Smart Universal Sustainability System Yielding Biodegradability Analysis of Known Agents

Drusany J, Ploj Y

2023-01-04

Exploration

```
data <- read.csv("train.csv")
df <- data
testing_data <- read.csv("test.csv")
head(data)
```

```
##      V1     V2 V3 V4 V5 V6 V7   V8 V9 V10 V11    V12   V13   V14   V15 V16
## 3 3.932 3.2512  0  0  0  0  0 26.7  2   4   0  0.000 3.279 2.585 9.110   0
## 5 4.236 3.3944  0  0  0  0  0 29.4  2   4   0 -0.271 3.449 2.753 9.528   2
## 6 4.236 3.4286  0  0  0  0  0 28.6  2   4   0 -0.275 3.313 2.522 9.383   1
## 7 5.000 5.0476  1  0  0  0  0 11.1  0   3   0  0.000 2.872 0.722 9.657   0
## 8 4.525 3.8301  0  0  0  0  0 31.6  3   2   0 -0.039 3.418 2.468 9.786   5
## 9 4.596 3.0777  0  0  0  0  2 44.4  2   0   0  0.000 2.970 0.875 9.540   0
##     V17   V18 V19 V20 V21   V22 V23 V24 V25 V26   V27    V28 V29    V30   V31
## 3 1.009 1.152   0   0   0 1.092   0   0   0   0 1.942 -0.008   0  0.000 4.891
## 5 1.004 1.147   0   0   0 1.137   0   0   0   0 1.985 -0.002   0 10.348 5.588
## 6 1.014 1.149   0   0   0 1.119   0   0   0   0 1.980 -0.008   0 10.276 4.746
## 7 1.092 1.153   0   0   0 1.125   0   0   0   0 2.000  0.446   0 18.375 0.800
## 8 0.980 1.142   0   0   0 1.179   0   0   0   0 2.119 -0.002   0 11.115 3.889
## 9 0.968 1.115   0   0   0 1.328   1   0   0   0 2.175  0.041   0  0.000 1.069
##   V32 V33 V34 V35   V36   V37 V38   V39 V40 V41 Class
## 3   0   0   0   1 3.076 2.417   0 7.601   0   0     2
## 5   0   0   0   0 3.351 2.405   0 8.003   0   0     2
## 6   0   0   0   0 3.351 2.556   0 7.904   0   0     2
## 7   0   0   0   1 4.712 4.583   0 9.303   0   0     2
## 8   0   0   0   0 3.379 2.143   0 7.950   0   0     2
## 9   0   0   0   0 3.626 1.917   0 7.939   0   0     2
```

We prepared some functions to help us with the analysis.

```
mcf <- function(data) {
    #' Return the major class frequency per each column
    return(sapply(data, function(x) max(table(x)) / sum(table(x))))
}
correlation <- function(data, threshold) {
    #' Return pairs of features with correlations above a threshold
    mat = cor(na.omit(data[, -ncol(data)]))
    mat[lower.tri(mat, diag = TRUE)] <- 0
    correlations <- which(abs(mat) > threshold, arr.ind = TRUE)
    rownames(correlations) <- NULL
    colnames(correlations) <- NULL
```

```
    annotated = t(apply(
        correlations,
        1,
        function(x) c(x[1], x[2], mat[x[1], x[2]])
    ))
    return(annotated)
}
```

General data analysis

Percentage of rows with missing features:

```
print(sum(apply(data, 1, function(x) any(is.na(x)))) / nrow(data))
```

```
## [1] 0.09574468
```

Number of missing features per row:

```
print(table(apply(data, 1, function(x) sum(is.na(x)))))
```

```
##
##   0   1   2
## 765  80   1
```

Total missing features by column:

```
missing_values <- apply(data, 2, function(x) sum(is.na(x)))
print(missing_values[missing_values > 0])
```

```
##  V4 V22 V27 V29 V37
##  25  16   8   8  25
```

Class counts:

```
groups = split(data, data$Class)
print(sapply(groups, nrow))
```

```
##   1   2
## 564 282
```

Number of unique values per column:

```
print(sapply(data, function(x) length(unique(x))))
```

```
##    V1    V2    V3    V4    V5    V6    V7    V8    V9   V10   V11   V12   V13
##   379   823    11     4    16    10    14   172    15    11    21   307   638
##   V14   V15   V16   V17   V18   V19   V20   V21   V22   V23   V24   V25   V26
##   321   436    24   158   115     2     4     3   322    13     2     2     3
##   V27   V28   V29   V30   V31   V32   V33   V34   V35   V36   V37   V38   V39
##   291   174     3   372   468     8    11    16     8   603   536     8   711
##   V40   V41 Class
##     5    16     2
```

Major class frequency per column:

```
MCF = mcf(data)
print(MCF)
```

```
##          V1          V2          V3          V4          V5          V6
## 0.024822695 0.002364066 0.680851064 0.974421437 0.696217494 0.862884161
```

```
##          V7          V8          V9         V10         V11         V12
## 0.479905437 0.065011820 0.438534279 0.304964539 0.659574468 0.622931442
##         V13         V14         V15         V16         V17         V18
## 0.009456265 0.053191489 0.024822695 0.360520095 0.042553191 0.030732861
##         V19         V20         V21         V22         V23         V24
## 0.995271868 0.938534279 0.981087470 0.010843373 0.413711584 0.962174941
##         V25         V26         V27         V28         V29         V30
## 0.843971631 0.976359338 0.042959427 0.302600473 0.974940334 0.542553191
##         V31         V32         V33         V34         V35         V36
## 0.021276596 0.937352246 0.609929078 0.630023641 0.498817967 0.005910165
##         V37         V38         V39         V40         V41       Class
## 0.010962241 0.608747045 0.004728132 0.966903073 0.777777778 0.666666667
```

```
print(MCF[MCF > 0.95])
```

```
##        V4       V19       V21       V24       V26       V29       V40
## 0.9744214 0.9952719 0.9810875 0.9621749 0.9763593 0.9749403 0.9669031
```

Correlation between features:

```
print(correlation(data, 0.90))
```

```
##      [,1] [,2]      [,3]
## [1,]    1   15 0.9100482
## [2,]    1   27 0.9218017
## [3,]   15   27 0.9222515
## [4,]   36   39 0.9186995
```

Variance of features:

```
print(apply(data, 2, var))
```

```
##           V1           V2           V3           V4           V5           V6
## 2.830144e-01 6.625676e-01 2.263656e+00 3.931553e-02 5.373499e+00 1.093162e+00
##           V7           V8           V9          V10          V11          V12
## 4.929383e+00 8.154105e+01 4.074072e+00 3.146564e+00 1.006504e+01 5.754003e-01
##          V13          V14          V15          V16          V17          V18
## 3.318507e-01 6.031522e-01 8.212696e-01 2.101845e+01 2.009771e-03 8.224612e-04
##          V19          V20          V21          V22          V23          V24
## 4.711346e-03 9.065285e-02 3.606950e-02 8.856541e-03 2.802252e+01 3.643739e-02
##          V25          V26          V27          V28          V29          V30
## 1.318394e-01 2.660204e-02 4.908201e-02 2.287145e-02 2.446087e-02 1.402481e+02
##          V31          V32          V33          V34          V35          V36
## 4.267224e+00 4.506735e-01 2.329056e+00 5.056578e+00 1.535451e+00 9.853253e-01
##          V37          V38          V39          V40          V41        Class
## 3.906513e-01 1.196033e+00 1.497441e+00 1.172129e-01 4.602722e+00 2.224852e-01
```

Outliers per column:

```
for (col in 1:ncol(data)) {
    s = colnames(data)[col]
    if (typeof(data[,col]) != "double")
        num = 0
    else
        num = sum(
            data[,col] < quantile(data[,col], 0.05, na.rm=TRUE) |
            data[,col] > quantile(data[,col], 0.95, na.rm=TRUE)
        ) / length(data[,col])
```

```
    print(paste(s, num, sep=": "))
}
```

```
## [1] "V1: 0.099290780141844"
## [1] "V2: 0.101654846335697"
## [1] "V3: 0"
## [1] "V4: 0"
## [1] "V5: 0"
## [1] "V6: 0"
## [1] "V7: 0"
## [1] "V8: 0.0921985815602837"
## [1] "V9: 0"
## [1] "V10: 0"
## [1] "V11: 0"
## [1] "V12: 0.101654846335697"
## [1] "V13: 0.101654846335697"
## [1] "V14: 0.049645390070922"
## [1] "V15: 0.100472813238771"
## [1] "V16: 0"
## [1] "V17: 0.100472813238771"
## [1] "V18: 0.0969267139479905"
## [1] "V19: 0"
## [1] "V20: 0"
## [1] "V21: 0"
## [1] "V22: NA"
## [1] "V23: 0"
## [1] "V24: 0"
## [1] "V25: 0"
## [1] "V26: 0"
## [1] "V27: NA"
## [1] "V28: 0.101654846335697"
## [1] "V29: 0"
## [1] "V30: 0.0508274231678487"
## [1] "V31: 0.0874704491725768"
## [1] "V32: 0"
## [1] "V33: 0"
## [1] "V34: 0"
## [1] "V35: 0"
## [1] "V36: 0.101654846335697"
## [1] "V37: NA"
## [1] "V38: 0"
## [1] "V39: 0.101654846335697"
## [1] "V40: 0"
## [1] "V41: 0"
## [1] "Class: 0"
```

Visualizations

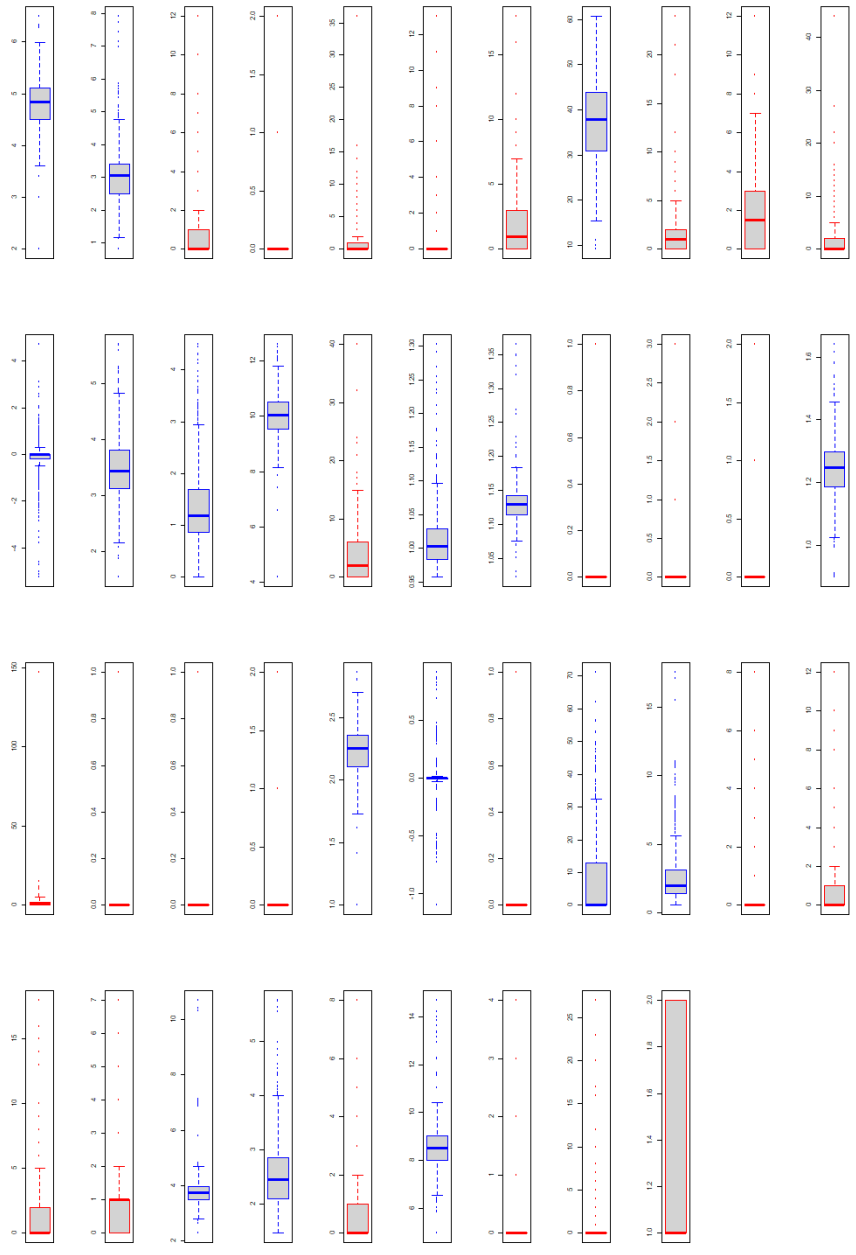
```
colour <- function(datatype) ifelse(datatype == "double", "blue", "red")

save_plots <- function(data, width, height, plot_function, plots_per_page = c(1,1),
    plot_count=NaN, filename=function(page) page) {
    #' Generate N plots
```

```
    if (is.na(plot_count)) plot_count = ncol(data)
    if (typeof(filename) == "character")
        filename = local(function(page) filename, list2env(list(filename=filename)))
    for (page in 1:floor(plot_count / prod(plots_per_page))) {
            name = as.character(filename(page))
        png(paste(name, ".png", sep=""), width, height)
        par(mfrow=plots_per_page)
        start = (page-1) * prod(plots_per_page) + 1
        end = min(plot_count, page * prod(plots_per_page))
        for (col in start:end) {
            plot_function(col)
        }
        dev.off()
    }
}
save_boxplots <- function(data, width=1000, height=1500, filename="boxplot") {
    #' Generate boxplots of each column
    save_plots(
        data=data,
        width=width,
        height=height,
        plot_function=function(col)
            boxplot(data[,colnames(data)[col]], width=1, border=colour(typeof(data[,col]))),
        plots_per_page=c(4,11),
        filename=filename
    )
}
save_barplots <- function(data, width=1000, height=2000, filename="barplot") {
    #' Generate barplots of each column
    save_plots(
        data=data,
        width=width,
        height=height,
        plot_function=function(col)
            barplot(table(data[,col]), main=colnames(data)[col], border=colour(typeof(data[,col]))),
        plots_per_page=c(11,4),
        filename=filename
    )
}
save_scatterplots <- function(data, width=10000, height=10000, filename="scatterplot") {
    #' Generate scatterplots of each column, colour them by class
    save_plots(
        data=data,
        width=width,
        height=height,
        plot_function=function(col)
            pairs(data, col=adjustcolor(c("red", "blue", alpha=0.5)[data$Class])),
        plots_per_page=c(1,1),
        plot_count=1,
        filename=filename
    )
}
```

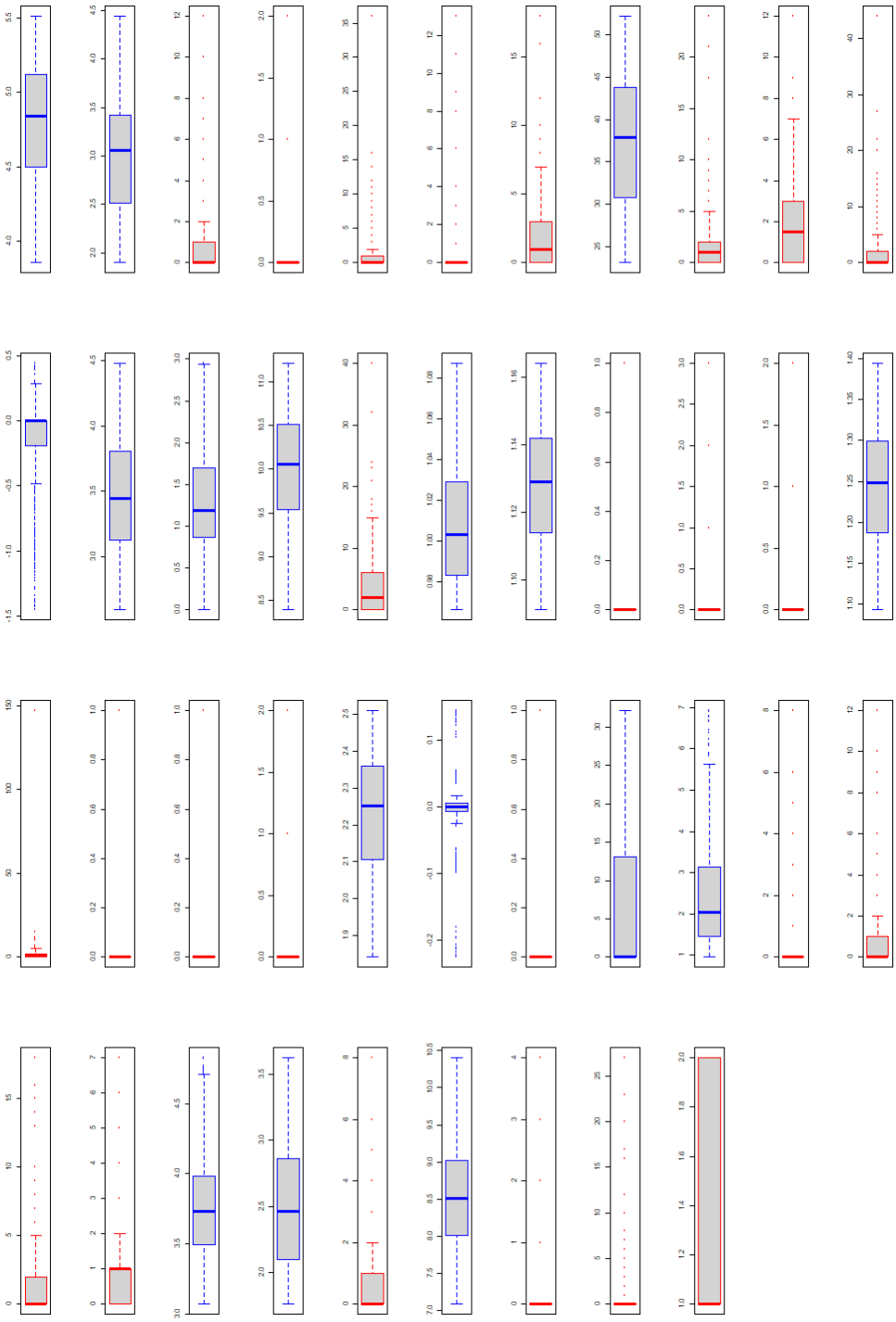
```
save_boxplots(df, filename="images/boxes")
save_boxplots(outliers.winsorize(df), filename="images/boxes_winsorized")
save_barplots(outliers.winsorize(df), filename="images/histograms")
save_scatterplots(df, filename="images/scatters")
save_scatterplots(transform.pca(impute.mean(df)), filename="images/scatters_pca")
```

The following plots were generated from the data (blue = continuous, red = categorical):

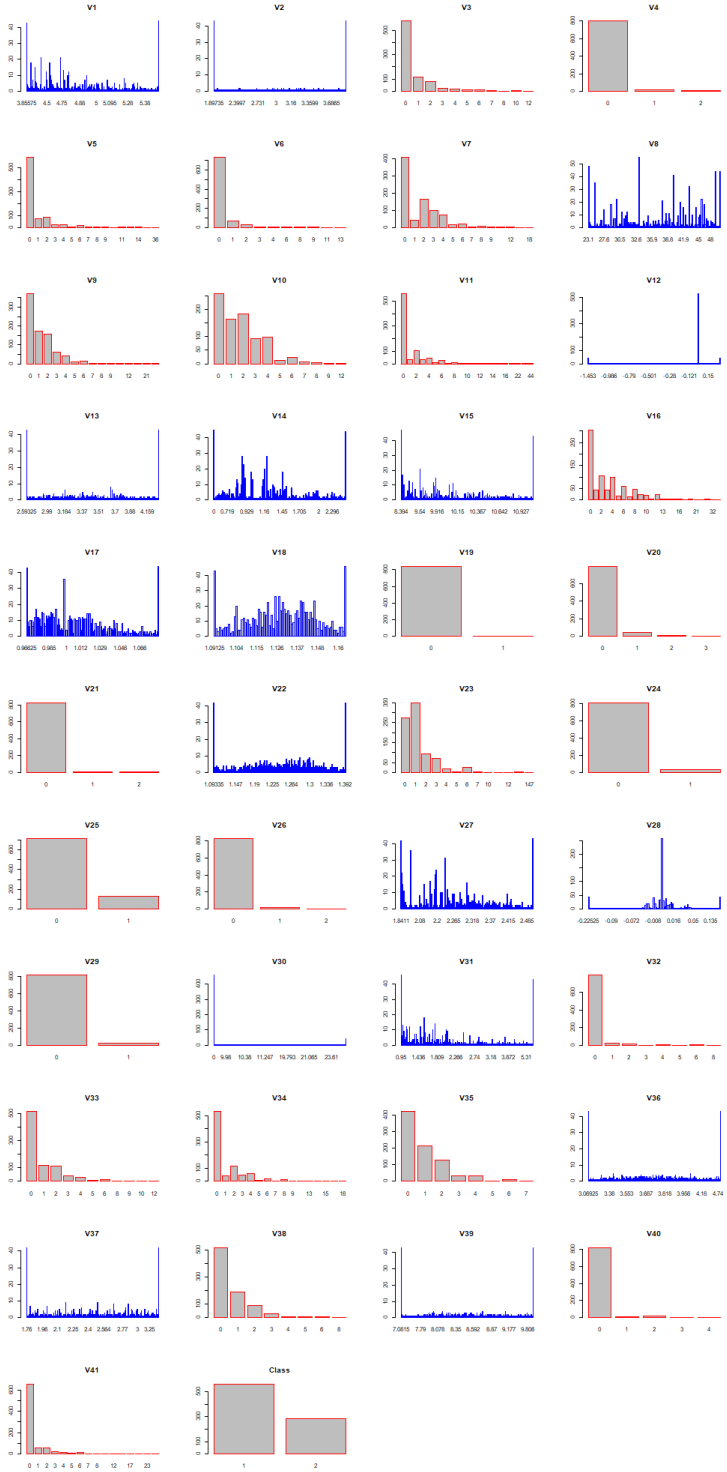
Raw attributes box plots



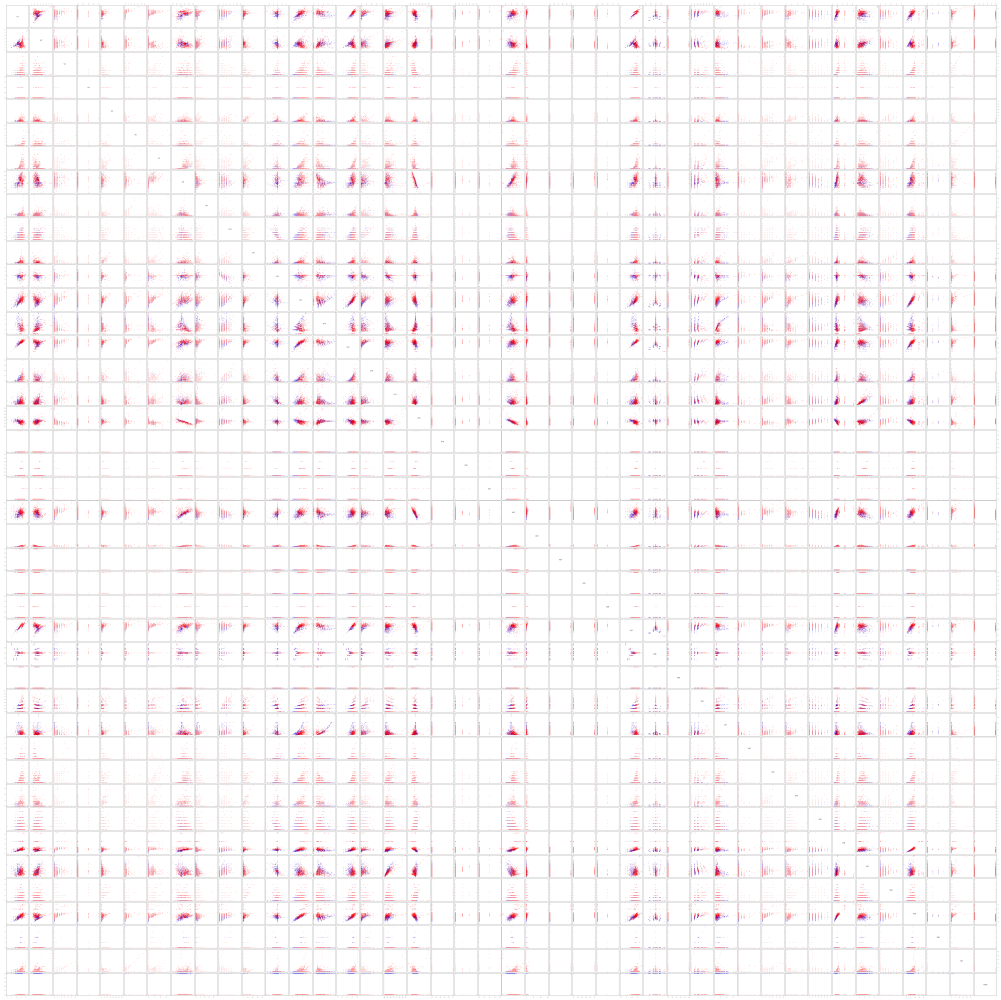
Winsorized attributes



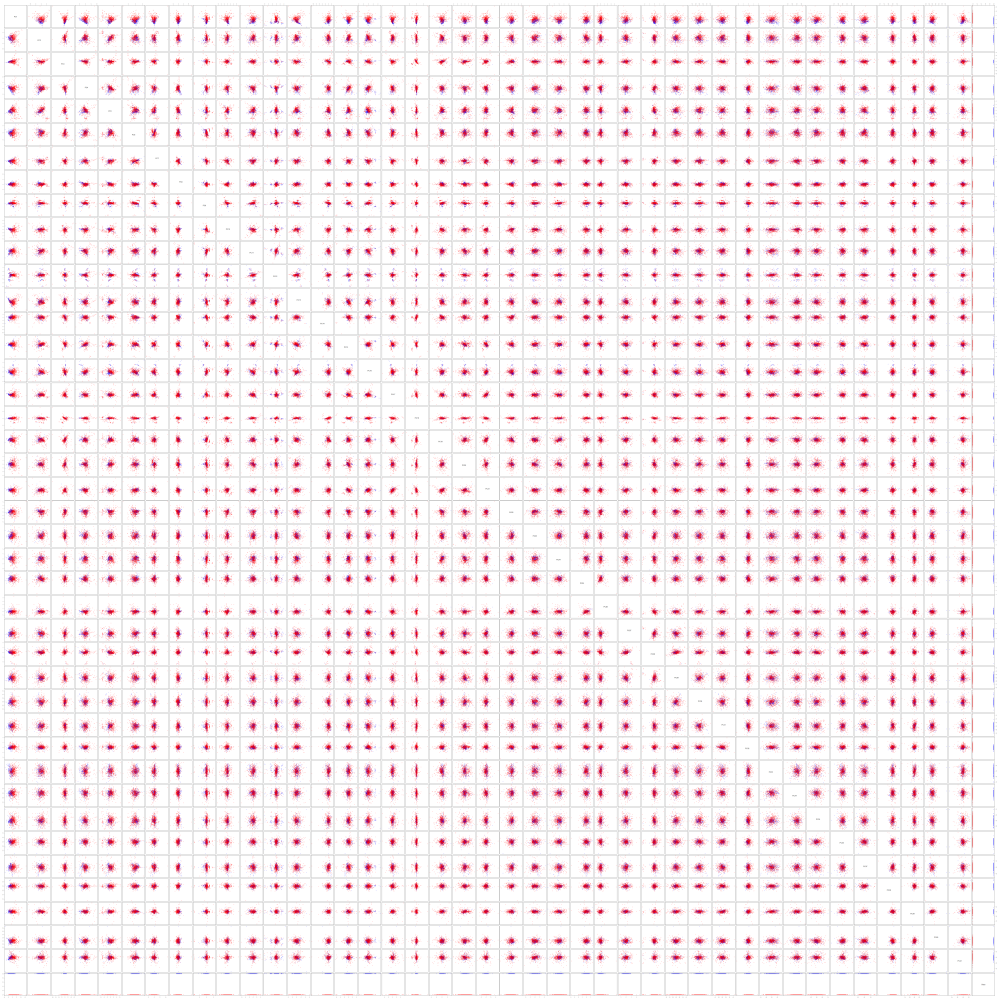
Attribute histograms



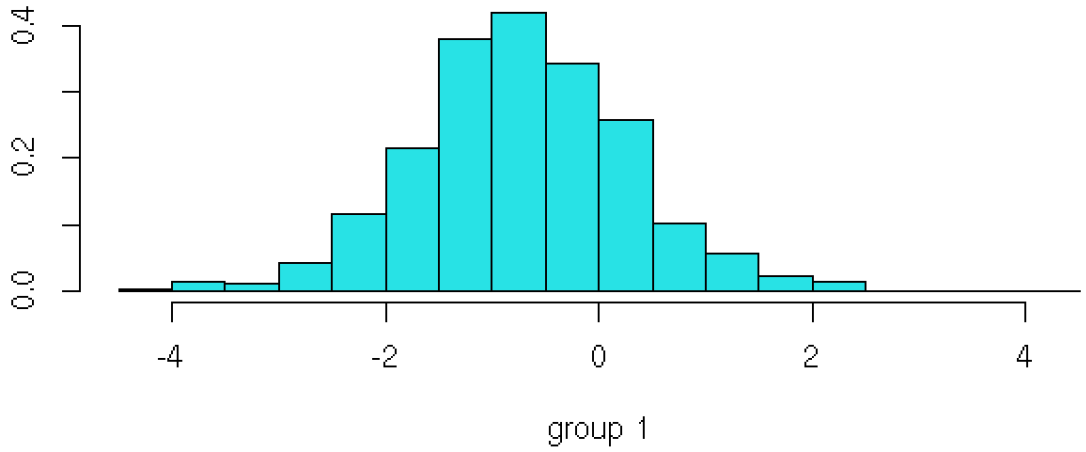
Attribute scatterplots

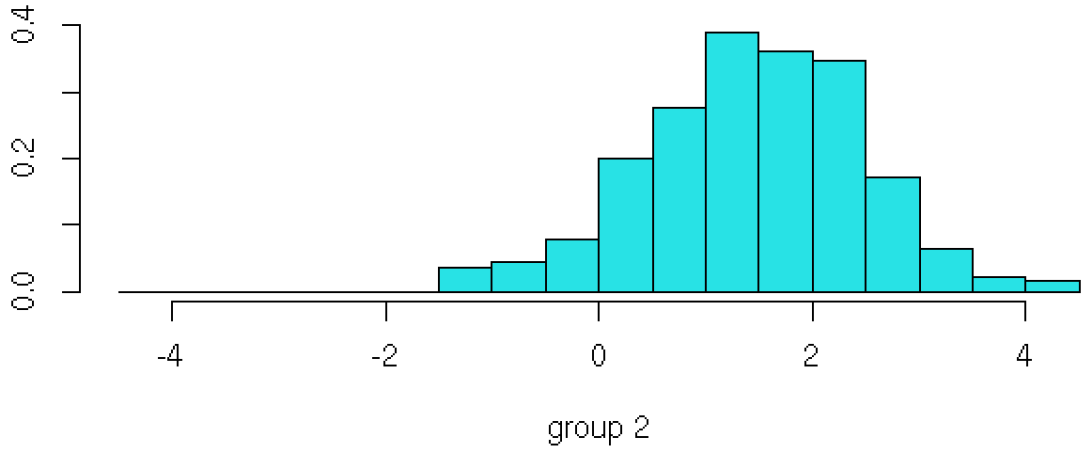


Principal components' scatterplots

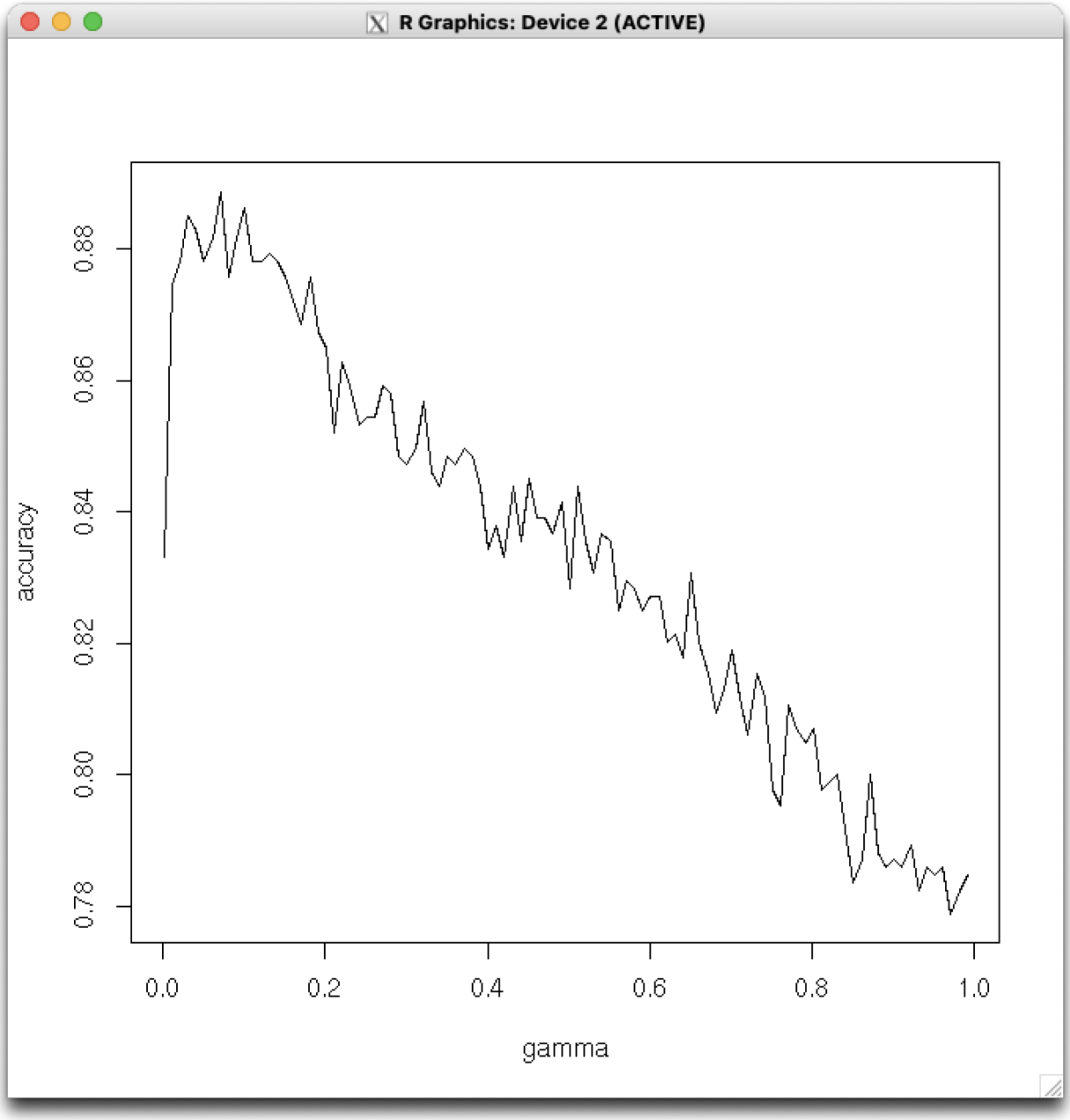


LDA space





SVM accuracy gamma



After analyzing the data, we found the following:

- ~10% of rows have a missing value (all but one are missing precisely one value)
- Most of the missing data appear in the same few columns.
- There are 2 classes with prior probabilities of 2/3 and 1/3.
- We counted unique values per column to determine which columns are categorical and which are continuous.
- 7 attributes have a majority class frequency of 0.95 or higher. These are likely to be useless.
- 4 pairs of attributes have a correlation of 0.90 or higher. These are likely to be redundant.

- None of the continuous columns had a significantly low variance.
- Several columns had a high percentage (up to 10%) of outliers.

Preprocessing transformers

In light of the findings above, we designed the following transformers:

```
dist <- function(datum, data) {
    #' Calculate the distance from datum to each row in data
    return(apply(data, 1, function(x) sqrt(sum((as.integer(datum) - as.integer(x))^2))))
}

transform.pca <- function(data) {
    #' Transform data using PCA
    pca <- prcomp(data[,-ncol(data)], center=TRUE, scale=TRUE)
    ret = data.frame(pca$x)
    ret$Class <- data$Class
    return(ret)
}

outliers.winsorize <- function(data, threshold=0.05) {
    #' Winsorize outliers
    for (i in 1:ncol(data)) {
        if (typeof(data[,i]) != "double") next
        quantiles <- quantile(
            data[,i],
            probs=c(threshold, 1-threshold),
            na.rm=TRUE
        )
        data[,i] <- ifelse(data[,i] < quantiles[1],
            quantiles[1],
            ifelse(data[,i] > quantiles[2],
                quantiles[2],
                data[,i]
            )
        )
    }
    return(data)
}

impute.mean <- function(data) {
    #' Impute missing values with the mean of the column
    for (i in 1:ncol(data)) {
        if (typeof(data[,i]) %in% c("double", "integer")) {
            data[,i] <- ifelse(is.na(data[,i]),
                mean(data[,i], na.rm=TRUE),
                data[,i]
            )
        }
    }
    return(data.frame(data))
}
impute.filter <- function(data) {
    #' Remove rows with missing data
```

```
    return(data[complete.cases(data),])
}
impute.knn <- function(data, k=1) {
    #' Impute using k nearest neighbours
    to_impute = data[!complete.cases(data),]
    for (row in rownames(to_impute)) {
        incomplete_row = colnames(data[row,])[apply(data[row,], 2, anyNA)]
        incomplete_filter = (names(data[row,]) %in% incomplete_row) # booleans
        distances = dist(data[row,!incomplete_filter], data[complete.cases(data),!incomplete_filter])
        k_nearest = data[complete.cases(data),][order(distances)[1:k],]

        # replace only na's
        for (col in incomplete_row) {
            if (typeof(data[,col]) %in% c("double", "integer")) {
                data[row,col] <- ifelse(is.na(data[row,col]),
                    mean(k_nearest[,col], na.rm=TRUE),
                    data[row,col]
                )
            }
        }
    }
    return(data.frame(data))
}

prune.mcf <- function(data, cutoff = 0.95) {
    #' Prune features with major class frequency > cutoff
    MCF = mcf(data)
    return(data[, names(MCF[MCF <= cutoff])])
}
prune.variance <- function(data, cutoff = 0.1) {
    #' Prune features with variance < cutoff
    return(data[, apply(data, 2, function(x) var(x, na.rm=TRUE)) >= cutoff])
}
prune.correlated <- function(data, threshold) {
    #' Prune features that are highly correlated with each other
    correlations = correlation(data, threshold)
    return(data[, !colnames(data) %in% correlations[,1]])
}
```

Classifiers

```
classify.majority.train <- function(data, params) {
    #' Train a majority class classifier
    return(names(which.max(table(data$Class))))
}
classify.majority.execute <- function(classifier, datum) {
    #' Classify using majority class
    return(classifier)
}

classify.random.train <- function(data, params) {
    #' Train a random classifier
    return(c())
```

```
}
classify.random.execute <- function(classifier, datum) {
    #' Classify using random class
    return(sample(2, 1))
}

classify.lda.train <- function(data, params) {
    #' Train a linear discriminant analysis classifier
    return(lda(Class ~ ., data=data))
}
classify.lda.execute <- function(classifier, datum) {
    #' Classify using linear discriminant analysis
    return(predict(classifier, datum)$class)
}

classify.bayes.train <- function(data, params) {
    #' Train a naive bayes classifier
    return(naive_bayes(Class ~ ., data=data, laplace=1))
}
classify.bayes.execute <- function(classifier, datum) {
    #' Classify using naive bayes
    return(predict(classifier, datum))
}

classify.random_forest.train <- function(data, params) {
    #' Train a random forest classifier
    return(randomForest(Class ~ ., data=data))
}
classify.random_forest.execute <- function(classifier, datum) {
    #' Classify using random forest
    return(predict(classifier, datum))
}

classify.svm.train <- function(data, params) {
    #' Train a svm classifier
    return(svm(Class ~ ., data=data, gamma=params$gamma))
}
classify.svm.execute <- function(classifier, datum) {
    #' Classify using svm
    return(predict(classifier, datum))
}


# Combine classify.X.train and classify.X.execute into classify.X
# classify.X(training_data) returns a function that can be used to classify.
classifiers = c("majority", "random", "bayes", "lda", "random_forest", "svm")
for (classifier in classifiers) {
    train = match.fun(paste("classify.", classifier, ".train", sep=""))
    execute = match.fun(paste("classify.", classifier, ".execute", sep=""))

    name = paste("classify.", classifier, sep="")

    fun = local(function(training_data, parameters) {
```

```
        classifier = train(training_data, parameters)
        columns = names(training_data)
        columns = columns[columns != "Class"]
        return(function(testing_datum) {
            results = list()
            for (i in rownames(testing_datum)) {
                results[[i]] <- execute(classifier, testing_datum[i,columns])
            }
            return(results)
        })
    }, list2env(list(train=train, execute=execute)))
    assign(name, fun)
}

get_classifier = function(name, parameters=list()) {
    classifier = match.fun(paste("classify.", name, sep=""))
    return(local(function(training_data) {
        return(classifier(training_data, parameters))
    }, list2env(list(classifier=classifier, parameters=parameters))))
}
```

Cross-validation and testing

We ran k-fold validation on the following metrics:

- Accuracy
- Precision
- Recall
- F1 score
- AUC

```
metrics <- function(data, results) {
    contingency_table = table(factor(as.integer(results), c(1, 2)), factor(data$Class, c(1, 2)))

    tp = contingency_table[1,1] # true positive
    fp = contingency_table[1,2] # false positive
    fn = contingency_table[2,1] # false negative
    tn = contingency_table[2,2] # true negative
    p = tp + fn # positive
    n = tn + fp # negative
    pp = tp + fp # predicted positive
    pn = tn + fn # predicted negative

    shita = as.vector(data$Class, mode="numeric")
    shitb = as.vector(results, mode="numeric")
    return(list(
        "Accuracy"=((tp+tn)/(tp+tn+fp+fn)),
        "F1"=(2*tp/(2*tp + fp + fn)),
        "Precision"=(tp/pp),
        "Recall"=(tp/p),
        "AUC"=(roc(shita, shitb, quiet=TRUE)$auc)
    ))
```

```
}

k_fold_validate <- function(data, k, classifier) {
    #' Perform k-fold cross validation on a dataset
    data = data[sample(nrow(data)),]
    fold_size = round(nrow(data) / k)
    fold_indices = seq(1, nrow(data), fold_size)
    fold_indices = c(fold_indices, nrow(data)+1)
    fold_accuracies = array(0, c(k, 5))
    for (fold in 1:k) {
        test_indices = seq(fold_indices[fold], fold_indices[fold+1]-1)
        fold_data = data[-test_indices,]
        fold_test = data[test_indices,]
        fold_classifier = classifier(fold_data)
        fold_accuracies[fold,] = as.numeric(metrics(fold_test, fold_classifier(fold_test)))
    }
    return(fold_accuracies)
}
```

```
df$Class = as.factor(df$Class)
testing_data$Class = as.factor(testing_data$Class)
```

```
preprossesses = list(
    "LDA"=function(data) {
        preprossess = compose(impute.knn, prune.variance)

        datalda = preprossess(data)
        lda_classifier = classify.lda.train(datalda)
        data[,ncol(data) + 1] = predict(lda_classifier, datalda)$x[,1]
        testing_data[,ncol(testing_data) + 1] = predict(lda_classifier, testing_data)$x[,1]

        return(preprossess(data))
    },
    "PCA"=function(data) {
        preprossess = compose(impute.knn, prune.variance)
        data[,(ncol(data) + 1):(ncol(data) + 3)] = transform.pca(preprossess(data))$x[,1:3]
        return(preprossess(data))
    },
    "knn+var"=compose(impute.knn, prune.variance),
    "prep1"=compose(outliers.winsorize, prune.mcf, prune.variance, impute.knn),
    "prep2"=compose(impute.knn, outliers.winsorize, prune.mcf, prune.variance),
    "prep3"=compose(impute.knn, prune.mcf, prune.variance, outliers.winsorize),
    "prep4"=compose(impute.knn, prune.variance, outliers.winsorize)
)

clacifiers = list(
    "majority"=get_classifier("majority"),
    "random"=get_classifier("random"),
    "bayes"=get_classifier("bayes"),
    "lda"=get_classifier("lda"),
    "random_forest"=get_classifier("random_forest"),
    "svm"=get_classifier("svm", list(gamma=0.1))
)
```

```
results = array(0,
    c(length(clacifiers),length(preprossesses), 5),
    dimnames=list(
        names(clacifiers),
        names(preprossesses),
        c("Accuracy","F1","Precision","Recall","AUC")
    )
)

i=1
for (classifier in clacifiers) {
    j=1
    for (preprossess in preprossesses) {
        print(paste(names(clacifiers)[i], names(preprossesses)[j], sep=" / "))
        results[i,j,] = colMeans(k_fold_validate(preprossess(df), 5, classifier))
        j = j + 1
    }
    i = i + 1
}
print(results)
```

```
, , Accuracy

                   LDA       PCA   knn+var     prep1     prep2     prep3     prep4
majority      0.6662722 0.6674556 0.6662722 0.6662722 0.6662722 0.6662722 0.6662722
random        0.5183432 0.5053254 0.5017751 0.4887574 0.4710059 0.5147929 0.5183432
bayes         0.7526627 0.7455621 0.7396450 0.7360947 0.7408284 0.7420118 0.7502959
lda           0.8556213 0.8556213 0.8662722 0.8532544 0.8532544 0.8615385 0.8556213
random_forest 0.8792899 0.8769231 0.8745562 0.8686391 0.8733728 0.8733728 0.8804734
svm           0.8816568 0.8792899 0.8852071 0.8721893 0.8792899 0.8792899 0.8662722

, , F1

                   LDA       PCA   knn+var     prep1     prep2     prep3     prep4
majority      0.7995775 0.8002009 0.7993917 0.7995438 0.7992534 0.7996404 0.7996574
random        0.5938008 0.5788811 0.5723456 0.5539403 0.5447596 0.5867403 0.5879583
bayes         0.7852539 0.7752716 0.7710351 0.7674772 0.7722090 0.7746837 0.7822387
lda           0.8935090 0.8929773 0.9011534 0.8908631 0.8913139 0.8967991 0.8926158
random_forest 0.9109033 0.9086232 0.9074074 0.9035810 0.9071326 0.9057051 0.9118149
svm           0.9141361 0.9126602 0.9168114 0.9072357 0.9122279 0.9123226 0.9028204

, , Precision

                   LDA       PCA   knn+var     prep1     prep2     prep3     prep4
majority      0.6662722 0.6674556 0.6662722 0.6662722 0.6662722 0.6662722 0.6662722
random        0.6792740 0.6667601 0.6649574 0.6613434 0.6375190 0.6808722 0.6846878
bayes         0.9333294 0.9341836 0.9274315 0.9317007 0.9327365 0.9249671 0.9285232
lda           0.8764114 0.8838444 0.8832680 0.8821365 0.8838687 0.8868206 0.8838979
random_forest 0.8908420 0.8907236 0.8904661 0.8832091 0.8896705 0.8914590 0.8963595
svm           0.8886048 0.8833421 0.8890476 0.8799291 0.8795615 0.8806433 0.8736328

, , Recall

                   LDA       PCA   knn+var     prep1     prep2     prep3     prep4
```

```
majority      1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
random        0.5287772 0.5124994 0.5035652 0.4773145 0.4763796 0.5168643 0.5171131
bayes         0.6782743 0.6638535 0.6605020 0.6534251 0.6592458 0.6674902 0.6761123
lda           0.9116336 0.9050837 0.9202519 0.9001445 0.8991870 0.9073656 0.9023994
random_forest 0.9323053 0.9276150 0.9255639 0.9252566 0.9262283 0.9212216 0.9288681
svm           0.9417164 0.9451932 0.9469596 0.9383180 0.9479875 0.9464702 0.9347249

, , AUC

                   LDA       PCA   knn+var     prep1     prep2     prep3     prep4
majority      0.5000000 0.5000000 0.5000000 0.5000000 0.5000000 0.5000000 0.5000000
random        0.5300538 0.5310166 0.5503669 0.5400970 0.5357013 0.5468420 0.5166997
bayes         0.7898685 0.7867730 0.7784342 0.7773750 0.7803360 0.7771968 0.7866831
lda           0.8278887 0.8350527 0.8396338 0.8279863 0.8287297 0.8397150 0.8324609
random_forest 0.8527645 0.8488336 0.8493560 0.8407737 0.8478734 0.8485350 0.8561589
svm           0.8503806 0.8466836 0.8535373 0.8406499 0.8461658 0.8452618 0.8318942
```

The best classifier (according to the accuracy metric) turned out to be the random forest classifier with the 4th preprossessing preset (impute.knn ← prune.variance ← outliers.winsorize). We ran these settings on the test dataset to evaluate our true success.

```
classifier = classify.random_forest(preprossesses[["prep4"]](df))
results = c()
for (row in rownames(testing_data)) {
    test = testing_data[row, names(testing_data) != "Class"]
    results = c(results, tryCatch(as.integer(classifier(test)), error=function(e) "error"))
}
contingency_table = table(results, testing_data$Class)

print(contingency_table)
```

```
##
## results   1   2
##       1 122  21
##       2  13  53
```

```
print(sum(diag(contingency_table))/sum(contingency_table))
```

```
## [1] 0.8373206
```

The final achieved accuracy is 84%.