|  |
| --- |
|  |
| STAT 5104 Introduction to Data Mining |
| Group Assignment |

|  |
| --- |
| CHAN Yiu Fung (1155010561)  CHUNG Wai Tung (1155118014)  LAM Siu Hung (1006201460)  LAU Chiu Kit (1155120306)  WONG Tsz Wing (1004666311)  WONG Yiu Chung (1155017920) |

Table of Contents

[Executive summary 1](#_Toc8410559)

[Introduction 1](#_Toc8410560)

[Descriptions of the dataset 1](#_Toc8410561)

[Data preparation 2](#_Toc8410562)

[Methods to use 2](#_Toc8410563)

[Findings 2](#_Toc8410564)

[Comparison of the results 2](#_Toc8410565)

[Conclusion 3](#_Toc8410566)

# Executive summary

# 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 “which” activity was performed at a specific point of time. Meanwhile, latest researchers 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 (Velloso, E.; Bulling, A.; Gellersen, H.; Ugulino, W.; Fuks, H., 2013) 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 the body. 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. The outcomes 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 with three numbers.

# Descriptions of the dataset

The data for this project comes from [this source](https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv) (https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv).

The dataset contains 160 variables, which include one target variable “Classe” and 159 readings from the sensors. 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 tantamount to each other.

The target variable “Classe” is defined as below:

1. Class A: exactly according to the specification (i.e. correctly perform the exercise);
2. Class B: throwing the elbows to the front;
3. Class C: lifting the dumbbell only halfway;
4. Class D: lowering the dumbbell only halfway; and
5. Class E: throwing the hips to the front.

# Data preparation

Since each predictor has well-defined meaning. They should not be scaled because this will cause distortion. \* Same scale \* Same range \* All continuous\* So no transformation needed.

We then performed data cleaning to the dataset. We first removed the first seven variables which are just descriptive data, then variables with near zero variance were also removed. In addition, variables with more than 10% missing data were excluded. After data cleaning, the dataset remained 53 variables. The variables were listed in the table below:

| **Column** | **Name** | **Description** | **Examples of the data** |
| --- | --- | --- | --- |
| 1 | roll\_belt | Numeric | 1.41, 1.41, 1.42 |
| 2 | pitch\_belt | Numeric | 8.07, 8.07, 8.07 |
| 3 | yaw\_belt | Numeric | -94.4, -94.4, -94.4 |
| 4 | total\_accel\_belt | Integer | 3, 3, 3 |
| 5 | gyros\_belt\_x | Numeric | 0, 0.02, 0 |
| 6 | gyros\_belt\_y | Numeric | 0, 0, 0 |
| 7 | gyros\_belt\_z | Numeric | -0.02, -0.02, -0.02 |
| 8 | accel\_belt\_x | Integer | -21, -22, -20 |
| 9 | accel\_belt\_y | Integer | 4, 4, 5 |
| 10 | accel\_belt\_z | Integer | 22, 22, 23 |
| 11 | magnet\_belt\_x | Integer | -3, -7, -2 |
| 12 | magnet\_belt\_y | Integer | 599, 608, 600 |
| 13 | magnet\_belt\_z | Integer | -313, -311, -305 |
| 14 | roll\_arm | Numeric | -128, -128, -128 |
| 15 | pitch\_arm | Numeric | 22.5, 22.5, 22.5 |
| 16 | yaw\_arm | Numeric | -161, -161, -161 |
| 17 | total\_accel\_arm | Integer | 34, 34, 34 |
| 18 | gyros\_arm\_x | Numeric | 0, 0.02, 0.02 |
| 19 | gyros\_arm\_y | Numeric | 0, -0.02, -0.02 |
| 20 | gyros\_arm\_z | Numeric | -0.02, -0.02, -0.02 |
| 21 | accel\_arm\_x | Integer | -288, -290, -289 |
| 22 | accel\_arm\_y | Integer | 109, 110, 110 |
| 23 | accel\_arm\_z | Integer | -123, -125, -126 |
| 24 | magnet\_arm\_x | Integer | -368, -369, -368 |
| 25 | magnet\_arm\_y | Integer | 337, 337, 344 |
| 26 | magnet\_arm\_z | Integer | 516, 513, 513 |
| 27 | roll\_dumbbell | Numeric | 13.1, 13.1, 12.9 |
| 28 | pitch\_dumbbell | Numeric | -70.5, -70.6, -70.3 |
| 29 | yaw\_dumbbell | Numeric | -84.9, -84.7, -85.1 |
| 30 | total\_accel\_dumbbell | Integer | 37, 37, 37 |
| 31 | gyros\_dumbbell\_x | Numeric | 0, 0, 0 |
| 32 | gyros\_dumbbell\_y | Numeric | -0.02, -0.02, -0.02 |
| 33 | gyros\_dumbbell\_z | Numeric | 0, 0, 0 |
| 34 | accel\_dumbbell\_x | Integer | -234, -233, -232 |
| 35 | accel\_dumbbell\_y | Integer | 47, 47, 46 |
| 36 | accel\_dumbbell\_z | Integer | -271, -269, -270 |
| 37 | magnet\_dumbbell\_x | Integer | -559, -555, -561 |
| 38 | magnet\_dumbbell\_y | Integer | 293, 296, 298 |
| 39 | magnet\_dumbbell\_z | Numeric | -65, -64, -63 |
| 40 | roll\_forearm | Numeric | 28.4, 28.3, 28.3 |
| 41 | pitch\_forearm | Numeric | -63.9, -63.9, -63.9 |
| 42 | yaw\_forearm | Numeric | -153, -153, -152 |
| 43 | total\_accel\_forearm | Integer | 36, 36, 36 |
| 44 | gyros\_forearm\_x | Numeric | 0.03, 0.02, 0.03 |
| 45 | gyros\_forearm\_y | Numeric | 0, 0, -0.02 |
| 46 | gyros\_forearm\_z | Numeric | -0.02, -0.02, 0 |
| 47 | accel\_forearm\_x | Integer | 192, 192, 196 |
| 48 | accel\_forearm\_y | Integer | 203, 203, 204 |
| 49 | accel\_forearm\_z | Integer | -215, -216, -213 |
| 50 | magnet\_forearm\_x | Integer | -17, -18, -18 |
| 51 | magnet\_forearm\_y | Numeric | 654, 661, 658 |
| 52 | magnet\_forearm\_z | Numeric | 476, 473, 469 |
| 53 | classe | Text | Factor with 5 levels: A, B, C, D, E |

In our next step, we sliced 80% of the dataset as training and the remaining 20% as testing. We then further reduced the dimension of the training dataset using Principal Component Analysis (PCA). 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 reduced the dimension of the datasets from 52 to 38 while retaining 99% of the information. This reduced model complexity and improved scalability.

As a side note, PCA is usually performed on scaled/standardised 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 similar in nature, scaling or standardisation provides no added benefits. Hence such procedures are not used in our analysis.

# Methods to use

## *Learning models*

Seven learning methods were adopted in our report. They are i) Decision Tree; ii) K-Nearest Neighbor; iii) Multinomial Logistic Regression; iv) Naïve Bayes; v) Neuro Network; vi) Random Forest; and vii) Tree Bagging. The methods can be classified as eager learner (Decision Tree, Tree Bagging, Random Forest, and Neuro Network) and lazy learner (K-Nearest Neighbor and Naïve Bayes). Caret package was used in our R program.

## *Cross validation* *(CV) - Choosing between LOOCV and K-Fold*

In order to get a better assessment of the above models, cross validation has been performed. 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. This has to do with a bias-variance trade-off.

Estimates produced by LOOCV is plagued by high variance compared to that produced by K-fold CV. This is because each of the validation errors in LOOCV are produced by models trained on virtually identical dataset. The final statistic is an average of the n validation errors which are highly positively correlated. On the other hand, K-fold CV outputs K (which is usually much less than n) validation errors which are less correlated as there is less overlap among models. The average of strongly correlated quantities has higher variance than the average of weakly correlated quantities; hence the estimated test error from LOOCV tends to have higher variance that that from K-fold.

The dataset in the report consists of relatively large number of variables (38 columns). Hence a 10-fold cross-validation was performed throughout the analysis.

## *Performance Measures for Multi-Class Problems*

While running the above learning model, we incorporated the following performance measuring syntax in the R program:

* Accuracy and Kappa
* Area Under ROC Curve
* Logarithmic Loss

## *Tree based models*

Tree-based methods, including Decision Tree, Random Forest, and Tree Bagging, 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: tress 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 predictors out of all , 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 is often the square root of but other method of choosing also exists.

## *K-Nearest Neighbor*

## *Multinomial Logistic Regression*

## *Naïve Bayes*

## *Neuro Network*

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 scaled 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.

neuro code

The parameter `sizespecifies 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

# Findings

# Comparison of the results

LogLoss, Accuracy, F1 and AUC are used to assess the performances of the models, and the definitions of the indicators

**AUC**

Receiver Operating Characteristic (ROC) curve is the plot of true positive rate (tpr) vs false positive rate (fpr). We can assess the performance of the model by the area under the ROC curve. As a rule of thumb, 0.9-1 = excellent; 0.8-0.9=good; 0.7-0.8=fair; 0.6-0.7=poor; 0.5-0.6=fail.

**Accuracy**

When measuring accuracy of a classification model, we can use F1 score. F1 score is harmonic mean of precision and recall. By using it, we can avoid some misleading situations when using error rate for comparing the performance.

F1=2 x Precision x Recall / (Precision+Recall)

where

Precision = True Positive / (False Postive + True Postive)

Recall = True Positive / (True Positive + False Negative)

**Logarithmic loss**

Logarithmic loss measures the performance of a classification model where the prediction input is a probability value between 0 and 1. The value of Log loss increases when the predicted probability diverges from the actual label.

When the predicted probability approaches 1, log loss slowly decreases. But if the predicted probability decreases, the log loss value would increase rapidly.

<http://wiki.fast.ai/index.php/Log_Loss>

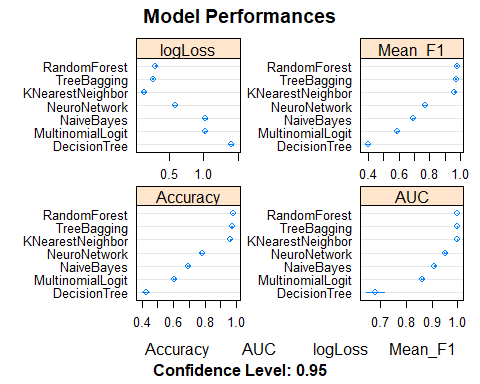
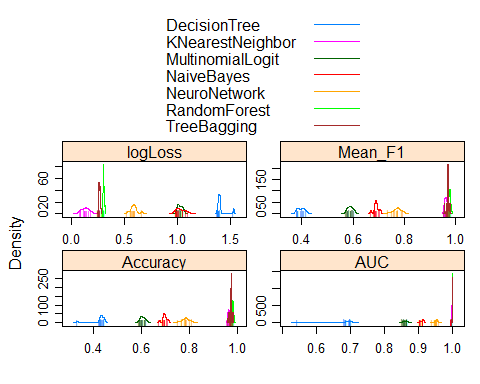
Among the seven models, the model performances of Random Forest, Tree Bagging and KNN outperform the other four models.

However, the model performance of KNN is the best among the seven models, with the lowest value (??) of log Loss, and the highest value of Mean F1 (??), Accuracy (??) and AUC (??) at the 95% confidence level.

In addition, in the aspect of time consumption in running the model, XXX seconds / minutes are used for running KNN, which is also the 2nd quickest model, follow by Decision Tree.

The variables in this dataset were not scaled / standardized, while the performance of KNN still outperforms the other models. This finding is surprise, as KNN heavily relies on the Euclidean distance.

Thus, KNN is used as the final model.



## Confusion Matrices of final model

confusionMatrix(predict(knn), trainPC$classe);

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction A B C D E  
## A 4442 25 0 2 0  
## B 8 2980 6 0 7  
## C 8 29 2710 51 4  
## D 5 4 19 2515 7  
## E 1 0 3 5 2868  
##   
## Overall Statistics  
##   
## Accuracy : 0.9883   
## 95% CI : (0.9865, 0.9899)  
## No Information Rate : 0.2843   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.9852   
##   
## Mcnemar's Test P-Value : 3.233e-09   
##   
## Statistics by Class:  
##   
## Class: A Class: B Class: C Class: D Class: E  
## Sensitivity 0.9951 0.9809 0.9898 0.9775 0.9938  
## Specificity 0.9976 0.9983 0.9929 0.9973 0.9993  
## Pos Pred Value 0.9940 0.9930 0.9672 0.9863 0.9969  
## Neg Pred Value 0.9980 0.9954 0.9978 0.9956 0.9986  
## Prevalence 0.2843 0.1935 0.1744 0.1639 0.1838  
## Detection Rate 0.2829 0.1898 0.1726 0.1602 0.1827  
## Detection Prevalence 0.2847 0.1912 0.1785 0.1624 0.1833  
## Balanced Accuracy 0.9963 0.9896 0.9913 0.9874 0.9965

confusionMatrix(predict(knn, testPC), testPC$classe); #don't use testPC until end of report

From the above confusion matrix, the accuracy rate of the training data is 0.9883 with the 95% confidence level in the range of 0.9865 and 0.9899. The p-value is approaching 0.

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction A B C D E  
## A 1105 15 0 3 0  
## B 3 727 7 1 4  
## C 4 16 667 23 3  
## D 3 1 7 613 1  
## E 1 0 3 3 713  
##   
## Overall Statistics  
##   
## Accuracy : 0.975   
## 95% CI : (0.9696, 0.9797)  
## No Information Rate : 0.2845   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.9684   
##   
## Mcnemar's Test P-Value : 0.0008391   
##   
## Statistics by Class:  
##   
## Class: A Class: B Class: C Class: D Class: E  
## Sensitivity 0.9901 0.9578 0.9751 0.9533 0.9889  
## Specificity 0.9936 0.9953 0.9858 0.9963 0.9978  
## Pos Pred Value 0.9840 0.9798 0.9355 0.9808 0.9903  
## Neg Pred Value 0.9961 0.9899 0.9947 0.9909 0.9975  
## Prevalence 0.2845 0.1935 0.1744 0.1639 0.1838  
## Detection Rate 0.2817 0.1853 0.1700 0.1563 0.1817  
## Detection Prevalence 0.2863 0.1891 0.1817 0.1593 0.1835  
## Balanced Accuracy 0.9919 0.9765 0.9805 0.9748 0.9934

#talk about why choose specific model first.

The final model produced by KNN applied to the testing data that the accuracy rate is 0.975, and the 95% confidence level lies within 0.9696 and 0.9797. The p-value also approach 0.

# Conclusion

The KNN model is chosen due to its best performance in Log Loss while its performance in Mean F1, Accuracy and AUC are similar among the top three models. The accuracies of KNN as shown in the confusion matrices using training and testing dataset are of high performance with accuracy rates of 0.9883 and 0.975 respectively.

# Reference

[Velloso, E.](http://groupware.les.inf.puc-rio.br/collaborator.jsf?p1=evelloso); Bulling, A.; Gellersen, H.; [Ugulino, W.](http://groupware.les.inf.puc-rio.br/collaborator.jsf?p1=ugulino); [Fuks, H.](http://groupware.les.inf.puc-rio.br/collaborator.jsf?p1=hugo) [**Qualitative Activity Recognition of Weight Lifting Exercises**](http://groupware.les.inf.puc-rio.br/work.jsf?p1=11201). Proceedings of 4th International Conference in Cooperation with SIGCHI (Augmented Human '13) . Stuttgart, Germany: ACM SIGCHI, 2013.  
  
Read more: <http://groupware.les.inf.puc-rio.br/har#ixzz5nWTcVmRG>

[Ugulino, W.](http://groupware.les.inf.puc-rio.br/collaborator.jsf?p1=ugulino); [Cardador, D.](http://groupware.les.inf.puc-rio.br/collaborator.jsf?p1=debora); [Vega, K.](http://groupware.les.inf.puc-rio.br/collaborator.jsf?p1=katia); [Velloso, E.](http://groupware.les.inf.puc-rio.br/collaborator.jsf?p1=evelloso); Milidiu, R.; [Fuks, H.](http://groupware.les.inf.puc-rio.br/collaborator.jsf?p1=hugo) [**Wearable Computing: Accelerometers' Data Classification of Body Postures and Movements**](http://groupware.les.inf.puc-rio.br/work.jsf?p1=10335). Proceedings of 21st Brazilian Symposium on Artificial Intelligence. Advances in Artificial Intelligence - SBIA 2012. In: Lecture Notes in Computer Science. , pp. 52-61. Curitiba, PR: Springer Berlin / Heidelberg, 2012. ISBN 978-3-642-34458-9. DOI: 10.1007/978-3-642-34459-6\_6.   
Cited by 2 (Google Scholar)  
  
Read more: <http://groupware.les.inf.puc-rio.br/har#ixzz5nWUsP3lL>