Predicting measurement type of UCI Ionosphere radar data

cgoldner

2021-07-16

Contents

1 Overview **2**
1.1 UCI Ionosphere radar dataset . . . 2
1.2 Goal of the project . . . 2
1.2.1 Measure of performance . . . 2
1.3 Key steps . . . 2

2 Methods **2**
2.1 Data wrangling . . . 2
2.2 Data exploration . . . 3
2.2.1 Summary of data exploration . . . 6
2.3 Building the final model . . . 6
2.3.1 Comparing different models . . . 7
2.3.2 Evaluating tuning parameters . . . 7
2.3.3 Ensemble model . . . 8
2.3.4 Summary of model building . . . 8

3 Results **8**
3.1 Accuracy . . . 9
3.2 Evaluation of errors . . . 9

4 Conclusion **11**
4.1 Summary . . . 11
4.2 Limitations . . . 11
4.3 Future improvements . . . 11

1 Overview

1.1 UCI Ionosphere radar dataset

The Ionosphere radar data was collected by a system in Goose Bay, Labrador. The system collected data via high-frequency antennas. The "good" radar returns are those showing evidence of some type of structure in the ionosphere. "Bad" returns are those that do not; their signals pass through the ionosphere. Received signals were processed such that the outputs of each measurement are 17 *pulses*. Each pulse is described by 2 *attributes*, corresponding to the real and imaginary part of a complex number. The data is freely available from the UCI Machine Learning Repository

https://archive.ics.uci.edu/ml/datasets/Ionosphere.

1.2 Goal of the project

The goal of this project is to build a data-driven classification model which predicts whether a measurement is "good" or "bad" given the measured data, such that the model's performance is as good as possible.

1.2.1 Measure of performance

The model's performance is evaluated by measuring the *accuracy* of the model on a *validation set*. Accuracy is defined as

$$\text{accuracy} = \frac{\text{number of right predictions on the validation set}}{\text{number of elements in the validation set}}.$$

1.3 Key steps

The most important steps performed in order to build a model for classifying "good" and "bad" measurements are:

(1) Download the data, convert it into a data frame and split off a validation set that is only used for evaluation purposes.

(2) Analyze the data to confirm that the classification problem is non-trivial (i.e. it does not depend on one or two variables only) and to detect variables (not) useful for classification.

(3) Choose models to train, estimate the optimal parameters of these models, train them and compare their performance.

(4) Choose the best performing model or ensemble several models into a new one to obtain a final model.

(5) Evaluate the final model.

2 Methods

2.1 Data wrangling

The raw data was downloaded from

https://archive.ics.uci.edu/ml/machine-learning-databases/ionosphere/ionosphere.data.

It can also be downloaded from the author's Github account following the link

https://github.com/cgoldner/radar-ionosphere/tree/main/data.

Since the raw data is already in an easy-to-handle format, a minimum of data wrangling is required. More precisely, the column names are set to $x_1, \dots, x_{34}, y$, where $x_1, \dots, x_{34}$ are the measurement results such that x_i, x_{i+1} correspond to the real and imaginary part of a complex number associated to one pulse if i is odd. We may also refer to these variables as *predictor variables*. They are stored as numeric values. The variable

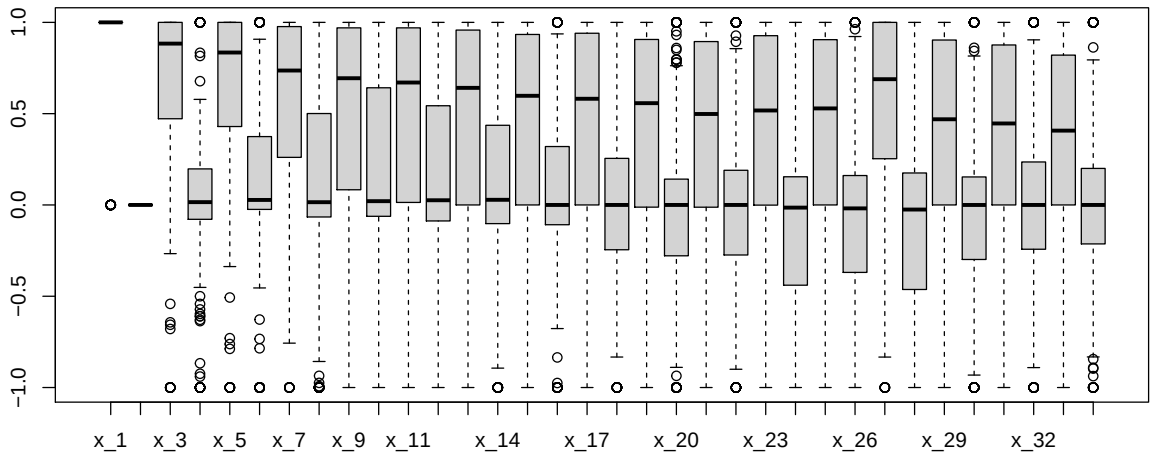
y, which should be predicted, represents if a measurement is "good" `g` or "bad" `b`. The variable y is stored as factor with levels `g` and `b`. It is important to note that there are no missing values.

Before starting with data exploration and model building, a `validation` set is split off the dataset. It is a hold out set to evaluate the final model's performance. The `validation` set consists of 20% of the overall data. The overall data consists of 351 observations and the `validation` set consists of 71 observations.

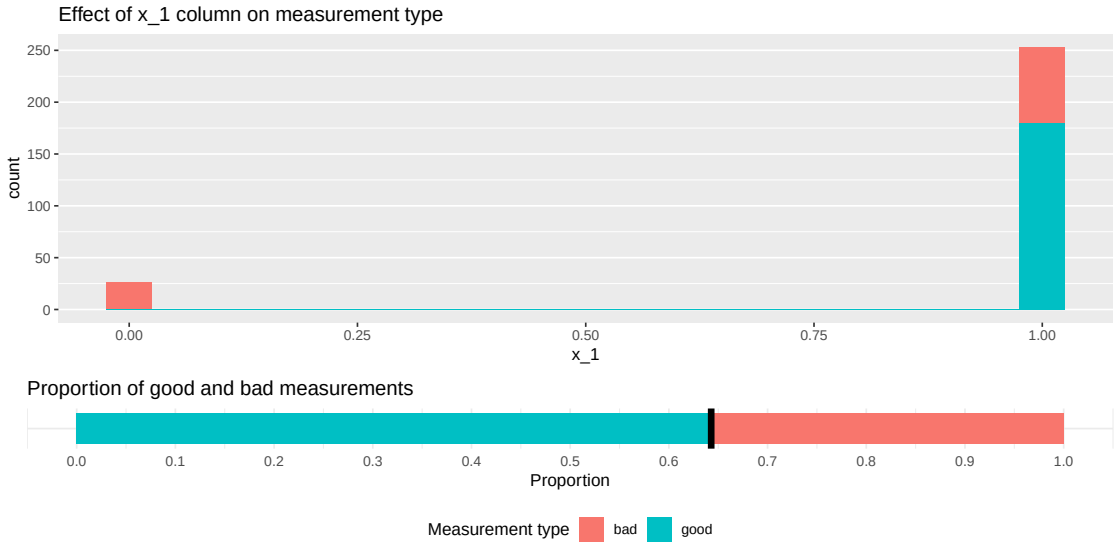
```
set.seed(1, sample.kind="Rounding") # set seed to make splitting reproducible
val_index <- createDataPartition(ionosphere$y, times = 1, p = 0.2, list = FALSE)
validation <- ionosphere %>% slice(val_index)
training <- ionosphere %>% slice(-val_index)
```

2.2 Data exploration

The `training` set is used to explore the data. The following plot clearly shows that real and imaginary parts are differently distributed for each pulse. Interestingly, the first pulse with real part `x_1` and imaginary part `x_2` shows very little variability. More precisely, `x_2` is always equal to zero and thus has no predictive power. Therefore the variable `x_2` can be excluded when building a model.

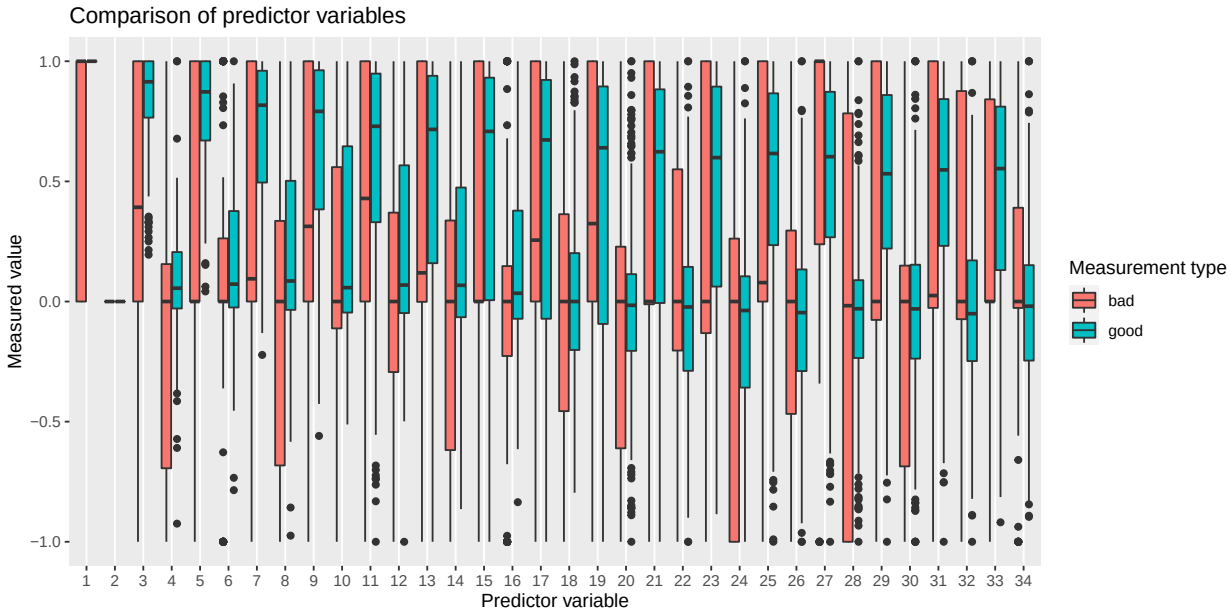


To explore how `x_1` can be used to make a prediction, its effect on the measurement type (`g` or `b`) is explored in the following plot.

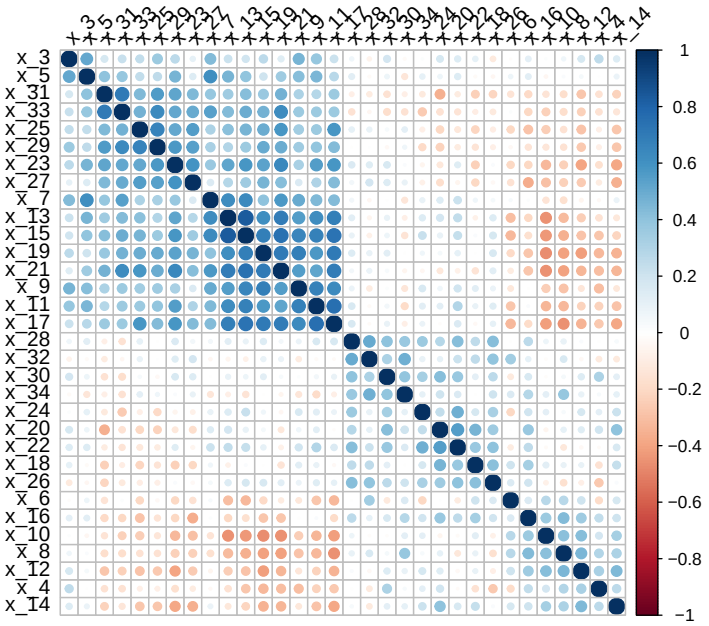


Notice that `x_1` takes only the values 0 and 1, and there are very few 0 values for `x_1`. Observe that if `x_1` is 0, then the measurement is a "bad" one. Thus `x_1` can be used to predict "bad" measurements. Moreover, notice that the plot above shows that the proportion of "good" measurements is around 0.65. Hence the baseline for the accuracy of our model is 0.65.

With the following plot, we try to isolate more predictor variables that can be used for classification — we also need to make sure that our classification problem is non-trivial, i.e. that there are not one or two variables which determine the whole classification.



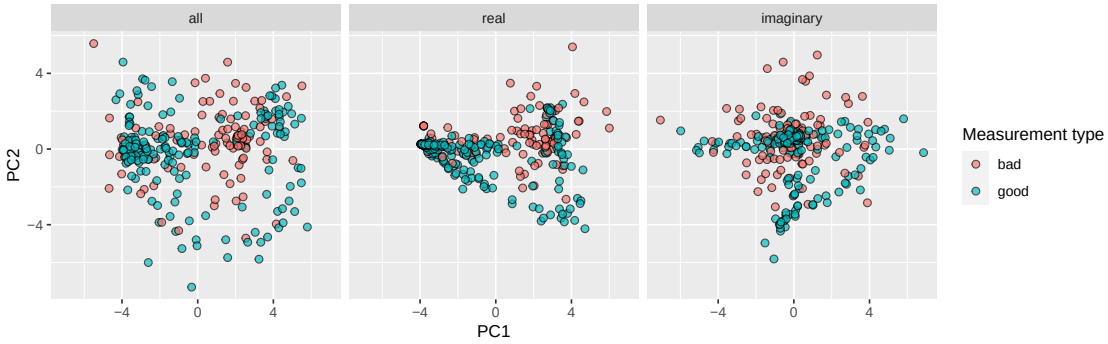
Most interquartile ranges overlap and almost all predictor variables show a large variability. Hence the classification problem is non-trivial. However, let us take a closer look to see if the remaining predictor variables (`x_1` and `x_2` are excluded) are correlated in a usable way.



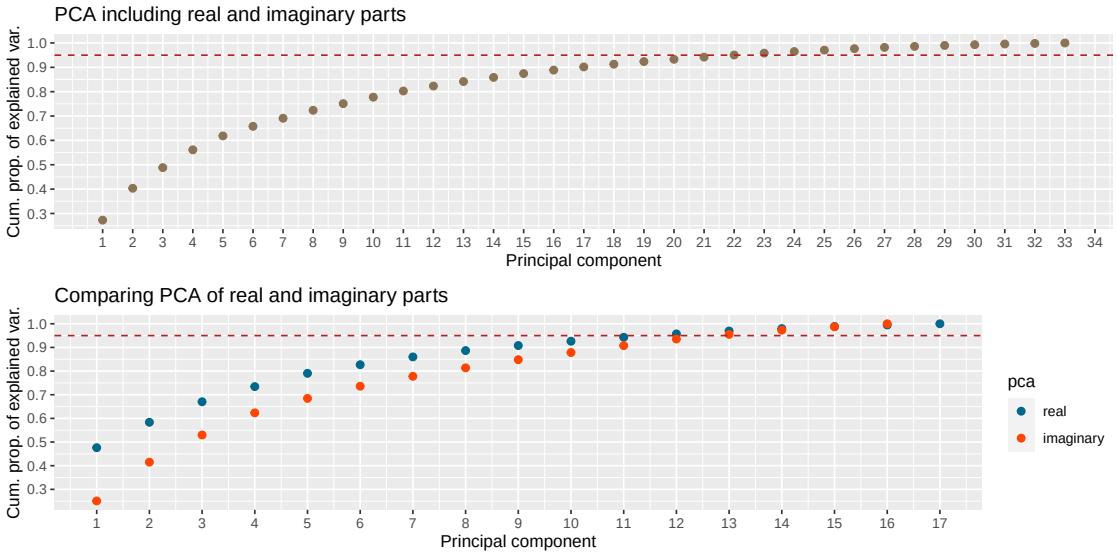
Notice that the correlation matrix above is decomposed into 4 squares, where the upper left and the lower right square show mostly positive correlations. The upper left square belongs to real parts of pulses and the lower right square belongs to imaginary parts of pulses. Thus, if we want to reduce the number of predictor variables, distinguishing between real and imaginary parts could be useful.

```
# distinguish between real and imaginary parts
training_real <- training[seq(1, 33, 2)]
training_imaginary <- training[seq(2, 34, 2)]
```

To see, if the number of predictor variables could be reduced, three *principal component analyses* (PCAs) are performed: One including all predictor variables, one including only real parts of pulses (`x_i` with i odd) and one including only imaginary parts of pulses (`x_i` with i even).



Observe from the plot above, that the first two principal components do a bad job at distinguishing between "good" and "bad" measurements. The plots below show that more than half of the predictor variables are required in order to explain at least 95% of all variance. Thus a PCA cannot be used to remove most predictor variables. Hence we decide not to remove any more predictor variables due to their relatively low number.



2.2.1 Summary of data exploration

The predictor variable `x_1` can be used to predict "bad" measurements. The predictor variable `x_2` has no predictive power and can be excluded when building a model. All of the remaining predictor variables are used when building a model.

2.3 Building the final model

To build a final model, different models are compared and the best ones are chosen.

```
# the "caret" package is used to train models
library(caret)
```

In order to compare different models, they need to be trained and evaluated with optimal tuning parameters. We do this as follows:

(1) Randomly split `training` into `test` and `train`, where the `train` set consists of 20% of the `training` data.

(2) Optimize model parameters on the `train` set via `caret's` build-in bootstrapping procedure.

(3) Compute accuracy of the optimized model on the `test` set.

(4) Repeat steps 1–3 a number of times, say k, and take the average accuracy and average the optimization parameters.

(5) Apply steps 1–4 to all different models.

Having found the best models and their parameters, a final model can be build depending on how each of the different models performed:

(6) Ensemble the best models via majority vote into one model.

(7) Determine if the ensemble performs much better than the best performing model (using steps 1–4). If yes, then the final model is the ensemble. If not, then the final model is defined as the best performing model from above.

The different models that are used are:

Regression To use Regression techniques, the outcomes "good" and "bad" are encoded as 1 and 0, respectively, and a cutoff probability of 0.5 is used.

- Linear model `lm`.
- Logistic regression model `glm`.
- k-nearest neighbors `knn`[1].

Classification

- Generative models `lda` and `qda`.
- Random forest `rf`.
- Support vector machine `svm`.

Notice that no unsupervised models are used since all observations are already labeled.

2.3.1 Comparing different models

We run the different models each k times, record the accuracies and take their averages to obtain the table below.

Table 1: Average accuracies of models over $k = 10$ runs.

model	run_1	run_2	run_3	run_4	run_5	run_6	run_7	run_8	run_9	run_10	avg
rf	0.964	0.911	0.964	0.946	0.911	0.964	0.964	0.929	0.946	0.964	0.946
svm	0.911	0.893	0.929	0.946	0.893	0.893	0.929	0.893	0.911	0.911	0.911
lm	0.857	0.946	0.893	0.893	0.875	0.893	0.946	0.929	0.911	0.875	0.902
glm	0.857	0.946	0.893	0.893	0.875	0.893	0.946	0.929	0.911	0.875	0.902
lda	0.857	0.946	0.893	0.893	0.875	0.893	0.946	0.929	0.911	0.875	0.902
knn	0.804	0.893	0.857	0.893	0.911	0.875	0.911	0.893	0.929	0.839	0.880
qda	0.839	0.857	0.929	0.893	0.839	0.911	0.857	0.893	0.821	0.911	0.875

Notice that `lm`, `glm` and `lda` perform the same, which is not surprising since they classify the outcomes by using a linear boundary. Hence the predictions of `lm`, `glm` and `lda` should be the same. This can be confirmed using the following code.

```
# Check if lm, glm, lda yield the same results
for (s in 1:k) {
  print(identical(model_list_train[[s]]$predictions$pred_lm,
                  model_list_train[[s]]$predictions$pred_lda))
}

for (s in 1:k) {
  print(identical(model_list_train[[s]]$predictions$pred_glm,
                  model_list_train[[s]]$predictions$pred_lda))
}
```

Therefore we do not use `lm` and `glm` to build the final model. Thus the three best performing models are `rf`, `svm` and `lda`.

2.3.2 Evaluating tuning parameters

To find the best tuning parameter `mtry` for `rf`, the mean of all optimal `mtry` parameters of the k different runs is taken, which is 6.

[1] Although kNN can be used for classification directly, it is used here with 1s and 0s

To tune `svm`, different ranges to find the optimal parameters in the single runs were tested. It can be observed that the performance decreased when allowing `svm` to fit with degrees > 2, i.e. these degrees lead to overfitting of `svm`. However, notice that, if tuning parameters are chosen correctly, then, in general, `svm` should not perform worse than linear models since it generalizes linear approaches. After comparing perfomances of degree 1 and degree 2, degree 2 is taken. We observe that optimizing the remaining tuning parameters yields the same results in all runs. Thus these are the parameters we choose when building a final model.

2.3.3 Ensemble model

The three best performing models `rf`, `svm` and `lda` are now ensembled into a single model using majority vote. Then, as before, the ensembled model is run k times and the resulting accuracies are averaged.

Table 2: Accuracies of different models.

	rf	svm	lda	ensemble
avg	0.946	0.911	0.902	0.9428571

```
# Compare types of errors of rf and ensemble ------------------------------
# ensemble misclassifies "bad" as "good" more often than rf does
# i.e. lower specificity

model_ensemble <- train_ensemble(s = 1, final_training = FALSE)

# confusion matrix rf
confusionMatrix(model_ensemble$pred_rf, model_ensemble$y)$table
```

```
##           Reference
## Prediction  g  b
##          g 35  1
##          b  1 19
```

```
# confusion matrix ensemble
confusionMatrix(model_ensemble$pred_ensemble, model_ensemble$y)$table
```

```
##           Reference
## Prediction  g  b
##          g 35  4
##          b  1 16
```

The results above show that the ensemble does not perform much better than `rf`. Thus the simpler model, namely `rf`, is chosen as final model.

2.3.4 Summary of model building

Random forest `rf` turned out to be the best performing model. Its performance is similar to that of an ensemble via majority vote of `rf`, `svm` and `lda`. Since `rf` is the simpler model, it is chosen as the final model with optimized model parameter `mtry`.

3 Results

3.1 Accuracy

After defining `rf` to be our final model (with optimized tuning parameter `mtry` given by 6), its accuracy is measured on the final hold set `validation` as 0.93, and, additionally, its F_1 value is 0.946. The following confusion matrix shows that the final model still suffers from misclassifying "bad" as "good", i.e. it has lower specificity than sensitivity if the positive class is "good"

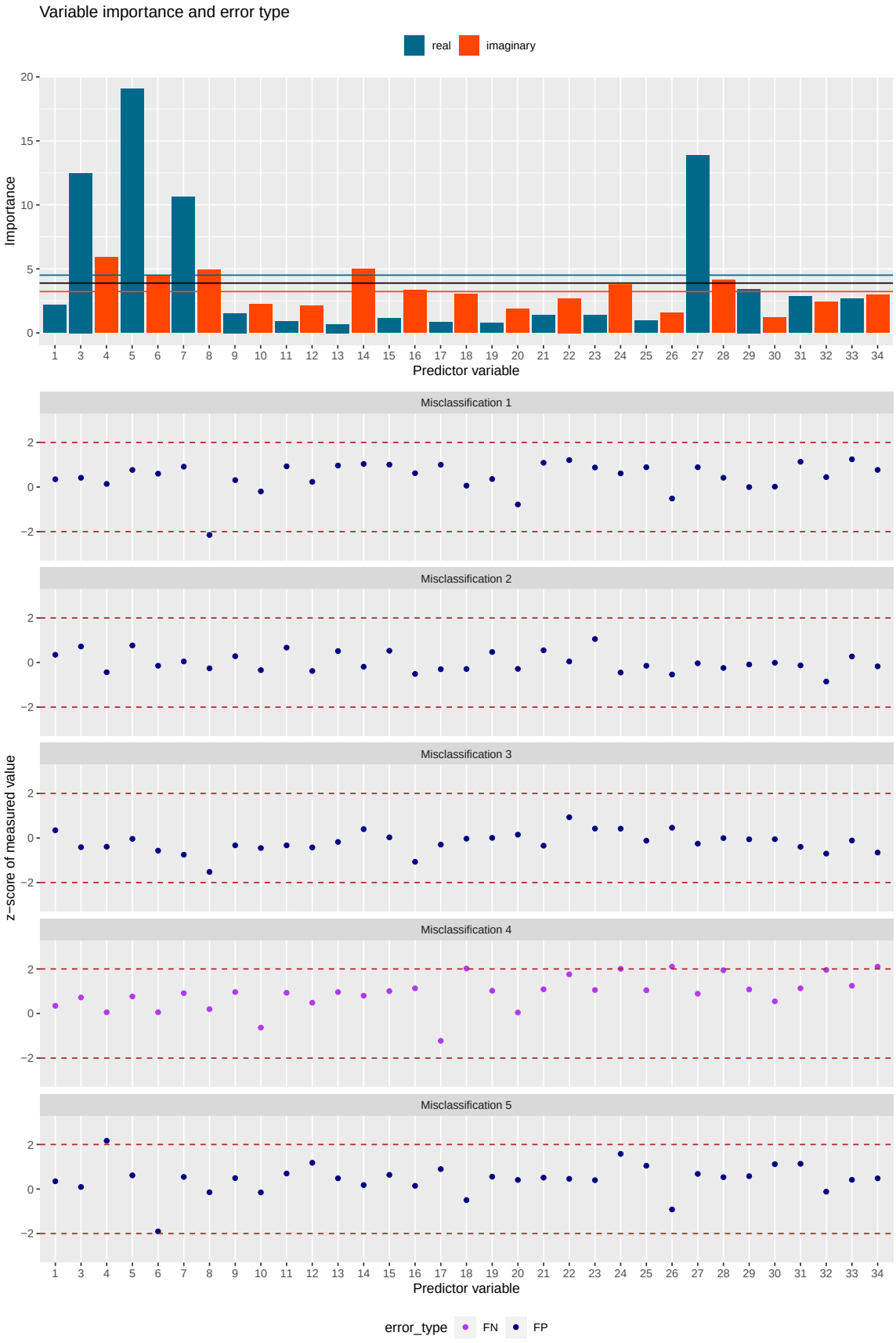
```
# confusion matrix for final model on validation set
confusionMatrix(final_model$predictions$pred_rf, final_model$predictions$y)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  g  b
##          g 44  4
##          b  1 22
##
##                Accuracy : 0.9296
##                  95% CI : (0.8433, 0.9767)
##     No Information Rate : 0.6338
##     P-Value [Acc > NIR] : 8.324e-09
##
##                   Kappa : 0.8445
##
##  Mcnemar's Test P-Value : 0.3711
##
##             Sensitivity : 0.9778
##             Specificity : 0.8462
##          Pos Pred Value : 0.9167
##          Neg Pred Value : 0.9565
##              Prevalence : 0.6338
##          Detection Rate : 0.6197
##    Detection Prevalence : 0.6761
##       Balanced Accuracy : 0.9120
##
##        'Positive' Class : g
##
```

3.2 Evaluation of errors

To better understand the errors made by our final model, the variable importance of each variable is evaluated and compared with the value of that variable in each missclassified measurement. More precisely, the upper part of the plot below shows the variable importance of each variable in the final model, where the overall average importance, and the averages of real and imaginary parts of pulses are indicated. The lower part of the plot below shows the "unusualness" of each measurement of each variable using z–scores. Notice that, the claim that "misclassified observations are those whose real and imaginary parts of the pulses are very unusual, or at least unusual in the most important variables" is only partly true. The false negative classification "Misclassification 4" seems to be "unusual" in a lot of variables associated to imaginary parts: the z–scores of `x_18`, `x_22`, `x_24`, `x_26`, `x_28`, `x_32` and `x_34` are around 2. Since there are not enough false negatives, it cannot be deduced that an accumulation of unusual values in the imaginary parts leads to misclassification. Moreover, in case of false positives, there are less "unusual" values. Observe that no single predictor variable of "Misclassification 2" has an unusual value (i.e. z–score > 2).

In total, we are not able to trace the misclassifications back to unusual values of important variables.


Variable importance and error type
real
imaginary
Importance
Predictor variable
Misclassification 1
Misclassification 2
Misclassification 3
Misclassification 4
Misclassification 5
z−score of measured value
Predictor variable
error_type
FN
FP

4 Conclusion

4.1 Summary

Random forest turned out to be the best model to classify the given measurements into "good" and "bad" ones. The accuracy achieved was 0.93 which is about 0.28 higher than the baseline.

4.2 Limitations

The total number of observations used to build and to test the final model was very small. Thus validating the model produces very few errors (less than 10), which in turn makes it hard to identify common properties of these errors.

4.3 Future improvements

The ensemble of `rf`, `svm` and `lda` we build showed a similar performance to the final model `rf` in the testing phase. Thus by increasing the performance of the weakest model among `rf`, `svm`, `lda`, which is `lda`, it could be possible to make the ensemble perform better than `rf`. For that, the `lda` model could be adapted to take prevalence into account. This would increase the performance of `lda` since "good" and "bad" measurements do not appear with the same frequency .