Practical Machine Learning Project

Alfonso R. Reyes

11/10/2018

Data Exploration

One of the first things we notice is the abundant number of NAs and blank cells in the train dataset.

I tried to detect a pattern of this occurrence and found that two different kind of measurements are being tried to be combined in one dataframe. There are measurements that occur continuously, and other measurements that are dependent of the status of the variable `new_window`. This maybe a different type of sensor that activates on certain times, but it does not produce a continuous output like the rest.

There are variables that have meaningful values only when the variable `new_window` takes the value of `yes`. When it is `no`, the variables are all blank or filled with NAs. This happens with 402 observations.

Examples of these variables are:

```
kurtosis_roll_belt
skewness_roll_belt
max_yaw_belt
amplitude_roll_belt
var_total_accel_belt
stddev_roll_belt
var_accel_arm
stddev_yaw_arm
kurtosis_roll_arm
kurtosis_picth_arm
max_roll_arm
amplitude_roll_arm
kurtosis_roll_dumbbell
max_roll_dumbbell
var_accel_dumbbell
var_yaw_dumbbell
kurtosis_roll_forearm
skewness_yaw_forearm
...
...
...
```

Variables related with `skewness_` and `kurtosis_` are also all blank unless the variable `new_window` is set to yes.

One of the first ideas was to separate these NA/blank variables and turn them in key/value pairs (a tidy dataset) using dplyr function `gather`. Like so:

```
key                     value
kurtosis_roll_belt      5.587755
kurtosis_picth_belt     -1.29859
max_roll_belt           -94.3
amplitude_pitch_belt    0
amplitude_yaw_belt      0
avg_roll_belt           1.5
avg_yaw_belt            -94.4
...
...
```

This idea was not viable since it created more than a million rows with no apparent gain; we still had NAs from other variables that did not confirm to the size and frequency of the second sensor.

Variables with #DIV/0!

There are few variables that contain all values #DIV/0!. Such as:

```
kurtosis_yaw_belt
skewness_yaw_belt
kurtosis_yaw_dumbbell
skewness_yaw_dumbbell
kurtosis_yaw_forearm
skewness_yaw_forearm
```

From all the 19622 observations, there are 405 rows that have the `new_window` variable set to `yes`. This means that 405 observations that occur at a different frequency that the majority of measurements was causing a great number of NAs and blank cells.

A second idea was detecting the abnormal values (NAs, blanks, #DIV/0!) in the training dataset, make an inventory, and compare it against the test set. One key question: were these variables with the abnormal values really included in the test dataset?

Blank/NA variables detection

1. Detect columns that contain mostly blanks
2. Detect columns that contain mostly NAs

Detect #DIV/0! variables

1. Detect columns that have all div0 values
2. Detect columns that sometimes have div0 values

General characteristics of the training dataset

1. Rows have a sequential index from 1 to 19622.
2. The index column does not have a name
3. There is a variable that turns on and off 402 unique measurements and seems to cause a great number of NAs and blanks in the rest of the variables.
4. By removing these low ocurrance variables, we obtain a clean dataset, ready for machine learning.

Main strategy for this project

The strategy followed to deal with abnormal values in the variables was this:

1. Detect the variables that have blanks, NAs and division-by-zero and build an inventory, meaning a count of variables that have all, or mostly, NAs, blanks or #DIV/0!

2. Make a vector of the names of the variables that we tagged as '`normal`. Normal variables are considered any variables that do no contain mostly NAs, blanks or #DIV/0! values. A variable may be totally NAs, but may not contain all blanks nor #DIV/0!.

3. We start scanning the dataframe for patterns of occurance of NAs and blanks taking and analyzing the tip of the sampled dataframe. For instance, we could take 10% of the dataframe split and analyze it for presence of NAs or blanks.

4. Create two functions to get the "health" of the sampled dataframe. The first function will interrogate for the health of every variable and count if it is NA, blank or div-0. The second function iterates through all the variables and produce the total count.

5. At the end of this process, we get four categories of variables:

- NAs
- blanks
- #DIV/0!
- Normal, or none of the above

6. We select the variables that are tagged as `normal`, then subtract them for the entire train dataset. We do the same with the test dataset.

7. Coincidentally, the variables that are tagged as normal in the train dataset are the same variables that are tagged as normal in the test dataset. And the opposite is true as well: variables that are NAs or blank in the train dataset also present the same characteristics in the test dataset.

8. Once the "healthy" variables are in place, we proceed with some Feature Engineering where we make cyclical variables really cyclical by using the trigonometric functions since and cosine. Then, we remove the original variables, or any other that is constant, like the year.

9. Finally, we apply the machine learning algorithms to the healthy train and test datasets.

Load the main libraries:

```r
library(caret)
library(dplyr)
library(tictoc)
```

Load and Analyze the train dataset

Read the train dataset and see the structure.

```r
train_raw <- read.csv(file = "./data/pml-training.csv", header = TRUE,
                      stringsAsFactors = FALSE)

glimpse(train_raw)
```

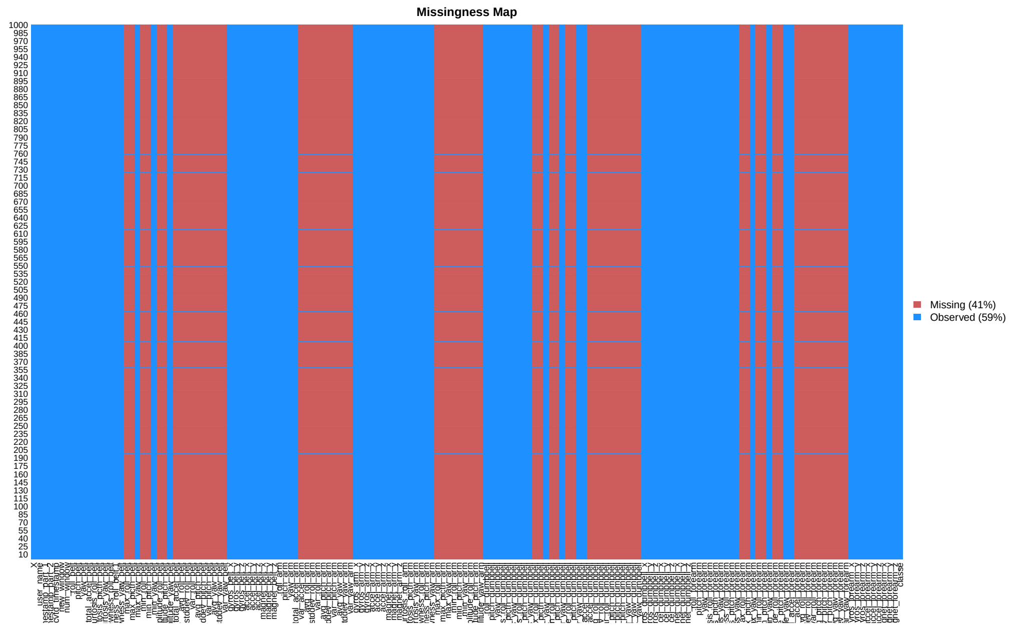
```
#> Observations: 19,622
#> Variables: 160
#> $ X                        <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12...
#> $ user_name                <chr> "carlitos", "carlitos", "carlitos", "...
#> $ raw_timestamp_part_1     <int> 1323084231, 1323084231, 1323084231, 1...
#> $ raw_timestamp_part_2     <int> 788290, 808298, 820366, 120339, 19632...
#> $ cvtd_timestamp           <chr> "05/12/2011 11:23", "05/12/2011 11:23...
#> $ new_window               <chr> "no", "no", "no", "no", "no", "no", "...
#> $ num_window               <int> 11, 11, 11, 12, 12, 12, 12, 12, 12, 1...
#> $ roll_belt                <dbl> 1.41, 1.41, 1.42, 1.48, 1.48, 1.45, 1...
#> $ pitch_belt               <dbl> 8.07, 8.07, 8.07, 8.05, 8.07, 8.06, 8...
#> $ yaw_belt                 <dbl> -94.4, -94.4, -94.4, -94.4, -94.4, -9...
#> $ total_accel_belt         <int> 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3...
#> $ kurtosis_roll_belt       <chr> "", "", "", "", "", "", "", "", "", "...
#> $ kurtosis_picth_belt      <chr> "", "", "", "", "", "", "", "", "", "...
#> $ kurtosis_yaw_belt        <chr> "", "", "", "", "", "", "", "", "", "...
#> $ skewness_roll_belt       <chr> "", "", "", "", "", "", "", "", "", "...
#> $ skewness_roll_belt.1     <chr> "", "", "", "", "", "", "", "", "", "...
#> $ skewness_yaw_belt        <chr> "", "", "", "", "", "", "", "", "", "...
#> $ max_roll_belt            <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ max_picth_belt           <int> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ max_yaw_belt             <chr> "", "", "", "", "", "", "", "", "", "...
#> $ min_roll_belt            <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ min_pitch_belt           <int> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ min_yaw_belt             <chr> "", "", "", "", "", "", "", "", "", "...
#> $ amplitude_roll_belt      <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ amplitude_pitch_belt     <int> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ amplitude_yaw_belt       <chr> "", "", "", "", "", "", "", "", "", "...
#> $ var_total_accel_belt     <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ avg_roll_belt            <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ stddev_roll_belt         <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ var_roll_belt            <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ avg_pitch_belt           <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ stddev_pitch_belt        <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ var_pitch_belt           <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ avg_yaw_belt             <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ stddev_yaw_belt          <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ var_yaw_belt             <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ gyros_belt_x             <dbl> 0.00, 0.02, 0.00, 0.02, 0.02, 0.02, 0...
#> $ gyros_belt_y             <dbl> 0.00, 0.00, 0.00, 0.00, 0.02, 0.00, 0...
#> $ gyros_belt_z             <dbl> -0.02, -0.02, -0.02, -0.03, -0.02, -0...
#> $ accel_belt_x             <int> -21, -22, -20, -22, -21, -21, -22, -2...
#> $ accel_belt_y             <int> 4, 4, 5, 3, 2, 4, 3, 4, 2, 4, 2, 2, 4...
#> $ accel_belt_z             <int> 22, 22, 23, 21, 24, 21, 21, 21, 24, 2...
#> $ magnet_belt_x            <int> -3, -7, -2, -6, -6, 0, -4, -2, 1, -3,...
#> $ magnet_belt_y            <int> 599, 608, 600, 604, 600, 603, 599, 60...
#> $ magnet_belt_z            <int> -313, -311, -305, -310, -302, -312, -...
#> $ roll_arm                 <dbl> -128, -128, -128, -128, -128, -128, -...
#> $ pitch_arm                <dbl> 22.5, 22.5, 22.5, 22.1, 22.1, 22.0, 2...
#> $ yaw_arm                  <dbl> -161, -161, -161, -161, -161, -161, -...
#> $ total_accel_arm          <int> 34, 34, 34, 34, 34, 34, 34, 34, 34, 3...
#> $ var_accel_arm            <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ avg_roll_arm             <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ stddev_roll_arm          <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ var_roll_arm             <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ avg_pitch_arm            <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ stddev_pitch_arm         <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ var_pitch_arm            <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ avg_yaw_arm              <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ stddev_yaw_arm           <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ var_yaw_arm              <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ gyros_arm_x              <dbl> 0.00, 0.02, 0.02, 0.02, 0.00, 0.02, 0...
#> $ gyros_arm_y              <dbl> 0.00, -0.02, -0.02, -0.03, -0.03, -0....
#> $ gyros_arm_z              <dbl> -0.02, -0.02, -0.02, 0.02, 0.00, 0.00...
#> $ accel_arm_x              <int> -288, -290, -289, -289, -289, -289, -...
#> $ accel_arm_y              <int> 109, 110, 110, 111, 111, 111, 111, 11...
#> $ accel_arm_z              <int> -123, -125, -126, -123, -123, -122, -...
#> $ magnet_arm_x             <int> -368, -369, -368, -372, -374, -369, -...
#> $ magnet_arm_y             <int> 337, 337, 344, 344, 337, 342, 336, 33...
#> $ magnet_arm_z             <int> 516, 513, 513, 512, 506, 513, 509, 51...
#> $ kurtosis_roll_arm        <chr> "", "", "", "", "", "", "", "", "", "...
#> $ kurtosis_picth_arm       <chr> "", "", "", "", "", "", "", "", "", "...
#> $ kurtosis_yaw_arm         <chr> "", "", "", "", "", "", "", "", "", "...
#> $ skewness_roll_arm        <chr> "", "", "", "", "", "", "", "", "", "...
#> $ skewness_pitch_arm       <chr> "", "", "", "", "", "", "", "", "", "...
#> $ skewness_yaw_arm         <chr> "", "", "", "", "", "", "", "", "", "...
#> $ max_roll_arm             <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ max_picth_arm            <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ max_yaw_arm              <int> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ min_roll_arm             <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ min_pitch_arm            <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ min_yaw_arm              <int> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ amplitude_roll_arm       <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ amplitude_pitch_arm      <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ amplitude_yaw_arm        <int> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ roll_dumbbell            <dbl> 13.05217, 13.13074, 12.85075, 13.4312...
#> $ pitch_dumbbell           <dbl> -70.49400, -70.63751, -70.27812, -70....
#> $ yaw_dumbbell             <dbl> -84.87394, -84.71065, -85.14078, -84....
#> $ kurtosis_roll_dumbbell   <chr> "", "", "", "", "", "", "", "", "", "...
#> $ kurtosis_picth_dumbbell  <chr> "", "", "", "", "", "", "", "", "", "...
#> $ kurtosis_yaw_dumbbell    <chr> "", "", "", "", "", "", "", "", "", "...
#> $ skewness_roll_dumbbell   <chr> "", "", "", "", "", "", "", "", "", "...
#> $ skewness_pitch_dumbbell  <chr> "", "", "", "", "", "", "", "", "", "...
#> $ skewness_yaw_dumbbell    <chr> "", "", "", "", "", "", "", "", "", "...
#> $ max_roll_dumbbell        <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ max_picth_dumbbell       <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ max_yaw_dumbbell         <chr> "", "", "", "", "", "", "", "", "", "...
#> $ min_roll_dumbbell        <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ min_pitch_dumbbell       <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ min_yaw_dumbbell         <chr> "", "", "", "", "", "", "", "", "", "...
#> $ amplitude_roll_dumbbell  <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ amplitude_pitch_dumbbell <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ amplitude_yaw_dumbbell   <chr> "", "", "", "", "", "", "", "", "", "...
#> $ total_accel_dumbbell     <int> 37, 37, 37, 37, 37, 37, 37, 37, 37, 3...
#> $ var_accel_dumbbell       <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ avg_roll_dumbbell        <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ stddev_roll_dumbbell     <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ var_roll_dumbbell        <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ avg_pitch_dumbbell       <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ stddev_pitch_dumbbell    <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ var_pitch_dumbbell       <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ avg_yaw_dumbbell         <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ stddev_yaw_dumbbell      <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ var_yaw_dumbbell         <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ gyros_dumbbell_x         <dbl> 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0...
#> $ gyros_dumbbell_y         <dbl> -0.02, -0.02, -0.02, -0.02, -0.02, -0...
#> $ gyros_dumbbell_z         <dbl> 0.00, 0.00, 0.00, -0.02, 0.00, 0.00, ...
#> $ accel_dumbbell_x         <int> -234, -233, -232, -232, -233, -234, -...
#> $ accel_dumbbell_y         <int> 47, 47, 46, 48, 48, 48, 47, 46, 47, 4...
#> $ accel_dumbbell_z         <int> -271, -269, -270, -269, -270, -269, -...
#> $ magnet_dumbbell_x        <int> -559, -555, -561, -552, -554, -558, -...
#> $ magnet_dumbbell_y        <int> 293, 296, 298, 303, 292, 294, 295, 30...
#> $ magnet_dumbbell_z        <dbl> -65, -64, -63, -60, -68, -66, -70, -7...
#> $ roll_forearm             <dbl> 28.4, 28.3, 28.3, 28.1, 28.0, 27.9, 2...
#> $ pitch_forearm            <dbl> -63.9, -63.9, -63.9, -63.9, -63.9, -6...
#> $ yaw_forearm              <dbl> -153, -153, -152, -152, -152, -152, -...
#> $ kurtosis_roll_forearm    <chr> "", "", "", "", "", "", "", "", "", "...
#> $ kurtosis_picth_forearm   <chr> "", "", "", "", "", "", "", "", "", "...
#> $ kurtosis_yaw_forearm     <chr> "", "", "", "", "", "", "", "", "", "...
#> $ skewness_roll_forearm    <chr> "", "", "", "", "", "", "", "", "", "...
#> $ skewness_pitch_forearm   <chr> "", "", "", "", "", "", "", "", "", "...
#> $ skewness_yaw_forearm     <chr> "", "", "", "", "", "", "", "", "", "...
#> $ max_roll_forearm         <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ max_picth_forearm        <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ max_yaw_forearm          <chr> "", "", "", "", "", "", "", "", "", "...
#> $ min_roll_forearm         <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ min_pitch_forearm        <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ min_yaw_forearm          <chr> "", "", "", "", "", "", "", "", "", "...
#> $ amplitude_roll_forearm   <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ amplitude_pitch_forearm  <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ amplitude_yaw_forearm    <chr> "", "", "", "", "", "", "", "", "", "...
#> $ total_accel_forearm      <int> 36, 36, 36, 36, 36, 36, 36, 36, 36, 3...
#> $ var_accel_forearm        <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ avg_roll_forearm         <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ stddev_roll_forearm      <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ var_roll_forearm         <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ avg_pitch_forearm        <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ stddev_pitch_forearm     <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ var_pitch_forearm        <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ avg_yaw_forearm          <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ stddev_yaw_forearm       <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ var_yaw_forearm          <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ gyros_forearm_x          <dbl> 0.03, 0.02, 0.03, 0.02, 0.02, 0.02, 0...
#> $ gyros_forearm_y          <dbl> 0.00, 0.00, -0.02, -0.02, 0.00, -0.02...
#> $ gyros_forearm_z          <dbl> -0.02, -0.02, 0.00, 0.00, -0.02, -0.0...
#> $ accel_forearm_x          <int> 192, 192, 196, 189, 189, 193, 195, 19...
#> $ accel_forearm_y          <int> 203, 203, 204, 206, 206, 203, 205, 20...
#> $ accel_forearm_z          <int> -215, -216, -213, -214, -214, -215, -...
#> $ magnet_forearm_x         <int> -17, -18, -18, -16, -17, -9, -18, -9,...
#> $ magnet_forearm_y         <dbl> 654, 661, 658, 658, 655, 660, 659, 66...
#> $ magnet_forearm_z         <dbl> 476, 473, 469, 469, 473, 478, 470, 47...
#> $ classe                   <chr> "A", "A", "A", "A", "A", "A", "A", "A...
```

Notice the large quantity of variables with NA and blank character values at the tip of the train dataframe.

Plot of train dataset variables

Take a look at the visual structure of the train dataset. Notice the variables with NAs.

```r
library(Amelia)
train_raw %>%
    sample_n(1000) %>%
    missmap(rank.order = FALSE)
```



Health of the train dataset

If we take a sample of the train dataset we could see that there are some non-data values that repeat in different areas:

1. There are variables with NAs
2. There are variables with blanks
3. There are observations with `#DIV/0!`

This is a sample near the tip of the train dataset:

```
train_raw[10:30, 15:20]
```

```
#>    skewness_roll_belt skewness_roll_belt.1 skewness_yaw_belt max_roll_belt
#> 10                                                                      NA
#> 11                                                                      NA
#> 12                                                                      NA
#> 13                                                                      NA
#> 14                                                                      NA
#> 15                                                                      NA
#> 16                                                                      NA
#> 17                                                                      NA
#> 18                                                                      NA
#> 19                                                                      NA
#> 20                                                                      NA
#> 21                                                                      NA
#> 22                                                                      NA
#> 23                                                                      NA
#> 24           2.713152              #DIV/0!           #DIV/0!         -94.3
#> 25                                                                      NA
#> 26                                                                      NA
#> 27                                                                      NA
#> 28                                                                      NA
#> 29                                                                      NA
#> 30                                                                      NA
#>    max_picth_belt max_yaw_belt
#> 10             NA
#> 11             NA
#> 12             NA
#> 13             NA
#> 14             NA
#> 15             NA
#> 16             NA
#> 17             NA
#> 18             NA
#> 19             NA
#> 20             NA
#> 21             NA
#> 22             NA
#> 23             NA
#> 24              3          5.6
#> 25             NA
#> 26             NA
#> 27             NA
#> 28             NA
#> 29             NA
#> 30             NA
```

Load and Analyze the test dataset

Read the test dataset. It is rather small with 20 observations but the same number of variables.

```
test_raw <- read.csv(file = "./data/pml-testing.csv", header = TRUE,
                      stringsAsFactors = FALSE)

glimpse(test_raw)
```

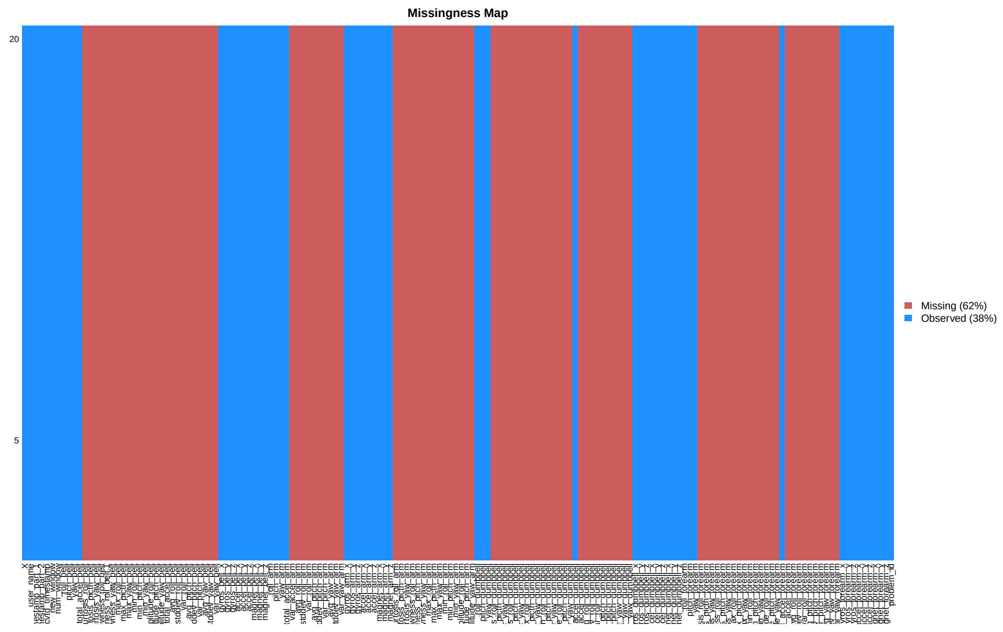
```
#> Observations: 20
#> Variables: 160
#> $ X                        <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12...
#> $ user_name                <chr> "pedro", "jeremy", "jeremy", "adelmo"...
#> $ raw_timestamp_part_1     <int> 1323095002, 1322673067, 1322673075, 1...
#> $ raw_timestamp_part_2     <int> 868349, 778725, 342967, 560311, 81477...
#> $ cvtd_timestamp           <chr> "05/12/2011 14:23", "30/11/2011 17:11...
#> $ new_window               <chr> "no", "no", "no", "no", "no", "no", "...
#> $ num_window               <int> 74, 431, 439, 194, 235, 504, 485, 440...
#> $ roll_belt                <dbl> 123.00, 1.02, 0.87, 125.00, 1.35, -5....
#> $ pitch_belt               <dbl> 27.00, 4.87, 1.82, -41.60, 3.33, 1.59...
#> $ yaw_belt                 <dbl> -4.75, -88.90, -88.50, 162.00, -88.60...
#> $ total_accel_belt         <int> 20, 4, 5, 17, 3, 4, 4, 4, 4, 18, 3, 5...
#> $ kurtosis_roll_belt       <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ kurtosis_picth_belt      <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ kurtosis_yaw_belt        <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ skewness_roll_belt       <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ skewness_roll_belt.1     <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ skewness_yaw_belt        <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ max_roll_belt            <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ max_picth_belt          <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ max_yaw_belt            <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ min_roll_belt           <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ min_pitch_belt          <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ min_yaw_belt            <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ amplitude_roll_belt     <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ amplitude_pitch_belt    <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ amplitude_yaw_belt      <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ var_total_accel_belt    <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ avg_roll_belt           <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ stddev_roll_belt        <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ var_roll_belt           <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ avg_pitch_belt          <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ stddev_pitch_belt       <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ var_pitch_belt          <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ avg_yaw_belt            <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ stddev_yaw_belt         <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ var_yaw_belt            <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ gyros_belt_x            <dbl> -0.50, -0.06, 0.05, 0.11, 0.03, 0.10,...
#> $ gyros_belt_y            <dbl> -0.02, -0.02, 0.02, 0.11, 0.02, 0.05,...
#> $ gyros_belt_z            <dbl> -0.46, -0.07, 0.03, -0.16, 0.00, -0.1...
#> $ accel_belt_x            <int> -38, -13, 1, 46, -8, -11, -14, -10, -...
#> $ accel_belt_y            <int> 69, 11, -1, 45, 4, -16, 2, -2, 1, 63,...
#> $ accel_belt_z            <int> -179, 39, 49, -156, 27, 38, 35, 42, 3...
#> $ magnet_belt_x           <int> -13, 43, 29, 169, 33, 31, 50, 39, -6,...
#> $ magnet_belt_y           <int> 581, 636, 631, 608, 566, 638, 622, 63...
#> $ magnet_belt_z           <int> -382, -309, -312, -304, -418, -291, -...
#> $ roll_arm                <dbl> 40.70, 0.00, 0.00, -109.00, 76.10, 0....
#> $ pitch_arm               <dbl> -27.80, 0.00, 0.00, 55.00, 2.76, 0.00...
#> $ yaw_arm                 <dbl> 178.0, 0.0, 0.0, -142.0, 102.0, 0.0, ...
#> $ total_accel_arm         <int> 10, 38, 44, 25, 29, 14, 15, 22, 34, 3...
#> $ var_accel_arm           <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ avg_roll_arm            <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ stddev_roll_arm         <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ var_roll_arm            <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ avg_pitch_arm           <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ stddev_pitch_arm        <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ var_pitch_arm           <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ avg_yaw_arm             <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ stddev_yaw_arm          <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ var_yaw_arm             <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ gyros_arm_x             <dbl> -1.65, -1.17, 2.10, 0.22, -1.96, 0.02...
#> $ gyros_arm_y             <dbl> 0.48, 0.85, -1.36, -0.51, 0.79, 0.05,...
#> $ gyros_arm_z             <dbl> -0.18, -0.43, 1.13, 0.92, -0.54, -0.0...
#> $ accel_arm_x             <int> 16, -290, -341, -238, -197, -26, 99, ...
#> $ accel_arm_y             <int> 38, 215, 245, -57, 200, 130, 79, 175,...
#> $ accel_arm_z             <int> 93, -90, -87, 6, -30, -19, -67, -78, ...
#> $ magnet_arm_x            <int> -326, -325, -264, -173, -170, 396, 70...
#> $ magnet_arm_y            <int> 385, 447, 474, 257, 275, 176, 15, 215...
#> $ magnet_arm_z            <int> 481, 434, 413, 633, 617, 516, 217, 38...
#> $ kurtosis_roll_arm       <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ kurtosis_picth_arm      <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ kurtosis_yaw_arm        <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ skewness_roll_arm       <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ skewness_pitch_arm       <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ skewness_yaw_arm         <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ max_roll_arm             <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ max_picth_arm            <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ max_yaw_arm              <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ min_roll_arm             <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ min_pitch_arm            <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ min_yaw_arm              <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ amplitude_roll_arm       <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ amplitude_pitch_arm      <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ amplitude_yaw_arm        <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ roll_dumbbell            <dbl> -17.737480, 54.477605, 57.070308, 43....
#> $ pitch_dumbbell           <dbl> 24.96085, -53.69758, -51.37303, -30.0...
#> $ yaw_dumbbell             <dbl> 126.235964, -75.514799, -75.202873, -...
#> $ kurtosis_roll_dumbbell   <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ kurtosis_picth_dumbbell  <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ kurtosis_yaw_dumbbell    <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ skewness_roll_dumbbell   <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ skewness_pitch_dumbbell  <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ skewness_yaw_dumbbell    <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ max_roll_dumbbell        <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ max_picth_dumbbell       <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ max_yaw_dumbbell         <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ min_roll_dumbbell        <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ min_pitch_dumbbell       <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ min_yaw_dumbbell         <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ amplitude_roll_dumbbell  <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ amplitude_pitch_dumbbell <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ amplitude_yaw_dumbbell   <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ total_accel_dumbbell     <int> 9, 31, 29, 18, 4, 29, 29, 29, 3, 2, 1...
#> $ var_accel_dumbbell       <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ avg_roll_dumbbell        <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ stddev_roll_dumbbell     <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ var_roll_dumbbell        <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ avg_pitch_dumbbell       <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ stddev_pitch_dumbbell    <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ var_pitch_dumbbell       <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ avg_yaw_dumbbell         <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ stddev_yaw_dumbbell      <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ var_yaw_dumbbell         <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ gyros_dumbbell_x         <dbl> 0.64, 0.34, 0.39, 0.10, 0.29, -0.59, ...
#> $ gyros_dumbbell_y         <dbl> 0.06, 0.05, 0.14, -0.02, -0.47, 0.80,...
#> $ gyros_dumbbell_z         <dbl> -0.61, -0.71, -0.34, 0.05, -0.46, 1.1...
#> $ accel_dumbbell_x         <int> 21, -153, -141, -51, -18, -138, -145,...
#> $ accel_dumbbell_y         <int> -15, 155, 155, 72, -30, 166, 150, 159...
#> $ accel_dumbbell_z         <int> 81, -205, -196, -148, -5, -186, -190,...
#> $ magnet_dumbbell_x        <int> 523, -502, -506, -576, -424, -543, -4...
#> $ magnet_dumbbell_y        <int> -528, 388, 349, 238, 252, 262, 354, 3...
#> $ magnet_dumbbell_z        <int> -56, -36, 41, 53, 312, 96, 97, 53, -3...
#> $ roll_forearm             <dbl> 141.0, 109.0, 131.0, 0.0, -176.0, 150...
#> $ pitch_forearm            <dbl> 49.30, -17.60, -32.60, 0.00, -2.16, 1...
#> $ yaw_forearm              <dbl> 156.0, 106.0, 93.0, 0.0, -47.9, 89.7,...
#> $ kurtosis_roll_forearm    <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ kurtosis_picth_forearm   <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ kurtosis_yaw_forearm     <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ skewness_roll_forearm    <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ skewness_pitch_forearm   <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ skewness_yaw_forearm     <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ max_roll_forearm         <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ max_picth_forearm        <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ max_yaw_forearm          <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ min_roll_forearm         <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ min_pitch_forearm        <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ min_yaw_forearm          <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ amplitude_roll_forearm   <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ amplitude_pitch_forearm  <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ amplitude_yaw_forearm    <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ total_accel_forearm      <int> 33, 39, 34, 43, 24, 43, 32, 47, 36, 2...
#> $ var_accel_forearm        <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ avg_roll_forearm         <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ stddev_roll_forearm      <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ var_roll_forearm         <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ avg_pitch_forearm        <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ stddev_pitch_forearm     <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ var_pitch_forearm        <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ avg_yaw_forearm          <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ stddev_yaw_forearm       <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ var_yaw_forearm          <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
#> $ gyros_forearm_x          <dbl> 0.74, 1.12, 0.18, 1.38, -0.75, -0.88,...
#> $ gyros_forearm_y          <dbl> -3.34, -2.78, -0.79, 0.69, 3.10, 4.26...
#> $ gyros_forearm_z          <dbl> -0.59, -0.18, 0.28, 1.80, 0.80, 1.35,...
#> $ accel_forearm_x          <int> -110, 212, 154, -92, 131, 230, -192, ...
#> $ accel_forearm_y          <int> 267, 297, 271, 406, -93, 322, 170, -3...
#> $ accel_forearm_z          <int> -149, -118, -129, -39, 172, -144, -17...
#> $ magnet_forearm_x         <int> -714, -237, -51, -233, 375, -300, -67...
#> $ magnet_forearm_y         <int> 419, 791, 698, 783, -787, 800, 284, -...
#> $ magnet_forearm_z         <int> 617, 873, 783, 521, 91, 884, 585, -32...
#> $ problem_id               <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12...
```

In the test dataset, we see more variables with NA than character blanks.

Plot of test dataset variables

A visual representation of the test dataset gives us some clue about the similarity with the train dataset.

```r
library(Amelia)
test_raw %>%
    missmap(rank.order = FALSE)
```



Analysis of the variables health

Build a function to qualify a variable health

This function will find what variables have NAs, blanks, #DIV/0!. Anything else we will considered as a **normal** variable.

The function will return a vector of the names of the columns and the health of the variable. By health, we mean that the variable contains any of the non-desirable values (NAs, blanks, or #DIV/0!).

```
# function that return the type of variable. There are four:
#     normal, is_na, blank and div_0

get_variable_health <- function(df, col, row_depth=50) {
    kum_df <- data.frame()

    for (i in 1:row_depth) {
        blank <- FALSE; na <- FALSE; normal <- FALSE; div_0 <- FALSE
        if (is.na(df[i, col])) {
            is_na  <- TRUE
            blank  <- FALSE
            normal <- FALSE
            div_0  <- FALSE
        } else if (df[i, col] == "") {
            blank  <- TRUE
            is_na  <- FALSE
            normal <- FALSE
            div_0  <- FALSE
        } else if (df[i, col] == "#DIV/0!") {
            div_0  <- TRUE
            is_na  <- FALSE
            blank  <- FALSE
```

```r
            normal <- FALSE
        } else {
            normal <- TRUE
            is_na  <- FALSE
            blank  <- FALSE
            div_0  <- FALSE
        }
        # create a dataframe of the health for one row
        row <- data.frame(normal = normal,
                          blank = blank,
                          is_na = is_na,
                          div_0 = div_0)
        # cumulate the row results
        kum_df <- rbind(kum_df, row)
    }
    ap <- apply(kum_df, 2, sum)  # what is the total for each of the cases
                                 # is_na, blank, normal, div_0
    # get the name of the column with maximum value which tell us the most
    # predominant case. For instance, "#DIV/0!" occurs in some variables but
    # they are sporadic, so we count the occurrences, and if it the greater sum
    # if will be considered predominant in the variable.
    names(which.max(ap))
}
```

Iterate through columns to find the variable health status

Now, we start iterating through every one of the variables and apply the function `get_variable_health()`. We constrain the detection to just the tip of the dataframe; we don't analyze all the observations at this time. To find how many observations are enough to give an opinion about the health of the variable, we use a formula which should give a representative chunk of the dataframe.

$$ceiling\left(\sqrt{\frac{nrow(dataframe)}{\log(nrow(dataframe))}}\right)$$

This formula gives us about 45 rows to investigate the health of the variables in the train set, and 3 rows at the top of the test set.

We use this formula: `ceiling(sqrt(nrow(df) / log(nrow(df))))` \

```r
# function to iterate all the variables in the dataframe
iterate_vars <- function(df) {
    # how many rows to look ahead
    row_depth <- ceiling(sqrt(nrow(df) / log(nrow(df))))
    var_health <- rep("", ncol(df))  # create vector to load results
    var_names <- names(df)           # assign names for each element of the vector

    for (col in var_names) {
        health <- get_variable_health(df, col, row_depth = row_depth)
        ncol <- which(colnames(df) == col)  # get column number given its name
        names(var_health)[ncol] <- col      # assign the name of the column to vector
        var_health[ncol] <- health          # assign the health the variable
        # cat(ncol, col, health, "\n")
    }
    var_health
}
```

Get the `normal` variables in the train set

Iterate through the variables of the train set and get a health inventory.

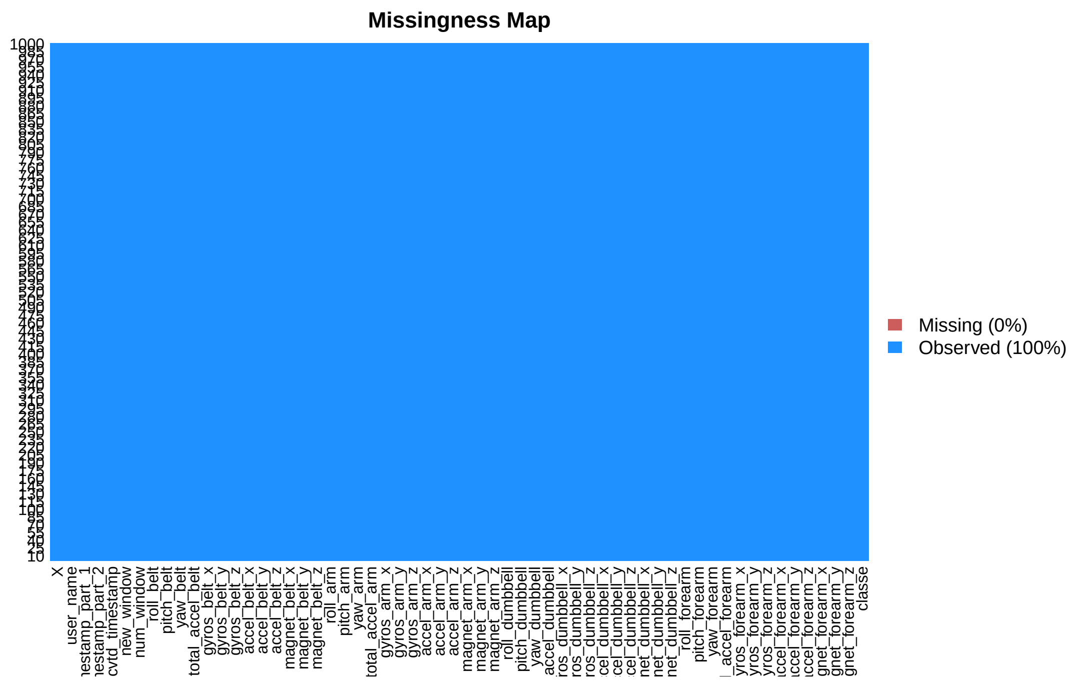
```
train_var_health <- iterate_vars(train_raw)
table(train_var_health)
```

```
#> train_var_health
#>  blank  is_na normal
#>     33     67     60
```

Get the variables that are tagged as "normal" which are the variables that we want to keep:

```
# train dataset with variables tagged as normal
train_set_raw <- train_raw[, train_var_health == "normal"]
```

```
# Visual representation of the clean dataset
train_set_raw %>%
    sample_n(1000) %>%
    missmap(rank.order = FALSE)
```



Taking random samples of the train dataset

Because we want to know if the behavior at the tip of the dataset is the same anywhere in the dataset, we will take random samples of the dataset and apply the `iterate_vars()` function.

Taking a sample of 10% of the dataset

```
# we take in chunks 10% of total number of rows
train_var_health <- iterate_vars(sample_n(train_raw, 0.1 * nrow(train_raw)))
table(train_var_health)
```

```
#> train_var_health
#>  blank  is_na normal 
#>     33     67     60
```

Taking multiple samples from the train dataset

We do the same thing again, but this time in automated mode. We will take random samples of the train dataset to confirm that the same health status of the variables is the same anywehere in the train dataset. The size of the sample is 10%:

`sample_n(train_raw, 0.1 * nrow(train_raw))`

```
# take ten random samples and find if the health of the variables is the same
# on each sample
for (i in 1:10) {
    df <- sample_n(train_raw, 0.1 * nrow(train_raw))  # 10% of the dataset
    # print(head(df))
    # print(t(as.data.frame(table(iterate_vars(df)))), row.names = NULL)
    print(table(iterate_vars(df)))
}
```

```
#> 
#>  blank  is_na normal 
#>     33     67     60 
#> 
#>  blank  is_na normal 
#>     33     67     60 
#> 
#>  blank  is_na normal 
#>     33     67     60 
#> 
#>  blank  is_na normal 
#>     33     67     60 
#> 
#>  blank  is_na normal 
#>     33     67     60 
#> 
#>  blank  is_na normal 
#>     33     67     60 
#> 
#>  blank  is_na normal 
#>     33     67     60 
#> 
#>  blank  is_na normal 
#>     33     67     60 
#> 
#>  blank  is_na normal 
#>     33     67     60
```

```
#>
#>  blank  is_na normal
#>     33     67     60
```

The result tell us that the finding at the tip of the training dataset is consistent with the what happen in other parts of the dataset, since we are taking random samples of the dataset.

Get normal variables in the test set

No it is the turn of the test set. We apply the same functions to the test set getting the vector with the health for each variable.

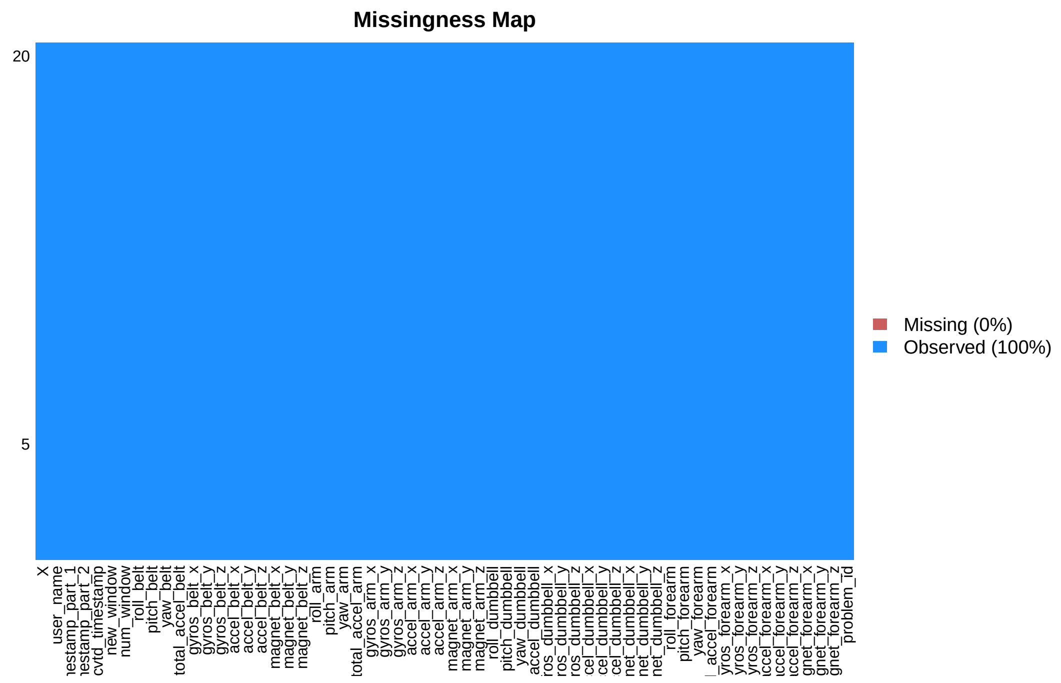
```
test_var_health <- iterate_vars(test_raw)
table(test_var_health)
```

```
#> test_var_health
#>  is_na normal
#>    100     60
```

And we only select the variables that are tagged as "normal".

```
# test dataset with variables tagged as normal
test_set_raw <- test_raw[, test_var_health == "normal"]
```

```
# visual representation of the clean test dataset
test_set_raw %>%
    missmap(rank.order = FALSE)
```



Comparing the `normal` variables in the train and test datasets

```r
# variables in the train dataset tagged as normal only
(train_var_normal <- train_var_health[train_var_health == "normal"])
```

```
#>                    X            user_name raw_timestamp_part_1
#>             "normal"             "normal"             "normal"
#> raw_timestamp_part_2       cvtd_timestamp           new_window
#>             "normal"             "normal"             "normal"
#>           num_window            roll_belt           pitch_belt
#>             "normal"             "normal"             "normal"
#>             yaw_belt     total_accel_belt         gyros_belt_x
#>             "normal"             "normal"             "normal"
#>         gyros_belt_y         gyros_belt_z         accel_belt_x
#>             "normal"             "normal"             "normal"
#>         accel_belt_y         accel_belt_z        magnet_belt_x
#>             "normal"             "normal"             "normal"
#>        magnet_belt_y        magnet_belt_z             roll_arm
#>             "normal"             "normal"             "normal"
#>            pitch_arm              yaw_arm      total_accel_arm
#>             "normal"             "normal"             "normal"
#>          gyros_arm_x          gyros_arm_y          gyros_arm_z
#>             "normal"             "normal"             "normal"
#>          accel_arm_x          accel_arm_y          accel_arm_z
#>             "normal"             "normal"             "normal"
#>         magnet_arm_x         magnet_arm_y         magnet_arm_z
#>             "normal"             "normal"             "normal"
#>        roll_dumbbell       pitch_dumbbell         yaw_dumbbell
#>             "normal"             "normal"             "normal"
#> total_accel_dumbbell     gyros_dumbbell_x     gyros_dumbbell_y
#>             "normal"             "normal"             "normal"
#>     gyros_dumbbell_z     accel_dumbbell_x     accel_dumbbell_y
#>             "normal"             "normal"             "normal"
#>     accel_dumbbell_z    magnet_dumbbell_x    magnet_dumbbell_y
#>             "normal"             "normal"             "normal"
#>    magnet_dumbbell_z         roll_forearm        pitch_forearm
#>             "normal"             "normal"             "normal"
#>          yaw_forearm  total_accel_forearm      gyros_forearm_x
#>             "normal"             "normal"             "normal"
#>      gyros_forearm_y      gyros_forearm_z      accel_forearm_x
#>             "normal"             "normal"             "normal"
#>      accel_forearm_y      accel_forearm_z     magnet_forearm_x
#>             "normal"             "normal"             "normal"
#>     magnet_forearm_y     magnet_forearm_z               classe
#>             "normal"             "normal"             "normal"
```

```r
# variables in the test dataset tagged as normal only
(test_var_normal <- test_var_health[test_var_health == "normal"])
```

```
#>                    X            user_name raw_timestamp_part_1
#>             "normal"             "normal"             "normal"
#> raw_timestamp_part_2       cvtd_timestamp           new_window
#>             "normal"             "normal"             "normal"
#>           num_window            roll_belt           pitch_belt
#>             "normal"             "normal"             "normal"
#>             yaw_belt     total_accel_belt         gyros_belt_x
#>             "normal"             "normal"             "normal"
#>          gyros_belt_y          gyros_belt_z          accel_belt_x
#>              "normal"              "normal"              "normal"
#>          accel_belt_y          accel_belt_z         magnet_belt_x
#>              "normal"              "normal"              "normal"
#>         magnet_belt_y         magnet_belt_z              roll_arm
#>              "normal"              "normal"              "normal"
#>             pitch_arm               yaw_arm       total_accel_arm
#>              "normal"              "normal"              "normal"
#>           gyros_arm_x           gyros_arm_y           gyros_arm_z
#>              "normal"              "normal"              "normal"
#>           accel_arm_x           accel_arm_y           accel_arm_z
#>              "normal"              "normal"              "normal"
#>          magnet_arm_x          magnet_arm_y          magnet_arm_z
#>              "normal"              "normal"              "normal"
#>         roll_dumbbell        pitch_dumbbell          yaw_dumbbell
#>              "normal"              "normal"              "normal"
#>  total_accel_dumbbell      gyros_dumbbell_x      gyros_dumbbell_y
#>              "normal"              "normal"              "normal"
#>      gyros_dumbbell_z      accel_dumbbell_x      accel_dumbbell_y
#>              "normal"              "normal"              "normal"
#>      accel_dumbbell_z     magnet_dumbbell_x     magnet_dumbbell_y
#>              "normal"              "normal"              "normal"
#>     magnet_dumbbell_z          roll_forearm         pitch_forearm
#>              "normal"              "normal"              "normal"
#>           yaw_forearm   total_accel_forearm       gyros_forearm_x
#>              "normal"              "normal"              "normal"
#>       gyros_forearm_y       gyros_forearm_z       accel_forearm_x
#>              "normal"              "normal"              "normal"
#>       accel_forearm_y       accel_forearm_z      magnet_forearm_x
#>              "normal"              "normal"              "normal"
#>      magnet_forearm_y      magnet_forearm_z            problem_id
#>              "normal"              "normal"              "normal"
```

Intersect and Outersect the train and test variables

There are 59 variables that are similarly tagged as "normal" in the train and test datasets. We use the `intersect()` function to find what variables are commonly tagged as normal in both datasets.

```r
intersect(names(train_var_normal), names(test_var_normal))
```

```
#>  [1] "X"                    "user_name"            "raw_timestamp_part_1"
#>  [4] "raw_timestamp_part_2" "cvtd_timestamp"       "new_window"
#>  [7] "num_window"           "roll_belt"            "pitch_belt"
#> [10] "yaw_belt"             "total_accel_belt"     "gyros_belt_x"
#> [13] "gyros_belt_y"         "gyros_belt_z"         "accel_belt_x"
#> [16] "accel_belt_y"         "accel_belt_z"         "magnet_belt_x"
#> [19] "magnet_belt_y"        "magnet_belt_z"        "roll_arm"
#> [22] "pitch_arm"            "yaw_arm"              "total_accel_arm"
#> [25] "gyros_arm_x"          "gyros_arm_y"          "gyros_arm_z"
#> [28] "accel_arm_x"          "accel_arm_y"          "accel_arm_z"
#> [31] "magnet_arm_x"         "magnet_arm_y"         "magnet_arm_z"
#> [34] "roll_dumbbell"        "pitch_dumbbell"       "yaw_dumbbell"
#> [37] "total_accel_dumbbell" "gyros_dumbbell_x"     "gyros_dumbbell_y"
#> [40] "gyros_dumbbell_z"     "accel_dumbbell_x"     "accel_dumbbell_y"
#> [43] "accel_dumbbell_z"     "magnet_dumbbell_x"    "magnet_dumbbell_y"
#> [46] "magnet_dumbbell_z"    "roll_forearm"         "pitch_forearm"
#> [49] "yaw_forearm"          "total_accel_forearm"  "gyros_forearm_x"
#> [52] "gyros_forearm_y"      "gyros_forearm_z"      "accel_forearm_x"
#> [55] "accel_forearm_y"      "accel_forearm_z"      "magnet_forearm_x"
#> [58] "magnet_forearm_y"     "magnet_forearm_z"
```

And what are the variables that are not common to both datasets, train and test?

```r
# function to find what variables are not common
outersect <- function(x, y) {
    c(
    setdiff(x, y),
    setdiff(y, x)
    )
}

outersect(names(train_var_normal), names(test_var_normal))
```

```
#> [1] "classe"     "problem_id"
```

There are only two variables that are not common in the train and test datasets.

Feature Engineering

Now it's time to perform some feature engineering.

Generic function to perform feature engineering

We will create a function that will perform the same actions on the train and test sets:

1. Remove redundant or unneccesary variables, or variables which carry duplicate or no meaningful information.
2. Extract the year, month, day, hour and minutes from the character variable named `cvtd_timestamp`.
3. Convert the date to cyclical variables using sine and cosine functions.
4. Remove intermediate variables at the end of the process.
5. Remove constants or features that do not change. Example: `year`.

```r
do_feature_eng <- function(df) {
    df %>%
    # remove unnecessary variables
    select(-c(X, raw_timestamp_part_1, raw_timestamp_part_2)) %>%
    # convert date to numeric
    mutate(
        year   = as.integer(substr(cvtd_timestamp, 7, 11)),
        day    = as.integer(substr(cvtd_timestamp, 1, 2)),
        month  = as.integer(substr(cvtd_timestamp, 4, 5)),
        minute = as.integer(substr(cvtd_timestamp, 15, 16)),
        hour   = as.integer(substr(cvtd_timestamp, 12, 13)) + minute / 60.0
        ) %>%
    # convert month, day, hour to cyclical
    mutate(
        month_sin = sin((month - 1) * (2.0 * pi / 12)),
        month_cos = cos((month - 1) * (2.0 * pi / 12)),
        day_sin   = sin(day * 2.0 * pi / 7),
        day_cos   = cos(day * 2.0 * pi / 7),
        hour_sin  = sin(hour * 2.0 * pi / 24),
        hour_cos  = cos(hour * 2.0 * pi / 24)

        ) %>%
    mutate(
        user_name = as.factor(user_name)
    ) %>%
    # { if ("classe" %in% names(.)) mutate(classe = as.factor(classe)) } %>%

    # order of variables
    select(user_name,  year, month, month_sin, month_cos,
           day, day_sin, day_cos,
           hour, minute, hour_sin, hour_cos,
           everything()) %>%
    # remove one more unnecessary variables
    select(-c(month, day, hour, minute)) %>%    # converted to cyclic
    select(-new_window, -cvtd_timestamp) %>%       # not meaningful
    # removing year because is constant or the same in the whole dataset
    select(-year)
}
```

FE on the train dataset

Apply feature engineering function to the train set:

```
train_set <- do_feature_eng(train_set_raw)
train_set$classe <- as.factor(train_set$classe) # convert class to factor
dim(train_set)
```

```
#> [1] 19622    61
```

FE on the test dataset

Apply feature engineering function to the test set:

```
# the test set does not have a class feature
test_set <- do_feature_eng(test_set_raw)
dim(test_set)
```

```
#> [1] 20 61
```

What about the dates?

This is optional. Just trying to find any relationship between the dates where the measurements occurred in the train dataset versus those in the test.

Find how many unique values of the time stamp `cvtd_timestamp` there are.

```
unique(train_raw$cvtd_timestamp)
```

```
#>  [1] "05/12/2011 11:23" "05/12/2011 14:22" "05/12/2011 14:23"
#>  [4] "02/12/2011 13:32" "02/12/2011 13:33" "02/12/2011 14:57"
#>  [7] "28/11/2011 14:13" "28/11/2011 14:14" "30/11/2011 17:10"
#> [10] "05/12/2011 11:24" "30/11/2011 17:11" "02/12/2011 14:56"
#> [13] "02/12/2011 13:34" "02/12/2011 14:58" "05/12/2011 14:24"
#> [16] "05/12/2011 11:25" "30/11/2011 17:12" "28/11/2011 14:15"
#> [19] "02/12/2011 13:35" "02/12/2011 14:59"
```

```r
unique(test_raw$cvtd_timestamp)
```

```
#>  [1] "05/12/2011 14:23" "30/11/2011 17:11" "02/12/2011 13:33"
#>  [4] "28/11/2011 14:13" "30/11/2011 17:12" "05/12/2011 11:24"
#>  [7] "02/12/2011 14:57" "28/11/2011 14:14" "30/11/2011 17:10"
#> [10] "28/11/2011 14:15" "05/12/2011 14:22"
```

```r
intersect(unique(train_raw$cvtd_timestamp), unique(test_raw$cvtd_timestamp))
```

```
#>  [1] "05/12/2011 14:22" "05/12/2011 14:23" "02/12/2011 13:33"
#>  [4] "02/12/2011 14:57" "28/11/2011 14:13" "28/11/2011 14:14"
#>  [7] "30/11/2011 17:10" "05/12/2011 11:24" "30/11/2011 17:11"
#> [10] "30/11/2011 17:12" "28/11/2011 14:15"
```

There are 11 timestamps of 20 that are common to both datasets.

```r
outersect(unique(train_raw$cvtd_timestamp), unique(test_raw$cvtd_timestamp))
```

```
#> [1] "05/12/2011 11:23" "02/12/2011 13:32" "02/12/2011 14:56"
#> [4] "02/12/2011 13:34" "02/12/2011 14:58" "05/12/2011 14:24"
#> [7] "05/12/2011 11:25" "02/12/2011 13:35" "02/12/2011 14:59"
```

And nine timestamps that are either in the train or test datasets but there are not common at all in both.

Run machine learning models

These are the machine learning algorithms used:

- K-Nearest Neighbors
- Random Forest
- Bagged CART
- Single C5 Tree
- Neural Networks

Run model with KNN

Using K Nearest Neighbors algorithm for multiclass classification.

```r
# KNN train
tic()
set.seed(7)
trainControl <- trainControl(method="cv", number=5)

fit.knn <- train(classe~., data = train_set,
                 method = "knn",
                 metric = "Accuracy",
                 trControl = trainControl)
# summarize fit
print(fit.knn)
```

```
#> k-Nearest Neighbors
#>
#> 19622 samples
#>    60 predictor
#>     5 classes: 'A', 'B', 'C', 'D', 'E'
#>
#> No pre-processing
#> Resampling: Cross-Validated (5 fold)
#> Summary of sample sizes: 15698, 15698, 15697, 15698, 15697
#> Resampling results across tuning parameters:
#>
#>   k  Accuracy   Kappa
#>   5  0.9279383  0.9088403
#>   7  0.9109167  0.8872951
#>   9  0.8961885  0.8686400
#>
#> Accuracy was used to select the optimal model using the largest value.
#> The final value used for the model was k = 5.
```

```
(knn.pred <- predict(fit.knn, test_set[, -61]))
```

```
#>  [1] B A B A A E D B A A B C B A E E A B B B
#> Levels: A B C D E
```

```
table(knn.pred)
```

```
#> knn.pred
#> A B C D E
#> 7 8 1 1 3
```

```
toc()
```

```
#> 125.02 sec elapsed
```

```
# B A B A A E D B A A B C B A E E A B B B
# A B C D E
# 7 8 1 1 3
# Accuracy = 0.9279383
# 113.88 sec elapsed
# 104.61
```

Random Forest

```
train.x <- data.matrix(train_set[, -61])
train.y <- train_set[, 61]

set.seed(7)
bestmtry <- tuneRF(train.x, train.y, stepFactor=1.5, improve=1e-5, ntree = 2000)
print(bestmtry)
# ntree mtry
#  500   10
# 1000   15
# 2000   22
```

```
# https://machinelearningmastery.com/tune-machine-learning-algorithms-in-r
```

```r
# Random Forest: takes more than 30 minutes
tic()
set.seed(7)
trainControl <- trainControl(method = "cv", number = 5)

mtry <- sqrt(ncol(train_set))
mtry <- 15
tunegrid <- expand.grid(.mtry = mtry)


fit.rf <- train(classe~., data = train_set,
                 method = "rf",
                 metric = "Accuracy",
                 tuneGrid = tunegrid,
                 tuneLength = 15,
                 ntree = 1000,
                 trControl = trainControl,
                allowParallel=TRUE)
# summarize fit
print(fit.rf)
```

```
#> Random Forest
#>
#> 19622 samples
#>    60 predictor
#>     5 classes: 'A', 'B', 'C', 'D', 'E'
#>
#> No pre-processing
#> Resampling: Cross-Validated (5 fold)
#> Summary of sample sizes: 15698, 15698, 15697, 15698, 15697
#> Resampling results:
#>
#>   Accuracy   Kappa
#>   0.9987769  0.9984529
#>
#> Tuning parameter 'mtry' was held constant at a value of 15
```

```r
rf.pred <- predict(fit.rf, test_set[, -61])
rf.pred
```

```
#>  [1] B A B A A E D B A A B C B A E E A B B B
#> Levels: A B C D E
```

```r
table(rf.pred)
```

```
#> rf.pred
#> A B C D E
#> 7 8 1 1 3
```

```r
toc()
```

```
#> 580.94 sec elapsed
```

```r
# [1] B A B A A E D B A A B C B A E E A B B B
# Accuracy = 0.9952096
# 803.96 sec elapsed
# acc       mtry    ntr  time   tuneL
# 0.9986241  10     500  568.22  15
# 0.998726   15    1000  288.09  15
# 0.998726   15    1000  303.36  NA
# 0.9989807  22    2000 1278.56  NA
# 0.9980635   7.81   NA  280.75  15
# 0.9987769  15    1000  572.12  15
```

Bagged CART

```
# Bagged CART
tic()
set.seed(7)
trainControl <- trainControl(method="cv", number=5)

fit.treebag <- train(classe~., data = train_set,
                 method = "treebag",
                 metric = "Accuracy",
                 trControl = trainControl)
# summarize fit
print(fit.treebag)
```

```
#> Bagged CART
#>
#> 19622 samples
#>    60 predictor
#>     5 classes: 'A', 'B', 'C', 'D', 'E'
#>
#> No pre-processing
#> Resampling: Cross-Validated (5 fold)
#> Summary of sample sizes: 15698, 15698, 15697, 15698, 15697
#> Resampling results:
#>
#>   Accuracy   Kappa
#>   0.9954134  0.9941984
```

```
(treebag.pred <- predict(fit.treebag, test_set[, -61]))
```

```
#>  [1] B A B A A E D B A A B C B A E E A B B B
#> Levels: A B C D E
```

```
table(treebag.pred)
```

```
#> treebag.pred
#> A B C D E
#> 7 8 1 1 3
```

```
# A B C D E
# 7 8 1 1 3
toc()
```

```
#> 71.58 sec elapsed
```

```
# [1] B A B A A E D B A A B C B A E E A B B B
# 73.3 sec elapsed
# Accuracy   0.9954134 71.68 sec elapsed
```

Single C5.0 Tree

```r
# Single C5.0 Tree
tic()
set.seed(7)
trainControl <- trainControl(method="cv", number=5)

fit.c5 <- train(classe~., data = train_set,
                 method = "C5.0Tree",
                 metric = "Accuracy",
                 # preProcess=c("center", "scale"),
                 trControl = trainControl)
# summarize fit
print(fit.c5)
```

```
#> Single C5.0 Tree
#>
#> 19622 samples
#>    60 predictor
#>     5 classes: 'A', 'B', 'C', 'D', 'E'
#>
#> No pre-processing
#> Resampling: Cross-Validated (5 fold)
#> Summary of sample sizes: 15698, 15698, 15697, 15698, 15697
#> Resampling results:
#>
#>   Accuracy   Kappa
#>   0.9887882  0.9858175
```

```r
(c5.pred <- predict(fit.c5, test_set[, -61]))
```

```
#>  [1] B A B A A E D B A A B C B A E E A B B B
#> Levels: A B C D E
```

```r
table(c5.pred)
```

```
#> c5.pred
#> A B C D E
#> 7 8 1 1 3
```

```r
# A B C D E
# 7 8 1 1 3
#  [1] B A B A A E D B A A B C B A E E A B B B
# 44.94 sec elapsed Accuracy 0.9887882  25.39 sec elapsed
toc()
```

```
#> 25.05 sec elapsed
```

Neural Networks with `mxnet`

The last algorithm will be Neural Networks with `mxnet`. This happens to be the fastest from all the algorithms tested previously. What really takes time is finding the right parameters for the model. I am including a table measuring the time for different combination of the parameters. After tweaking some, I found out that the best activation was `relu`.

The fact that `mxnet` gave the most accurate results at the shortest time, made possible to generate a table of results. With the other algorithms could have taken a very long time to get such table.

Neural Network with `mxnet` standalone

In this case we convert the outcome to numeric, and then convert them back to character to show the results table.

```r
# classification with mxnet
library(mxnet)

# x part of data should be as a matrix type
train.x <- data.matrix(train_set[, -61])
train.y <- as.numeric(train_set[, 61]) - 1

# Get activation parameters from:
# https://media.readthedocs.org/pdf/mxnet-test/latest/mxnet-test.pdf
tic()
mx.set.seed(0)
model <- mx.mlp(train.x, train.y,      # multiple layer perceptron
                hidden_node = 60,
                activation = "relu",   # "tanh", "sigmoid", "softrelu"
                out_node = 5,
                out_activation = "softmax",
                num.round = 242,
                array.batch.size = 550,
                learning.rate = 0.001,
                momentum = 0.9,
                eval.metric = mx.metric.accuracy,
                array.layout = "rowmajor")
toc()
```

```
#> 17.06 sec elapsed
```

Table of Results for Neural Network, `mxnet` standalone

```
 nround act  bsize acc   lrate hidden time  A B C D E
   100 tanh  3000 0.834 0.01    60    3.65  7 6 2 2 3
   100 tanh  4000 0.856 0.01    60    3.34  7 6 2 3 2
   100 tanh  5500 0.867 0.01    60    3.58  7 6 1 2 4
   200 tanh  5500 0.888 0.01    60    6.95  7 5 1 4 3
   400 tanh  5500 0.932 0.01    60   14.19  7 7 2 2 2
   600 tanh  5500 0.943 0.01    60   21.64  7 7 2 1 3
   600 tanh  6000 0.959 0.001   60   24.52  7 7 1 2 3
   800 tanh  6000 0.962 0.001   65   33.78  7 7 1 2 3
   800 tanh  6000 0.953 0.001   55   31.89  7 7 1 2 3
   800 tanh  6000 0.924 0.001   30   29.62  7 7 1 2 3
   800 tanh  1000 0.963 0.001   60   39.18  7 8 1 1 3
   800 tanh  2000 0.971 0.001   60   32.16  7 8 1 1 3
   800 tanh  1800 0.972 0.001   62   33.22  7 8 1 1 3
   800 tanh  1800 0.970 0.001   58   35.97  7 8 1 1 3
```

```
    1000 tanh  1800 0.977 0.0005 60  50.54  7 8 1 1 3
    1200 tanh  1800 0.978 0.001  60  52.25  7 8 1 1 3
    1500 tanh  1800 0.979 0.001  60  69.61  7 8 1 1 3
    1000 tanh  1800 0.977 0.001  60  43.06  7 8 1 1 3
     500 tanh  1800 0.960 0.001  60  20.02  7 8 1 1 3
     600 tanh  1800 0.964 0.001  60  26.00  7 8 1 1 3
     400 tanh  1800 0.961 0.001  60  16.06  7 7 1 2 3
     500 tanh  1800 0.960 0.0008 60  20.55  7 8 1 1 3
     500 tanh  1800 0.960 0.002  60  20.63  8 7 1 1 3
    1200 tanh  1800 0.934 0.005  60  47.43  7 8 1 1 3
     800 tanh  1700 0.971 0.001  60  34.27  7 7 1 2 3
     700 tanh  2000 0.968 0.001  60  30.99  7 8 1 1 3
     600 tanh  3000 0.965 0.001  60  26.75  7 8 1 1 3
     600 tanh  4000 0.962 0.001  60  22.88  7 8 1 1 3
     300 relu  1800 0.986 0.001  60  11.29  7 8 1 1 3
     242 relu   550 0.992 0.001  60  14.18  7 8 1 1 3
     300 relu   800 0.990 0.001  60  16.94  7 8 1 1 3
     255 relu   800 0.989 0.001  60  14.41  7 8 1 1 3
     243 relu   800 0.989 0.001  60  14.18  7 8 1 1 3
     300 relu   400 0.982 0.001  60  29.40  7 7 1 2 3
     300 relu   200 0.962 0.001  60  55.07  7 8 1 1 3
     300 tanh  1800 0.951 0.001  60  12.41  7 8 1 1 3
     300 srelu 1800 0.881 0.001  60  25.89  7 7 1 2 3

lrate: learning rate; nround: number of iterations;
acc: accuracy; hid: hidden layers; time: time iterations;
bsize: batch size
```

```r
# test predictors. exclude last column which is the identifier
test.x <- data.matrix(test_set[, -61])

# these are the correct labels of the target
test.y <- c("B", "A", "B", "A", "A", "E", "D", "B", "A", "A",
            "B", "C", "B", "A", "E", "E", "A", "B", "B", "B")

pred <- predict(model, test.x, array.layout = "rowmajor")
pred.label <- max.col(t(pred)) - 1

# convert integers of outcome back to letters
pred.label.ltr <- LETTERS[pred.label + 1]

# confusion matrix
cfm <- confusionMatrix(as.factor(pred.label.ltr), as.factor(test.y))
print(cfm)
```

```
#> Confusion Matrix and Statistics
#> 
#>           Reference
#> Prediction A B C D E
#>          A 7 0 0 0 0
#>          B 0 8 0 0 0
#>          C 0 0 1 0 0
#>          D 0 0 0 1 0
#>          E 0 0 0 0 3
#> 
```

```
#> Overall Statistics
#>
#>                Accuracy : 1
#>                  95% CI : (0.8316, 1)
#>     No Information Rate : 0.4
#>     P-Value [Acc > NIR] : 1.1e-08
#>
#>                   Kappa : 1
#>  Mcnemar's Test P-Value : NA
#>
#> Statistics by Class:
#>
#>                      Class: A Class: B Class: C Class: D Class: E
#> Sensitivity              1.00      1.0     1.00     1.00     1.00
#> Specificity              1.00      1.0     1.00     1.00     1.00
#> Pos Pred Value           1.00      1.0     1.00     1.00     1.00
#> Neg Pred Value           1.00      1.0     1.00     1.00     1.00
#> Prevalence               0.35      0.4     0.05     0.05     0.15
#> Detection Rate           0.35      0.4     0.05     0.05     0.15
#> Detection Prevalence     0.35      0.4     0.05     0.05     0.15
#> Balanced Accuracy        1.00      1.0     1.00     1.00     1.00
```

```r
table(LETTERS[pred.label + 1])
```

```
#>
#> A B C D E
#> 7 8 1 1 3
```

```r
LETTERS[pred.label + 1]
```

```
#>  [1] "B" "A" "B" "A" "A" "E" "D" "B" "A" "A" "B" "C" "B" "A" "E" "E" "A"
#> [18] "B" "B" "B"
```

```r
# read CSV with mxnet results
mxnet_results <- read.csv(file = "mxnet.csv", stringsAsFactors = FALSE)
```

```r
# save mxnet results to RDA file
save(mxnet_results, file = "mxnet.rda")
```

Neural Network with `mxnet` and `caret`

Since we will compare all the algorithms, we need to do some cross-validation and resampling with caret. We will call `mxnet` from `caret` to produce the output for the comparison. There are a couple of parameter that we used with mxnet that are not available in the standard `caret`, so we will make a customized model for running `mxnet` with caret.

The parameters that cannot be sent from caret are:

```
array.batch.size = 550
out_activation = "softmax"
mx.init.Xavier (the initializer)
```

The code for this custom made method can be found in the file `custom_mxnet.R`.

```r
# using customized mxnet
# with array.batch.size = 550,
library(mxnet)
```

```
tic()
mx.set.seed(0)
source("custom_mxnet.R")

# x part of data should be as a matrix type
train.x <- data.matrix(train_set[, -61])
train.y <- train_set[, 61]

mlp_grid <- expand.grid(
  layer1 = 60,
  layer2 = 0,
  layer3 = 0,
  learning.rate = 0.00075,
  momentum = 0.95,
  dropout = 0,
  activation = "relu"
  )

fit.mxnet <- train(x = train.x, y = train.y,
                   method = new.modelInfo,
                   trControl = trainControl(method = "cv", number = 5),
                   tuneGrid = mlp_grid,
                   num.round = 700
                   )
toc()
```

```
#> 218.25 sec elapsed
```

Table of Results for Neural Network, `mxnet` with caret

```
  fac lrate   mag    nround acc       hid     time   mom  bsize
  avg 0.00075 0.0003 1200   0.9849664 60x0x0  613.67 0.95  550
  avg 0.00075 0.0003  700   0.9836917 60x0x0  242.61 0.95  750
  avg 0.00075 0.0003 1200   0.9826220 60x0x0  242.61 0.95  750

  avg 0.00075 0.0003  700   0.9825182 60x0x0  334.42 0.95  550
  avg 0.0007  0.0003  700   0.9817039 60x0x0  308.65 0.95  550

  avg 0.00075 0.0003  700   0.9809403 60x0x0  247.2  0.95  700
  avg 0.0008  0.0003  700   0.9804814 60x0x0  321.05 0.95  550
  avg 0.0008  0.0003  700   0.9802274 60x0x0  182.26 0.95 1550

  avg 0.00075 0.0003  700   0.9774747 60x0x0  208.55 0.95  800
  avg 0.00075 0.0003  700   0.9749770 60x0x0  208.55 0.95 1000
  avg 0.001   0.0003  400   0.9727351 60x0x0  163.79 0.95  550
  avg 0.001   0.0003  500   0.9732951 60x0x0  196.61 0.95  550
  avg 0.001   0.0003  242   0.9631016 60x0x0  97.28  0.95  550
  avg 0.001   0.0003  100   0.9552554 60x0x0  37.72  0.95  550
  avg 0.0001  0.0003  700   0.9734995 60x0x0  37.16  0.95  550
  avg 0.001   0.0003  700   0.9759978 60x0x0  291.11 0.95  550
  avg 0.0001  0.0003  100   0.9531142 60x0x0  37.16  0.94  550
  avg 0.0001  0.0003  100   0.9538278 60x0x0  36.95  0.95  550
  avg 0.00075 0.0003  100   0.9572933 60x0x0  23.52  0.95 1550
  avg 0.0001  0.0003  100  0.9350735 60x0x0  37.27  0.96  550
  avg 0.0001  0.0003  100  0.7590508 60x0x0  36.93  0.99  550
  avg 0.0001  0.0003  100  0.9499029 60x0x0  36.22  0.80  550
  out 0.0001  0.0003  242  0.9703391 60x0x0
  avg 0.0001  0.0003  242  0.9713581 60x0x0
  avg 0.0001  0.0003  300  0.9751811 60x0x0
  avg 0.0001  0.0003  100  0.9493936 60x0x0         0.90  550


fac: factor type; lrate: learning rate; nround: num.round;
acc: accuracy; hid: hidden layers; time: time iterations;
mom: momentum; bsize: batch size
```

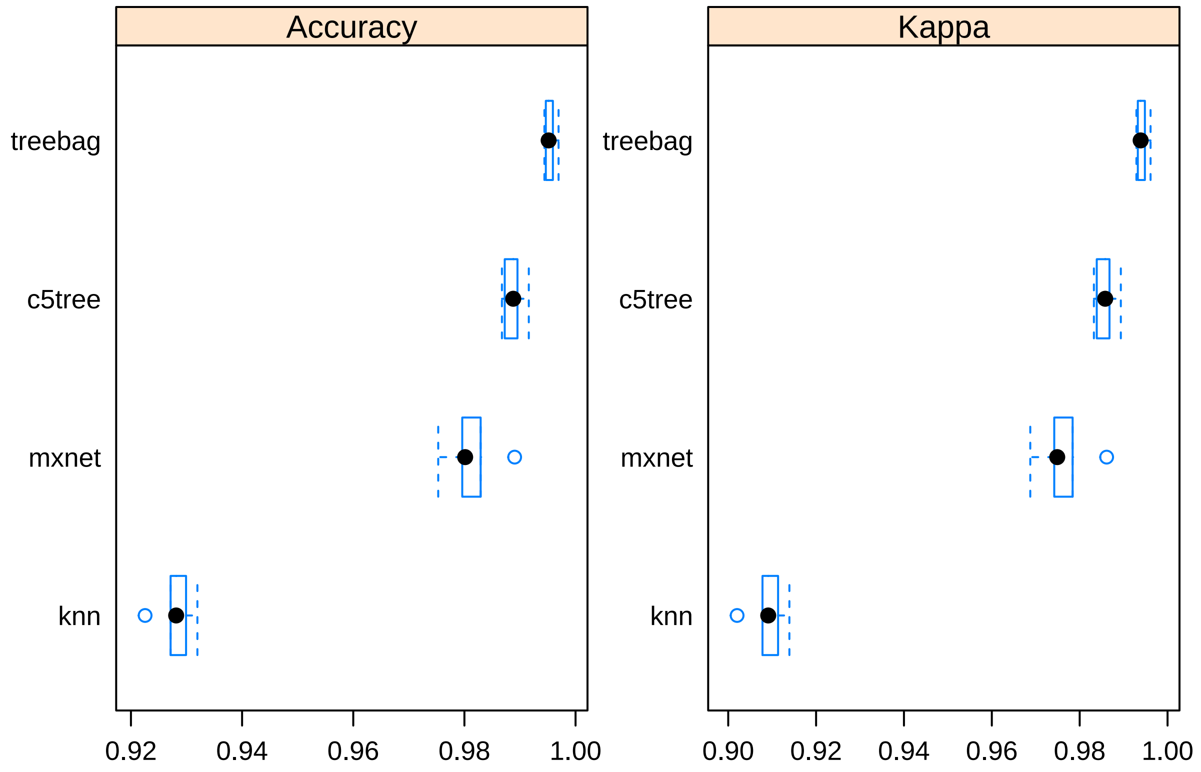
Final comparison

In this section we compare all the machine learning algorithms used. The comparison is made by using the caret function `resamples`.

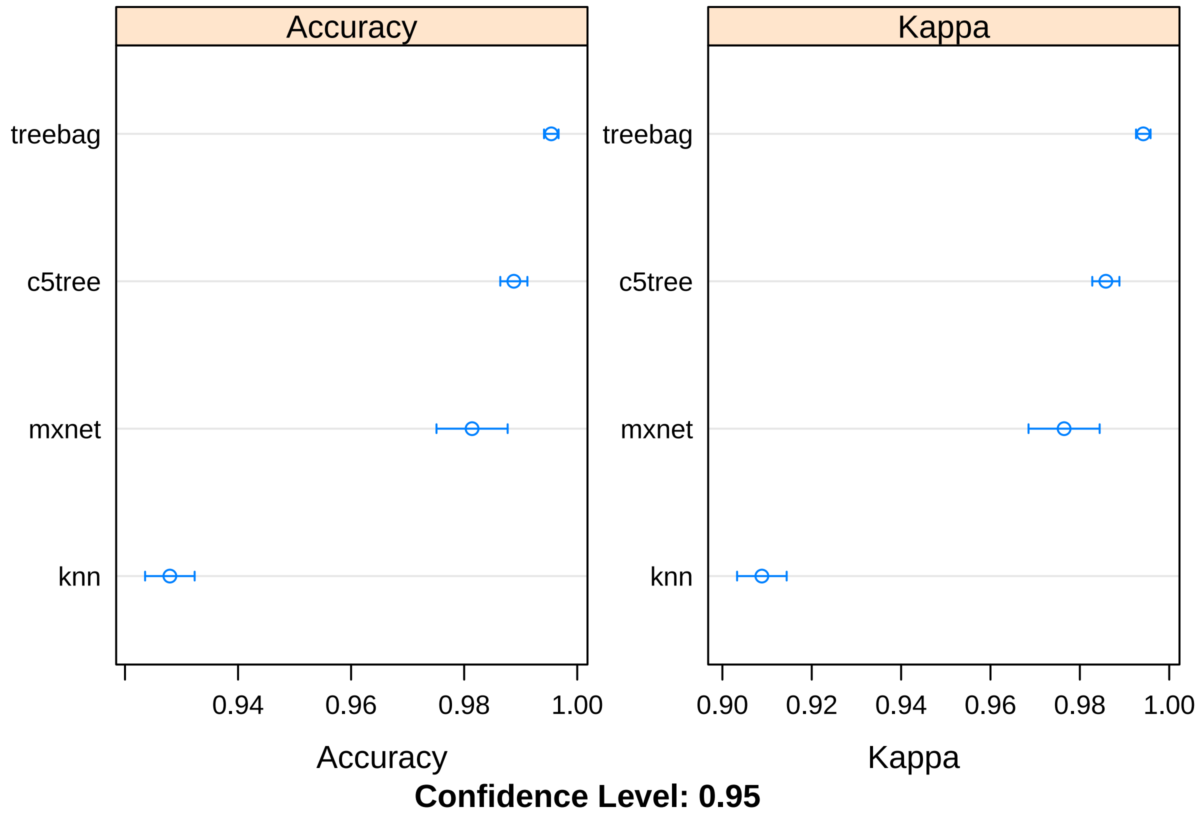
```
results <- resamples(list(
    knn     = fit.knn,
    c5tree  = fit.c5,
    treebag = fit.treebag,
    # nnet    = fit.nnet,
    mxnet   = fit.mxnet
))


summary(results)
```

```
#>
#> Call:
#> summary.resamples(object = results)
#>
#> Models: knn, c5tree, treebag, mxnet
#> Number of resamples: 5
#>
#> Accuracy
#>              Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
#> knn     0.9225478 0.9271338 0.9281346 0.9279383 0.9299185 0.9319572    0
#> c5tree  0.9867516 0.9872579 0.9887898 0.9887882 0.9895515 0.9915902    0
#> treebag 0.9943949 0.9946497 0.9951580 0.9954134 0.9959225 0.9969419    0
#> mxnet   0.9752866 0.9796126 0.9801375 0.9813994 0.9829212 0.9890390    0
#>
#> Kappa
#>              Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
#> knn     0.9020216 0.9077985 0.9091062 0.9088403 0.9113511 0.9139240    0
#> c5tree  0.9832379 0.9838840 0.9858173 0.9858175 0.9867832 0.9893652    0
#> treebag 0.9929103 0.9932318 0.9938756 0.9941984 0.9948423 0.9961318    0
#> mxnet   0.9687543 0.9742079 0.9748882 0.9764755 0.9783901 0.9861369    0
```

```
# box and whisker plots to compare models
scales <- list(x=list(relation="free"), y=list(relation="free"))
bwplot(results, scales=scales)
```



```r
# dot plots of accuracy
scales <- list(x=list(relation="free"), y=list(relation="free"))
dotplot(results, scales=scales)
```



Conclusion

1. The prediction on the test dataset takes the following form:
   B A B A A E D B A A B C B A E E A B B B
2. The prediction corresponds to the following combination, represented by the table:

```
A B C D E
7 8 1 1 3
```

3. The most accurate machine learning algorithm is `treebag` with 0.995 and out of sample error of 0.005.
4. The least accurate algorithm is `knn` with 0.928 and out of sample error of 0.072.