PCOS modeling

Erick Navarro & Timo Tolppa

2023-01-25

Contents

Creating the classificators

This report contains the details of the generation and comparison of several classifiers whose aim is to classify people with and without polycystic ovary syndrome (PCOS). This report is part of a larger project with a goal to develop and validate a model that takes easily collected variables, and tests its performance against models using variables obtained with increasingly invasive procedures. For a summary of the results and a clearer picture of the project, please read the Final_report.pdf file in this repository.

Setup

Load packages and data

Data splitting

After loading the data and packages, we will proceed to split the data into training and validation sets using a 70:30 random split.

```
data = data %>%
  column_to_rownames("id")

# Make the code reproducible
set.seed(504)

# Create the data split
train.index <- caret::createDataPartition(data$pcos, p = .7, list=FALSE)

# Define the training and validation data sets
train <- data[ train.index,]
valid <- data[-train.index,]

# Check the ratio of cases with and without the main outcome measure (i.e. PCOS diagnosis) in the train
table(train$pcos)
```

```
##
##  No Yes
## 255 124
```

```
table(train$pcos)[1]/ table(train$pcos)[2]
```

```
##       No
## 2.056452
```

```
# Check the ratio of cases with and without the main outcome measure (i.e. PCOS diagnosis) in the valid
table(valid$pcos)
```

```
##
##  No Yes
## 109  53
```

```
table(valid$pcos)[1]/ table(valid$pcos)[2]
```

```
##       No
## 2.056604
```

```
# Check the ratio of cases with and without the main outcome measure (i.e. PCOS diagnosis) in the origi
table(data$pcos)
```

```
##
##  No Yes
## 364 177
```

```
table(data$pcos)[1]/ table(data$pcos)[2]
```

```
##       No
## 2.056497
```

The ratio of cases with and without the main outcome measure (i.e. PCOS diagnosis) is similar between the original (2.0565), validation (2.0566) and training (2.0565) data.

Imputation on training set

Following the data splitting, we will impute missing values for the training set. As observed in the exploratory data analysis, the proportion of missing data is very low overall, which makes variables with missingness suitable for imputation.

```
# Explore misingness in training set
sapply(train, function(x) sum(is.na(x)))
```

```
##            pcos             age          weight          height             bmi
##               0               0               0               0               0
##     blood_group      pulse_rate              rr              hb           cycle
##               0               2               0               0               0
##    cycle_length marriage_status        pregnant no_of_abortions       i_betahcg
##               0               0               0               0               0
##      ii_betahcg             fsh              lh    fsh_lh_ratio             hip
##               0               0               1               1               0
##           waist waist_hip_ratio             tsh             amh             prl
##               0               0               0               2               0
##           vitd3             prg             rbs     weight_gain     hair_growth
##               1               0               0               0               0
##  skin_darkening       hair_loss         pimples       fast_food    reg_exercise
##               0               0               0               1               0
##     bp_systolic    bp_diastolic   follicle_no_l   follicle_no_r    avg_f_size_l
##               0               0               0               0               0
##    avg_f_size_r     endometrium
##               0               0
```

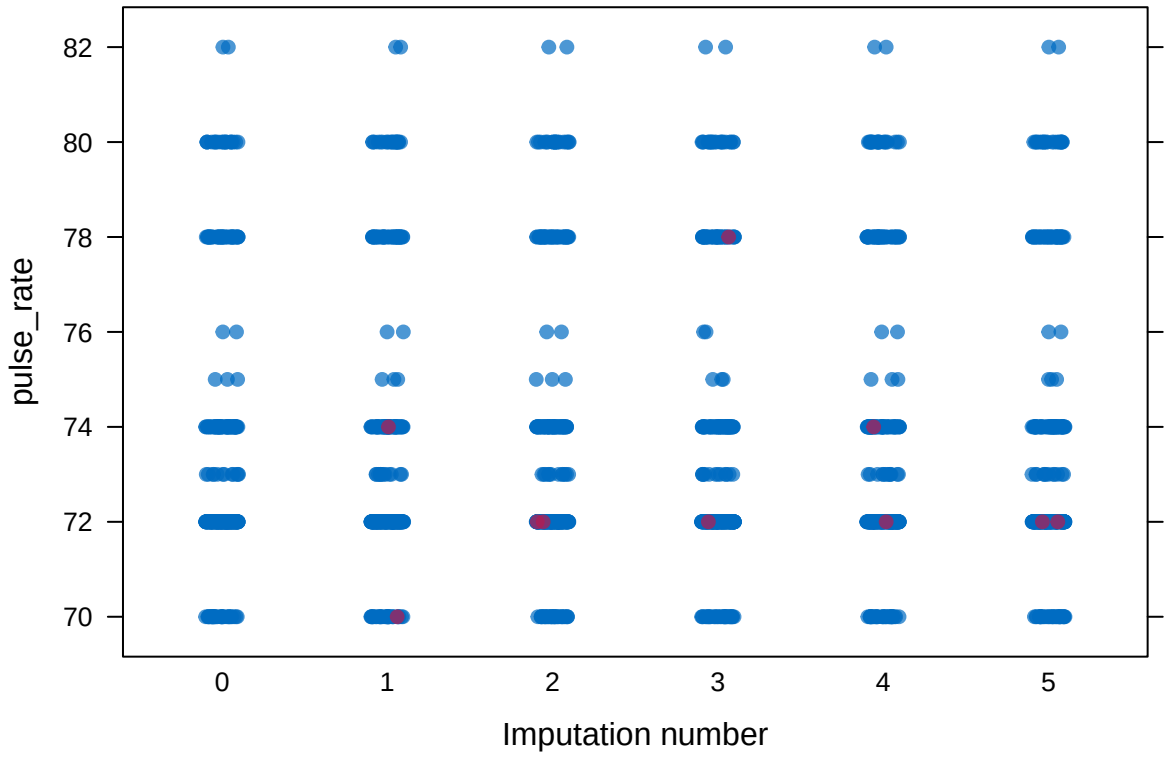
For imputation, we will use the mice R package, a widely used software to handle missing data. We will use the default options, which use the most appropiate methodology depending on the class of the data to be imputed, which are the following: "By default, the method uses pmm, predictive mean matching (numeric data) logreg, logistic regression imputation (binary data, factor with 2 levels) polyreg, polytomous regression imputation for un-ordered categorical data (factor > 2 levels) polr, proportional odds model for (ordered, > 2 levels)."

```
chained_train = mice::mice(train)
```

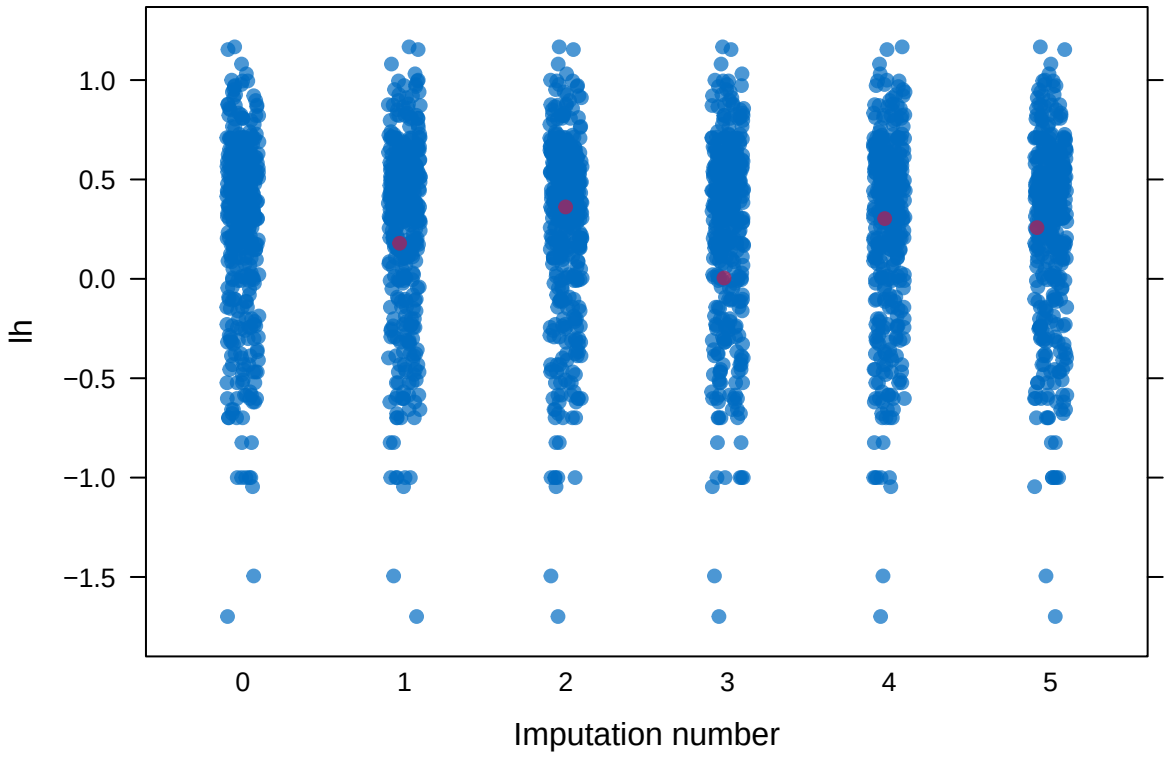
```
##
##  iter imp variable
##   1   1  pulse_rate  lh  fsh_lh_ratio  amh  vitd3  fast_food
##   1   2  pulse_rate  lh  fsh_lh_ratio  amh  vitd3  fast_food
##   1   3  pulse_rate  lh  fsh_lh_ratio  amh  vitd3  fast_food
##   1   4  pulse_rate  lh  fsh_lh_ratio  amh  vitd3  fast_food
```

```
##    1   5  pulse_rate  lh  fsh_lh_ratio  amh  vitd3  fast_food
##    2   1  pulse_rate  lh  fsh_lh_ratio  amh  vitd3  fast_food
##    2   2  pulse_rate  lh  fsh_lh_ratio  amh  vitd3  fast_food
##    2   3  pulse_rate  lh  fsh_lh_ratio  amh  vitd3  fast_food
##    2   4  pulse_rate  lh  fsh_lh_ratio  amh  vitd3  fast_food
##    2   5  pulse_rate  lh  fsh_lh_ratio  amh  vitd3  fast_food
##    3   1  pulse_rate  lh  fsh_lh_ratio  amh  vitd3  fast_food
##    3   2  pulse_rate  lh  fsh_lh_ratio  amh  vitd3  fast_food
##    3   3  pulse_rate  lh  fsh_lh_ratio  amh  vitd3  fast_food
##    3   4  pulse_rate  lh  fsh_lh_ratio  amh  vitd3  fast_food
##    3   5  pulse_rate  lh  fsh_lh_ratio  amh  vitd3  fast_food
##    4   1  pulse_rate  lh  fsh_lh_ratio  amh  vitd3  fast_food
##    4   2  pulse_rate  lh  fsh_lh_ratio  amh  vitd3  fast_food
##    4   3  pulse_rate  lh  fsh_lh_ratio  amh  vitd3  fast_food
##    4   4  pulse_rate  lh  fsh_lh_ratio  amh  vitd3  fast_food
##    4   5  pulse_rate  lh  fsh_lh_ratio  amh  vitd3  fast_food
##    5   1  pulse_rate  lh  fsh_lh_ratio  amh  vitd3  fast_food
##    5   2  pulse_rate  lh  fsh_lh_ratio  amh  vitd3  fast_food
##    5   3  pulse_rate  lh  fsh_lh_ratio  amh  vitd3  fast_food
##    5   4  pulse_rate  lh  fsh_lh_ratio  amh  vitd3  fast_food
##    5   5  pulse_rate  lh  fsh_lh_ratio  amh  vitd3  fast_food

## Warning: Number of logged events: 98
```

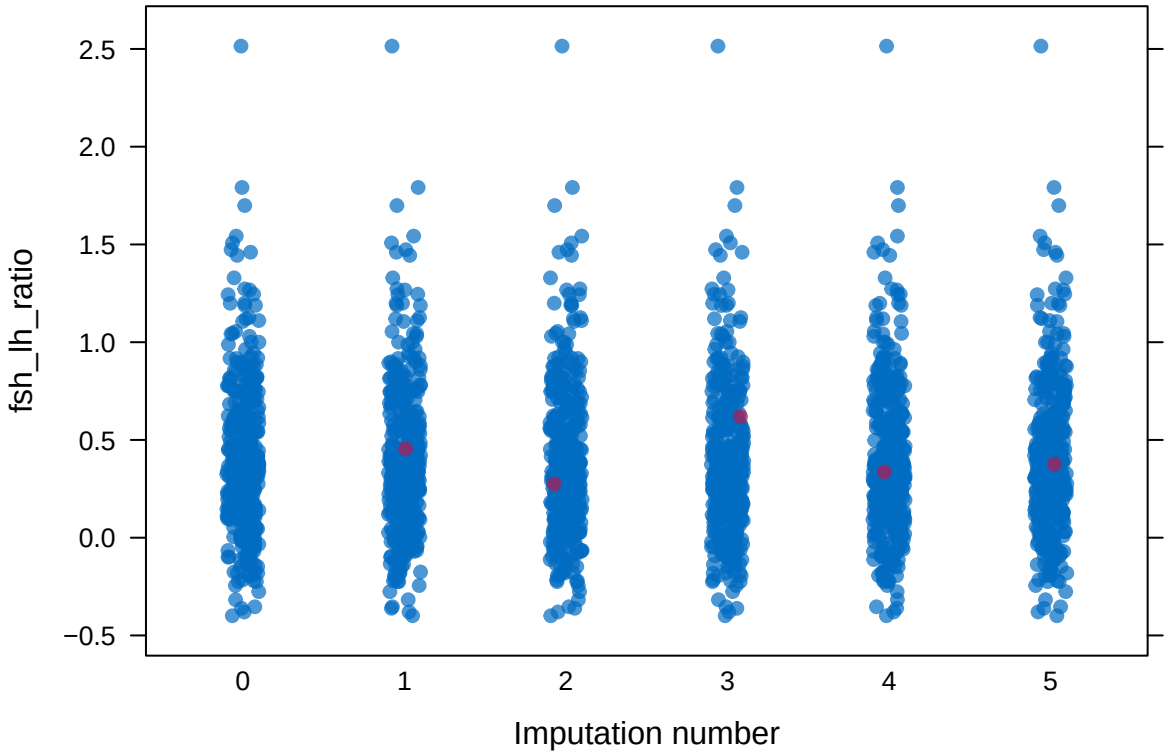
```r
# Explore the imputed data and check that the generated value are plausible
stripplot(chained_train, pulse_rate, pch = 19, xlab = "Imputation number")
```



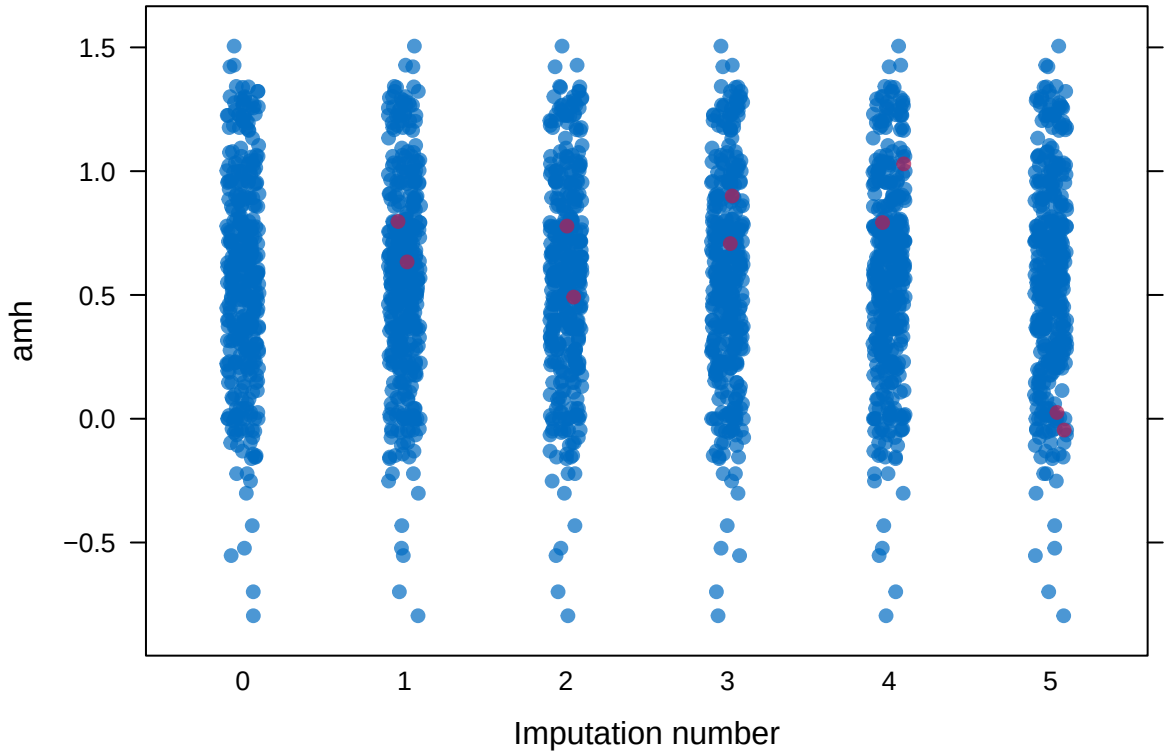
```
stripplot(chained_train, lh, pch = 19, xlab = "Imputation number")
```



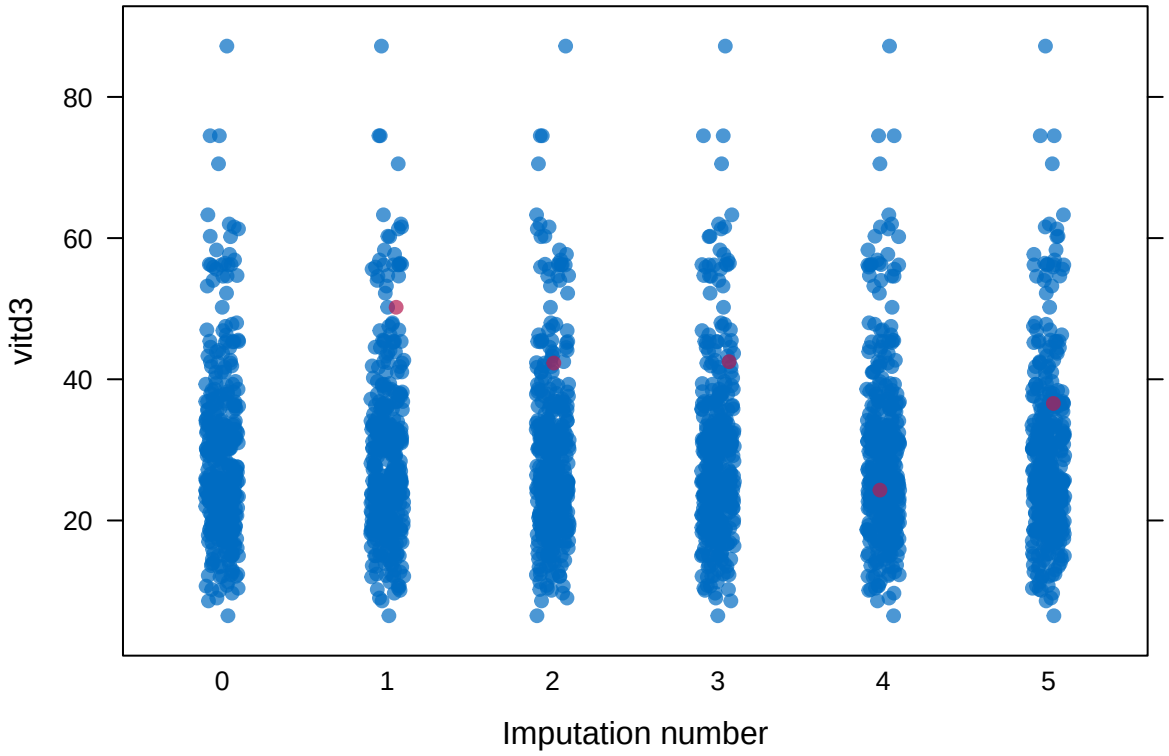

```
stripplot(chained_train, fsh_lh_ratio, pch = 19, xlab = "Imputation number")
```



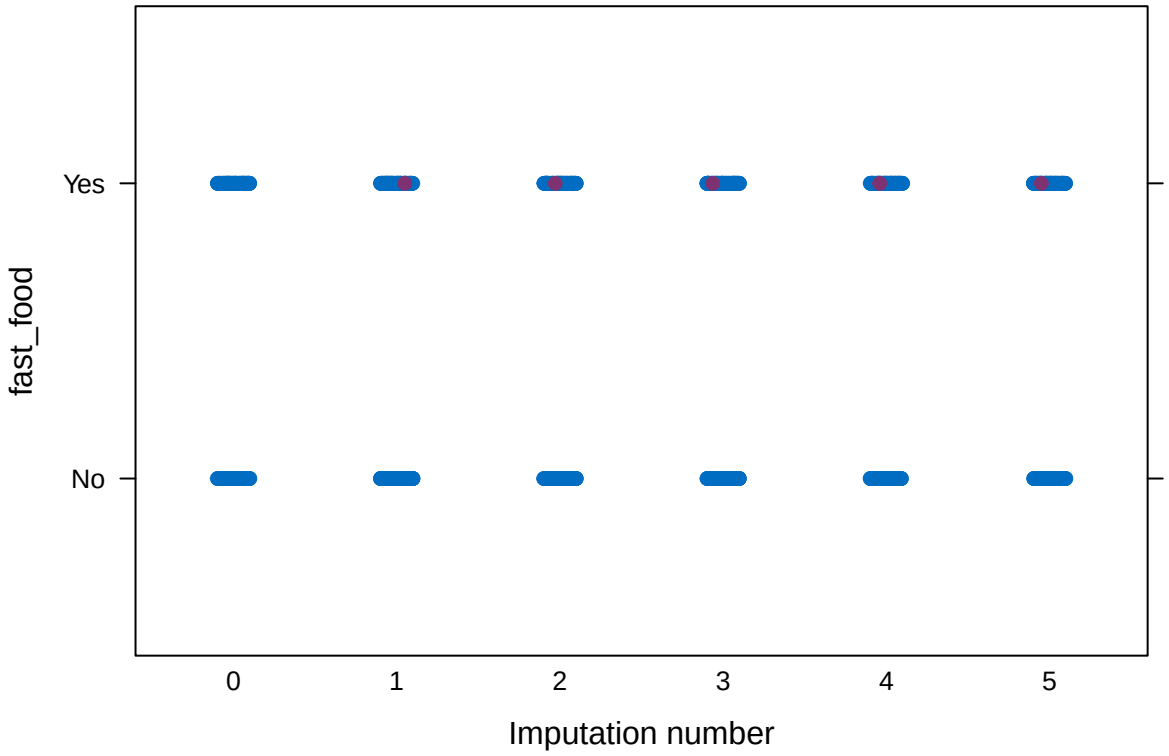

```
stripplot(chained_train, amh, pch = 19, xlab = "Imputation number")
```




```
stripplot(chained_train, vitd3, pch = 19, xlab = "Imputation number")
```




```
stripplot(chained_train, fast_food, pch = 19, xlab = "Imputation number")
```



The values appear to be plausible, therefore, we will proceed with extracting the imputed data

```
chained_train = complete(data=chained_train)
sapply(chained_train, function(x) sum(is.na(x)))
```

```
##            pcos             age          weight          height             bmi
##               0               0               0               0               0
##     blood_group      pulse_rate              rr              hb           cycle
##               0               0               0               0               0
##    cycle_length marriage_status        pregnant no_of_abortions       i_betahcg
##               0               0               0               0               0
##      ii_betahcg             fsh              lh    fsh_lh_ratio             hip
##               0               0               0               0               0
##           waist waist_hip_ratio             tsh             amh             prl
##               0               0               0               0               0
##           vitd3             prg             rbs     weight_gain     hair_growth
##               0               0               0               0               0
##  skin_darkening      hair_loss         pimples       fast_food    reg_exercise
##               0               0               0               0               0
##     bp_systolic    bp_diastolic   follicle_no_l   follicle_no_r    avg_f_size_l
##               0               0               0               0               0
##    avg_f_size_r     endometrium
##               0               0
```

Define the models

For the aim of this project, we will develop 4 different models with an increasing number of variables according to how easy they are to collect. Model 1 will contain only variables that are collected through patient history, model 2 will add variables collected through clinical examination, model 3 adds results of blood tests and model 4 will include all the relevant variables available in the data, including results from the transvaginal ultrasound.

It is worth mentioning that we removed some non-relevant variables from the dataset based on the scientific literature of the field. A more detailed information about this selection can be found in the final report available in this repository.

```
# Create dataset 1 using only variables obtained through through patient history
model1_vars = c("pcos","age","cycle","cycle_length",
                "no_of_abortions", "weight_gain", "hair_growth", "skin_darkening",
                "hair_loss","pimples","fast_food", "reg_exercise")

# Create dataset 2 using variables obtained through patient history and clinical examination
model2_vars  = c(model1_vars, "weight","height","bmi",
                "hip", "waist","waist_hip_ratio",
                "bp_systolic", "bp_diastolic")

# Create dataset 3 using variables obtained through patient history, clinical examination and blood tes
model3_vars = c(model2_vars, "fsh", "lh", "fsh_lh_ratio",
              "amh", "prl", "vitd3", "prg", "rbs")

# Create dataset 4 using variables obtained through patient history, clinical examination, blood tests
model4_vars = c(model3_vars, "follicle_no_l", "follicle_no_r" ,
              "avg_f_size_l", "avg_f_size_r", "endometrium")
```

Define cross-validation parameters

```
# Set the cross-validation parameters for all models
fitControl <- trainControl(
    method = 'cv',
    number = 5,
    savePredictions = 'final',
    classProbs = TRUE,
    summaryFunction=twoClassSummary)
```

Modeling

Logistic regression modeling

First, we will fit a logistic regression model with the features mentioned above.

```
# Set the seed to ensure reproducibility
set.seed(504)

#Since we have few samples, we will use bootstrapping instead of cross fold validation
models_logreg = list()
```

```
for (model in list(model1_vars, model2_vars, model3_vars, model4_vars)){
  cv_model = caret::train(
    pcos ~ .,
    data = chained_train %>%
      dplyr::select(all_of(model)),
    method = "glm",
    family = "binomial",
    trControl = fitControl)
  models_logreg = append(models_logreg, list(cv_model))
}
```

```
## Warning in train.default(x, y, weights = w, ...): The metric "Accuracy" was not
## in the result set. ROC will be used instead.

## Warning in train.default(x, y, weights = w, ...): The metric "Accuracy" was not
## in the result set. ROC will be used instead.

## Warning in train.default(x, y, weights = w, ...): The metric "Accuracy" was not
## in the result set. ROC will be used instead.

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning in train.default(x, y, weights = w, ...): The metric "Accuracy" was not
## in the result set. ROC will be used instead.

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
names(models_logreg) = c("model1","model2", "model3", "model4")

models_logreg
```

```
## $model1
## Generalized Linear Model
##
## 379 samples
##  11 predictor
##   2 classes: 'No', 'Yes'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 304, 303, 303, 303, 303
```

```
## Resampling results:
##
##   ROC        Sens       Spec
##   0.9050523  0.9019608  0.718
##
##
## $model2
## Generalized Linear Model
##
## 379 samples
##  19 predictor
##   2 classes: 'No', 'Yes'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 303, 304, 303, 303, 303
## Resampling results:
##
##   ROC        Sens       Spec
##   0.8943791  0.9019608  0.6946667
##
##
## $model3
## Generalized Linear Model
##
## 379 samples
##  27 predictor
##   2 classes: 'No', 'Yes'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 303, 304, 303, 303, 303
## Resampling results:
##
##   ROC        Sens       Spec
##   0.8718627  0.8941176  0.6943333
##
##
## $model4
## Generalized Linear Model
##
## 379 samples
##  32 predictor
##   2 classes: 'No', 'Yes'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 303, 304, 303, 303, 303
## Resampling results:
##
##   ROC        Sens       Spec
##   0.9350327  0.9137255  0.815
```

Elastic net regression

We will also fit an elastic net model, which is a method that performs feature selection and remove redundant terms in the models.

```
models_EN = list()

for (model in list(model1_vars, model2_vars, model3_vars, model4_vars)){
  en_model = caret::train(
    pcos ~ .,
    trControl = fitControl,
    data = chained_train %>%
      dplyr::select(all_of(model)),
    method = "glmnet",
    tuneGrid = expand.grid(alpha = seq(0.1,.2,by = 0.05),
                           lambda = seq(0.05,0.3,by = 0.05)),
    verbose = FALSE,
    metric="ROC")
  models_EN = append(models_EN, list(en_model))
}

names(models_EN) = c("model1","model2", "model3", "model4")
models_EN
```

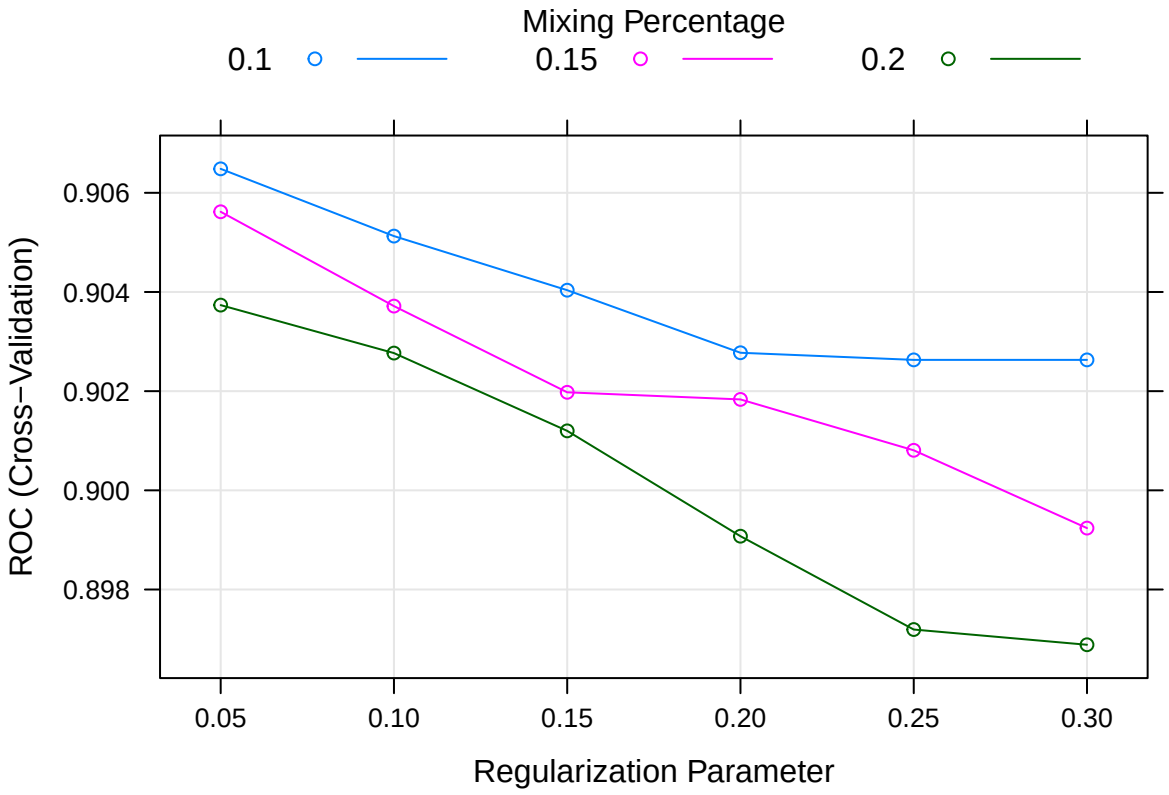
```
## $model1
## glmnet
##
## 379 samples
##  11 predictor
##   2 classes: 'No', 'Yes'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 303, 304, 303, 303, 303
## Resampling results across tuning parameters:
##
##   alpha  lambda  ROC        Sens       Spec
##   0.10   0.05    0.9064837  0.9254902  0.7253333
##   0.10   0.10    0.9051275  0.9372549  0.6933333
##   0.10   0.15    0.9040359  0.9490196  0.6613333
##   0.10   0.20    0.9027745  0.9529412  0.6373333
##   0.10   0.25    0.9026307  0.9568627  0.6053333
##   0.10   0.30    0.9026307  0.9568627  0.5650000
##   0.15   0.05    0.9056176  0.9254902  0.7253333
##   0.15   0.10    0.9037157  0.9411765  0.6933333
##   0.15   0.15    0.9019771  0.9490196  0.6533333
##   0.15   0.20    0.9018333  0.9568627  0.6133333
##   0.15   0.25    0.9008072  0.9568627  0.5730000
##   0.15   0.30    0.8992386  0.9568627  0.5086667
##   0.20   0.05    0.9037353  0.9215686  0.7253333
##   0.20   0.10    0.9027680  0.9411765  0.6773333
##   0.20   0.15    0.9011993  0.9529412  0.6533333
##   0.20   0.20    0.8990752  0.9568627  0.5810000
##   0.20   0.25    0.8971928  0.9568627  0.5246667
```

```
##   0.20   0.30    0.8968856  0.9568627  0.4446667
##
## ROC was used to select the optimal model using the largest value.
## The final values used for the model were alpha = 0.1 and lambda = 0.05.
##
## $model2
## glmnet
##
## 379 samples
##  19 predictor
##   2 classes: 'No', 'Yes'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 304, 303, 303, 303, 303
## Resampling results across tuning parameters:
##
##   alpha  lambda  ROC        Sens       Spec
##   0.10   0.05    0.9011961  0.9215686  0.6930000
##   0.10   0.10    0.9031307  0.9372549  0.6770000
##   0.10   0.15    0.9031438  0.9490196  0.6610000
##   0.10   0.20    0.9033137  0.9490196  0.6366667
##   0.10   0.25    0.9041176  0.9529412  0.5876667
##   0.10   0.30    0.9044379  0.9529412  0.5636667
##   0.15   0.05    0.9012288  0.9254902  0.6933333
##   0.15   0.10    0.9044052  0.9411765  0.6853333
##   0.15   0.15    0.9045882  0.9450980  0.6526667
##   0.15   0.20    0.9050850  0.9529412  0.5880000
##   0.15   0.25    0.9044575  0.9529412  0.5636667
##   0.15   0.30    0.9034118  0.9607843  0.5316667
##   0.20   0.05    0.9015556  0.9294118  0.7013333
##   0.20   0.10    0.9039477  0.9372549  0.6770000
##   0.20   0.15    0.9052614  0.9411765  0.6363333
##   0.20   0.20    0.9042222  0.9529412  0.5956667
##   0.20   0.25    0.9026601  0.9568627  0.5556667
##   0.20   0.30    0.9014706  0.9647059  0.4590000
##
## ROC was used to select the optimal model using the largest value.
## The final values used for the model were alpha = 0.2 and lambda = 0.15.
##
## $model3
## glmnet
##
## 379 samples
##  27 predictor
##   2 classes: 'No', 'Yes'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 304, 303, 303, 303, 303
## Resampling results across tuning parameters:
##
##   alpha  lambda  ROC        Sens       Spec
##   0.10   0.05    0.8999869  0.9058824  0.6930000
```

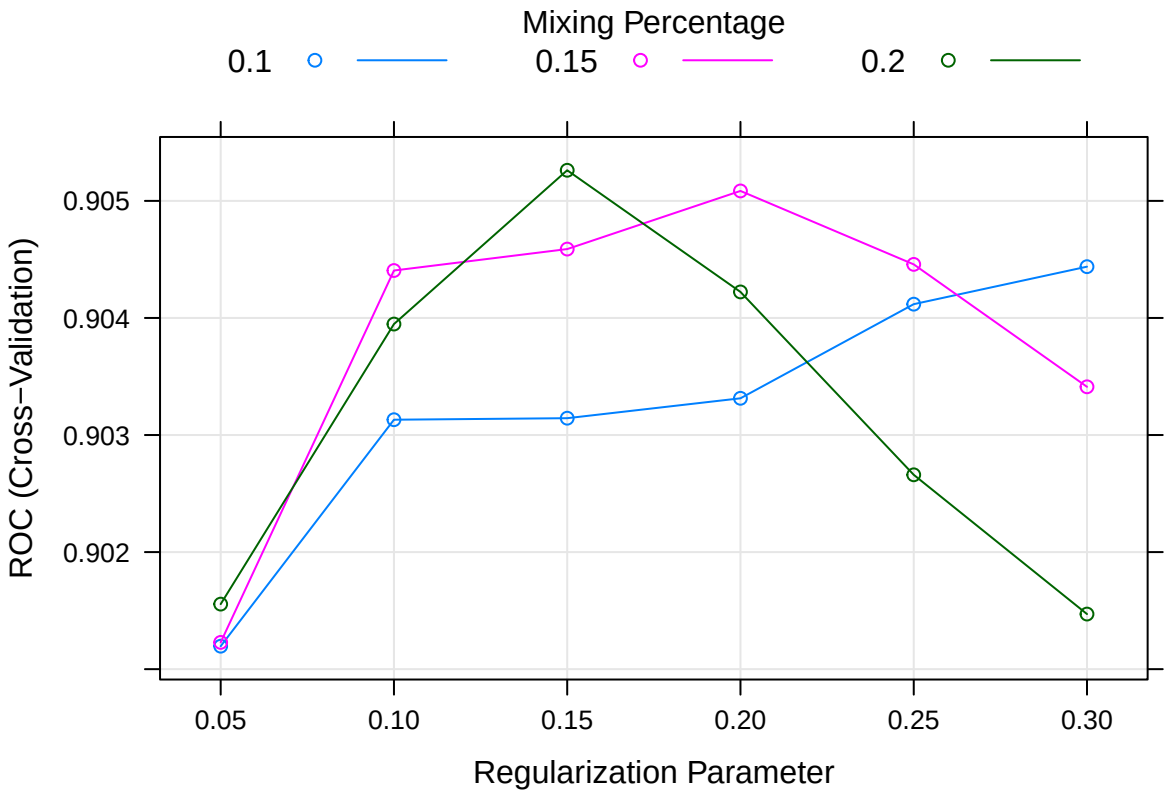
```
##   0.10   0.10    0.9050327  0.9176471  0.6930000
##   0.10   0.15    0.9045882  0.9294118  0.6686667
##   0.10   0.20    0.9049150  0.9372549  0.6366667
##   0.10   0.25    0.9041111  0.9450980  0.6206667
##   0.10   0.30    0.9040980  0.9529412  0.5806667
##   0.15   0.05    0.9015686  0.9058824  0.6930000
##   0.15   0.10    0.9049085  0.9176471  0.6930000
##   0.15   0.15    0.9038105  0.9294118  0.6526667
##   0.15   0.20    0.9033203  0.9411765  0.6286667
##   0.15   0.25    0.9032941  0.9529412  0.5806667
##   0.15   0.30    0.9017190  0.9529412  0.5323333
##   0.20   0.05    0.9031699  0.9098039  0.6930000
##   0.20   0.10    0.9046144  0.9215686  0.6930000
##   0.20   0.15    0.9025294  0.9333333  0.6366667
##   0.20   0.20    0.9025359  0.9529412  0.6126667
##   0.20   0.25    0.9009542  0.9529412  0.5483333
##   0.20   0.30    0.9011111  0.9568627  0.4603333
##
## ROC was used to select the optimal model using the largest value.
## The final values used for the model were alpha = 0.1 and lambda = 0.1.
##
## $model4
## glmnet
##
## 379 samples
##  32 predictor
##   2 classes: 'No', 'Yes'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 304, 303, 303, 303, 303
## Resampling results across tuning parameters:
##
##   alpha  lambda  ROC        Sens       Spec
##   0.10   0.05    0.9597255  0.9529412  0.7803333
##   0.10   0.10    0.9595621  0.9607843  0.7403333
##   0.10   0.15    0.9592288  0.9647059  0.7163333
##   0.10   0.20    0.9587451  0.9647059  0.7083333
##   0.10   0.25    0.9590784  0.9725490  0.7000000
##   0.10   0.30    0.9590784  0.9803922  0.6920000
##   0.15   0.05    0.9597320  0.9490196  0.7723333
##   0.15   0.10    0.9597190  0.9607843  0.7323333
##   0.15   0.15    0.9597124  0.9647059  0.7163333
##   0.15   0.20    0.9582745  0.9686275  0.7000000
##   0.15   0.25    0.9573203  0.9843137  0.6920000
##   0.15   0.30    0.9573268  0.9843137  0.6513333
##   0.20   0.05    0.9587647  0.9490196  0.7723333
##   0.20   0.10    0.9595556  0.9568627  0.7403333
##   0.20   0.15    0.9585948  0.9647059  0.7163333
##   0.20   0.20    0.9579673  0.9764706  0.7000000
##   0.20   0.25    0.9582876  0.9843137  0.6430000
##   0.20   0.30    0.9594052  0.9882353  0.6270000
##
## ROC was used to select the optimal model using the largest value.
```

```
## The final values used for the model were alpha = 0.15 and lambda = 0.05.
```

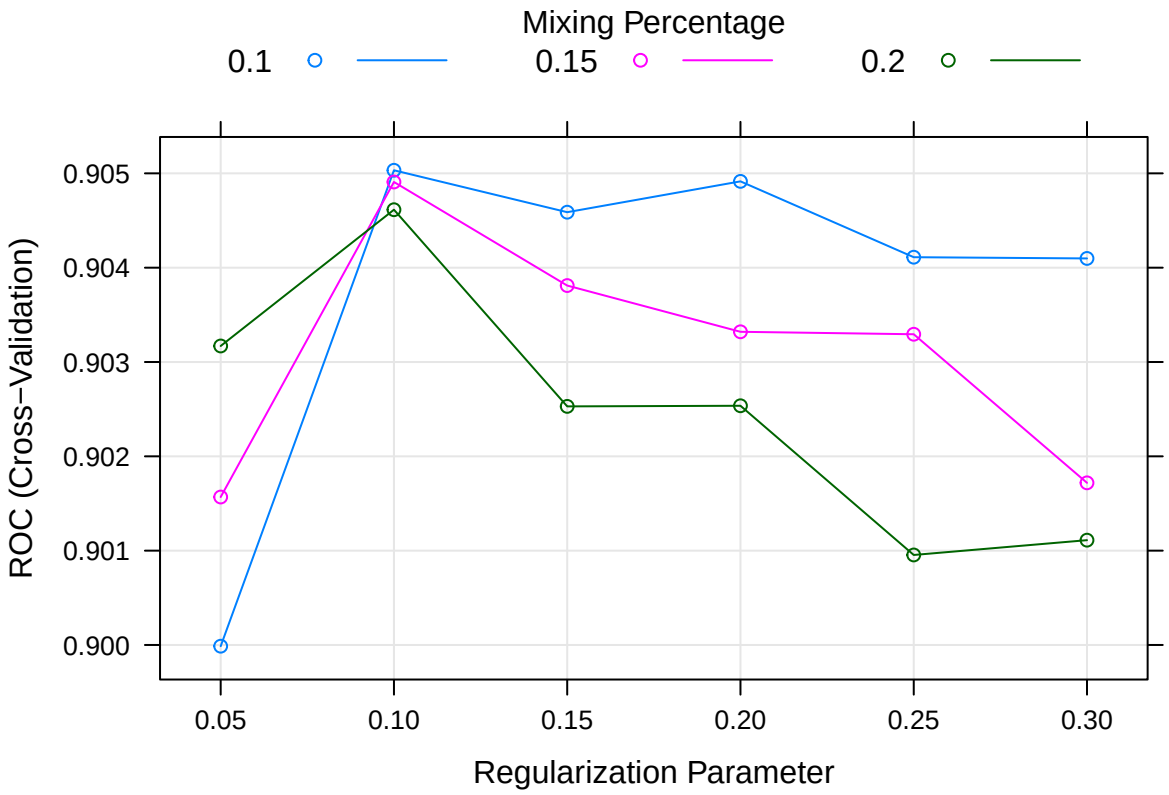
```
#Plot parameter tuning of model 1
plot(models_EN[[1]])
```



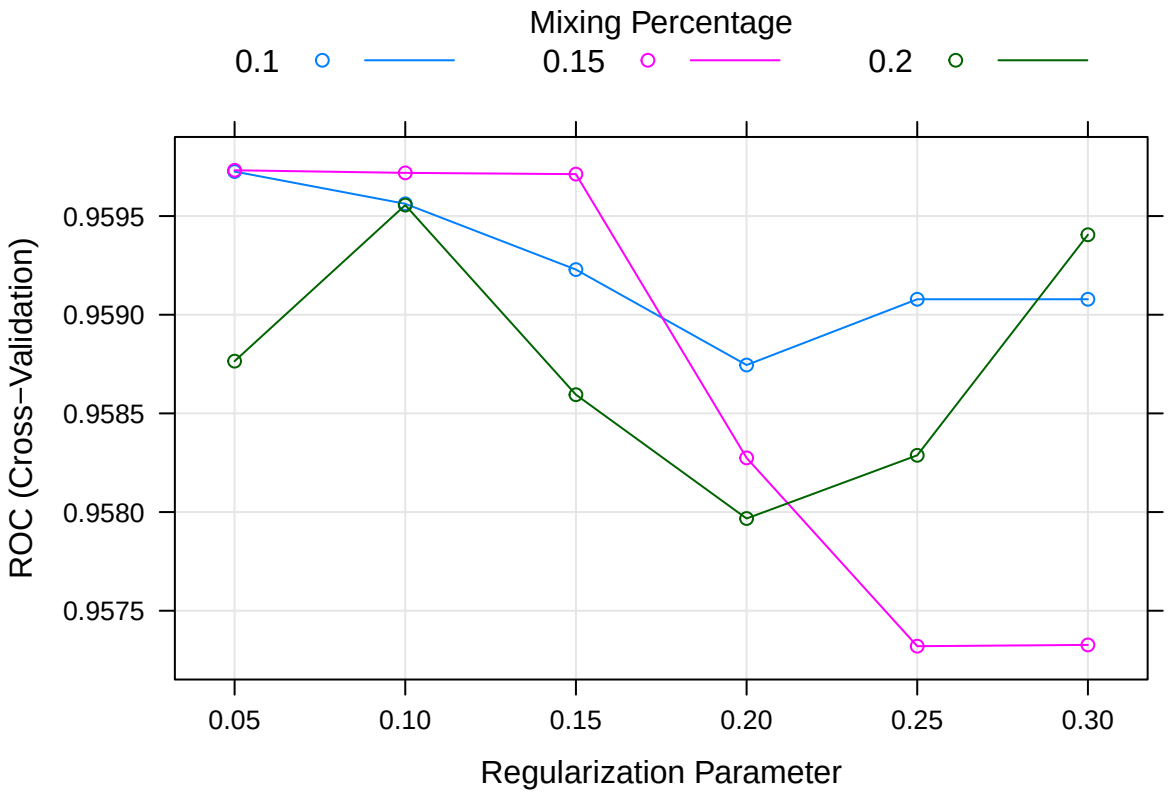

```
#Plot parameter tuning of model 2
plot(models_EN[[2]])
```




```
#Plot parameter tuning of model 3
plot(models_EN[[3]])
```




```
#Plot parameter tuning of model 4
plot(models_EN[[4]])
```



Random Forest

Finally, we will fit a model of the CART family.

```
# Set the seed to ensure reproducibility
set.seed(504)

# Train the random forest model for all four sets of data
models_rf = list()

for (model in list(model1_vars, model2_vars, model3_vars, model4_vars)){
  rf_model = caret::train(
    pcos ~ .,
    data = chained_train %>%
      dplyr::select(all_of(model)),
    method = "ranger",
    num.trees = 500,
    tuneLength = 5,
    metric = "Sens",
    trControl = fitControl,
    importance="impurity")
  models_rf = append(models_rf, list(rf_model))
}

names(models_rf) = c("model1","model2", "model3", "model4")
```

```
models_rf
```

```
## $model1
## Random Forest
##
## 379 samples
##  11 predictor
##   2 classes: 'No', 'Yes'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 304, 303, 303, 303, 303
## Resampling results across tuning parameters:
##
##   mtry  splitrule   ROC        Sens       Spec
##    2    gini        0.9067255  0.9098039  0.7020000
##    2    extratrees  0.9043922  0.9137255  0.7260000
##    4    gini        0.8963203  0.8901961  0.7100000
##    4    extratrees  0.8933203  0.8862745  0.7176667
##    6    gini        0.8884575  0.8745098  0.7263333
##    6    extratrees  0.8869673  0.8823529  0.7256667
##    8    gini        0.8834641  0.8627451  0.7180000
##    8    extratrees  0.8850458  0.8627451  0.7340000
##   11    gini        0.8799837  0.8549020  0.7183333
##   11    extratrees  0.8803072  0.8588235  0.7173333
##
## Tuning parameter 'min.node.size' was held constant at a value of 1
## Sens was used to select the optimal model using the largest value.
## The final values used for the model were mtry = 2, splitrule = extratrees
##  and min.node.size = 1.
##
## $model2
## Random Forest
##
## 379 samples
##  19 predictor
##   2 classes: 'No', 'Yes'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 303, 304, 303, 303, 303
## Resampling results across tuning parameters:
##
##   mtry  splitrule   ROC        Sens       Spec
##    2    gini        0.9013791  0.9215686  0.6366667
##    2    extratrees  0.9013203  0.9176471  0.7016667
##    6    gini        0.8963399  0.8901961  0.6850000
##    6    extratrees  0.8971569  0.8980392  0.7176667
##   10    gini        0.8883954  0.8862745  0.7010000
##   10    extratrees  0.8913497  0.8823529  0.7420000
##   14    gini        0.8824967  0.8862745  0.6853333
##   14    extratrees  0.8836863  0.8784314  0.7340000
##   19    gini        0.8795000  0.8823529  0.6770000
```

```
##   19    extratrees  0.8864706  0.8823529  0.7423333
##
## Tuning parameter 'min.node.size' was held constant at a value of 1
## Sens was used to select the optimal model using the largest value.
## The final values used for the model were mtry = 2, splitrule = gini
##  and min.node.size = 1.
##
## $model3
## Random Forest
##
## 379 samples
##  27 predictor
##   2 classes: 'No', 'Yes'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 303, 303, 303, 303, 304
## Resampling results across tuning parameters:
##
##   mtry  splitrule   ROC        Sens       Spec
##    2    gini        0.9082157  0.9294118  0.6053333
##    2    extratrees  0.9030458  0.9333333  0.6703333
##    8    gini        0.8923301  0.8980392  0.6943333
##    8    extratrees  0.8944020  0.8980392  0.6783333
##   14    gini        0.8788856  0.8862745  0.7103333
##   14    extratrees  0.8931830  0.8862745  0.7100000
##   20    gini        0.8764902  0.8862745  0.7023333
##   20    extratrees  0.8860131  0.8862745  0.6860000
##   27    gini        0.8673660  0.8627451  0.6863333
##   27    extratrees  0.8798007  0.8823529  0.6696667
##
## Tuning parameter 'min.node.size' was held constant at a value of 1
## Sens was used to select the optimal model using the largest value.
## The final values used for the model were mtry = 2, splitrule = extratrees
##  and min.node.size = 1.
##
## $model4
## Random Forest
##
## 379 samples
##  32 predictor
##   2 classes: 'No', 'Yes'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 303, 303, 303, 303, 304
## Resampling results across tuning parameters:
##
##   mtry  splitrule   ROC        Sens       Spec
##    2    gini        0.9608105  0.9686275  0.7433333
##    2    extratrees  0.9599477  0.9490196  0.7026667
##    9    gini        0.9513007  0.9490196  0.8150000
##    9    extratrees  0.9595131  0.9333333  0.7913333
##   17    gini        0.9432712  0.9372549  0.8150000
```

```
##   17    extratrees  0.9565098  0.9294118  0.7990000
##   24    gini        0.9419216  0.9372549  0.8070000
##   24    extratrees  0.9556340  0.9215686  0.8070000
##   32    gini        0.9413399  0.9294118  0.7906667
##   32    extratrees  0.9578758  0.9254902  0.8310000
##
## Tuning parameter 'min.node.size' was held constant at a value of 1
## Sens was used to select the optimal model using the largest value.
## The final values used for the model were mtry = 2, splitrule = gini
##  and min.node.size = 1.
```

The number of trees was kept at the default of 500, as increases in the number of trees to 1000, 5000 and 10000 improved model performance by less than 1% for ROC, sensitivity and specificity. However, the processing time was increased significantly and thus the default value was not changed. Main tuning parameters were determined using the inbuilt 'tuneLength' parameter by trying different default grid values for the main parameters, which were optimized to 'sensitivity' as per the aims of this project. Further detail and justification for the metrics used in this project can be found in the Final Report in this repository.

Model comparison

Test the performance in the validation set

For this step, we will first fit the models in the validation data set to get the performance metrics.

We will concatenate all the models and estimate their performance in the validation set.

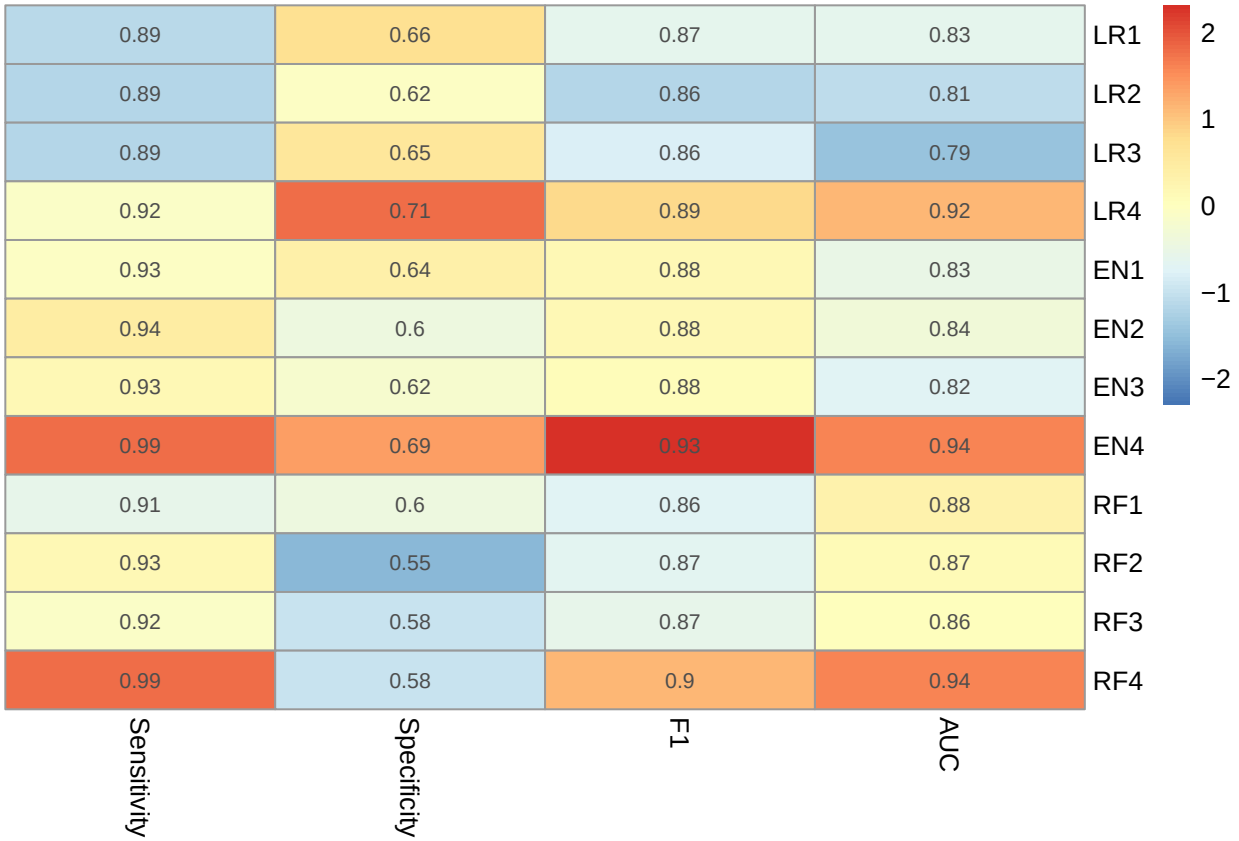
```
#Get the outcomes in the validation set.
outcomes = list(valid %>%
                  dplyr::select(all_of(model1_vars)) %>%
                  drop_na() %>%
                  pull(pcos),
                valid %>%
                  dplyr::select(all_of(model2_vars)) %>%
                  drop_na() %>%
                  pull(pcos),
                valid %>%
                  dplyr::select(all_of(model3_vars)) %>%
                  drop_na() %>%
                  pull(pcos),
                valid %>%
                  dplyr::select(all_of(model4_vars)) %>%
                  drop_na() %>%
                  pull(pcos))

models_aggregated = c(models_logreg, models_EN, models_rf)

#Generate prediction metrics
pred_models = models_aggregated %>%
  purrr::map(\(x) predict(object = x, newdata = valid, type = "raw")) %>%
  purrr::map2(c(rep(outcomes,3)), \(x,y) caret::confusionMatrix(x,y))

AUC = models_aggregated %>%
  purrr::map(\(x) predict(object = x, newdata = valid, type = "prob")) %>%
  purrr::map2(c(rep(outcomes,3)), \(x,y) pROC::auc(y, x$Yes))
```

```
## Setting levels: control = No, case = Yes
## Setting direction: controls < cases
## Setting levels: control = No, case = Yes
## Setting direction: controls < cases
## Setting levels: control = No, case = Yes
## Setting direction: controls < cases
## Setting levels: control = No, case = Yes
## Setting direction: controls < cases
## Setting levels: control = No, case = Yes
## Setting direction: controls < cases
## Setting levels: control = No, case = Yes
## Setting direction: controls < cases
## Setting levels: control = No, case = Yes
## Setting direction: controls < cases
## Setting levels: control = No, case = Yes
## Setting direction: controls < cases
## Setting levels: control = No, case = Yes
## Setting direction: controls < cases
## Setting levels: control = No, case = Yes
## Setting direction: controls < cases
## Setting levels: control = No, case = Yes
## Setting direction: controls < cases
## Setting levels: control = No, case = Yes
## Setting direction: controls < cases
```

```r
#Create data frame with the metrics
(metrics = tibble(Method = c(rep("LR",4),
                             rep("EN",4),
                             rep("RF",4)),
                  Model = c(rep(c("1", "2","3", "4"), 3)),
                  Specificity = map(pred_models, "byClass") %>%
                    map_dbl("Specificity"),
                  Sensitivity = map(pred_models, "byClass") %>%
                    map_dbl("Sensitivity"),
                  F1 = map(pred_models, "byClass") %>%
                    map_dbl("F1"),
                  AUC = unlist(AUC)
                  ))
```

```
## # A tibble: 12 x 6
##    Method Model Specificity Sensitivity    F1   AUC
##    <chr>  <chr>       <dbl>       <dbl> <dbl> <dbl>
##  1 LR     1           0.660       0.890 0.866 0.828
##  2 LR     2           0.623       0.888 0.856 0.806
##  3 LR     3           0.654       0.887 0.862 0.785
##  4 LR     4           0.712       0.925 0.895 0.918
##  5 EN     1           0.642       0.927 0.882 0.832
##  6 EN     2           0.604       0.944 0.882 0.842
##  7 EN     3           0.615       0.934 0.88  0.824
##  8 EN     4           0.692       0.991 0.925 0.942
##  9 RF     1           0.604       0.908 0.865 0.875
## 10 RF     2           0.547       0.935 0.866 0.867
## 11 RF     3           0.577       0.925 0.867 0.861
## 12 RF     4           0.577       0.991 0.901 0.942
```

```r
pheatmap(metrics %>%
           unite(name, Method, Model, sep = "") %>%
           column_to_rownames(var= "name") %>%
           select(c(Sensitivity, Specificity, F1, AUC)),
         cluster_rows = F,
         cluster_cols = F,
         scale = "column",
         display_numbers = round(metrics %>%
           select(c(Sensitivity, Specificity, F1, AUC)),2)
         )
```



In the plot above, we can observe that models 1,2 and 3 have a very similar performance in all of the methods applied. Model 4 has consistently a better performance than 1/2/3, which suggests that ultrasound variables are informative. However, getting that information usually requires a more specialized medical equipement, which is not that easily accessible.

If we look at Sensitivity and F1, we can see that model 1 usually outperforms models 2 and 3. Since the aim of our project is to create a model that balances performance and easily collected variables, model 1 is the best option.

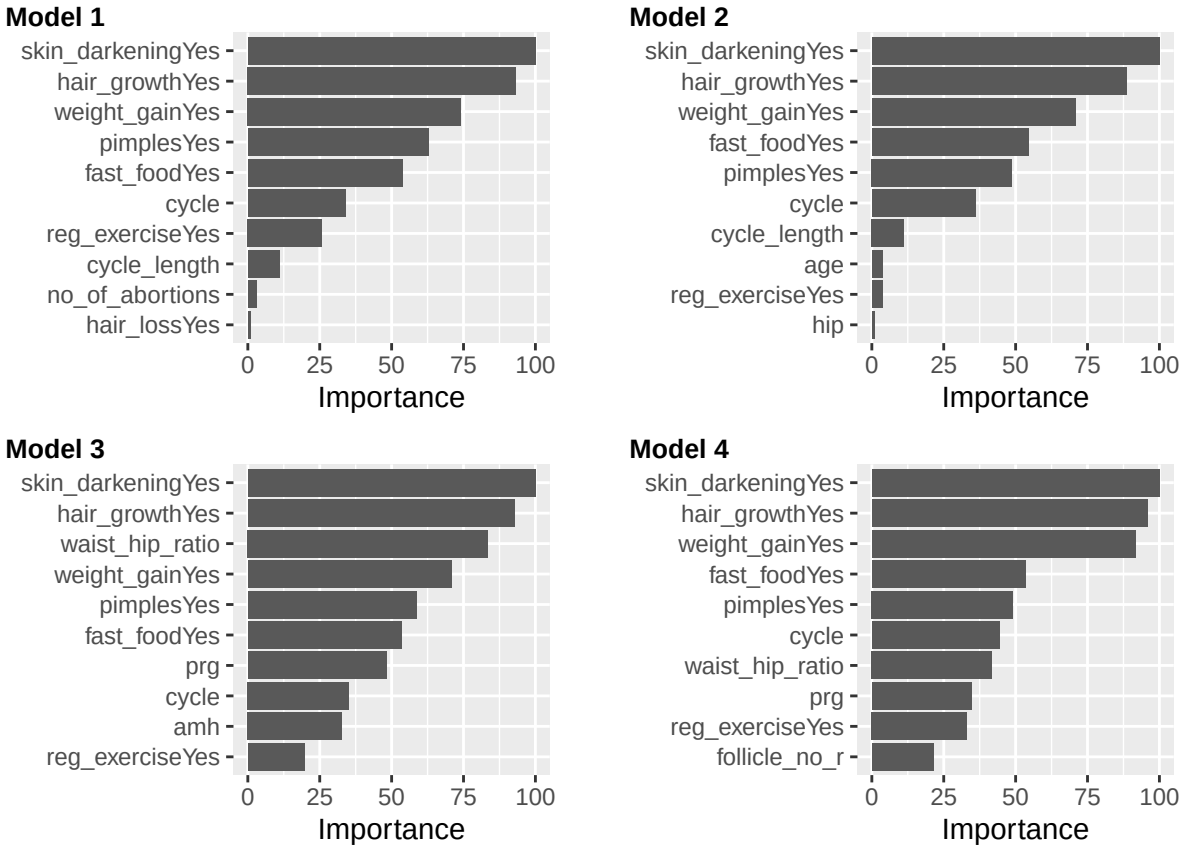
Furthermore, out of all of the models 1, the method that creates the best model is Elastic Net based on the metrics. Therefore, we will propose it as the final model in our project.

Explore EN model 1

Variable importance

Next, we will explore the most important variables in the elastic net models

```
#Create a combined graph for all variable importance graphs
plot_grid(vip::vip(models_EN[[1]]),
          vip::vip(models_EN[[2]]),
          vip::vip(models_EN[[3]]),
          vip::vip(models_EN[[4]]), labels = c('Model 1', 'Model 2', 'Model 3', 'Model 4'), label_size =
```



Effect of class imbalance on the best model (if time allows)