STAT5104 Data Mining Project

CHAN Yiu Fung (1155010561)

CHUNG Wai Tung (1155118104)

LAM Siu Hung (1006201460)

LAU Chiu Kit (1155120306)

WONG Tsz Wing (1004666311)

WONG Yiu Chung (1155017920)

13 May, 2019

Abstract

This report explores the possibility of predicting human movement using spatial data of exercises. Seven data mining models are used to extract latent pattern from the dataset. The present study is successful in predicting various types of human physical movements with extremely high accuracy. Work are divided evenly among authors.

All code, R objects, and other supporting files can be found at this Github repository: https://github.com/emailyc/STAT_5104_Project.git

Contents

1 Introduction

The research topic, Human Activity Recognition (HAR), is becoming more and more popular among the computing research community. In the traditional HAR research, researchers mainly focused on predicting what activity a person was performing at a specific point of time. Meanwhile, latest researches have shifted the focus on how well the activities have been performed. In real-life, we can apply the ideas, for example, in sports training. In this report, we explored the Weight Lifting Exercises Dataset and attempted to assess if the participants performed the specific weight lifting exercise, Unilateral Dumbbell Biceps Curl (hereafter refers to the exercise), correctly from the data collected via various sensors attached on different parts of the body, which includes arm, belt, forearm, and dumbbell. The type of mistakes in the exercise can also be identified. Six male participants aged between 20-28 years were asked to wear a number of body sensors to perform one set of 10 repetitions of the exercise. Based on the sensor data collected, we can trace the outcome of the performance accordingly. The performance outcome can be grouped into five classes, one corresponding to the specified execution of the exercise, while the other 4 classes corresponding to some common mistakes. Each sensor generated a set of readings in three dimensions (Velloso, Bulling, Gellersen, Ugulino, & Fuks, 2013).

2 The Data

The data for this project come from this source: https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv.

The dataset contains 160 variables, which include one target variable "Class" and 159 readings from the sensors. Each "Class" represents a specific performance outcome. This dataset is unique in a way that while there are many variables, each are fundamentally the same, i.e. each set of three columns represents a sensor attached on different parts of the body. Each sensor generates data according to its rotation around a spatial axis, giving spatial data on three dimensions. Hence all 159 columns of data are highly similar to each other (Velloso et al., 2013).

The target variable has the following levels:

- Class A: exactly according to the specification (i.e. performing the exercise correctly);
- Class B: throwing the elbows to the front;
- Class C: lifting the dumbbell only halfway;
- Class D: lowering the dumbbell only halfway; and

- Class E: throwing the hips to the front.

2.1 Data preparation

For the following reasons:

- Predictor with well-defined meaning
- Similar scale
- Similar range
- All continuous

scaling / standardising may not yield the best result since this may cause distortion. Data are not rescaled or normalised in this report.

2.1.1 Load data

2.1.2 Data cleaning

The data are further processed by:

* Removing the first seven fields which are just descriptive data

* Removing near zero variance fields

* Removing columns with more than 10% missing values

53 columns remains in the dataset post-cleaning. The following table lists the remaining variables

variable	classe	first_values
roll_belt	double	1.41, 1.41, 1.42, 1.48, 1.48, 1.45
pitch_belt	double	8.07, 8.07, 8.07, 8.05, 8.07, 8.06
yaw_belt	double	-94.4, -94.4, -94.4, -94.4, -94.4, -94.4
total_accel_belt	integer	3, 3, 3, 3, 3, 3
gyros_belt_x	double	0, 0.02, 0, 0.02, 0.02, 0.02
gyros_belt_y	double	0, 0, 0, 0, 0.02, 0
gyros_belt_z	double	-0.02, -0.02, -0.02, -0.03, -0.02, -0.02
accel_belt_x	integer	-21, -22, -20, -22, -21, -21
accel_belt_y	integer	4, 4, 5, 3, 2, 4
accel_belt_z	integer	22, 22, 23, 21, 24, 21

variable	classe	first_values
magnet_belt_x	integer	-3, -7, -2, -6, -6, 0
magnet_belt_y	integer	599, 608, 600, 604, 600, 603
magnet_belt_z	integer	-313, -311, -305, -310, -302, -312
roll_arm	double	-128, -128, -128, -128, -128, -128
pitch_arm	double	22.5, 22.5, 22.5, 22.1, 22.1, 22
yaw_arm	double	-161, -161, -161, -161, -161, -161
total_accel_arm	integer	34, 34, 34, 34, 34, 34
gyros_arm_x	double	0, 0.02, 0.02, 0.02, 0, 0.02
gyros_arm_y	double	0, -0.02, -0.02, -0.03, -0.03, -0.03
gyros_arm_z	double	-0.02, -0.02, -0.02, 0.02, 0, 0
accel_arm_x	integer	-288, -290, -289, -289, -289, -289
accel_arm_y	integer	109, 110, 110, 111, 111, 111
accel_arm_z	integer	-123, -125, -126, -123, -123, -122
magnet_arm_x	integer	-368, -369, -368, -372, -374, -369
magnet_arm_y	integer	337, 337, 344, 344, 337, 342
magnet_arm_z	integer	516, 513, 513, 512, 506, 513
roll_dumbbell	double	13.05217456, 13.13073959, 12.85074981, 13.43119971, 13.37871611, 13.38245941
pitch_dumbbell	double	-70.49400371, -70.63750507, -70.27811982, -70.39379464, -70.42855971, -70.81758832
yaw_dumbbell	double	-84.87393888, -84.71064711, -85.14078134, -84.87362553, -84.85305745, -84.46500278
total_accel_dumbbell	integer	37, 37, 37, 37, 37, 37
gyros_dumbbell_x	double	0, 0, 0, 0, 0, 0
gyros_dumbbell_y	double	-0.02, -0.02, -0.02, -0.02, -0.02, -0.02
gyros_dumbbell_z	double	0, 0, 0, -0.02, 0, 0
accel_dumbbell_x	integer	-234, -233, -232, -232, -233, -234
accel_dumbbell_y	integer	47, 47, 46, 48, 48, 48
accel_dumbbell_z	integer	-271, -269, -270, -269, -270, -269
magnet_dumbbell_x	integer	-559, -555, -561, -552, -554, -558
magnet_dumbbell_y	integer	293, 296, 298, 303, 292, 294
magnet_dumbbell_z	double	-65, -64, -63, -60, -68, -66
roll_forearm	double	28.4, 28.3, 28.3, 28.1, 28, 27.9
pitch_forearm	double	-63.9, -63.9, -63.9, -63.9, -63.9, -63.9

variable	classe	first_values
yaw_forearm	double	-153, -153, -152, -152, -152, -152
total_accel_forearm	integer	36, 36, 36, 36, 36, 36
gyros_forearm_x	double	0.03, 0.02, 0.03, 0.02, 0.02, 0.02
gyros_forearm_y	double	0, 0, -0.02, -0.02, 0, -0.02
gyros_forearm_z	double	-0.02, -0.02, 0, 0, -0.02, -0.03
accel_forearm_x	integer	192, 192, 196, 189, 189, 193
accel_forearm_y	integer	203, 203, 204, 206, 206, 203
accel_forearm_z	integer	-215, -216, -213, -214, -214, -215
magnet_forearm_x	integer	-17, -18, -18, -16, -17, -9
magnet_forearm_y	double	654, 661, 658, 658, 655, 660
magnet_forearm_z	double	476, 473, 469, 469, 473, 478
classe	integer	A, A, A, A, A, A

2.1.3 Slicing into training and testing sets

The training data set is sliced into 80% for training and 20% for testing.

2.1.4 Overview of cleaned dataset

```
## [1] 15699    53
```

2.2 Principal Component Analysis

Principal Component Analysis (PCA) is a dimension reduction technique. A reduced dataset allows faster processing and smaller storage. In the context of data mining, PCA reduces the number of variables to be used in a model by focusing only on the components accounting for the majority of the variance. Highly correlated variables are also removed as a result of PCA.

Here, PCA is able to reduce the dimension (number of predictors) of the datasets from 52 to 37 while retaining 99% of the information. This reduces model complexity and improves scalability.

As a side note, PCA is usually performed on scaled or normalised dataset to prevent the resulting principle sub-space from being dominated by variables with large scales. As mentioned above, because the variables in the dataset are of similar nature, scaling or normalised provides little added benefits. Hence such procedures are not used.

3 Methods

3.1 Learning Models

Seven learning methods are adopted in this report. Namely:

1. Naive Bayes;
2. K-Nearest Neighbor;
3. Multinomial Logistic Regression;
4. Decision Tree;
5. Tree Bagging.
6. Random Forest;
7. Neuro Network;

The methods can be classified as eager learner (Decision Tree, Tree Bagging, Random Forest, and Neuro Network) and lazy learner (K-Nearest Neighbor and Naive Bayes). The library `Caret` is used to generate training models .

3.2 Resampling: Cross Validation

Cross Validation is performed on each training methods to infer model performance.

3.2.1 Choosing between LOOCV and k-Fold

Leave-One-Out Cross-Validation (LOOCV) and k-Fold are common resampling methods for accessing model performance. While LOOCV estimates test error with lowest bias (averaging validation errors across n models), k-Fold CV is much less computationally intensive.

Yet there is another advantage to using k-Fold CV: k-Fold CV often gives more accurate estimates of the test error; estimates produced by LOOCV is often plagued by high variance compared to that produced by k-Fold CV. This is because test errors in LOOCV are produced by models trained on virtually identical datasets. The final averaged statistic is an average of statistics from n models which are highly positively correlated. On the other hand, k-Fold CV outputs k (which is usually much less than n) statistics which are less correlated as there are less overlap among models. The average of strongly correlated quantities has higher variance than the average of weakly correlated quantities; hence the estimated statistics from LOOCV tends to have higher variance than that from k-Fold (James, Witten, Hastie, & Tibshirani, 2013).

The dataset in the report consists of relatively large number of observations (15699). Hence a 10 fold cross-validation is performed.

3.2.2 Performance Measures for Multi-Class Problems

The following are some of many viable model performance metrics for choosing the best model out of the many models `caret::train` create using different parameters. For example, `caret::train` tries different k in KNN.

- Accuracy and Kappa
- Area Under ROC Curve
- F1
- Logarithmic Loss

Log loss often works well with multi-class problems. By setting the parameter metric to `logLoss`, model selection will be based on lowest log loss.

3.3 Lazy Learners

Lazy learners simply store the training data without performing further munging, until a test dataset is presented (Thakurta, 2015). During model training, Lazy Learners require significantly less computational operation as there is no new algorithm being developed; for the same reason, Lazy Learners are slow when used for prediction because new data are used to compute predictions instead of relying on a pre-calculated algorithm.

Naive Bayes and K-Nearest-Neighbor (KNN) are used in this section. Both lazy learners are expected to perform quickly on large datasets like the one used in this report. KNN relies heavily on Euclidean distance (L2 norm) between observations and is more appropriate on scaled or normalised data. Hence, this model is expected to perform less well than other data mining models used in this report.

3.4 Multinomial logistic regression

Multinomial logistic regression posts no assumptions such as normality, linearity, or homoscedasticity. This makes it more flexible than other more powerful techniques such as discriminant analysis (Starkweather & Moske, 2011).

3.5 Tree based models

Tree-based methods tend to perform well on unprocessed data (i.e. without normalizing, centering, scaling features).

Decision Trees often produce predictions with low bias but high variance. The more complex the tree, the more apparent this becomes (overfitting). Methods have been proposed to overcome this issue. This includes Bootstrap Aggregation (Bagging), as well as Random Forest.

The idea behind tree bagging is to create many trees, each trained from bootstrapped data from the original dataset. Each tree is slightly different from each other because they are trained with mildly different datasets. Classification decision is then performed by popular vote across all trees. This method reduces variance by averaging decisions among many trees. There is a caveat though: trees turn out to be very similar to each other when there exists a (or few) extremely strong predictor, following by some moderately strong predictors. Each tree will have similar node splitting because of these strong predictors, which renders each tree to have practicality the same decision rules. Unfortunately, as mentioned above, the variance of the averages of highly correlated quantities is also high. This means tree bagging provides little improvements in terms of variance reduction.

Random Forest enhances tree bagging through a tweak: at each node split, the algorithm randomly picks a subset of size m predictors out of all p, then choose the best predictor for this node split as normally seen in decision trees. This way, each tree is more likely to be different from each other. And hence their averages are less varying. The choice of m is often the square root of p but other method of choosing m also exists (James et al., 2013).

Note that in the code above, both models `treebag` and `rf` employ the training method rf. This is because tree bagging is in fact a special case of Random Forest where $m = p$.

3.6 Neuro-Net

R doesn't provide an easy way to model multilayer perceptron (Neuro Network). Hence a single-layer perceptron is modelled below. Neuro Networks tend to be scale invariant (just like tree based models): rescaling the input vector is equivalent to changing the weights and biases of the network, resulting in the exact same outputs as before.

The parameter `size` specifies the number of units in the hidden layer. Sizes ranging from 1 to 10 are experimented for best results. The parameter `decay` specifies the regularisation of the number of nodes: model with high node counts are more heavily penalised

3.7 Compare Models

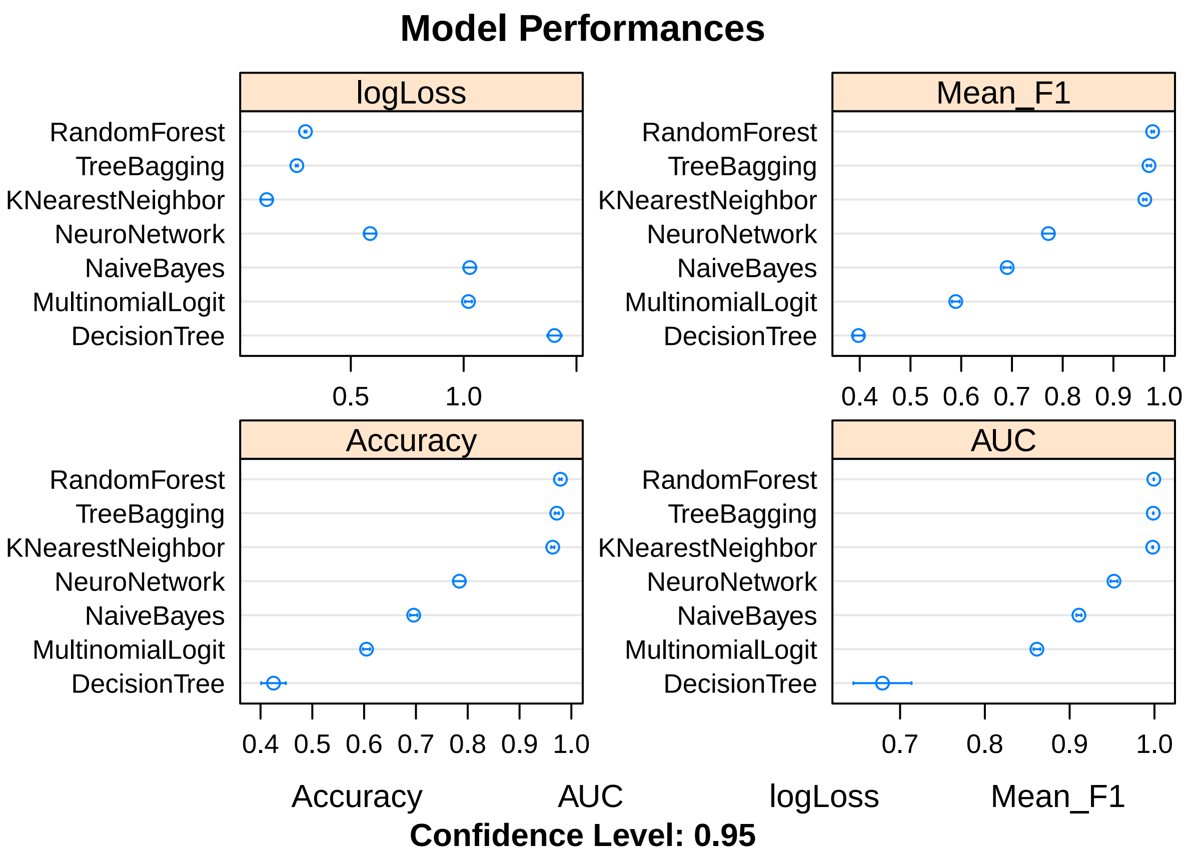
```
##  [1] "Accuracy"               "AUC"
##  [3] "Kappa"                  "logLoss"
##  [5] "Mean_Balanced_Accuracy" "Mean_Detection_Rate"
##  [7] "Mean_F1"                "Mean_Neg_Pred_Value"
##  [9] "Mean_Pos_Pred_Value"    "Mean_Precision"
## [11] "Mean_Recall"            "Mean_Sensitivity"
## [13] "Mean_Specificity"       "prAUC"
```

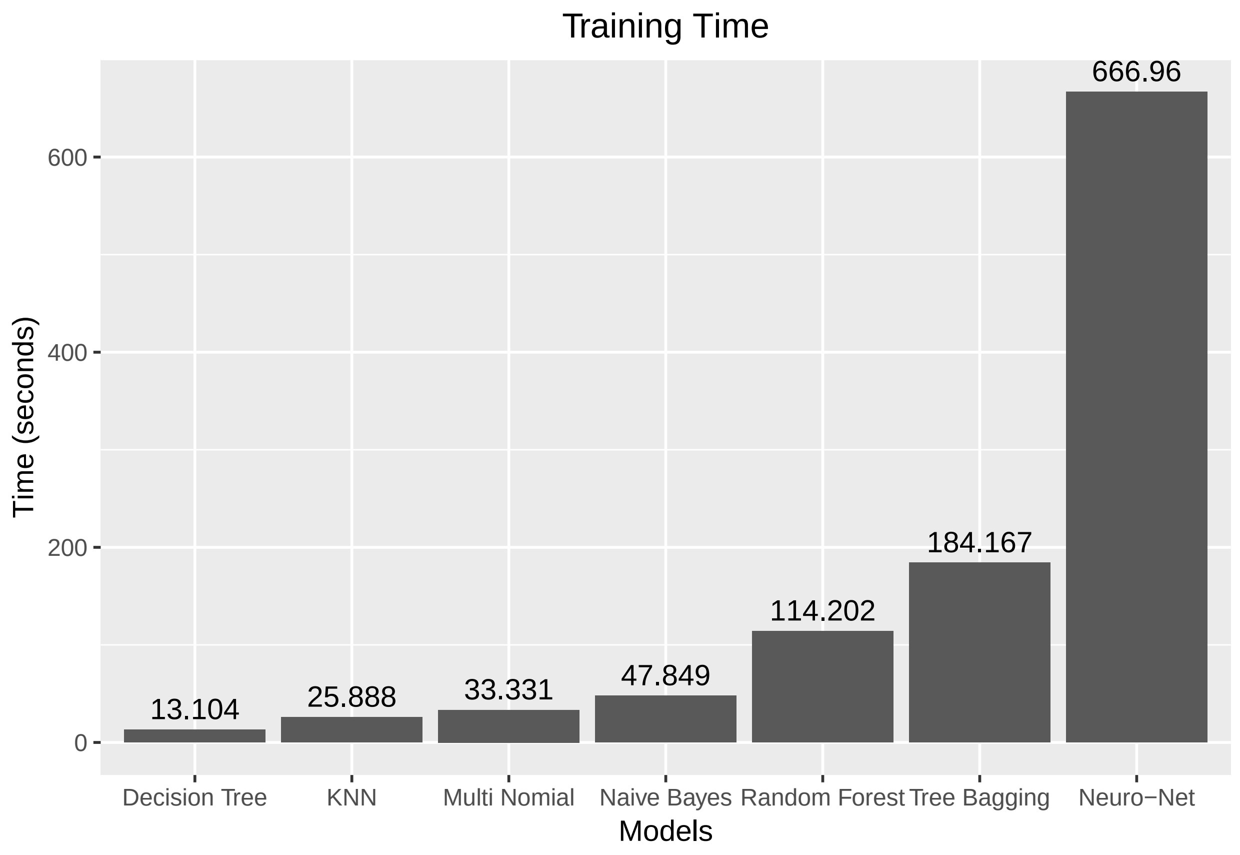
There are a total of 14 metrics for comparing models.

4 Findings

4.1 Comparing models

Averages of LogLoss, Accuracy, F1 and AUC are used to assess the performances of the models.


Model Performances
logLoss
Mean_F1
Accuracy
AUC
RandomForest
TreeBagging
KNearestNeighbor
NeuroNetwork
NaiveBayes
MultinomialLogit
DecisionTree
0.5
1.0
0.4 0.5 0.6 0.7 0.8 0.9 1.0
0.7 0.8 0.9 1.0
Accuracy AUC logLoss Mean_F1
Confidence Level: 0.95



Among the seven models, Random Forest, Tree Bagging and KNN outperform the other four models. Surprisingly, KNN appears to outperform all other models holistically: lowest log Loss value at 0.1274666, highest Mean F1, Accuracy, and AUC at 0.9618464, 0.9640746, 0.9979557 respectively. In addition, KNN has the second lowest learning time (which is less surprising given its lazy learning nature) at 25.888 seconds (wall clock), beaten by Decision Tree only.

The training data used in the report remain at their original scale. KNN is supposed to suffer from neighbors being aligned along the direction of the axis with the smaller range. This somewhat reaffirms the notion of the dataset having variables with similar scales and ranges.

KNN performs well at various metrics, as well as having a low training time. Thus, KNN is chosen as the final model to be tested.

As a last note, Decision Tree has high variance across different metrics. This confirms earlier analysis on Tree based models regarding its drawback of having high variance.

5 Model Performance

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1104   15    0    3    0
##          B    4  727    7    0    4
##          C    4   16  665   23    3
##          D    3    1    7  614    1
##          E    1    0    5    3  713
##
## Overall Statistics
##
##                Accuracy : 0.9745
##                  95% CI : (0.9691, 0.9792)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9678
##
##  Mcnemar's Test P-Value : 0.0008817
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9892   0.9578   0.9722   0.9549   0.9889
## Specificity            0.9936   0.9953   0.9858   0.9963   0.9972
## Pos Pred Value         0.9840   0.9798   0.9353   0.9808   0.9875
## Neg Pred Value         0.9957   0.9899   0.9941   0.9912   0.9975
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2814   0.1853   0.1695   0.1565   0.1817
## Detection Prevalence   0.2860   0.1891   0.1812   0.1596   0.1840
## Balanced Accuracy      0.9914   0.9765   0.9790   0.9756   0.9930
```

`KNN` is able to predict future data with 0.9745093 accuracy and 0.9677523 Kappa.

6 Discussion

Predictive accuracy and trivial computational requirement have led the authors to conclude the KNN algorithm as the final model. One might question why the more accurate models aren't used (Random Forest, Tree Bagging), given the fact that this report aims at tackling a prediction problem. First, a look at the model objects may answer this question

```
## 12497440 bytes
```

```
## 61133528 bytes
```

One is about 5 times the size of the other. If a model is to be run on a smaller device, which sports devices usually are, a bigger model may cause storage problem.

Another reason for choosing a simpler model is interpretability. Using KNN, one can safely construe each particular exercise have similar spatial readings (close Euclidean distance). On the other hand, one may find elucidating exercise grouping using the Random Forest paradigm difficult.

6.1 Final Remarks

This report pays pittance regard to exploratory data analysis: apart from delineating the background, there are no graphical nor analytically insight of how variables might correlate to each other. Given the success of the KNN algorithm on this dataset, which is heavily distance based, there is a good chance that graphical tools can yield good explanation for KNN's ascendancy. Given the high dimension (53) nature of the data, as well as writing space limit, the authors have opted not to use graphical exploratory data analysis.

This analysis has perhaps debased the Neuro-Network model somewhat unfairly. Looking at the code at Neuro-Net, 10 values are specified for the parameter `size`, and 8 values are specified for `decay`. At least 10 * 8 models with varying sizes are built and compared. This is significantly more model built than other methods. Nonetheless, the performance of such sophisticated algorithm under-performs other simpler models, hence discarded.

Had other data mining techniques been discussed in class, such as Support Vector Classifier, and Discriminant Analysis, more models would be tried here.

7 Session Info

```
## - Session info ----------------------------------------------------------
##  setting  value
##  version  R version 3.5.3 (2019-03-11)
##  os       Windows 10 x64
##  system   x86_64, mingw32
##  ui       RTerm
##  language (EN)
##  collate  English_Hong Kong SAR.1252
##  ctype    English_Hong Kong SAR.1252
##  tz       Asia/Taipei
##  date     2019-05-12
##
## - Packages --------------------------------------------------------------
##  package      * version    date       lib source
##  assertthat     0.2.1      2019-03-21 [1] CRAN (R 3.5.3)
##  backports      1.1.4      2019-04-10 [1] CRAN (R 3.5.3)
##  callr          3.2.0      2019-03-15 [1] CRAN (R 3.5.3)
##  caret        * 6.0-84     2019-04-27 [1] CRAN (R 3.5.3)
##  class          7.3-15     2019-01-01 [2] CRAN (R 3.5.3)
##  cli            1.1.0      2019-03-19 [1] CRAN (R 3.5.3)
##  codetools      0.2-16     2018-12-24 [2] CRAN (R 3.5.3)
##  colorspace     1.4-1      2019-03-18 [1] CRAN (R 3.5.3)
##  crayon         1.3.4      2017-09-16 [1] CRAN (R 3.5.0)
##  data.table     1.12.2     2019-04-07 [1] CRAN (R 3.5.3)
##  desc           1.2.0      2018-05-01 [1] CRAN (R 3.5.3)
##  devtools       2.0.2      2019-04-08 [1] CRAN (R 3.5.3)
##  digest         0.6.18     2018-10-10 [1] CRAN (R 3.5.2)
##  doParallel     1.0.14     2018-09-24 [1] CRAN (R 3.5.3)
##  dplyr        * 0.8.0.1    2019-02-15 [1] CRAN (R 3.5.2)
##  e1071          1.7-1      2019-03-19 [1] CRAN (R 3.5.3)
##  evaluate       0.13       2019-02-12 [1] CRAN (R 3.5.2)
## foreach        1.4.4      2017-12-12 [1] CRAN (R 3.5.1)
## fs             1.3.1      2019-05-06 [1] CRAN (R 3.5.3)
## generics       0.0.2      2018-11-29 [1] CRAN (R 3.5.1)
## ggplot2      * 3.1.1      2019-04-07 [1] CRAN (R 3.5.3)
## glue           1.3.1      2019-03-12 [1] CRAN (R 3.5.3)
## gower          0.2.0      2019-03-07 [1] CRAN (R 3.5.3)
## gtable         0.3.0      2019-03-25 [1] CRAN (R 3.5.3)
## highr          0.8        2019-03-20 [1] CRAN (R 3.5.3)
## htmltools      0.3.6      2017-04-28 [1] CRAN (R 3.5.1)
## ipred          0.9-9      2019-04-28 [1] CRAN (R 3.5.3)
## iterators      1.0.10     2018-07-13 [1] CRAN (R 3.5.1)
## knitr          1.22       2019-03-08 [1] CRAN (R 3.5.3)
## labeling       0.3        2014-08-23 [1] CRAN (R 3.5.0)
## lattice      * 0.20-38    2018-11-04 [2] CRAN (R 3.5.3)
## lava           1.6.5      2019-02-12 [1] CRAN (R 3.5.2)
## lazyeval       0.2.2      2019-03-15 [1] CRAN (R 3.5.3)
## lubridate      1.7.4      2018-04-11 [1] CRAN (R 3.5.1)
## magrittr       1.5        2014-11-22 [1] CRAN (R 3.5.3)
## MASS           7.3-51.1   2018-11-01 [2] CRAN (R 3.5.3)
## Matrix         1.2-15     2018-11-01 [2] CRAN (R 3.5.3)
## memoise        1.1.0      2017-04-21 [1] CRAN (R 3.5.3)
## ModelMetrics   1.2.2      2018-11-03 [1] CRAN (R 3.5.1)
## munsell        0.5.0      2018-06-12 [1] CRAN (R 3.5.1)
## nlme           3.1-137    2018-04-07 [2] CRAN (R 3.5.3)
## nnet           7.3-12     2016-02-02 [2] CRAN (R 3.5.3)
## pillar         1.3.1      2018-12-15 [1] CRAN (R 3.5.2)
## pkgbuild       1.0.3      2019-03-20 [1] CRAN (R 3.5.3)
## pkgconfig      2.0.2      2018-08-16 [1] CRAN (R 3.5.1)
## pkgload        1.0.2      2018-10-29 [1] CRAN (R 3.5.3)
## plyr           1.8.4      2016-06-08 [1] CRAN (R 3.5.0)
## prettyunits    1.0.2      2015-07-13 [1] CRAN (R 3.5.1)
## processx       3.3.1      2019-05-08 [1] CRAN (R 3.5.3)
## prodlim        2018.04.18 2018-04-18 [1] CRAN (R 3.5.1)
##  ps            1.3.0     2018-12-21 [1] CRAN (R 3.5.3)
##  purrr         0.3.2     2019-03-15 [1] CRAN (R 3.5.3)
##  R6            2.4.0     2019-02-14 [1] CRAN (R 3.5.2)
##  Rcpp          1.0.1     2019-03-17 [1] CRAN (R 3.5.3)
##  recipes       0.1.5     2019-03-21 [1] CRAN (R 3.5.3)
##  remotes       2.0.4     2019-04-10 [1] CRAN (R 3.5.3)
##  reshape2      1.4.3     2017-12-11 [1] CRAN (R 3.5.0)
##  rlang         0.3.4     2019-04-07 [1] CRAN (R 3.5.3)
##  rmarkdown     1.12      2019-03-14 [1] CRAN (R 3.5.3)
##  rpart         4.1-13    2018-02-23 [2] CRAN (R 3.5.3)
##  rprojroot     1.3-2     2018-01-03 [1] CRAN (R 3.5.1)
##  rstudioapi    0.10      2019-03-19 [1] CRAN (R 3.5.3)
##  scales        1.0.0     2018-08-09 [1] CRAN (R 3.5.1)
##  sessioninfo   1.1.1     2018-11-05 [1] CRAN (R 3.5.3)
##  stringi       1.4.3     2019-03-12 [1] CRAN (R 3.5.3)
##  stringr       1.4.0     2019-02-10 [1] CRAN (R 3.5.2)
##  survival      2.43-3    2018-11-26 [2] CRAN (R 3.5.3)
##  tibble        2.1.1     2019-03-16 [1] CRAN (R 3.5.3)
##  tidyselect    0.2.5     2018-10-11 [1] CRAN (R 3.5.1)
##  timeDate      3043.102  2018-02-21 [1] CRAN (R 3.5.1)
##  usethis       1.5.0     2019-04-07 [1] CRAN (R 3.5.3)
##  withr         2.1.2     2018-03-15 [1] CRAN (R 3.5.1)
##  xfun          0.6       2019-04-02 [1] CRAN (R 3.5.3)
##  yaml          2.2.0     2018-07-25 [1] CRAN (R 3.5.1)
##
## [1] C:/Users/YiuChung/Documents/R/win-library/3.5
## [2] C:/Program Files/R/R-3.5.3/library
```

References

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). *An introduction to statistical learning*. In *7th Ser.: Vol. 112* (1st ed., pp. 303–332). New York: Springer.

Jed Wing, M. K. C. from, Weston, S., Williams, A., Keefer, C., Engelhardt, A., Cooper, T., … Hunt., T. (2019). *Caret: Classification and regression training*. Retrieved from https://CRAN.R-project.org/package=caret

R Core Team. (2019). *R: A language and environment for statistical computing*. Retrieved from https://www.R-project.org/

Sarkar, D. (2008). *Lattice: Multivariate data visualization with r*. Retrieved from http://lmdvr.r-forge.r-project.org

Starkweather, J., & Moske, A. K. (2011). Multinomial logistic regression. *Consulted Page at September 10th: Http://Www.unt.edu/Rss/Class/Jon/Benchmarks/MLR_JDS_Aug2011.pdf*, *29*, 2825–2830.

Thakurta, A. G. (2015). What is the difference between eager learning and lazy learning? Retrieved May 12, 2019, from https://www.quora.com/What-is-the-difference-between-eager-learning-and-lazy-learning

Velloso, E., Bulling, A., Gellersen, H., Ugulino, W., & Fuks, H. (2013). Qualitative activity recognition of weight lifting exercises. *Proceedings of the 4th augmented human international conference*, 116–123. ACM.

Wickham, H. (2016). *Ggplot2: Elegant graphics for data analysis*. Retrieved from https://ggplot2.tidyverse.org

Wickham, H., François, R., Henry, L., & Müller, K. (2019). *Dplyr: A grammar of data manipulation*. Retrieved from https://CRAN.R-project.org/package=dplyr