csv", sep=",", header =T)

#Add labels to data

file1$y = 1

file2$y = 2

file3$y = 3

file4$y = 4

#view top rows of data

head(file1)

| **time** | **ax** | **ay** | **az** | **aT** | **y** |
| --- | --- | --- | --- | --- | --- |
| 0.002 | -0.3246 | 0.2748 | 0.1502 | 0.451 | 1 |
| 0.009 | 0.6020 | -0.1900 | -0.3227 | 0.709 | 1 |
| 0.019 | 0.9787 | 0.3258 | 0.0124 | 1.032 | 1 |
| 0.027 | 0.6141 | -0.4179 | 0.0471 | 0.744 | 1 |
| 0.038 | -0.3218 | -0.6389 | -0.4259 | 0.833 | 1 |
| 0.047 | -0.3607 | 0.1332 | -0.1291 | 0.406 | 1 |

Raw data

**Create some features from data**

The data used in this study is vibration data with different states. The data was collected at 100 Hz. The data to be used as is is high dimensional also, we do not have any good summary of the data. Hence, some statistical features are extracted. In this case, sample standard deviation, sample mean, sample min, sample max and sample median is calculated. Also, the data is aggregated by 1 second.

file1$group = as.factor(round(file1$time))

file2$group = as.factor(round(file2$time))

file3$group = as.factor(round(file3$time))

file4$group = as.factor(round(file4$time))

#(file1,20)

#list of all files

files = list(file1, file2, file3, file4)

#loop through all files and combine

features = NULL

for (i in 1:4){

res = files[[i]] %>%

group\_by(group) %>%

summarize(ax\_mean = mean(ax),

ax\_sd = sd(ax),

ax\_min = min(ax),

ax\_max = max(ax),

ax\_median = median(ax),

ay\_mean = mean(ay),

ay\_sd = sd(ay),

ay\_min = min(ay),

ay\_may = max(ay),

ay\_median = median(ay),

az\_mean = mean(az),

az\_sd = sd(az),

az\_min = min(az),

az\_maz = max(az),

az\_median = median(az),

aT\_mean = mean(aT),

aT\_sd = sd(aT),

aT\_min = min(aT),

aT\_maT = max(aT),

aT\_median = median(aT),

y = mean(y)

)

features = rbind(features, res)

}

features = subset(features, select = -group)

# store it in a df for future reference

actual.features = features

**Study data**

First, lets look at the size of our populations and summary of our features along with their data types.

# show data types

str(features)

Classes 'tbl\_df', 'tbl' and 'data.frame': 362 obs. of 21 variables:

$ ax\_mean : num -0.03816 -0.00581 0.06985 0.01155 0.04669 ...

$ ax\_sd : num 0.659 0.633 0.667 0.551 0.643 ...

$ ax\_min : num -1.26 -1.62 -1.46 -1.93 -1.78 ...

$ ax\_max : num 1.38 1.19 1.47 1.2 1.48 ...

$ ax\_median: num -0.0955 -0.0015 0.107 0.0675 0.0836 ...

$ ay\_mean : num -0.068263 0.003791 0.074433 0.000826 -0.017759 ...

$ ay\_sd : num 0.751 0.782 0.802 0.789 0.751 ...

$ ay\_min : num -1.39 -1.56 -1.48 -2 -1.66 ...

$ ay\_may : num 1.64 1.54 1.8 1.56 1.44 ...

$ ay\_median: num -0.19 0.0101 0.1186 -0.0027 -0.0253 ...

$ az\_mean : num -0.138 -0.205 -0.0641 -0.0929 -0.1399 ...

$ az\_sd : num 0.985 0.925 0.929 0.889 0.927 ...

$ az\_min : num -2.68 -3.08 -1.82 -2.16 -1.85 ...

$ az\_maz : num 2.75 2.72 2.49 3.24 3.55 ...

$ az\_median: num 0.0254 -0.2121 -0.1512 -0.1672 -0.1741 ...

$ aT\_mean : num 1.27 1.26 1.3 1.2 1.23 ...

$ aT\_sd : num 0.583 0.545 0.513 0.513 0.582 ...

$ aT\_min : num 0.4 0.41 0.255 0.393 0.313 0.336 0.275 0.196 0.032 0.358 ...

$ aT\_maT : num 3.03 3.2 2.64 3.32 3.6 ...

$ aT\_median: num 1.08 1.14 1.28 1.12 1.17 ...

$ y : num 1 1 1 1 1 1 1 1 1 1 ...

**Create observations with NA values in the end**

Next, we will impute some NA’s for this tutorial purpose at the end of the table.

features1 = features

for(i in 363:400){

features1[i,] = NA

}

**View at bottom 50 rows**

We see the missing values at the end of the table.

Disclaimer: here we introducing all of last 50 rows as NA. In real world, its highly unlikely. You might have only few values missing.

tail(features1, 50)

| **ax\_mean** | **ax\_sd** | **ax\_min** | **ax\_max** | **ax\_median** | **ay\_mean** | **ay\_sd** | **ay\_min** | **ay\_may** | **ay\_median** | **…** | **az\_sd** | **az\_min** | **az\_maz** | **az\_median** | **aT\_mean** | **aT\_sd** | **aT\_min** | **aT\_maT** | **aT\_median** | **y** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| -0.016097030 | 0.8938523 | -2.3445 | 2.3006 | -0.07360 | -0.009759406 | 1.311817 | -3.4215 | 2.5028 | 0.10890 | … | 1.264572 | -2.8751 | 3.3718 | -0.07070 | 1.866030 | 0.7808319 | 0.380 | 4.098 | 1.8200 | 4 |
| -0.015565347 | 0.8956615 | -2.2661 | 2.5089 | 0.08640 | 0.027313861 | 1.294063 | -2.9421 | 2.3497 | 0.15260 | … | 1.368576 | -3.3165 | 2.6989 | -0.01660 | 1.930426 | 0.7749686 | 0.127 | 4.463 | 1.8350 | 4 |
| 0.024006250 | 0.8653758 | -2.4099 | 2.5328 | -0.03170 | 0.008440625 | 1.376398 | -3.0422 | 2.3727 | 0.11390 | … | 1.449783 | -4.2171 | 4.7703 | 0.00110 | 2.003552 | 0.8300253 | 0.387 | 5.138 | 1.9920 | 4 |
| -0.015563000 | 0.8720967 | -2.3451 | 2.3269 | -0.05325 | 0.013962000 | 1.240091 | -3.1360 | 2.8563 | 0.09145 | … | 1.418988 | -3.3758 | 3.4279 | -0.10410 | 1.895380 | 0.8351505 | 0.173 | 4.458 | 1.8735 | 4 |
| 0.003894898 | 0.8806773 | -2.3098 | 3.1902 | -0.09260 | 0.022575510 | 1.301955 | -3.2561 | 2.7833 | -0.05380 | … | 1.271799 | -3.8035 | 3.1323 | -0.26115 | 1.852265 | 0.7909640 | 0.436 | 3.944 | 1.7570 | 4 |
| -0.039379208 | 0.8127135 | -2.1523 | 1.8828 | -0.11250 | 0.005454455 | 1.189519 | -2.8057 | 2.4852 | 0.03040 | … | 1.366368 | -3.3928 | 2.4507 | 0.05430 | 1.828059 | 0.7562042 | 0.580 | 3.573 | 1.6960 | 4 |
| 0.021469000 | 0.8272527 | -1.5895 | 3.7505 | -0.08995 | 0.011312000 | 1.285206 | -2.7423 | 2.6785 | -0.03640 | … | 1.177012 | -2.6649 | 2.1685 | 0.02755 | 1.785930 | 0.7120829 | 0.298 | 3.895 | 1.7575 | 4 |
| 0.005917000 | 0.9139808 | -2.3310 | 2.8131 | -0.07800 | -0.040868000 | 1.320873 | -2.9778 | 2.2841 | -0.01435 | … | 1.401567 | -3.3728 | 3.3165 | 0.19485 | 1.947570 | 0.8513573 | 0.397 | 4.191 | 1.8180 | 4 |
| -0.034448571 | 0.8640626 | -2.4917 | 2.4113 | -0.01960 | -0.013410476 | 1.235196 | -3.3305 | 2.4912 | 0.09420 | … | 1.327886 | -2.9864 | 2.8430 | -0.05300 | 1.882590 | 0.6971337 | 0.370 | 3.775 | 1.9030 | 4 |
| 0.046837374 | 0.9776022 | -1.8688 | 2.6644 | -0.03600 | 0.019817172 | 1.293644 | -2.7836 | 2.6166 | 0.12540 | … | 1.245906 | -2.4813 | 3.2677 | -0.11460 | 1.901646 | 0.7296095 | 0.283 | 3.813 | 1.8440 | 4 |
| -0.014453061 | 0.9553743 | -2.7118 | 2.4640 | -0.01000 | -0.037717347 | 1.285358 | -3.1225 | 2.4506 | 0.03085 | … | 1.457232 | -4.2512 | 3.3754 | 0.09325 | 1.984418 | 0.8511168 | 0.446 | 4.351 | 1.8600 | 4 |
| 0.046810870 | 0.9259427 | -1.5309 | 1.9420 | -0.11455 | 0.230676087 | 1.491983 | -2.8435 | 2.8405 | 0.33060 | … | 1.111205 | -2.1748 | 2.9009 | -0.03790 | 1.927174 | 0.7622031 | 0.491 | 3.355 | 2.1620 | 4 |
| NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | … | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA |
| NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | … | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA |
| NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | … | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA |
| NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | … | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA |
| NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | … | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA |
| NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | … | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA |
| NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | … | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA |
| NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | … | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA |
| NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | … | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA |
| NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | … | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA |
| NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | … | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA |
| NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | … | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA |
| NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | … | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA |
| NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | … | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA |
| NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | … | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA |
| NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | … | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA |
| NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | … | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA |
| NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | … | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA |
| NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | … | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA |
| NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | … | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA |
| NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | … | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA |
| NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | … | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA |
| NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | … | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA |
| NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | … | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA |
| NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | … | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA |
| NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | … | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA |
| NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | … | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA |
| NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | … | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA |
| NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | … | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA |
| NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | … | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA |
| NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | … | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA |
| NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | … | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA |
| NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | … | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA |
| NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | … | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA |
| NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | … | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA |
| NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | … | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA |
| NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | … | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA |
| NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | … | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA |

**Impute NA’s with best values using iteration method**

Next, to impute missing values we will use mice function. We will keep max iterations to 50 and method as ‘pmm’.

imputed\_Data = mice(features1,

m=1,

maxit = 50,

method = 'pmm',

seed = 999,

printFlag =FALSE)

**View imputed results**

Now we have imputed results. We will use the first imputed data frame for this study. You could actually test all the different imputations to see which works better.

imputedResultData = mice::complete(imputed\_Data,1)

tail(imputedResultData, 50)

|  | **ax\_mean** | **ax\_sd** | **ax\_min** | **ax\_max** | **ax\_median** | **ay\_mean** | **ay\_sd** | **ay\_min** | **ay\_may** | **ay\_median** | **…** | **az\_sd** | **az\_min** | **az\_maz** | **az\_median** | **aT\_mean** | **aT\_sd** | **aT\_min** | **aT\_maT** | **aT\_median** | **y** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **351** | -0.016097030 | 0.8938523 | -2.3445 | 2.3006 | -0.07360 | -0.009759406 | 1.3118166 | -3.4215 | 2.5028 | 0.10890 | … | 1.2645719 | -2.8751 | 3.3718 | -0.07070 | 1.8660297 | 0.7808319 | 0.380 | 4.098 | 1.8200 | 4 |
| **352** | -0.015565347 | 0.8956615 | -2.2661 | 2.5089 | 0.08640 | 0.027313861 | 1.2940627 | -2.9421 | 2.3497 | 0.15260 | … | 1.3685757 | -3.3165 | 2.6989 | -0.01660 | 1.9304257 | 0.7749686 | 0.127 | 4.463 | 1.8350 | 4 |
| **353** | 0.024006250 | 0.8653758 | -2.4099 | 2.5328 | -0.03170 | 0.008440625 | 1.3763983 | -3.0422 | 2.3727 | 0.11390 | … | 1.4497833 | -4.2171 | 4.7703 | 0.00110 | 2.0035521 | 0.8300253 | 0.387 | 5.138 | 1.9920 | 4 |
| **354** | -0.015563000 | 0.8720967 | -2.3451 | 2.3269 | -0.05325 | 0.013962000 | 1.2400913 | -3.1360 | 2.8563 | 0.09145 | … | 1.4189884 | -3.3758 | 3.4279 | -0.10410 | 1.8953800 | 0.8351505 | 0.173 | 4.458 | 1.8735 | 4 |
| **355** | 0.003894898 | 0.8806773 | -2.3098 | 3.1902 | -0.09260 | 0.022575510 | 1.3019546 | -3.2561 | 2.7833 | -0.05380 | … | 1.2717989 | -3.8035 | 3.1323 | -0.26115 | 1.8522653 | 0.7909640 | 0.436 | 3.944 | 1.7570 | 4 |
| **356** | -0.039379208 | 0.8127135 | -2.1523 | 1.8828 | -0.11250 | 0.005454455 | 1.1895194 | -2.8057 | 2.4852 | 0.03040 | … | 1.3663678 | -3.3928 | 2.4507 | 0.05430 | 1.8280594 | 0.7562042 | 0.580 | 3.573 | 1.6960 | 4 |
| **357** | 0.021469000 | 0.8272527 | -1.5895 | 3.7505 | -0.08995 | 0.011312000 | 1.2852056 | -2.7423 | 2.6785 | -0.03640 | … | 1.1770121 | -2.6649 | 2.1685 | 0.02755 | 1.7859300 | 0.7120829 | 0.298 | 3.895 | 1.7575 | 4 |
| **358** | 0.005917000 | 0.9139808 | -2.3310 | 2.8131 | -0.07800 | -0.040868000 | 1.3208731 | -2.9778 | 2.2841 | -0.01435 | … | 1.4015674 | -3.3728 | 3.3165 | 0.19485 | 1.9475700 | 0.8513573 | 0.397 | 4.191 | 1.8180 | 4 |
| **359** | -0.034448571 | 0.8640626 | -2.4917 | 2.4113 | -0.01960 | -0.013410476 | 1.2351957 | -3.3305 | 2.4912 | 0.09420 | … | 1.3278861 | -2.9864 | 2.8430 | -0.05300 | 1.8825905 | 0.6971337 | 0.370 | 3.775 | 1.9030 | 4 |
| **360** | 0.046837374 | 0.9776022 | -1.8688 | 2.6644 | -0.03600 | 0.019817172 | 1.2936436 | -2.7836 | 2.6166 | 0.12540 | … | 1.2459059 | -2.4813 | 3.2677 | -0.11460 | 1.9016465 | 0.7296095 | 0.283 | 3.813 | 1.8440 | 4 |
| **361** | -0.014453061 | 0.9553743 | -2.7118 | 2.4640 | -0.01000 | -0.037717347 | 1.2853576 | -3.1225 | 2.4506 | 0.03085 | … | 1.4572321 | -4.2512 | 3.3754 | 0.09325 | 1.9844184 | 0.8511168 | 0.446 | 4.351 | 1.8600 | 4 |
| **362** | 0.046810870 | 0.9259427 | -1.5309 | 1.9420 | -0.11455 | 0.230676087 | 1.4919834 | -2.8435 | 2.8405 | 0.33060 | … | 1.1112049 | -2.1748 | 2.9009 | -0.03790 | 1.9271739 | 0.7622031 | 0.491 | 3.355 | 2.1620 | 4 |
| **363** | 0.011238614 | 0.8127502 | -1.9602 | 2.1430 | 0.00680 | -0.013367308 | 1.3019546 | -3.0628 | 2.7338 | 0.00070 | … | 1.4534581 | -4.4325 | 2.9648 | -0.03520 | 1.9383000 | 0.8526128 | 0.373 | 4.351 | 1.8705 | 4 |
| **364** | -0.009812264 | 0.7680463 | -2.3492 | 1.3919 | 0.03110 | 0.013984158 | 0.6084791 | -1.4155 | 0.9273 | 0.11860 | … | 0.9997898 | -3.0031 | 3.5781 | -0.25930 | 1.2219510 | 0.6450616 | 0.233 | 3.603 | 1.0730 | 1 |
| **365** | -0.026760000 | 0.4780558 | -1.1826 | 0.9934 | 0.05560 | -0.035218269 | 0.5632648 | -1.0761 | 1.2307 | -0.08165 | … | 0.7635922 | -2.3115 | 1.8934 | 0.03005 | 0.9714200 | 0.4214891 | 0.214 | 2.180 | 0.9265 | 1 |
| **366** | 0.029083000 | 0.7515921 | -2.2628 | 2.4640 | -0.00820 | 0.011159596 | 1.3073606 | -3.1360 | 2.8527 | 0.04010 | … | 1.4534581 | -3.6751 | 2.6187 | -0.22680 | 1.9367549 | 0.7439326 | 0.354 | 4.156 | 1.8450 | 4 |
| **367** | 0.002401000 | 0.5641062 | -1.1533 | 1.4479 | -0.04215 | 0.011159596 | 1.0358946 | -1.9856 | 2.9217 | -0.07040 | … | 0.7141977 | -1.7791 | 1.3013 | -0.20785 | 1.2607358 | 0.4523664 | 0.376 | 2.106 | 1.2830 | 4 |
| **368** | 0.017670707 | 0.4158231 | -0.9785 | 1.0647 | 0.07680 | -0.026719608 | 0.4759174 | -0.9340 | 0.9077 | -0.03650 | … | 0.6919936 | -1.6094 | 2.0555 | -0.19365 | 0.8742105 | 0.3962710 | 0.230 | 2.123 | 0.8120 | 1 |
| **369** | -0.078038776 | 0.4413032 | -1.1099 | 0.9826 | -0.03910 | -0.010626042 | 0.4768587 | -0.9392 | 0.8497 | -0.04655 | … | 0.8165436 | -2.2936 | 2.1036 | -0.29570 | 0.9319524 | 0.4517633 | 0.193 | 2.380 | 0.8865 | 2 |
| **370** | 0.004372632 | 0.8352791 | -1.6966 | 2.3897 | 0.00845 | -0.010064000 | 1.2746954 | -2.7832 | 2.2841 | 0.03085 | … | 1.2177225 | -3.1289 | 3.0919 | 0.01905 | 1.7844653 | 0.7343952 | 0.489 | 3.764 | 1.7520 | 3 |
| **371** | 0.016103000 | 0.3997476 | -0.9537 | 1.1546 | 0.03655 | -0.031622772 | 0.4828770 | -0.9772 | 1.1237 | -0.14540 | … | 0.7672163 | -1.9821 | 1.8173 | -0.09240 | 0.9053800 | 0.4160549 | 0.201 | 2.053 | 0.8520 | 2 |
| **372** | -0.020355446 | 0.4178729 | -1.0524 | 0.9076 | -0.09340 | 0.044400000 | 0.5439558 | -0.9843 | 1.0798 | 0.14000 | … | 0.7552593 | -2.0607 | 1.6134 | -0.17990 | 0.9498911 | 0.3846176 | 0.222 | 1.752 | 0.8950 | 1 |
| **373** | 0.001363636 | 0.4868077 | -0.9027 | 1.5155 | 0.04820 | 0.031339000 | 1.0619675 | -2.3261 | 2.4081 | -0.00210 | … | 0.7598489 | -1.7482 | 1.3013 | -0.20075 | 1.3272772 | 0.4315494 | 0.478 | 2.288 | 1.3220 | 4 |
| **374** | -0.008122222 | 0.8831968 | -1.9394 | 3.3244 | -0.09610 | 0.017400971 | 1.3778757 | -3.7580 | 2.4527 | 0.16935 | … | 1.4260617 | -3.1893 | 3.5781 | 0.09325 | 1.9576857 | 0.9167571 | 0.295 | 4.830 | 1.9430 | 4 |
| **375** | -0.065401010 | 0.8489219 | -2.4871 | 2.1672 | -0.11250 | -0.043491753 | 0.5648206 | -1.5188 | 0.8497 | 0.05440 | … | 1.4259974 | -3.1893 | 4.6557 | 0.08010 | 1.4950297 | 0.8012418 | 0.198 | 4.290 | 1.2550 | 1 |
| **376** | 0.039720000 | 0.5946125 | -1.5250 | 1.7390 | 0.05040 | 0.061424510 | 0.8133879 | -1.2303 | 1.6255 | 0.05660 | … | 0.9355264 | -2.2936 | 2.9202 | 0.02420 | 1.2507900 | 0.5391791 | 0.294 | 3.081 | 1.1770 | 3 |
| **377** | 0.022841000 | 0.8646867 | -2.1253 | 2.6378 | 0.05720 | 0.052515306 | 1.1332836 | -2.5429 | 2.3692 | 0.10620 | … | 1.0360114 | -3.0924 | 3.0590 | 0.00110 | 1.5811275 | 0.7053254 | 0.326 | 3.742 | 1.5815 | 3 |
| **378** | -0.001924510 | 0.5975310 | -1.4775 | 1.4089 | -0.11455 | -0.040868000 | 1.0363392 | -2.3289 | 2.2123 | 0.03025 | … | 0.7546022 | -1.6175 | 1.2922 | -0.18510 | 1.3324845 | 0.5131552 | 0.305 | 2.091 | 1.2830 | 4 |
| **379** | 0.017975000 | 0.4780750 | -1.2011 | 1.4923 | -0.07450 | -0.022319802 | 0.5072372 | -1.1404 | 1.0361 | -0.04135 | … | 0.7439169 | -2.0052 | 1.7066 | -0.09450 | 0.9151400 | 0.4541700 | 0.262 | 2.264 | 0.8270 | 2 |
| **380** | -0.070804000 | 0.4780558 | -1.9254 | 0.9244 | -0.05830 | -0.074927551 | 0.5037149 | -1.0485 | 1.0710 | -0.07750 | … | 0.7598489 | -2.1735 | 2.0385 | -0.24560 | 0.9281400 | 0.4813814 | 0.150 | 2.084 | 0.7900 | 2 |
| **381** | -0.002204762 | 0.9310547 | -2.7832 | 2.5242 | -0.07875 | -0.019305882 | 1.3019546 | -2.4215 | 2.8615 | -0.02880 | … | 1.1771775 | -3.0903 | 2.4800 | -0.19155 | 1.8377451 | 0.7254306 | 0.377 | 3.348 | 1.7770 | 4 |
| **382** | 0.021469000 | 0.8646867 | -2.0001 | 2.4477 | -0.03400 | 0.051977895 | 1.3628383 | -2.6574 | 2.7414 | 0.15305 | … | 1.1474602 | -2.9516 | 2.6371 | 0.08870 | 1.7884124 | 0.7520192 | 0.400 | 3.651 | 1.9180 | 4 |
| **383** | -0.015468354 | 0.8127502 | -2.2034 | 2.3405 | -0.02150 | 0.046179798 | 1.3628383 | -2.8594 | 2.7288 | 0.02130 | … | 1.1112049 | -4.2171 | 1.7215 | 0.09600 | 1.7592828 | 0.7680118 | 0.295 | 3.671 | 1.7780 | 4 |
| **384** | -0.002143000 | 0.4442709 | -0.9949 | 1.0734 | -0.04265 | -0.007904000 | 0.5386439 | -1.2828 | 1.2250 | -0.06765 | … | 0.7335329 | -2.2694 | 2.1640 | -0.30150 | 0.9293627 | 0.4517633 | 0.266 | 2.407 | 0.8000 | 2 |
| **385** | 0.027587129 | 0.4551125 | -1.2785 | 1.0285 | 0.05660 | -0.035263725 | 0.4854652 | -1.0143 | 1.1332 | -0.03650 | … | 0.7048400 | -2.1237 | 1.8689 | 0.11100 | 0.8571800 | 0.4493956 | 0.164 | 2.222 | 0.8120 | 2 |
| **386** | 0.017670707 | 0.6981887 | -1.5387 | 2.1808 | -0.04500 | 0.043603191 | 1.2152972 | -2.6631 | 3.1973 | 0.09380 | … | 0.8017314 | -1.6094 | 1.2922 | -0.10680 | 1.4910700 | 0.5158915 | 0.376 | 2.428 | 1.5820 | 4 |
| **387** | 0.017401000 | 0.7680463 | -1.4528 | 2.2822 | -0.00350 | 0.055612871 | 1.0989870 | -2.7737 | 2.3134 | 0.16785 | … | 1.0468209 | -2.8051 | 1.7055 | -0.01470 | 1.5737525 | 0.6825190 | 0.428 | 2.988 | 1.5810 | 4 |
| **388** | 0.001363636 | 0.4354711 | -1.0677 | 0.9579 | 0.03655 | -0.017115842 | 0.5501718 | -1.1134 | 1.0798 | -0.01640 | … | 0.7466890 | -2.1237 | 2.0555 | 0.02230 | 0.9342100 | 0.4437911 | 0.266 | 2.222 | 0.8410 | 1 |
| **389** | 0.036087000 | 0.8741671 | -2.2967 | 3.3393 | -0.03330 | -0.019919792 | 1.4065464 | -2.9778 | 3.0511 | -0.04680 | … | 1.2155255 | -3.8281 | 1.9302 | 0.08820 | 1.8953800 | 0.7778120 | 0.242 | 4.098 | 1.9170 | 4 |
| **390** | 0.007588000 | 0.8409728 | -1.9602 | 2.2383 | -0.07985 | 0.025797000 | 1.3525870 | -3.1511 | 2.7414 | -0.02135 | … | 1.4189884 | -3.6947 | 2.7486 | -0.14945 | 1.9648889 | 0.8489206 | 0.397 | 3.963 | 1.8600 | 4 |
| **391** | 0.065754545 | 0.4533416 | -0.7769 | 1.1179 | 0.10470 | 0.047955446 | 0.5539467 | -0.9340 | 1.0356 | 0.03360 | … | 0.7569361 | -2.1362 | 2.3655 | -0.10495 | 0.9663913 | 0.4276036 | 0.285 | 2.353 | 0.8930 | 2 |
| **392** | -0.030526733 | 0.4442709 | -1.7119 | 1.0302 | 0.03000 | -0.021866667 | 0.6103892 | -1.0198 | 1.6418 | -0.01105 | … | 1.4149706 | -3.3599 | 5.0202 | -0.11600 | 1.3062900 | 0.7562042 | 0.131 | 4.443 | 1.1075 | 1 |
| **393** | -0.001643000 | 0.8086920 | -1.9033 | 2.5242 | -0.03200 | -0.033747959 | 1.3111909 | -3.0231 | 2.3208 | 0.01690 | … | 1.1671442 | -3.7451 | 2.0425 | -0.19155 | 1.7976224 | 0.7133729 | 0.326 | 3.651 | 1.7310 | 4 |
| **394** | -0.023916346 | 0.4139117 | -0.6977 | 1.1179 | -0.04360 | 0.011312000 | 0.4828770 | -1.2828 | 1.1237 | 0.04940 | … | 0.7135787 | -1.9553 | 1.8769 | -0.23950 | 0.8609714 | 0.4064190 | 0.054 | 2.031 | 0.7900 | 2 |
| **395** | 0.037914706 | 0.4369138 | -0.9701 | 0.9937 | 0.07080 | -0.011703810 | 0.4883374 | -1.0822 | 1.1166 | -0.08405 | … | 0.7141977 | -1.9285 | 2.0766 | 0.08010 | 0.8621584 | 0.4222442 | 0.193 | 2.180 | 0.7910 | 2 |
| **396** | -0.024820792 | 0.8127135 | -1.9299 | 2.6378 | 0.01800 | -0.044580000 | 1.1363141 | -2.5429 | 2.4081 | -0.12910 | … | 1.0066063 | -2.4043 | 1.5056 | -0.12860 | 1.6121359 | 0.5853224 | 0.052 | 2.517 | 1.6945 | 4 |
| **397** | -0.016237500 | 0.7620745 | -2.4099 | 1.7855 | -0.05150 | 0.032355102 | 1.1534694 | -2.6734 | 2.4506 | 0.07725 | … | 1.4259974 | -4.1238 | 4.2297 | -0.24790 | 1.7976224 | 0.9082928 | 0.212 | 5.397 | 1.6595 | 3 |
| **398** | -0.039379208 | 0.5614528 | -1.7119 | 1.4600 | -0.11620 | -0.032463000 | 1.1096189 | -2.4111 | 2.4533 | -0.09910 | … | 1.1076786 | -3.1215 | 2.2947 | -0.14000 | 1.5025833 | 0.7521618 | 0.168 | 3.790 | 1.4420 | 3 |
| **399** | 0.026206186 | 0.7980083 | -1.9033 | 2.3863 | 0.00210 | 0.009870874 | 1.2557210 | -2.8507 | 2.4343 | 0.13105 | … | 1.2135140 | -2.5112 | 2.1638 | -0.22680 | 1.7924158 | 0.6828006 | 0.397 | 3.197 | 1.7150 | 3 |
| **400** | 0.072777778 | 0.4051881 | -0.8386 | 0.8847 | 0.15575 | 0.015370408 | 0.4759174 | -0.9340 | 1.2039 | 0.01090 | … | 0.7135787 | -2.1186 | 1.5632 | -0.13970 | 0.9087400 | 0.3767882 | 0.170 | 2.507 | 0.8120 | 1 |

**Looking at distribution actual data and imputed data**

We will first compare basic statistics and then distributions of the couple of features. In the comparison of statistics between actual and imputed we can observe that the mean and SD for both imputed and actual are almost equal.

data.frame(actual\_ax\_mean = c(mean(features$ax\_mean), sd(features$ax\_mean))

, imputed\_ax\_mean = c(mean(imputedResultData$ax\_mean), sd(imputedResultData$ax\_mean))

, actual\_ax\_median = c(mean(features$ax\_median), sd(features$ax\_median))

, imputed\_ax\_median = c(mean(imputedResultData$ax\_median), sd(imputedResultData$ax\_median))

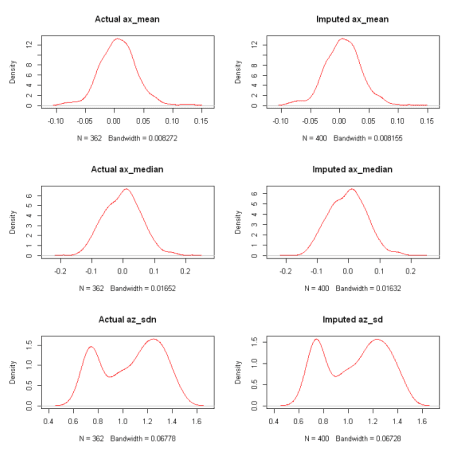
, actual\_az\_sd = c(mean(features$az\_sd), sd(features$az\_sd))

, imputed\_az\_sd = c(mean(imputedResultData$az\_sd), sd(imputedResultData$az\_sd))

, row.names = c("mean", "sd"))

|  | **actual\_ax\_mean** | **imputed\_ax\_mean** | **actual\_ax\_median** | **imputed\_ax\_median** | **actual\_az\_sd** | **imputed\_az\_sd** |
| --- | --- | --- | --- | --- | --- | --- |
| **mean** | 0.006307909 | 0.005851233 | -0.001328867 | -0.00214025 | 1.0588650 | 1.0528059 |
| **sd** | 0.030961085 | 0.031125848 | 0.059619834 | 0.06011342 | 0.2446782 | 0.2477697 |

Now, lets look at the distributions in the data. From the distribution below, we can observe that the distributions for actual data and imputed data is almost identical. We can confirm it with the bandwidth in the plots.



par(mfrow=c(3,2))

plot(density(features$ax\_mean), main = "Actual ax\_mean", type="l", col="red")

plot(density(imputedResultData$ax\_mean), main = "Imputed ax\_mean", type="l", col="red")

plot(density(features$ax\_median), main = "Actual ax\_median", type="l", col="red")

plot(density(imputedResultData$ax\_median), main = "Imputed ax\_median", type="l", col="red")

plot(density(features$az\_sd), main = "Actual az\_sdn", type="l", col="red")

plot(density(imputedResultData$az\_sd), main = "Imputed az\_sd", type="l", col="red")

Density plots

**Building a classification model based on actual data and Imputed data**

In the following data y will be our classification variable. We will build a classification model using a simple support vector machine(SVM) with actual and imputed data. No transformation will be done on the data. In the end we will compare the results

**Actual Data**

**Sample data creation**

Let’s split the data into train and test with ratio’s of 80:20.

#create samples of 80:20 ratio

features$y = as.factor(features$y)

sample = sample(nrow(features) , nrow(features)\* 0.8)

train = features[sample,]

test = features[-sample,]

**Build a SVM model**

Now, we can train the model using train set. We will not do any parameter tuning in this example.

library(e1071)

ibrary(caret)

actual.svm.model = svm(y ~., data = train)

summary(actual.svm.model)

Loading required package: ggplot2

Call:

svm(formula = y ~ ., data = train)

Parameters:

SVM-Type: C-classification

SVM-Kernel: radial

cost: 1

gamma: 0.05

Number of Support Vectors: 142

( 47 18 47 30 )

Number of Classes: 4

Levels:

1 2 3 4

**Validate SVM model**

In the below confusion matrix, we observe the following

1. accuary>NIR indicating model is very good
2. Higher accuray and kappa value indicates a very accurate model
3. Even the balanced accuracy is close to 1 indicating the model is highly accurate

# build a confusion matrix using caret package

confusionMatrix(predict(actual.svm.model, test), test$y)

Confusion Matrix and Statistics

Reference

Prediction 1 2 3 4

1 10 1 0 0

2 0 26 0 0

3 0 0 22 0

4 0 0 3 11

Overall Statistics

Accuracy : 0.9452

95% CI : (0.8656, 0.9849)

No Information Rate : 0.3699

P-Value [Acc > NIR] : < 2.2e-16

Kappa : 0.9234

Mcnemar's Test P-Value : NA

Statistics by Class:

Class: 1 Class: 2 Class: 3 Class: 4

Sensitivity 1.0000 0.9630 0.8800 1.0000

Specificity 0.9841 1.0000 1.0000 0.9516

Pos Pred Value 0.9091 1.0000 1.0000 0.7857

Neg Pred Value 1.0000 0.9787 0.9412 1.0000

Prevalence 0.1370 0.3699 0.3425 0.1507

Detection Rate 0.1370 0.3562 0.3014 0.1507

Detection Prevalence 0.1507 0.3562 0.3014 0.1918

Balanced Accuracy 0.9921 0.9815 0.9400 0.9758

**Imputed Data**

**Sample data creation**

# create samples of 80:20 ratio

imputedResultData$y = as.factor(imputedResultData$y)

sample = sample(nrow(imputedResultData) , nrow(imputedResultData)\* 0.8)

train = imputedResultData[sample,]

test = imputedResultData[-sample,]

**Build a SVM model**

imputed.svm.model = svm(y ~., data = train)

summary(imputed.svm.model)

Call:

svm(formula = y ~ ., data = train)

Parameters:

SVM-Type: C-classification

SVM-Kernel: radial

cost: 1

gamma: 0.05

Number of Support Vectors: 167

( 59 47 36 25 )

Number of Classes: 4

Levels:

1 2 3 4

**Validate SVM model**

In the below confusion matrix, we observe the following

1. accuary>NIR indicating model is very good
2. Higher accuray and kappa value indicates a very accurate model
3. Even the balanced accuracy is close to 1 indicating the model is highly accurate

confusionMatrix(predict(imputed.svm.model, test), test$y)

Confusion Matrix and Statistics

Reference

Prediction 1 2 3 4

1 15 0 0 0

2 1 21 0 0

3 0 0 17 0

4 0 0 0 26

Overall Statistics

Accuracy : 0.9875

95% CI : (0.9323, 0.9997)

No Information Rate : 0.325

P-Value [Acc > NIR] : < 2.2e-16

Kappa : 0.9831

Mcnemar's Test P-Value : NA

Statistics by Class:

Class: 1 Class: 2 Class: 3 Class: 4

Sensitivity 0.9375 1.0000 1.0000 1.000

Specificity 1.0000 0.9831 1.0000 1.000

Pos Pred Value 1.0000 0.9545 1.0000 1.000

Neg Pred Value 0.9846 1.0000 1.0000 1.000

Prevalence 0.2000 0.2625 0.2125 0.325

Detection Rate 0.1875 0.2625 0.2125 0.325

Detection Prevalence 0.1875 0.2750 0.2125 0.325

Balanced Accuracy 0.9688 0.9915 1.0000 1.000

**Overall results**

What we saw above and their interpretation is completely subjective. One way to truly validate them is to create random train and test samples multiple times (say 30), build a model, validate the model, capture kappa value. Finally use a simple t-test to see if there is a significant difference.

Null hypothesis:  
H0: there is no significant difference between two samples.

# lets create functions to simplify the process

test.function = (data){

# create samples

sample = sample(nrow(data) , nrow(data)\* 0.75)

train = data[sample,]

test = data[-sample,]

# build model

svm.model = svm(y ~., data = train)

# get metrics

metrics = confusionMatrix(predict(svm.model, test), test$y)

return(metrics$overall['Accuracy'])

}

# now lets calculate accuracy with actual data to get 30 results

actual.results = NULL

for(i in 1:100) {

actual.results[i] = test.function(features)

}

head(actual.results)

# 0.978021978021978

# 0.978021978021978

# 0.978021978021978

# 0.945054945054945

# 0.989010989010989

# 0.967032967032967

# now lets calculate accuracy with imputed data to get 30 results

imputed.results = NULL

for(i in 1:100) {

imputed.results[i] = test.function(imputedResultData)

}

head(imputed.results)

# 0.97

# 0.95

# 0.92

# 0.96

# 0.92

# 0.96

**T-test to test the results**

What’s better than statistically prove if there is significant difference right? So, we will do a t-test to see if there is any statistical difference in the accuracy.

# Do a simple t-test to see if there is a difference in accuracy when data is imputed

t.test(x= actual.results, y = imputed.results, conf.level = 0.95)

Welch Two Sample t-test

data: actual.results and imputed.results

t = 7.9834, df = 194.03, p-value = 1.222e-13

alternative hypothesis: true difference in means is not equal to 0

95 percent confidence interval:

0.01673213 0.02771182

sample estimates:

mean of x mean of y

0.968022 0.945800

In the above t-test we have set the confidence level at 95%. From the results we can observe that the p-value is less than 0.05 indicating that there is a significant difference in accuracy between actual data and imputed data. From the means we can notice that the average accuracy of actual data is about 96.5% while the accuracy of imputed data y is about 92.5%. There is a variation of 4%. So, does that mean imputing more data results in reducing the accuracy across various models?

Why not do a test to compare the results? let’s consider 4 other models for that and those will be

1. Random forest
2. Decision tree
3. KNN
4. Naive Bayes

**Random Forest**

Let’s use all the same steps as above and fit different models. The results of accuracy will be in the below table

library(randomForest)

# lets create functions to simplify the process

test.rf.function = function(data){

# create samples

sample = sample(nrow(data) , nrow(data)\* 0.75)

train = data[sample,]

test = data[-sample,]

# build model

rf.model = randomForest(y ~., data = train)

# get metrics

metrics = confusionMatrix(predict(rf.model, test), test$y)

return(metrics$overall['Accuracy'])

}

# now lets calculate accuracy with actual data to get 30 results

actual.rf.results = NULL

for(i in 1:100) {

actual.rf.results[i] = test.rf.function(features)

}

#head(actual.rf.results)

# now lets calculate accuracy with imputed data to get 30 results

imputed.rf.results = NULL

for(i in 1:100) {

imputed.rf.results[i] = test.rf.function(imputedResultData)

}

head(data.frame(Actual = actual.rf.results, Imputed = imputed.rf.results))

# Do a simple t-test to see if there is a difference in accuracy when data is imputed

t.test(x= actual.rf.results, y = imputed.rf.results, conf.level = 0.95)

| **Actual** | **Imputed** |
| --- | --- |
| 0.956044 | 0.95 |
| 1.000000 | 0.93 |
| 0.967033 | 0.96 |
| 0.967033 | 0.96 |
| 1.000000 | 0.97 |
| 0.967033 | 0.93 |

Random forest accuracy for actual and imputed data

Welch Two Sample t-test

data: actual.rf.results and imputed.rf.results

t = 11.734, df = 183.2, p-value 2.2e-16

alternative hypothesis: true difference in means is not equal to 0

95 percent confidence interval:

0.02183138 0.03065654

sample estimates:

mean of x mean of y

0.976044 0.949800

In the above t-test results we can come to a similar conclusion as above. There is a significant difference between the actual data and imputed data accuracy. We see approximately 2.5% difference.

**Decision Tree**

library(rpart)

# lets create functions to simplify the process

test.dt.function = function(data){

# create samples

sample = sample(nrow(data) , nrow(data)\* 0.75)

train = data[sample,]

test = data[-sample,]

# build model

dt.model = rpart(y ~., data = train, method="class")

# get metrics

metrics = confusionMatrix(predict(dt.model, test, type="class"), test$y)

return(metrics$overall['Accuracy'])

}

# now lets calculate accuracy with actual data to get 30 results

actual.dt.results = NULL

for(i in 1:100) {

actual.dt.results[i] = test.dt.function(features)

}

#head(actual.rf.results)

# now lets calculate accuracy with imputed data to get 30 results

imputed.dt.results = NULL

for(i in 1:100) {

imputed.dt.results[i] = test.dt.function(imputedResultData)

}

head(data.frame(Actual = actual.dt.results, Imputed = imputed.dt.results))

# Do a simple t-test to see if there is a difference in accuracy when data is imputed

t.test(x= actual.dt.results, y = imputed.dt.results, conf.level = 0.95)

| **Actual** | **Imputed** |
| --- | --- |
| 0.978022 | 0.92 |
| 0.967033 | 0.94 |
| 0.967033 | 0.95 |
| 0.956044 | 0.94 |
| 0.956044 | 0.94 |
| 0.978022 | 0.95 |

Decision tree accuracy for actual and imputed data

Welch Two Sample t-test

data: actual.dt.results and imputed.dt.results

t = 16.24, df = 167.94, p-value < 2.2e-16

alternative hypothesis: true difference in means is not equal to 0

95 percent confidence interval:

0.03331888 0.04254046

sample estimates:

mean of x mean of y

0.9703297 0.9324000

In the above t-test results we can come to a similar conclusion as above. There is a significant difference between the actual data and imputed data accuracy. We see approximately 3.5% difference.

**K-Nearest Neighbor (KNN)**

library(class)

# lets create functions to simplify the process

test.knn.function = function(data){

# create samples

sample = sample(nrow(data) , nrow(data)\* 0.75)

train = data[sample,]

test = data[-sample,]

# build model

knn.model = knn(train,test, cl=train$y, k=5)

# get metrics

metrics = confusionMatrix(knn.model, test$y)

return(metrics$overall['Accuracy'])

}

# now lets calculate accuracy with actual data to get 30 results

actual.dt.results = NULL

for(i in 1:100) {

actual.dt.results[i] = test.knn.function(features)

}

#head(actual.rf.results)

# now lets calculate accuracy with imputed data to get 30 results

imputed.dt.results = NULL

for(i in 1:100) {

imputed.dt.results[i] = test.knn.function(imputedResultData)

}

head(data.frame(Actual = actual.dt.results, Imputed = imputed.dt.results))

# Do a simple t-test to see if there is a difference in accuracy when data is imputed

t.test(x= actual.dt.results, y = imputed.dt.results, conf.level = 0.95)

| **Actual** | **Imputed** |
| --- | --- |
| 0.967033 | 0.97 |
| 1.000000 | 0.98 |
| 0.978022 | 0.99 |
| 0.978022 | 1.00 |
| 0.967033 | 1.00 |
| 0.978022 | 1.00 |

KNN accuracy for actual and imputed data

Welch Two Sample t-test

data: actual.dt.results and imputed.dt.results

t = 3.2151, df = 166.45, p-value = 0.001566

alternative hypothesis: true difference in means is not equal to 0

95 percent confidence interval:

0.002126868 0.008895110

sample estimates:

mean of x mean of y

0.989011 0.983500

In the above t-test results we can come to a similar conclusion as above. There is a significant difference between the actual data and imputed data accuracy. We see approximately 0.05% difference.

**Naive Bayes**

# lets create functions to simplify the process

test.nb.function = function(data){

# create samples

sample = sample(nrow(data) , nrow(data)\* 0.75)

train = data[sample,]

test = data[-sample,]

# build model

nb.model = naiveBayes(y ~., data = train)

# get metrics

metrics = confusionMatrix(predict(nb.model, test), test$y)

return(metrics$overall['Accuracy'])

}

# now lets calculate accuracy with actual data to get 30 results

actual.nb.results = NULL

for(i in 1:100) {

actual.nb.results[i] = test.nb.function(features)

}

#head(actual.rf.results)

# now lets calculate accuracy with imputed data to get 30 results

imputed.nb.results = NULL

for(i in 1:100) {

imputed.nb.results[i] = test.nb.function(imputedResultData)

}

head(data.frame(Actual = actual.nb.results, Imputed = imputed.nb.results))

# Do a simple t-test to see if there is a difference in accuracy when data is imputed

t.test(x= actual.nb.results, y = imputed.nb.results, conf.level = 0.95)

| **Actual** | **Imputed** |
| --- | --- |
| 0.989011 | 0.95 |
| 0.967033 | 0.92 |
| 0.978022 | 0.94 |
| 1.000000 | 0.95 |
| 0.989011 | 0.90 |
| 0.967033 | 0.93 |

Naive Bayes accuracy for actual and imputed data

Welch Two Sample t-test

data: actual.nb.results and imputed.nb.results

t = 18.529, df = 174.88, p-value < 2.2e-16

alternative hypothesis: true difference in means is not equal to 0

95 percent confidence interval:

0.04214191 0.05218996

sample estimates:

mean of x mean of y

0.9740659 0.9269000

In the above t-test results we can come to a similar conclusion as above. There is a significant difference between the actual data and imputed data accuracy. We see approximately 4.5% difference.

**Conclusion**

From the above results we observe that irrespective of the type of model built, we observed a standard variation in accuracy in the range of 3% – 5% between using actual data and imputed data. In all the cases, actual data helped in building a better model compared to using imputed data for building the model.