Smartphone Mood Data Analysis

Tommy Maaiveld

Introduction

This document shows the results of my preliminary analysis in R and presents some preliminary models for the task at hand. The document is in English to allow for potential reuse of text for the report. The work presented here was prepared in Rstudio.

- *I have attached questions to each section in italics.*

Data Processing

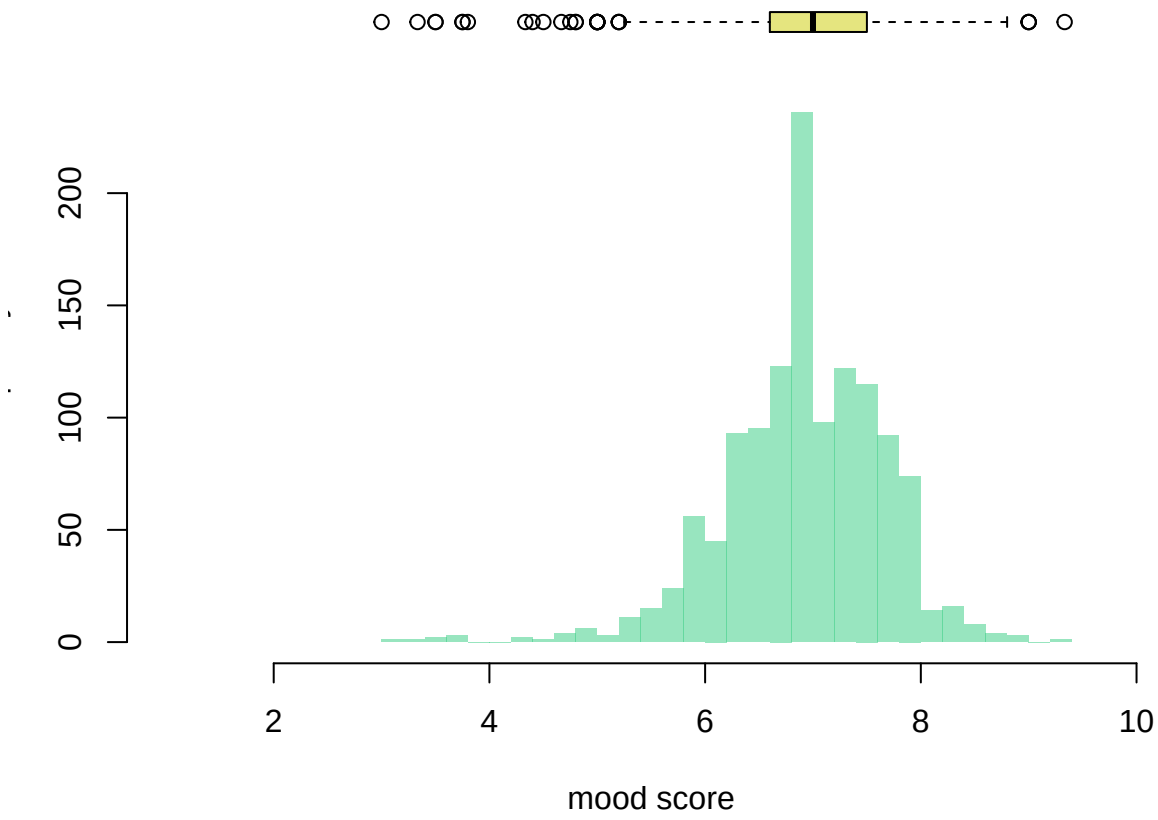
The (hidden) code block below displays the method used to transform the dataset from one instance per row to aggregated rows of daily averages. The result is a more even distribution of values across dates and an aggregation of several explanatory features that can be used to generate predictions over time for target variable values.

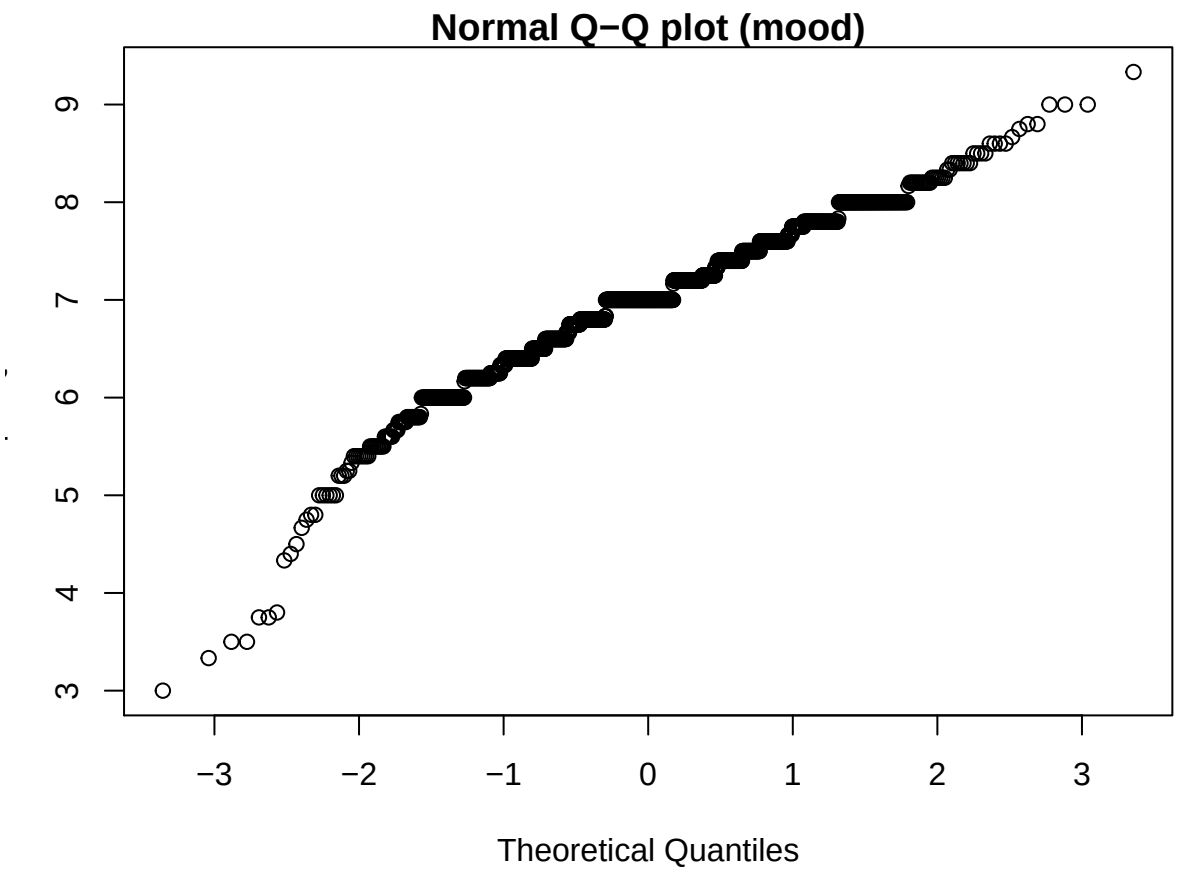
```
data <- read.csv(file="data/dataset_mood_smartphone.csv") %>%
  tbl_df() %>%
  separate(time, c("date", "time"), sep=" ") %>%
  spread(variable,value) %>% # organise
  mutate(appCat.builtin=replace(appCat.builtin, appCat.builtin<0, NA)) %>% #clean
  group_by(id,date)
data$date <- as.Date(data$date, origin = min(data$date))

ordvars = names(data)[1:4]
meanvars = c(names(data)[5],names(data)[19:21])
sumvars = names(data)[! names(data) %in% c(meanvars, ordvars)]

data_agg = merge(
  arrange(aggregate(data[meanvars], by=list(data$id, data$date),
                    FUN=mean, na.rm=TRUE), Group.1,Group.2),
  arrange(aggregate(data[sumvars], by=list(data$id, data$date),
                    FUN=sum,  na.rm=TRUE), Group.1,Group.2)
  )[, c(1,2,6,4,5,3,7:21)] %>%
  dplyr::rename(id=Group.1, date=Group.2) %>%
  tbl_df()
```

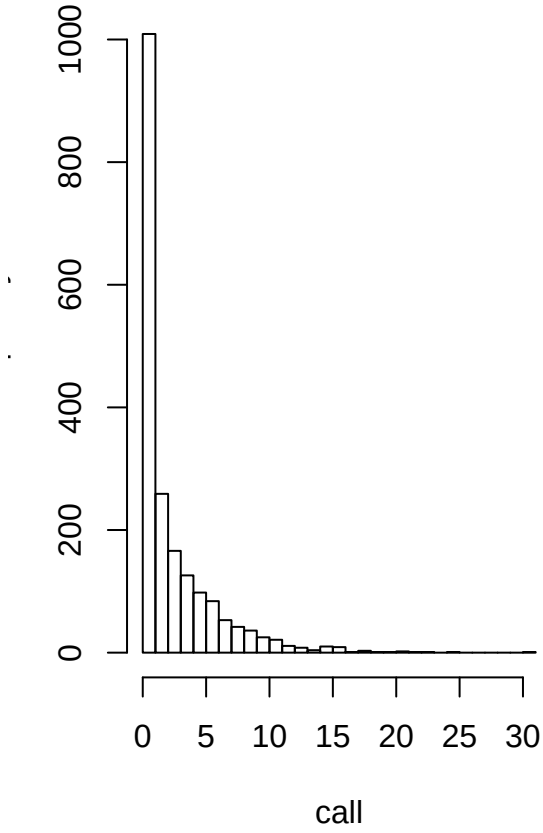
Missing values are filled in as `NaN` - I wanted to look into using them somehow, since simply removing them could remove potential information from the model. I explored normalising the data at this stage, but this did not seem to improve predictions. Transforming the dataset to fewer principal components (PCA) significantly worsened prediction. These measures were re-attempted after feature transformation (Section ?), but also failed to produce improved results.

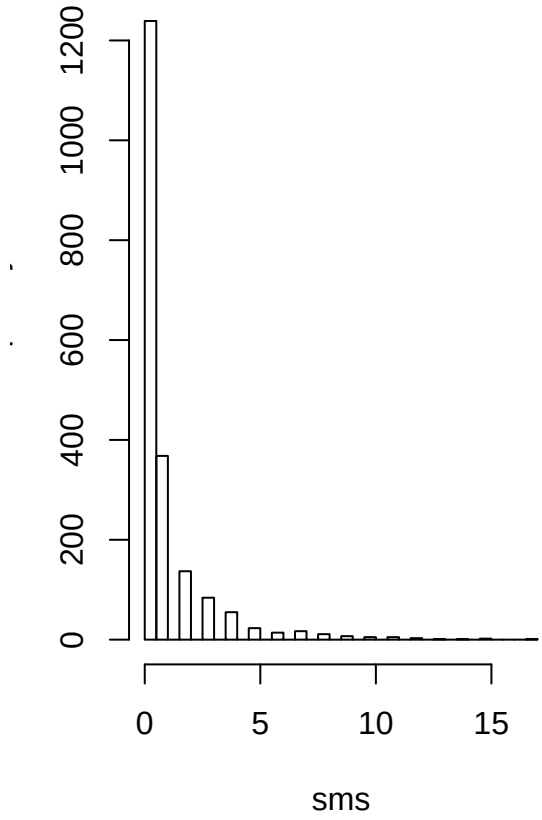
- *What is the reasoning behind applying data normalisation and standardisation means? Why does transforming the features at this level not affect any of the model predictions?*
- *PCA is not commonly used for neural network models, as they do not benefit from capturing more essential features of the data? Why does it produce worse results here?*
- *still have to try transforming the data at an earlier point*

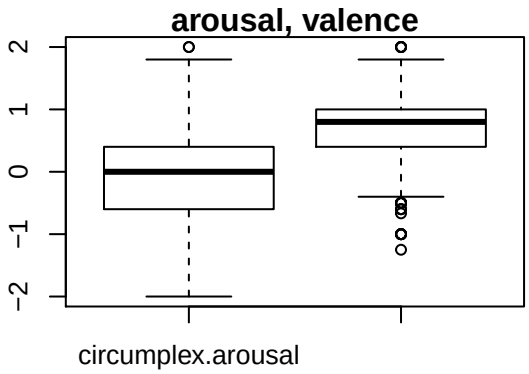
Data Exploration & Visualisation

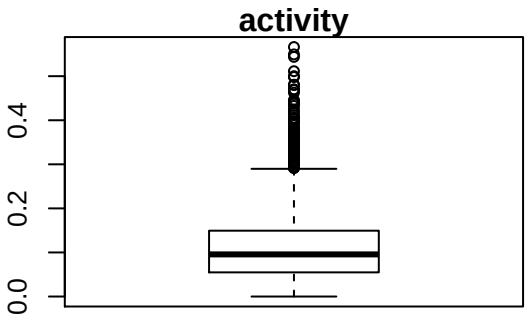


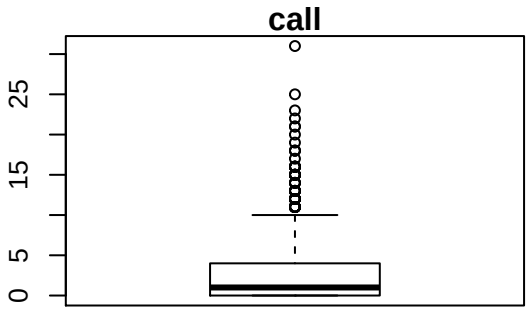

Normal Q−Q plot (mood)
Theoretical Quantiles
3
4
5
6
7
8
9
−3
−2
−1
0
1
2
3

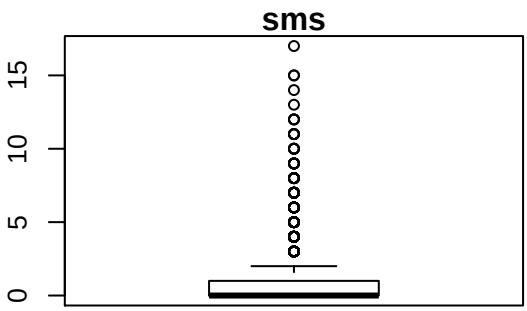

0
200
400
600
800
1000
0
5
10
15
20
25
30
call

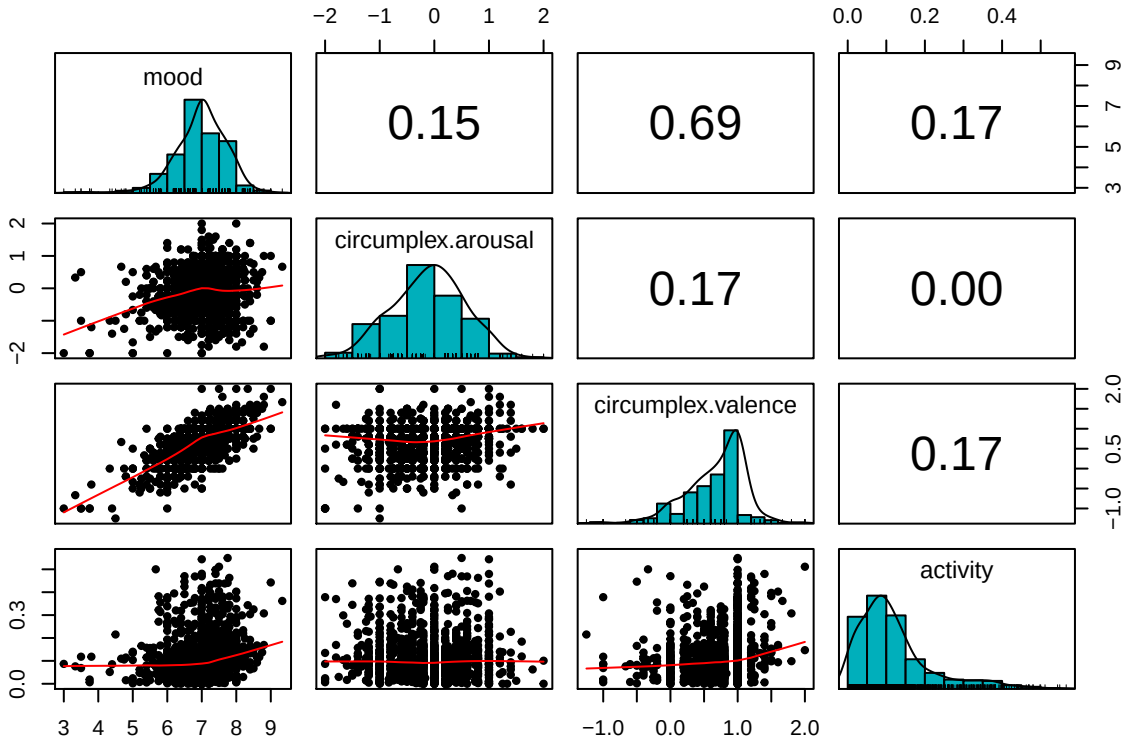

0
200
400
600
800
1000
1200
0
5
10
15
sms

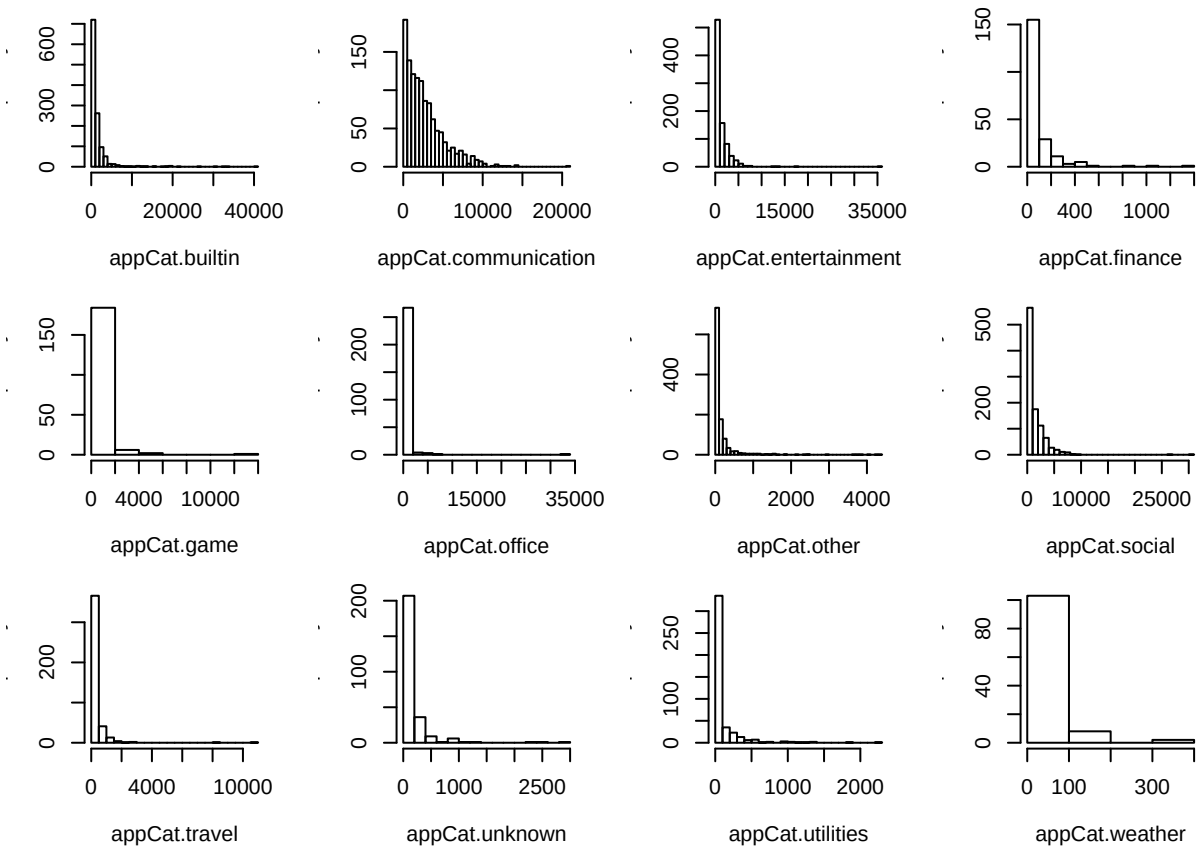

arousal, valence
2
1
0
−1
−2
circumplex.arousal

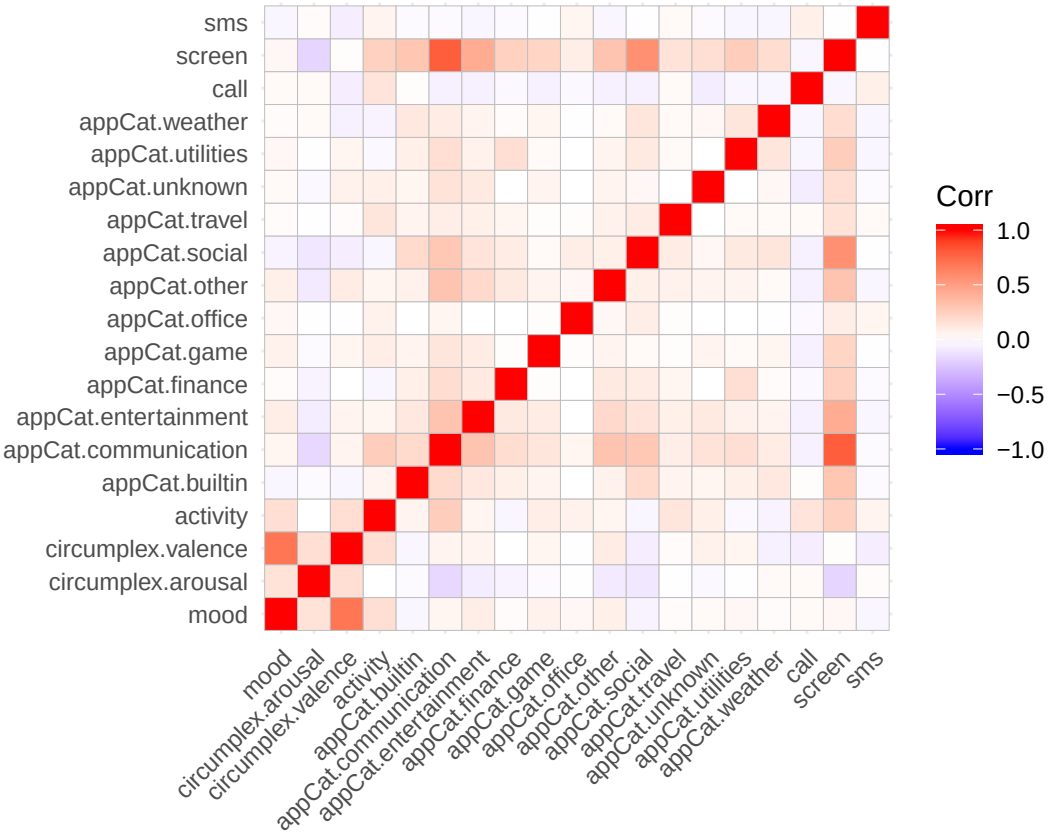

activity
0.4
0.2
0.0


call
25
15
5
0


sms
15
10
5
0


mood
0.15
0.69
0.17
circumplex.arousal
0.17
0.00
circumplex.valence
0.17
activity
−2 −1 0 1 2
0.0 0.2 0.4
3 5 7 9
−2 0 1 2
−1.0 0.5 2.0
0.0 0.3
3 4 5 6 7 8 9
−1.0 0.0 1.0 2.0


appCat.builtin
appCat.communication
appCat.entertainment
appCat.finance
appCat.game
appCat.office
appCat.other
appCat.social
appCat.travel
appCat.unknown
appCat.utilities
appCat.weather



The summary and plots of the data describe the target variable. It appears slightly skewed. `mood` and `circumplex.valence` appear to be highly collinear. The histograms for the variables starting with `appCat.`.

the `call` and `sms` data appear somewhat strange, as they seem to be linearly distributed. A portion of the data could be misrepresentative of the actual user's activity - in general, this dataset seems to contain a lot of outliers and missing values.

- *is it worth investigating removing collinear variables in the model?*
- *is it worth doing to perform transformations on the data? Boxcox etc.? Perhaps producing a mixed set of features individually transformed?*
- *Neural net responded well to standardized features, while other models not so much...*

Constructing Features

The initial features in this model were prepared by taking, for each timestep, the five preceding values and either taking the mean (`mood`, `circumplex.arousal`,`circumplex.variance`, `activity`) or the sum total (other variables).

- *is there any difference between using the sum or the mean as an aggregation in this step?*
- *what other features can be built using these? A better way to process missing values, perhaps?*

```
# create a feature set
Fset <- as.tibble(matrix(data=NA,
                         nrow=nrow(data_agg),
```

```
                          ncol=FEATURE_COUNT)) %>%
  bind_cols(data_agg[,1:2], .,data_agg[,"mood"])

# store and rename feature names
features <- sprintf("F%s", 1:(length(Fset)-3))
feature_interp <- paste("5d", colnames(data_agg)[3:21], sep = ".")
colnames(Fset)[3:length(Fset)] <- c(features, "target")

# Data imputation via the mean
data_missing <- data_agg
for (i in 4:length(data_agg)) {
data_agg[i] <-  data_agg[[i]] %>%
    replace_na(mean(data_agg[[i]], na.rm=TRUE))
}

# populate cells with 5-day estimates of each variable
attach(data_agg)
for (patient in levels(id)) {
  for(i in (6:nrow(data_agg[id==patient,]))) {
    window <- ((i-PREDICTION_WINDOW):(i-1)) + (match(patient,id)-1)

    for (k in (3:21)) {
      if (k<7) Fset[(i-1+match(patient, data_agg$id)),k]  <- mean(pull(data_agg[window,k]), na.rm = TRU
      if (k>6) Fset[(i-1+match(patient, data_agg$id)),k]  <- sum(pull(data_agg[window,k]), na.rm = TRUE
    }
  }
}
detach(data_agg)

# drop missing value rows after imputation (buggy?)
Fset <- Fset %>% drop_na()

# Feature transformation
#describe(Fset)

## standardized data
preprocessParams.std <- preProcess(Fset, method=c("center", "scale"))
Fset.std <- Fset %>%
  predict(preprocessParams.std, .) %>%
  {bind_cols((.[1:(length(.)-1)]),Fset[,"target"])}
#describe(Fset.std)

## normalized data
preprocessParams.nrm <- preProcess(Fset, method=c("range"))
Fset.nrm <- Fset %>%
  predict(preprocessParams.nrm, .) %>%
  {bind_cols(.[1:(length(Fset)-1)], Fset[,"target"])}
#describe(Fset.nrm)

## mixed data (norm sumvars, stand contvars, do BoxCox?
Fset.mix <- bind_cols(Fset.std[1:4], Fset.nrm[5:length(Fset)])
#describe(Fset.mix)
#implement boxcox, etc.? More custom dataset variation
```

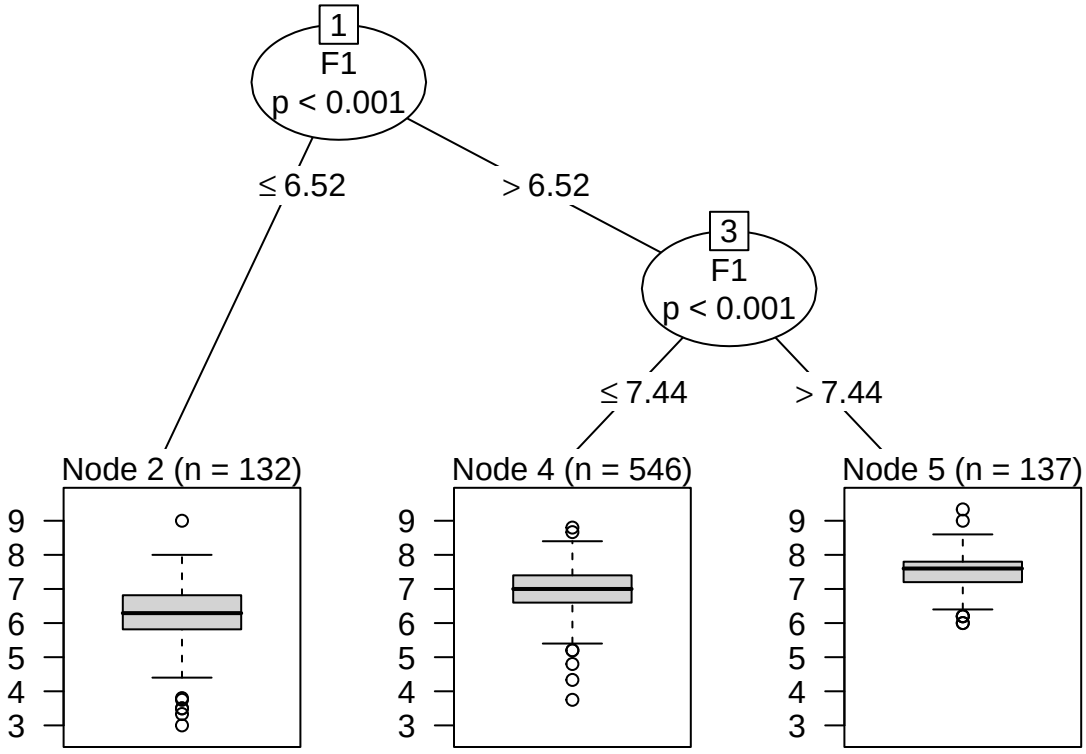
```
#could run PCA on these
```

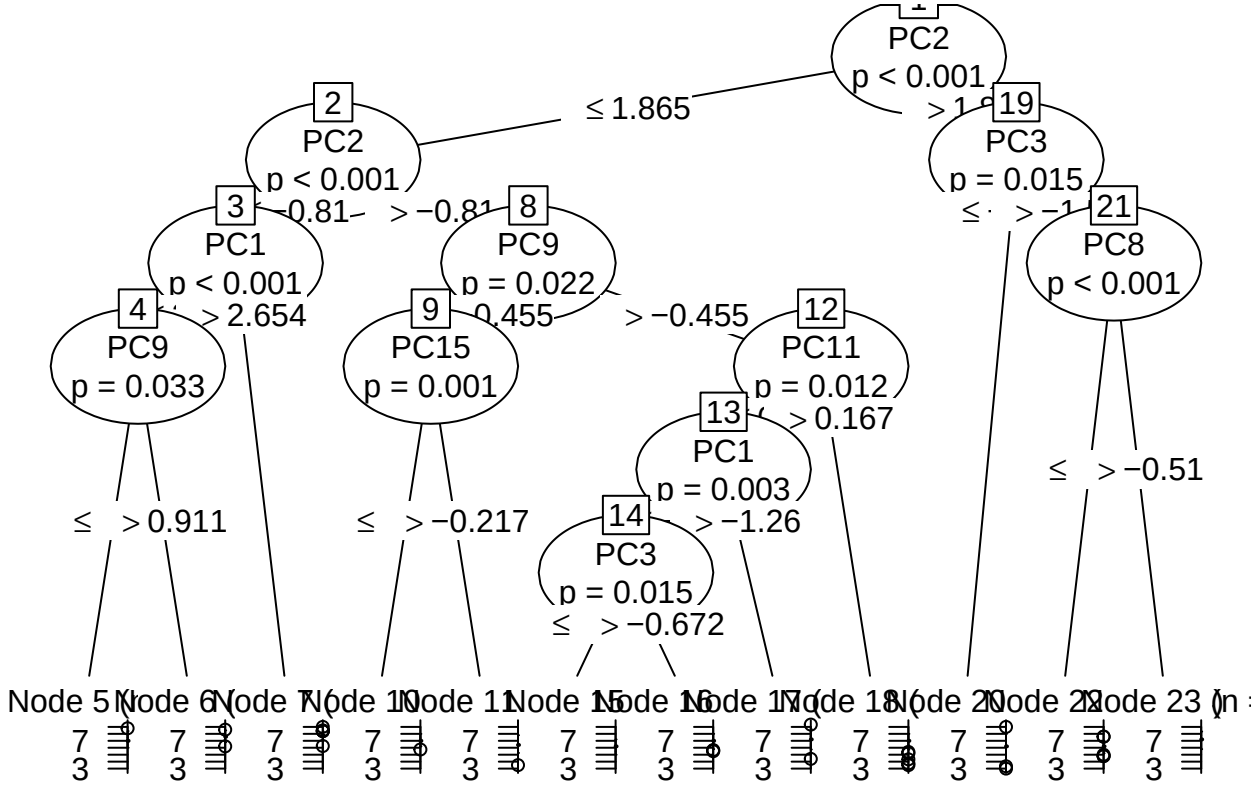
Prediction generation

Since several of the machine learning methods applied here rely on complete datasets, partially stripping `NaN` values was not feasible. Since removing all measurements with `NaN` values would remove a lot of information from the model, a method described in the literature was used (replacing missing values with the variable's mean). This also seemed to improve the predictive power of the various modeling approaches. The code generated the plots and the results shown in the tables below using mean square error and misclassified examples metrics. Misclassified examples were defined as inequal values and predictions after rounding to the nearest integer, discretizing results to blocks (... | 6.5-7.5 | 7.5-8.5 | ...). Only three tree plots are shown, as the normalised, standardised and untransformed datasets looked similar

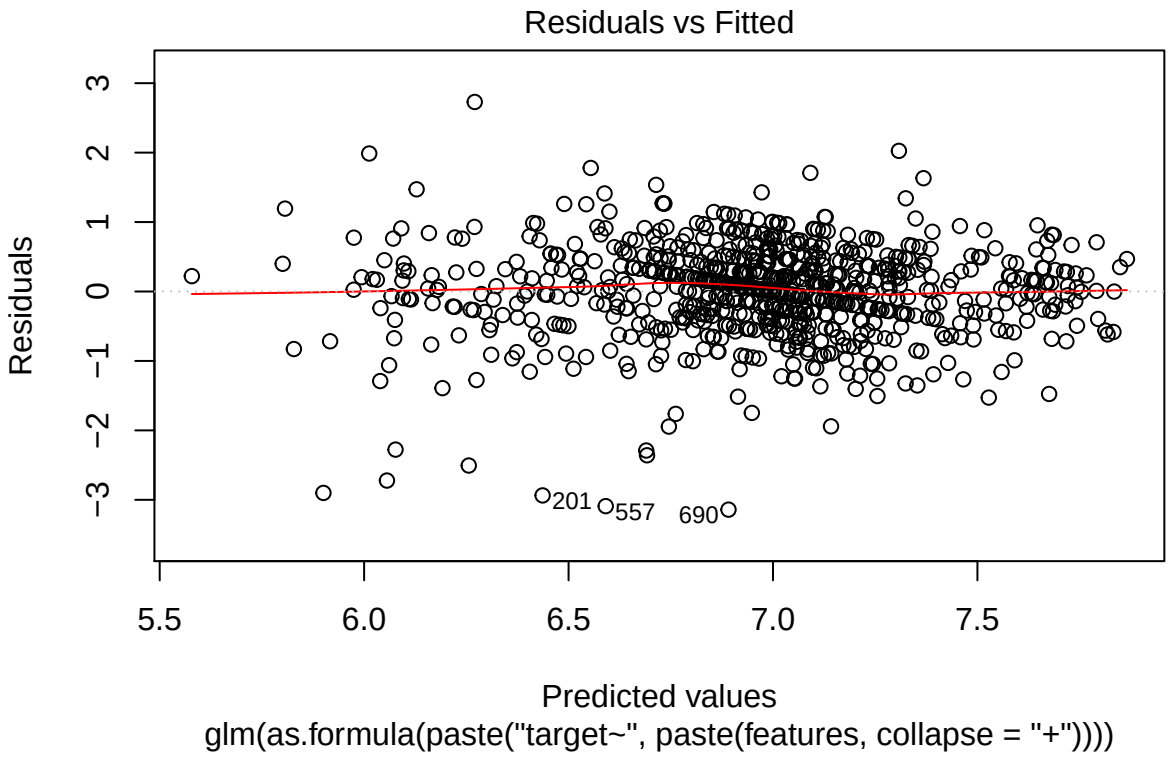
```
# Test / Train set split parameters
set.seed(SEED)
train_size = TEST_TRAIN_PROP * nrow(Fset)

### train indices for a train/test split
train_ind <- sample(seq_len(nrow(Fset)), size = train_size)

## PCA
preprocessParams.pca <- preProcess(Fset[train_ind,3:(length(Fset)-1)],
                                   method=c("center", "scale", "pca"))
Fset.pca <- Fset[,3:(length(Fset)-1)] %>%
  predict(preprocessParams.pca, .) %>%
  bind_cols(Fset[,"target"]) %>%
  tbl_df()
#describe(Fset.pca)

princomps <- colnames(Fset.pca[1:(length(Fset.pca)-1)])

# Models
## Decision Tree
tree.unt <- ctree(as.formula(paste("target~", paste(features, collapse="+"))),
                  data=Fset[train_ind,])
tree.std <- ctree(as.formula(paste("target~", paste(features, collapse="+"))),
                  data=Fset.std[train_ind,])
tree.nrm <- ctree(as.formula(paste("target~", paste(features, collapse="+"))),
                  data=Fset.nrm[train_ind,])
tree.pca <- ctree(as.formula(paste("target~", paste(princomps, collapse="+"))),
                  data=Fset.pca[train_ind,])
tree.mix <- ctree(as.formula(paste("target~", paste(features, collapse="+"))),
                  data=Fset.mix[train_ind,])
trees <- list(tree.unt,tree.std, tree.nrm, tree.pca, tree.mix)

## Generalized Linear Model
glm.unt <- glm(as.formula(paste("target~", paste(features, collapse="+"))),
               data=Fset[train_ind,])
glm.std <- glm(as.formula(paste("target~", paste(features, collapse="+"))),
               data=Fset.std[train_ind,])
glm.nrm <- glm(as.formula(paste("target~", paste(features, collapse="+"))),
               data=Fset.nrm[train_ind,])
```

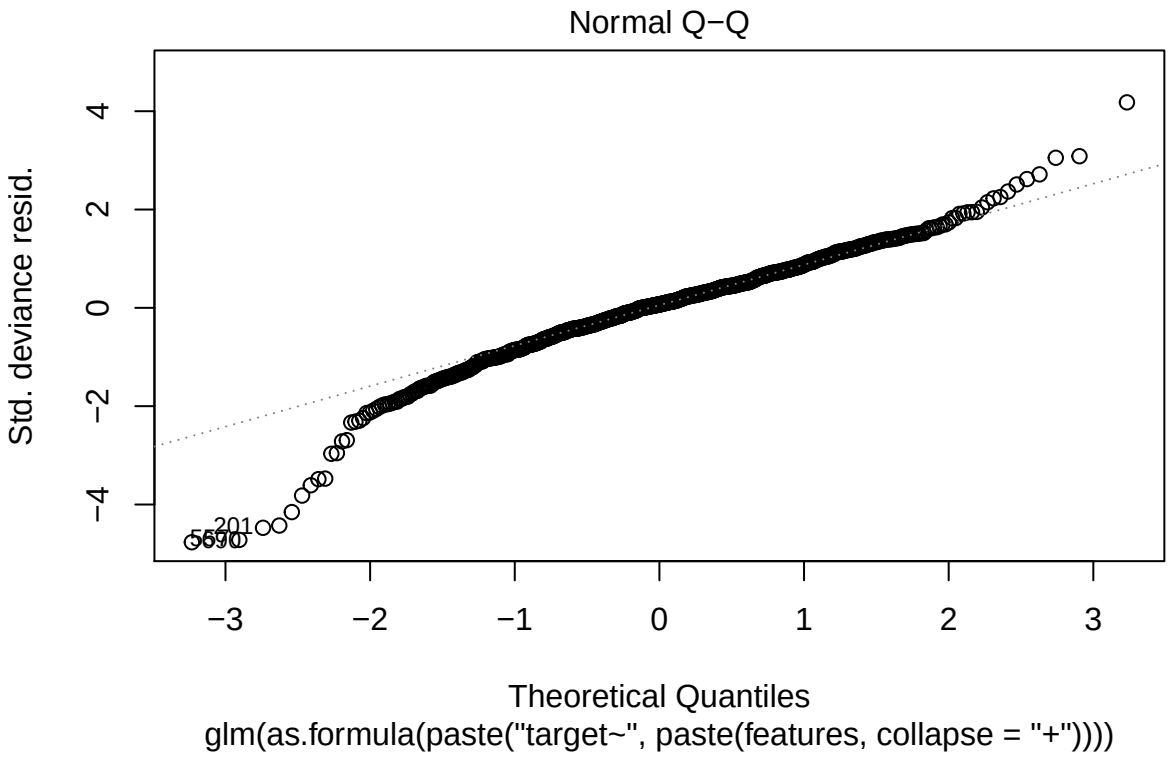
```
glm.pca <- glm(as.formula(paste("target~", paste(princomps, collapse="+"))),
               data=Fset.pca[train_ind,])
glm.mix <- glm(as.formula(paste("target~", paste(features, collapse="+"))),
               data=Fset.mix[train_ind,])
glms = list(glm.unt,glm.std,glm.nrm,glm.pca,glm.mix)

## SVM
svm.unt <- svm(as.formula(paste("target~", paste(features, collapse="+"))),
               data=Fset[train_ind,])
svm.std <- svm(as.formula(paste("target~", paste(features, collapse="+"))),
               data=Fset.std[train_ind,])
svm.nrm <- svm(as.formula(paste("target~", paste(features, collapse="+"))),
               data=Fset.nrm[train_ind,])
svm.pca <- svm(as.formula(paste("target~", paste(princomps, collapse="+"))),
               data=Fset.pca[train_ind,])
svm.mix <- svm(as.formula(paste("target~", paste(features, collapse="+"))),
               data=Fset.mix[train_ind,])
svms = list(svm.unt,svm.std,svm.nrm,svm.pca,svm.mix)

# Neural net
nn.unt <- Fset.std[train_ind,] %>%
  neuralnet(as.formula(paste("target~", paste(features, collapse="+"))),
            hidden=c(10,5),stepmax=1e6,threshold=0.1,linear.output=T,data=.)
nn.std <- Fset.std[train_ind,] %>%
  neuralnet(as.formula(paste("target~", paste(features, collapse="+"))),
            hidden=c(10,5),stepmax=1e6,threshold=0.1,linear.output=T,data=.)
nn.nrm <- Fset.nrm[train_ind,] %>%
  neuralnet(as.formula(paste("target~", paste(features, collapse="+"))),
            hidden=c(10,5),stepmax=1e6,threshold=0.1, linear.output=T,data=.)
nn.pca <- Fset.pca[train_ind,] %>%
  neuralnet(as.formula(paste("target~", paste(princomps, collapse="+"))),
            hidden=c(10,5),stepmax=1e6,threshold=0.1,linear.output=T,data=.)
nn.mix <- Fset.mix[train_ind,] %>%
  neuralnet(as.formula(paste("target~", paste(features, collapse="+"))),
            hidden=c(10,5),stepmax=1e6,threshold=0.1,linear.output=T,data=.)

# nns = list(nn.unt,nn.std,nn.nrm,nn.pca,nn.mix)

# generate plots of the models
plot(trees[[1]]);plot(trees[[4]])
```

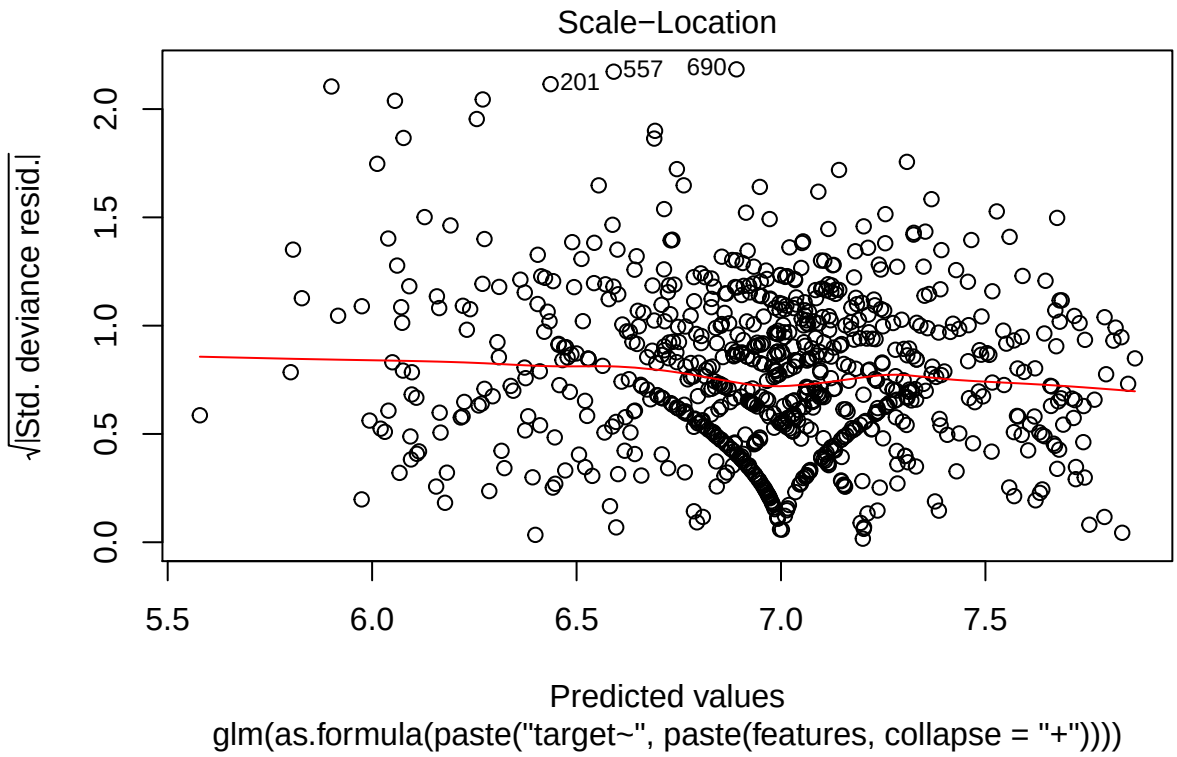

1
F1
p < 0.001
≤ 6.52
> 6.52
3
F1
p < 0.001
≤ 7.44
> 7.44
Node 2 (n = 132)
Node 4 (n = 546)
Node 5 (n = 137)
9
8
7
6
5
4
3

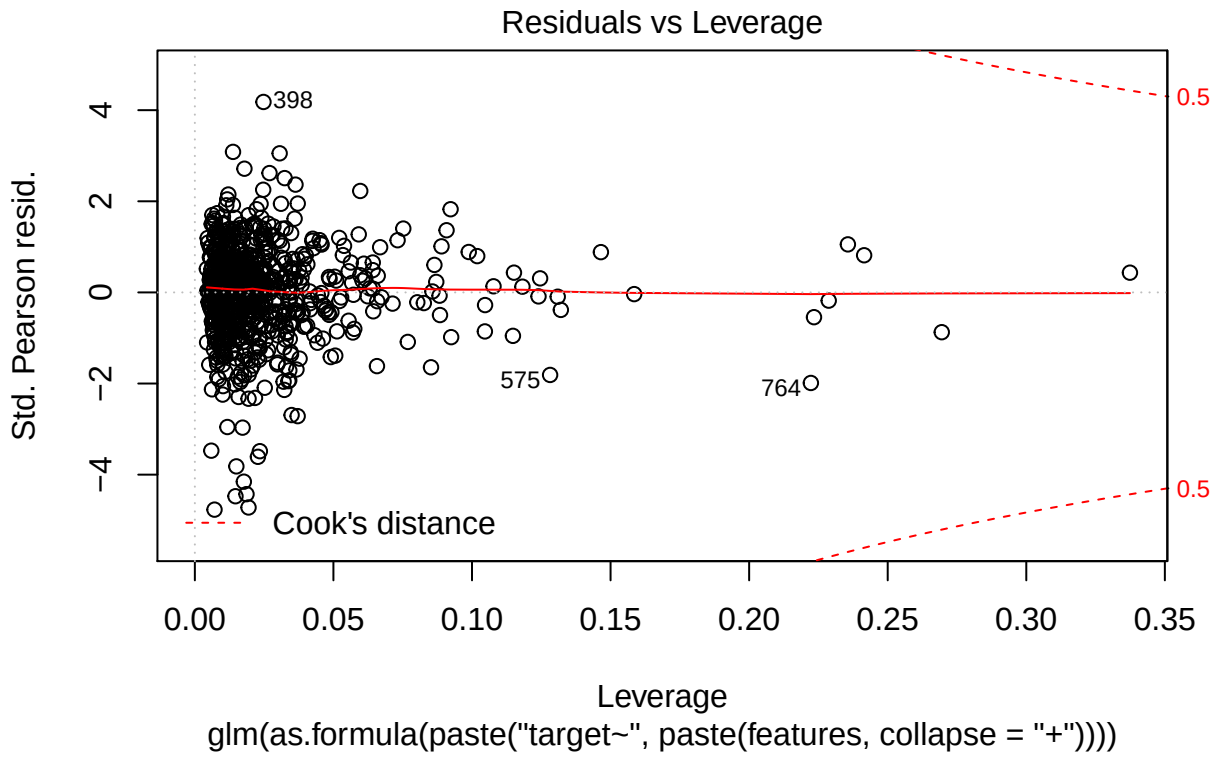

PC2
p < 0.001
≤ 1.865
2
PC2
p < 0.001
3
PC1
p < 0.001
4
PC9
p = 0.033
≤ > 0.911
> 2.654
8
PC9
p = 0.022
9
PC15
p = 0.001
≤ > −0.217
> −0.455
12
PC11
p = 0.012
> 0.167
13
PC1
p = 0.003
> −1.26
14
PC3
p = 0.015
≤ > −0.672
19
PC3
p = 0.015
21
PC8
p < 0.001
≤ > −0.51
Node 5 (n = 73)
Node 6 (n = 73)
Node 7 (n = 73)
Node 10 (n = 73)
Node 11 (n = 73)
Node 15 (n = 73)
Node 16 (n = 73)
Node 17 (n = 73)
Node 18 (n = 73)
Node 20 (n = 73)
Node 22 (n = 73)
Node 23 (n = 73)

```
plot(glm.std)
```


Residuals vs Fitted
Residuals
3
2
1
0
−1
−2
−3
201
557
690
5.5
6.0
6.5
7.0
7.5
Predicted values
glm(as.formula(paste("target~", paste(features, collapse = "+"))))


Normal Q−Q
Std. deviance resid.
Theoretical Quantiles
glm(as.formula(paste("target~", paste(features, collapse = "+"))))
201
5570
5690
−4
−2
0
2
4
−3
−2
−1
0
1
2
3


Scale−Location
201
557
690
√|Std. deviance resid.|
0.0
0.5
1.0
1.5
2.0
5.5
6.0
6.5
7.0
7.5
Predicted values
glm(as.formula(paste("target~", paste(features, collapse = "+"))))




```
plot(nn.unt)
```

```
##                       tree  glm  svm   nn benchmark
## Untransformed         0.34 0.33 0.33 0.91      0.59
## Standardized          0.34 0.33 0.33 1.02        NA
## Normalized            0.34 0.33 0.33 0.93        NA
## Principal Components  0.41 0.36 0.34 0.61        NA
## Mixed Variables       0.34 0.33 0.33 0.80        NA

##                       tree glm svm  nn benchmark
## Untransformed          165 176 173 238       212
## Standardized           165 176 173 198        NA
## Normalized             165 176 173 206        NA
## Principal Components   193 183 179 222        NA
## Mixed Variables        165 176 173 220        NA

##                       tree  glm  svm   nn benchmark
## Untransformed         0.08 0.09 0.09 0.12      0.11
## Standardized          0.08 0.09 0.09 0.10        NA
## Normalized            0.08 0.09 0.09 0.10        NA
## Principal Components  0.10 0.09 0.09 0.11        NA
## Mixed Variables       0.08 0.09 0.09 0.11        NA
```

The neural network may have performed poorly simply because of the limitations in the `neuralnet` package or incorrect parameters. Many networks could not converge, and data transformation seemed necessary to

achieve even reasonably close predictions, barely beating the benchmark (value of the day before). The tree models were mostly simple structures with four outcomes. Only the tree for the PCA values had a complicated structure, but this clearly significantly worsened prediction. Nonetheless, the mean square error was about equal to those of linear methods and in the classification results, the `tree` column is a clear winner. GLM and SVM both performed the best on the data in the mean square error category. In fact, both methods seem almost equivalent.

- *Can I generate iteration diagrams for the nn?*
- *Evaluation setup? Validation set/means?*