Part-2

Florencia Luque and Seyed Amirhossein Mosaddad

2024-10-18

Part 2

```r
library(caret)
library(caretEnsemble)
library(h2o)
library(tidymodels)
library(ROCR)
library(ConfusionTableR)
library(dplyr)
library(pander)
library(pROC)
library(xgboost)
library(e1071)
library(gridExtra)
library(ggplot2)
```

Introduction

This dataset is a data obtain from *kaggle* and is used to predict if a pacient will probably get a stroke based on characteristic of them like gender, age, bmi, glucose levels.

The stroke variable have a 4.8% of people have had one. We want to check the distributions of the variables and possibles explanations of which variable can make an impact to get a stroke before creating a model to proved or been proved wrong about it.

The data have the follow variables:

1) id: unique identifier
2) gender: "Male", "Female" or "Other"
3) age: age of the patient
4) hypertension: 0 if the patient doesn't have hypertension, 1 if the patient has hypertension
5) heart_disease: 0 if the patient doesn't have any heart diseases, 1 if the patient has a heart disease
6) ever_married: "No" or "Yes"
7) work_type: "children", "Govt_jov", "Never_worked", "Private" or "Self-employed"
8) Residence_type: "Rural" or "Urban"
9) avg_glucose_level: average glucose level in blood
10) bmi: body mass index

11) smoking_status: “formerly smoked”, “never smoked”, “smokes” or “Unknown”
12) stroke: 1 if the patient had a stroke or 0 if not

```
path = "/Users/soroush/Desktop/UC3M/courses/R/minitask1/Stroke_data"
stroke_data = read.csv("healthcare-dataset-stroke-data.csv",header = TRUE)

stroke_data = na.omit(stroke_data)
stroke_data = stroke_data[stroke_data$gender!="Other",]
stroke_data$gender = as.factor(stroke_data$gender)
stroke_data$hypertension = as.factor(stroke_data$hypertension)
stroke_data$heart_disease = as.factor(stroke_data$heart_disease)
stroke_data$work_type = as.factor(stroke_data$work_type)
stroke_data$Residence_type = as.factor(stroke_data$Residence_type)
stroke_data$ever_married = as.factor(stroke_data$ever_married)
stroke_data$smoking_status = as.factor(stroke_data$smoking_status)
stroke_data$stroke = as.factor(stroke_data$stroke)
stroke_data$bmi =as.numeric(stroke_data$bmi)

frec_table <- cut(stroke_data$bmi, breaks = c(0, 18.5, 24.9, 29.9, 34.9, Inf),
                labels = c("Underweight", "Normal", "Overweight", "Obese", "Extremely Obese"))
stroke_data$cat_weight = frec_table

glucose_categories <- cut(stroke_data$avg_glucose_level,
                          breaks = c(0, 99.9, 125.9, Inf),
                          labels = c("Normal", "Prediabetes", "Diabetes"))

# Reorder the levels to put Normal in the middle
glucose_categories <- factor(glucose_categories, levels = c("Prediabetes", "Normal", "Diabetes"))

# Add the categories to the data frame
stroke_data$glucose_category = glucose_categories
str(stroke_data)
```

```
## 'data.frame':    5109 obs. of  14 variables:
##  $ id               : int  9046 51676 31112 60182 1665 56669 53882 10434 27419 60491 ...
##  $ gender           : Factor w/ 2 levels "Female","Male": 2 1 2 1 1 2 2 1 1 1 ...
##  $ age              : num  67 61 80 49 79 81 74 69 59 78 ...
##  $ hypertension     : Factor w/ 2 levels "0","1": 1 1 1 1 2 1 2 1 1 1 ...
##  $ heart_disease    : Factor w/ 2 levels "0","1": 2 1 2 1 1 1 2 1 1 1 ...
##  $ ever_married     : Factor w/ 2 levels "No","Yes": 2 2 2 2 2 2 2 1 2 2 ...
##  $ work_type        : Factor w/ 5 levels "children","Govt_job",..: 4 5 4 4 5 4 4 4 4 4 ...
##  $ Residence_type   : Factor w/ 2 levels "Rural","Urban": 2 1 1 2 1 2 1 2 1 2 ...
##  $ avg_glucose_level: num  229 202 106 171 174 ...
##  $ bmi              : num  36.6 NA 32.5 34.4 24 29 27.4 22.8 NA 24.2 ...
##  $ smoking_status   : Factor w/ 4 levels "formerly smoked",..: 1 2 2 3 2 1 2 2 4 4 ...
##  $ stroke           : Factor w/ 2 levels "0","1": 2 2 2 2 2 2 2 2 2 2 ...
##  $ cat_weight       : Factor w/ 5 levels "Underweight",..: 5 NA 4 4 2 3 3 2 NA 2 ...
##  $ glucose_category : Factor w/ 3 levels "Prediabetes",..: 3 3 1 3 3 3 2 2 2 2 ...
```

Relation between variables

```r
tab_st_gd = table(stroke_data$stroke,stroke_data$gender)
chisq.test(tab_st_gd)
```

Stroke with Gender

```
##
##  Pearson's Chi-squared test with Yates' continuity correction
##
## data:  tab_st_gd
## X-squared = 0.34, df = 1, p-value = 0.5598
```

The *p-value* is larger than 0.05 this mean that there's not evidence of dependency between the variables gender and stroke.

```r
tab_st_hy = table(stroke_data$stroke,stroke_data$hypertension)
chisq.test(tab_st_hy)
```

Stroke with Hypertension

```
##
##  Pearson's Chi-squared test with Yates' continuity correction
##
## data:  tab_st_hy
## X-squared = 81.573, df = 1, p-value < 2.2e-16
```

The *p-value* is a lot smaller than 0.05. This mean that there's a relation between getting a stroke and hypertension. This is a could be a comprobation of the hypothesis that we established earlier about the existence of a relation between this two variables.

```r
tab_st_hd = table(stroke_data$stroke,stroke_data$heart_disease)
chisq.test(tab_st_hd)
```

Stroke with Heart Disease

```
##
##  Pearson's Chi-squared test with Yates' continuity correction
##
## data:  tab_st_hd
## X-squared = 90.229, df = 1, p-value < 2.2e-16
```

The *p-value* is a lot smaller than 0.05. This mean that there's a relation between getting a stroke and heart disease This is a could be a comprobation of the hypothesis that we established earlier about the existence of a relation between this two variables.

```
tab_st_rt = table(stroke_data$stroke,stroke_data$Residence_type)
chisq.test(tab_st_rt)
```

Stroke with Residence Type

```
##
##  Pearson's Chi-squared test with Yates' continuity correction
##
## data:  tab_st_rt
## X-squared = 1.075, df = 1, p-value = 0.2998
```

As we had seen in the graph there´s no evidence to say that there's a relation between the type of residence and getting a stroke.

```
tab_st_em = table(stroke_data$stroke,stroke_data$ever_married)
chisq.test(tab_st_em)
```

Stroke with Ever Married

```
##
##  Pearson's Chi-squared test with Yates' continuity correction
##
## data:  tab_st_em
## X-squared = 58.868, df = 1, p-value = 1.686e-14
```

Apparently there's a relation within this two variables. Having a stroke have a relation with have been or had been ever married.

```
tab_st_sk = table(stroke_data$stroke,stroke_data$smoking_status)
chisq.test(tab_st_sk)
```

Stroke with Smoking Status

```
##
##  Pearson's Chi-squared test
##
## data:  tab_st_sk
## X-squared = 29.226, df = 3, p-value = 2.008e-06
```

The *p-value* is a lot smaller than 0.05 so there's is a relation between the variables. This was something that we *dont know what to write here*

```
tab_st_wt = table(stroke_data$stroke,stroke_data$work_type)
chisq.test(tab_st_wt)
```

Stroke with Work Type

```
##
##  Pearson's Chi-squared test
##
## data:  tab_st_wt
## X-squared = 49.159, df = 4, p-value = 5.409e-10
```

There's a relation between the variables ($p\text{-}value<0.05$). This we think was because of the difference between the quantity of people who got a stroke and work independently and the people who work with children because the difference was big between them.

```
tab_hy_hd = table(stroke_data$heart_disease,stroke_data$hypertension)
chisq.test(tab_hy_hd)
```

Heart Disease and Hypertension

```
##
##  Pearson's Chi-squared test with Yates' continuity correction
##
## data:  tab_hy_hd
## X-squared = 58.31, df = 1, p-value = 2.239e-14
```

There's a relation between hypertension and heart disease and both variable are related to stroke. This could be a good indicator that within only one of this variables we could have the same information in the model.

```
tab_hd_em = table(stroke_data$heart_disease,stroke_data$ever_married)
chisq.test(tab_hd_em)
```

Heart Disease and Ever Married

```
##
##  Pearson's Chi-squared test with Yates' continuity correction
##
## data:  tab_hd_em
## X-squared = 66.036, df = 1, p-value = 4.428e-16
```

```r
tab_hd_sk= table(stroke_data$heart_disease,stroke_data$smoking_status)
chisq.test(tab_hd_sk)
```

Heart Disease and Smoking Status

```
##
##  Pearson's Chi-squared test
##
## data:  tab_hd_sk
## X-squared = 44.74, df = 3, p-value = 1.051e-09
```

```r
tab_hd_wt = table(stroke_data$heart_disease,stroke_data$work_type)
chisq.test(tab_hd_wt)
```

Heart Disease and Work Type

```
##
##  Pearson's Chi-squared test
##
## data:  tab_hd_wt
## X-squared = 70.689, df = 4, p-value = 1.623e-14
```

```r
tab_hy_em = table(stroke_data$heart_disease,stroke_data$ever_married)
chisq.test(tab_hy_em)
```

Hypertension and Ever Married

```
##
##  Pearson's Chi-squared test with Yates' continuity correction
##
## data:  tab_hy_em
## X-squared = 66.036, df = 1, p-value = 4.428e-16
```

```r
tab_hy_sk = table(stroke_data$heart_disease,stroke_data$smoking_status)
chisq.test(tab_hy_sk)
```

Hypertension and Smoking Status

```
##
##  Pearson's Chi-squared test
##
## data:  tab_hy_sk
## X-squared = 44.74, df = 3, p-value = 1.051e-09
```

```r
tab_hy_wt= table(stroke_data$heart_disease,stroke_data$work_type)
chisq.test(tab_hy_wt)
```

Hypertension and Work Type

```
##
##  Pearson's Chi-squared test
##
## data:  tab_hy_wt
## X-squared = 70.689, df = 4, p-value = 1.623e-14
```

Stroke with Age

```r
t.test(age ~ stroke, data = stroke_data)
```

```
##
##  Welch Two Sample t-test
##
## data:  age by stroke
## t = -29.682, df = 331.68, p-value < 2.2e-16
## alternative hypothesis: true difference in means between group 0 and group 1 is not equal to 0
## 95 percent confidence interval:
##  -27.46015 -24.04658
## sample estimates:
## mean in group 0 mean in group 1
##        41.97483        67.72819
```

As you can see we reject the null hypothesis that said that both group have the same mean and this tell as that this variable could have and impact in the probability of getting a stroke in this case getting older increase your chances.

Stroke with Average Glucose Levels

```r
t.test(avg_glucose_level ~ stroke, data = stroke_data)
```

```
##
##  Welch Two Sample t-test
##
## data:  avg_glucose_level by stroke
## t = -6.9844, df = 260.9, p-value = 2.373e-11
## alternative hypothesis: true difference in means between group 0 and group 1 is not equal to 0
## 95 percent confidence interval:
##  -35.58269 -19.93162
## sample estimates:
## mean in group 0 mean in group 1
##        104.7876        132.5447
```

Also we can say that the difference in means between the groups with a stroke are without is not zero.

Stroke with BMI

```
t.test(bmi ~ stroke, data = stroke_data)
```

```
##
##  Welch Two Sample t-test
##
## data:  bmi by stroke
## t = -3.6374, df = 237.84, p-value = 0.0003377
## alternative hypothesis: true difference in means between group 0 and group 1 is not equal to 0
## 95 percent confidence interval:
##  -2.5387991 -0.7549231
## sample estimates:
## mean in group 0 mean in group 1
##        28.82443        30.47129
```

Apparently all the continuous variables could have and impact in the chances or getting a stroke. This variable *bmi* also have a *p-value* less than 0.05 so we can say that the groups have significant different means.

Models

Logistic Regresion

```
set.seed(23)
index_split = createDataPartition(stroke_data$stroke,p=0.8,list=FALSE)
train = stroke_data[index_split,]
test = stroke_data[-index_split,]
train = subset(train,select = -c(id,cat_weight,glucose_category))
train = na.omit(train)
test = subset(test,select = -c(id,cat_weight,glucose_category))
test = na.omit(test)
```

We would start the models with one that includes all the variables and the reduce it with different test.

```
class_weight <- ifelse(train$stroke == 1, 25, 1.04)
log_reg_model = glm(stroke~(gender + age + hypertension + heart_disease + ever_married +
    work_type + Residence_type + avg_glucose_level + bmi + smoking_status),data = train,family = binomi
summary(log_reg_model)
```

```
##
## Call:
## glm(formula = stroke ~ (gender + age + hypertension + heart_disease +
##     ever_married + work_type + Residence_type + avg_glucose_level +
##     bmi + smoking_status), family = binomial(link = "logit"),
##     data = train, weights = class_weight)
##
## Coefficients:
##                                Estimate Std. Error z value Pr(>|z|)
## (Intercept)                -3.860e+00  2.485e-01 -15.532  < 2e-16 ***
## genderMale                 -5.040e-02  6.005e-02  -0.839 0.401304
## age                         7.914e-02  2.321e-03  34.104  < 2e-16 ***
## hypertension1               8.490e-01  7.759e-02  10.942  < 2e-16 ***
## heart_disease1              3.797e-01  1.003e-01   3.784 0.000154 ***
## ever_marriedYes            -1.144e-01  9.289e-02  -1.231 0.218197
## work_typeGovt_job          -1.359e+00  2.611e-01  -5.204 1.95e-07 ***
## work_typeNever_worked      -1.123e+01  1.329e+02  -0.084 0.932663
## work_typePrivate           -1.292e+00  2.549e-01  -5.070 3.98e-07 ***
## work_typeSelf-employed     -1.552e+00  2.679e-01  -5.793 6.91e-09 ***
## Residence_typeUrban         3.577e-02  5.762e-02   0.621 0.534725
## avg_glucose_level           3.118e-03  5.853e-04   5.326 1.00e-07 ***
## bmi                         1.599e-02  4.514e-03   3.543 0.000395 ***
## smoking_statusnever smoked -3.382e-01  7.671e-02  -4.409 1.04e-05 ***
## smoking_statussmokes        2.248e-01  8.857e-02   2.538 0.011138 *
## smoking_statusUnknown      -5.600e-01  9.272e-02  -6.040 1.54e-09 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 11203.1  on 3928  degrees of freedom
## Residual deviance:  7500.3  on 3913  degrees of freedom
## AIC: 7377.4
##
## Number of Fisher Scoring iterations: 12
```

```
fitted.results <- predict(log_reg_model,newdata = test,type='response')
fitted.results <- ifelse(fitted.results > 0.8,1,0)
test$accu = fitted.results
misClasificError <- mean(fitted.results != test$stroke)
print(paste('Accuracy',1-misClasificError))
```

```
## [1] "Accuracy 0.892747701736466"
```

```
pander(confusionMatrix(as.factor(test$accu),test$stroke))
```

- **positive**: 0
- **table**:

	0	1
0	851	19
1	86	23

- **overall**:

Table 2: Table continues below

Accuracy	Kappa	AccuracyLower	AccuracyUpper	AccuracyNull
0.8927	0.2587	0.8717	0.9114	0.9571

AccuracyPValue	McnemarPValue
1	1.187e-10

- **byClass**:

Table 4: Table continues below

Sensitivity	Specificity	Pos Pred Value	Neg Pred Value	Precision
0.9082	0.5476	0.9782	0.211	0.9782

Table 5: Table continues below

Recall	F1	Prevalence	Detection Rate	Detection Prevalence
0.9082	0.9419	0.9571	0.8693	0.8887

Balanced Accuracy
0.7279

- **mode**: sens_spec
- **dots**:

```
#pander(confusionMatrix(test$accu,test$stroke))
```

As you can see if we take 0.8 as the threshold we get an accuracy of 0.89 and a sensibility of 0.9. We will continue deleting the variable resident type because it's not hace any significance in the model.

```
class_weight <- ifelse(train$stroke == 1, 25, 1.04)
log_reg_model2 = glm(stroke~(gender + age + hypertension + heart_disease + ever_married +work_type + avg
summary(log_reg_model2)
```

```
##
## Call:
## glm(formula = stroke ~ (gender + age + hypertension + heart_disease +
##     ever_married + work_type + avg_glucose_level + bmi + smoking_status),
##     family = binomial(link = "logit"), data = train, weights = class_weight)
##
## Coefficients:
##                              Estimate Std. Error z value Pr(>|z|)
## (Intercept)                -3.845e+00  2.474e-01 -15.543  < 2e-16 ***
## genderMale                 -5.104e-02  6.004e-02  -0.850 0.395203
## age                         7.919e-02  2.319e-03  34.147  < 2e-16 ***
## hypertension1               8.483e-01  7.758e-02  10.935  < 2e-16 ***
## heart_disease1              3.795e-01  1.004e-01   3.781 0.000156 ***
## ever_marriedYes            -1.188e-01  9.254e-02  -1.284 0.199235
## work_typeGovt_job          -1.355e+00  2.610e-01  -5.192 2.08e-07 ***
## work_typeNever_worked      -1.122e+01  1.329e+02  -0.084 0.932721
## work_typePrivate           -1.288e+00  2.548e-01  -5.057 4.26e-07 ***
## work_typeSelf-employed     -1.549e+00  2.679e-01  -5.784 7.29e-09 ***
## avg_glucose_level           3.103e-03  5.849e-04   5.306 1.12e-07 ***
## bmi                         1.609e-02  4.509e-03   3.569 0.000359 ***
## smoking_statusnever smoked -3.382e-01  7.673e-02  -4.408 1.05e-05 ***
## smoking_statussmokes        2.259e-01  8.853e-02   2.552 0.010720 *
## smoking_statusUnknown      -5.592e-01  9.272e-02  -6.031 1.63e-09 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 11203.1  on 3928  degrees of freedom
## Residual deviance:  7500.7  on 3914  degrees of freedom
## AIC: 7375.8
##
## Number of Fisher Scoring iterations: 12
```

```r
fitted.results <- predict(log_reg_model2,newdata = test,type='response')
fitted.results <- ifelse(fitted.results > 0.8,1,0)
test$accu = fitted.results
misClasificError <- mean(fitted.results != test$stroke)
print(paste('Accuracy',1-misClasificError))
```

```
## [1] "Accuracy 0.892747701736466"
```

```r
pander(confusionMatrix(as.factor(test$accu),test$stroke))
```

- **positive**: 0
- **table**:

	0	1
0	851	19
1	86	23

- **overall**:

Table 8: Table continues below

Accuracy	Kappa	AccuracyLower	AccuracyUpper	AccuracyNull
0.8927	0.2587	0.8717	0.9114	0.9571

AccuracyPValue	McnemarPValue
1	1.187e-10

- **byClass**:

Table 10: Table continues below

Sensitivity	Specificity	Pos Pred Value	Neg Pred Value	Precision
0.9082	0.5476	0.9782	0.211	0.9782

Table 11: Table continues below

Recall	F1	Prevalence	Detection Rate	Detection Prevalence
0.9082	0.9419	0.9571	0.8693	0.8887

Balanced Accuracy
0.7279

- **mode**: sens_spec
- **dots**:

The AIC was reduce so the model improved just a little.

```r
anova(log_reg_model2,log_reg_model)
```

```
## Analysis of Deviance Table
##
## Model 1: stroke ~ (gender + age + hypertension + heart_disease + ever_married +
##     work_type + avg_glucose_level + bmi + smoking_status)
## Model 2: stroke ~ (gender + age + hypertension + heart_disease + ever_married +
##     work_type + Residence_type + avg_glucose_level + bmi + smoking_status)
##   Resid. Df Resid. Dev Df Deviance Pr(>Chi)
## 1      3914     7500.7
## 2      3913     7500.3  1  0.38543   0.5347
```

As the *p-value* is higher than 0.05 we cannot reject that the simpler model is better. So we are going to deleted the variable gender because i doesn't affect the model.

```r
class_weight <- ifelse(train$stroke == 1, 25, 1.04)
log_reg_model3 = glm(stroke~(ever_married + age + hypertension + heart_disease+work_type + avg_glucose_
summary(log_reg_model3)
```

```
##
## Call:
## glm(formula = stroke ~ (ever_married + age + hypertension + heart_disease +
##     work_type + avg_glucose_level + bmi + smoking_status), family = binomial(link = "logit"),
##     data = train, weights = class_weight)
##
## Coefficients:
##                                Estimate Std. Error z value Pr(>|z|)
## (Intercept)                  -3.871e+00  2.456e-01 -15.761  < 2e-16 ***
## ever_marriedYes              -1.170e-01  9.243e-02  -1.266 0.205554
## age                          7.913e-02  2.318e-03  34.146  < 2e-16 ***
## hypertension1                8.493e-01  7.758e-02  10.947  < 2e-16 ***
## heart_disease1               3.682e-01  9.943e-02   3.703 0.000213 ***
## work_typeGovt_job           -1.347e+00  2.608e-01  -5.163 2.43e-07 ***
## work_typeNever_worked       -1.122e+01  1.330e+02  -0.084 0.932782
## work_typePrivate            -1.283e+00  2.547e-01  -5.038 4.71e-07 ***
## work_typeSelf-employed      -1.542e+00  2.677e-01  -5.759 8.48e-09 ***
## avg_glucose_level            3.073e-03  5.837e-04   5.264 1.41e-07 ***
## bmi                          1.612e-02  4.508e-03   3.576 0.000349 ***
## smoking_statusnever smoked -3.298e-01  7.609e-02  -4.334 1.46e-05 ***
## smoking_statussmokes         2.259e-01  8.857e-02   2.550 0.010762 *
## smoking_statusUnknown       -5.577e-01  9.267e-02  -6.019 1.76e-09 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 11203.1  on 3928  degrees of freedom
## Residual deviance:  7501.4  on 3915  degrees of freedom
## AIC: 7374.5
##
## Number of Fisher Scoring iterations: 12
```

```
fitted.results <- predict(log_reg_model3,newdata = test,type='response')
fitted.results <- ifelse(fitted.results > 0.8,1,0)
test$accu = fitted.results
misClasificError <- mean(fitted.results != test$stroke)
print(paste('Accuracy',1-misClasificError))
```

```
## [1] "Accuracy 0.889683350357508"
```

```
pander(confusionMatrix(as.factor(test$accu),test$stroke))
```

- **positive**: 0
- **table**:

	0	1
0	848	19
1	89	23

- **overall**:

Table 14: Table continues below

Accuracy	Kappa	AccuracyLower	AccuracyUpper	AccuracyNull
0.8897	0.252	0.8684	0.9086	0.9571

AccuracyPValue	McnemarPValue
1	3.147e-11

- **byClass**:

Table 16: Table continues below

Sensitivity	Specificity	Pos Pred Value	Neg Pred Value	Precision
0.905	0.5476	0.9781	0.2054	0.9781

Table 17: Table continues below

Recall	F1	Prevalence	Detection Rate	Detection Prevalence
0.905	0.9401	0.9571	0.8662	0.8856

Balanced Accuracy
0.7263

- **mode**: sens_spec
- **dots**:

```
anova(log_reg_model3,log_reg_model2)
```

```
## Analysis of Deviance Table
##
## Model 1: stroke ~ (ever_married + age + hypertension + heart_disease +
##     work_type + avg_glucose_level + bmi + smoking_status)
## Model 2: stroke ~ (gender + age + hypertension + heart_disease + ever_married +
##     work_type + avg_glucose_level + bmi + smoking_status)
##   Resid. Df Resid. Dev Df Deviance Pr(>Chi)
## 1      3915     7501.4
## 2      3914     7500.7  1  0.72316   0.3951
```

As the *p-value* is higher than 0.05 we cannot reject that the simpler model is better. So we are going to deleted the variable ever married because i doesn't affect the model.

```
class_weight <- ifelse(train$stroke == 1, 25, 1.04)
log_reg_model4 = glm(stroke~(age + hypertension + heart_disease+work_type + avg_glucose_level + bmi + sm
summary(log_reg_model4)
```

```
##
## Call:
## glm(formula = stroke ~ (age + hypertension + heart_disease +
##     work_type + avg_glucose_level + bmi + smoking_status), family = binomial(link = "logit"),
##     data = train, weights = class_weight)
##
```

```
## Coefficients:
##                               Estimate Std. Error z value Pr(>|z|)
## (Intercept)                 -3.861e+00  2.455e-01 -15.729  < 2e-16 ***
## age                          7.861e-02  2.284e-03  34.412  < 2e-16 ***
## hypertension1                8.531e-01  7.740e-02  11.023  < 2e-16 ***
## heart_disease1               3.719e-01  9.928e-02   3.746 0.000180 ***
## work_typeGovt_job           -1.417e+00  2.556e-01  -5.544 2.96e-08 ***
## work_typeNever_worked       -1.120e+01  1.323e+02  -0.085 0.932489
## work_typePrivate            -1.355e+00  2.490e-01  -5.443 5.24e-08 ***
## work_typeSelf-employed      -1.615e+00  2.621e-01  -6.165 7.07e-10 ***
## avg_glucose_level            3.020e-03  5.822e-04   5.186 2.14e-07 ***
## bmi                          1.580e-02  4.505e-03   3.507 0.000454 ***
## smoking_statusnever smoked -3.190e-01  7.559e-02  -4.220 2.44e-05 ***
## smoking_statussmokes         2.288e-01  8.858e-02   2.583 0.009806 **
## smoking_statusUnknown       -5.505e-01  9.252e-02  -5.950 2.68e-09 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 11203  on 3928  degrees of freedom
## Residual deviance:  7503  on 3916  degrees of freedom
## AIC: 7374.1
##
## Number of Fisher Scoring iterations: 12
```

```
fitted.results <- predict(log_reg_model4,newdata = test,type='response')
fitted.results <- ifelse(fitted.results > 0.8,1,0)
test$accu = fitted.results
misClasificError <- mean(fitted.results != test$stroke)
print(paste('Accuracy',1-misClasificError))
```

```
## [1] "Accuracy 0.888661899897855"
```

```
pander(confusionMatrix(as.factor(test$accu),test$stroke))
```

- **positive**: 0
- **table**:

	0	1
0	848	20
1	89	22

- **overall**:

Table 20: Table continues below

Accuracy	Kappa	AccuracyLower	AccuracyUpper	AccuracyNull
0.8887	0.2403	0.8673	0.9077	0.9571

AccuracyPValue	McnemarPValue
1	7.356e-11

- **byClass**:

Table 22: Table continues below

Sensitivity	Specificity	Pos Pred Value	Neg Pred Value	Precision
0.905	0.5238	0.977	0.1982	0.977

Table 23: Table continues below

Recall	F1	Prevalence	Detection Rate	Detection Prevalence
0.905	0.9396	0.9571	0.8662	0.8866

Balanced Accuracy
0.7144

- **mode**: sens_spec
- **dots**:

```
anova(log_reg_model,log_reg_model4)
```

```
## Analysis of Deviance Table
##
## Model 1: stroke ~ (gender + age + hypertension + heart_disease + ever_married +
##     work_type + Residence_type + avg_glucose_level + bmi + smoking_status)
## Model 2: stroke ~ (age + hypertension + heart_disease + work_type + avg_glucose_level +
##     bmi + smoking_status)
##   Resid. Df Resid. Dev Df Deviance Pr(>Chi)
## 1      3913     7500.3
## 2      3916     7503.0 -3  -2.7142   0.4378
```

The simpler model is better so as all the variables are significance in the model we will leave it at that.

```
pander(exp(log_reg_model4$coefficients))
```

Table 25: Table continues below

(Intercept)	age	hypertension1	heart_disease1	work_typeGovt_job
0.02104	1.082	2.347	1.45	0.2425

Table 26: Table continues below

work_typeNever_worked	work_typePrivate	work_typeSelf-employed
1.362e-05	0.2579	0.1988

Table 27: Table continues below

avg_glucose_level	bmi	smoking_statusnever smoked	smoking_statussmokes
1.003	1.016	0.7269	1.257

smoking_statusUnknown
0.5767

With the exp of the coefficients we get the odds of getting a stroke. Like the intercept means that if all the other variables are zero you have a 2.1% odds of getting a stroke. As we said in the preliminary analysis if you got hypertension the odd of getting a stroke duplicate. If you got a heart disease you have a 4.5% increase in your odds.

VSM

```r
# Data preprocessing function
prepare_data <- function(data) {
  processed_data <- data %>%
    select(-c(id, cat_weight, glucose_category)) %>%
    na.omit() %>%
    mutate(
      gender = as.factor(gender),
      hypertension = as.factor(hypertension),
      heart_disease = as.factor(heart_disease),
      ever_married = as.factor(ever_married),
      work_type = as.factor(work_type),
      Residence_type = as.factor(Residence_type),
      smoking_status = as.factor(smoking_status),
      stroke = factor(stroke, levels = c("0", "1"),
                      labels = c("No_Stroke", "Stroke")),
      age = scale(age),
      avg_glucose_level = scale(avg_glucose_level),
      bmi = scale(bmi)
    )
  return(processed_data)
}

# Set random seed for reproducibility
set.seed(23)

# Prepare data
stroke_data_processed <- prepare_data(stroke_data)
```

```r
# Split data into training and testing sets (80-20)
train_index <- createDataPartition(stroke_data_processed$stroke, p = 0.8, list = FALSE)
train_data <- stroke_data_processed[train_index, ]
test_data <- stroke_data_processed[-train_index, ]

# Define training control with cross-validation and class weights
ctrl <- trainControl(
  method = "cv",
  number = 5,
  classProbs = TRUE,
  summaryFunction = twoClassSummary,
  sampling = "smote",
  verboseIter = FALSE
)

# Define parameter grid for tuning
svm_grid <- expand.grid(
  C = c(0.1, 1, 10, 100),
  sigma = c(0.01, 0.1, 1)
)

# Train SVM model with tuning
svm_tune <- suppressWarnings(suppressMessages({
  train(
    stroke ~ .,
    data = train_data,
    method = "svmRadial",
    trControl = ctrl,
    tuneGrid = svm_grid,
    metric = "ROC",
    preProcess = c("center", "scale")
  )
}))
```

```
## line search fails -4.057891 0.4469497 1.495077e-05 -8.623829e-06 -1.796847e-07 7.605055e-08 -3.34227
```

```r
# Train final model with best parameters
final_svm <- suppressWarnings({
  svm(
    stroke ~ .,
    data = train_data,
    kernel = "radial",
    cost = svm_tune$bestTune$C,
    gamma = svm_tune$bestTune$sigma,
    probability = TRUE,
    class.weights = c("No_Stroke" = 1, "Stroke" = sum(train_data$stroke == "No_Stroke")/sum(train_data$
  )
})

# Make predictions
pred_prob <- suppressWarnings(predict(final_svm, test_data, probability = TRUE))
pred_class <- suppressWarnings(predict(final_svm, test_data))
```

```r
# Calculate performance metrics
confusion_matrix <- confusionMatrix(pred_class, test_data$stroke)
roc_obj <- suppressMessages({
  roc(as.numeric(test_data$stroke) - 1,
      attr(predict(final_svm, test_data, probability = TRUE), "probabilities")[,2])
})
auc_value <- auc(roc_obj)

# Calculate feature importance using permutation method
calculate_feature_importance <- function(model, data, target, n_permutations = 10) {
  baseline_pred <- suppressWarnings(predict(model, data, probability = TRUE))
  baseline_auc <- suppressMessages({
    auc(roc(as.numeric(data[[target]]) - 1,
            attr(baseline_pred, "probabilities")[,2]))
  })

  importance <- data.frame(
    feature = names(data)[names(data) != target],
    importance = 0
  )

  for(feature in importance$feature) {
    auc_drops <- numeric(n_permutations)
    for(i in 1:n_permutations) {
      permuted_data <- data
      permuted_data[[feature]] <- sample(permuted_data[[feature]])
      perm_pred <- suppressWarnings(predict(model, permuted_data, probability = TRUE))
      perm_auc <- suppressMessages({
        auc(roc(as.numeric(permuted_data[[target]]) - 1,
                attr(perm_pred, "probabilities")[,2]))
      })
      auc_drops[i] <- baseline_auc - perm_auc
    }
    importance$importance[importance$feature == feature] <- mean(auc_drops)
  }

  return(importance[order(-importance$importance),])
}

# Calculate feature importance
feature_importance <- calculate_feature_importance(final_svm, test_data, "stroke")

# Create visualizations
roc_plot <- ggroc(roc_obj) +
  geom_abline(intercept = 1, slope = 1, linetype = "dashed", color = "gray") +
  theme_minimal() +
  labs(title = "ROC Curve for SVM Model",
       subtitle = paste("AUC =", round(auc_value, 3)))

importance_plot <- ggplot(feature_importance,
                         aes(x = reorder(feature, importance), y = importance)) +
  geom_bar(stat = "identity", fill = "steelblue") +
  coord_flip() +
  theme_minimal() +
  labs(title = "Feature Importance (Permutation)",
       x = "Feature",
       y = "Importance (AUC drop)")

# Print results
cat("\nModel Performance:\n")
```

```
##
## Model Performance:
```

```r
print(confusion_matrix)
```

```
## Confusion Matrix and Statistics
##
##            Reference
## Prediction  No_Stroke Stroke
##   No_Stroke       643     10
##   Stroke          296     31
##
##                Accuracy : 0.6878
##                  95% CI : (0.6577, 0.7167)
##     No Information Rate : 0.9582
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.1017
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.6848
##             Specificity : 0.7561
##          Pos Pred Value : 0.9847
##          Neg Pred Value : 0.0948
##              Prevalence : 0.9582
##          Detection Rate : 0.6561
##    Detection Prevalence : 0.6663
##       Balanced Accuracy : 0.7204
##
##        'Positive' Class : No_Stroke
##
```

```r
cat("\nAUC:", round(auc_value, 4), "\n\n")
```

```
##
## AUC: 0.8089
```

```r
print("Feature Importance:")
```

```
## [1] "Feature Importance:"
```

```
print(feature_importance)
```

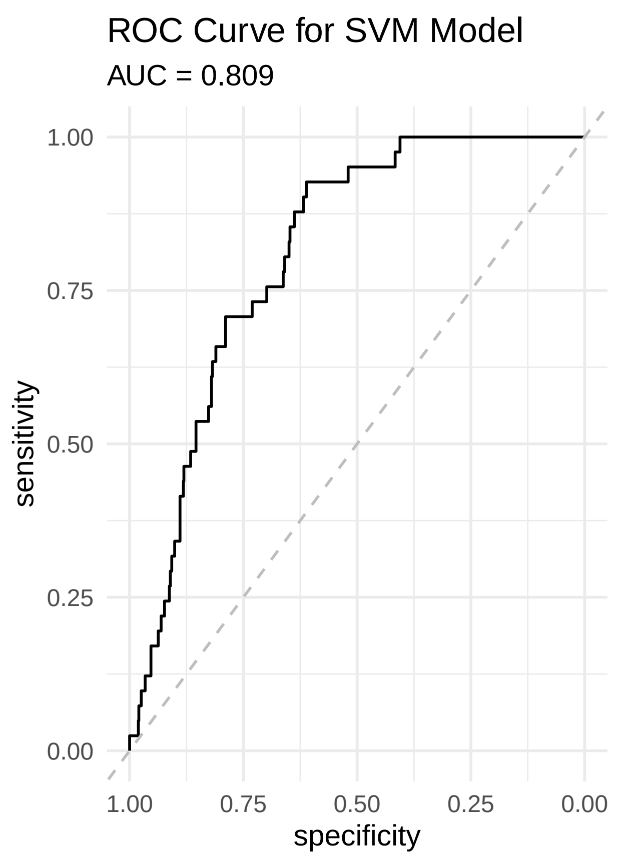
```
##              feature    importance
## 2                age  0.2852022130
## 3       hypertension  0.0095067404
## 8  avg_glucose_level  0.0025922751
## 9                bmi  0.0012857477
## 5       ever_married  0.0001168862
## 4      heart_disease  0.0001064963
## 6          work_type -0.0006233928
## 10    smoking_status -0.0015195200
## 7     Residence_type -0.0021896673
## 1             gender -0.0023143458
```

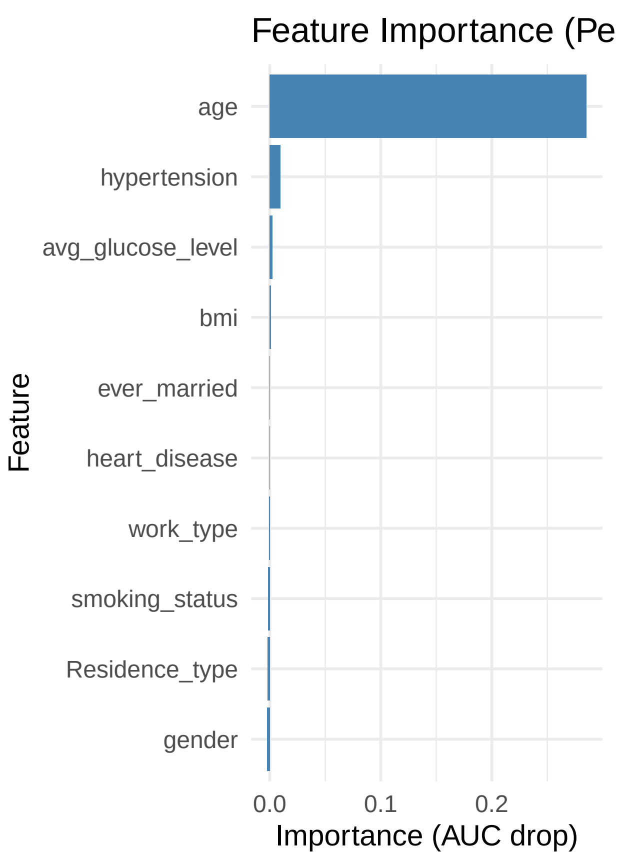
```
# Save the model and results
results <- list(
  model = final_svm,
  tuning_results = svm_tune$results,
  best_parameters = svm_tune$bestTune,
  performance = list(
    confusion_matrix = confusion_matrix,
    auc = auc_value
  ),
  feature_importance = feature_importance
)

saveRDS(results, "svm_stroke_prediction_results.rds")
```

The SVM model for stroke prediction demonstrates good discriminative ability with an AUC of 0.809, despite challenges with class imbalance in the dataset (95.82% non-stroke cases). While the overall accuracy is 68.78%, the model achieves a balanced accuracy of 72.04% with sensitivity of 68.48% and specificity of 75.61%, indicating reasonable performance in identifying both stroke and non-stroke cases. The feature importance analysis reveals that age is overwhelmingly the most significant predictor (0.285 AUC drop), followed distantly by hypertension (0.010), average glucose level (0.003), and BMI (0.001), while other features like residence type, smoking status, and gender show minimal or slightly negative impact. This suggests that the model could potentially be simplified by focusing on the key predictors, particularly age-based risk stratification, while maintaining its effectiveness. The model's strong AUC but relatively low negative predictive value (9.48%) indicates it's better suited for risk stratification rather than definitive diagnosis, and might benefit from additional techniques to improve prediction of actual stroke cases.

```
# Display plots
grid.arrange(roc_plot, importance_plot, ncol = 2)
```





The ROC curve on the left illustrates the model's classification performance in distinguishing stroke cases from non-stroke cases. The curve rises significantly above the diagonal baseline, achieving an AUC of 0.809, which reflects a strong discriminative capability. This AUC score suggests that the model effectively differentiates between stroke and non-stroke conditions.

On the right, the feature importance plot highlights each variable's contribution to the model. Age emerges as the most influential predictor, with an importance score substantially higher than other features. Notably, after age, there's a marked decline in feature importance: hypertension, average glucose level, and BMI show only minimal positive impact on stroke prediction, while variables such as work type, smoking status, residence type, and gender contribute little to none or even negatively. This stark contrast in feature importance implies that age could serve as a primary indicator for stroke risk, although the modest contributions of certain physiological factors suggest they should still be considered in comprehensive risk assessment.

XGBoost for classification

```r
# Function to prepare data for XGBoost
prepare_data <- function(data) {
  # Explicit conversion for each feature
  XGBoost_data <- data %>%
    mutate(
      gender = as.numeric(factor(gender)) - 1,
      hypertension = as.numeric(hypertension) - 1,
      heart_disease = as.numeric(heart_disease) - 1,
      ever_married = as.numeric(factor(ever_married)) - 1,
      work_type = as.numeric(factor(work_type)),
      Residence_type = as.numeric(factor(Residence_type)) - 1,
      smoking_status = as.numeric(factor(smoking_status)),
      stroke = as.numeric(factor(stroke)) - 1
    )

  return(XGBoost_data)
}

# Data cleaning and preparation
stroke_data_clean <- stroke_data %>%
  select(-c(id, cat_weight, glucose_category)) %>%
  na.omit()

# Prepare the data
stroke_data_xgb <- prepare_data(stroke_data_clean)

# Split the data (80-20)
set.seed(23)
train_index <- createDataPartition(stroke_data_xgb$stroke, p = 0.8, list = FALSE)
train_data <- stroke_data_xgb[train_index, ]
test_data <- stroke_data_xgb[-train_index, ]

# Define features explicitly
features <- c("gender", "age", "hypertension", "heart_disease", "ever_married",
             "work_type", "Residence_type", "avg_glucose_level", "bmi", "smoking_status")

# Create matrix format
train_matrix <- as.matrix(train_data[, features])
test_matrix <- as.matrix(test_data[, features])
train_label <- train_data$stroke
test_label <- test_data$stroke

# Create DMatrix objects
dtrain <- xgb.DMatrix(data = train_matrix, label = train_label)
dtest <- xgb.DMatrix(data = test_matrix, label = test_label)

# Set XGBoost parameters with class imbalance handling
pos_weight <- sum(train_label == 0) / sum(train_label == 1)
params <- list(
  objective = "binary:logistic",
  eval_metric = c("auc", "error"),
  max_depth = 6,
  eta = 0.1,
  subsample = 0.8,
  colsample_bytree = 0.8,
  min_child_weight = 1,
  scale_pos_weight = pos_weight
)

# Train the model with early stopping
xgb_model <- xgb.train(
  params = params,
  data = dtrain,
  nrounds = 100,
  watchlist = list(train = dtrain, test = dtest),
  early_stopping_rounds = 10,
  print_every_n = 10
)
```

```
## [1]  train-auc:0.884262  test-auc:0.771946
## Multiple eval metrics are present. Will use test_auc for early stopping.
## Will train until test_auc hasn't improved in 10 rounds.
##
## [11] train-auc:0.953179  test-auc:0.827286
## [21] train-auc:0.972242  test-auc:0.816776
## Stopping. Best iteration:
## [12] train-auc:0.954376  test-auc:0.828452
```

```r
# Make predictions
pred_prob <- predict(xgb_model, dtest)
pred_class <- ifelse(pred_prob > 0.5, 1, 0)

# Calculate comprehensive performance metrics
confusion_matrix <- confusionMatrix(factor(pred_class), factor(test_label))
roc_curve <- roc(test_label, pred_prob)
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

```r
auc_value <- auc(roc_curve)

# Print detailed results
cat("\nModel Performance Metrics:\n")
```

```
##
## Model Performance Metrics:
```

```r
print(confusion_matrix)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 786  21
##          1 153  21
##
##                Accuracy : 0.8226
##                  95% CI : (0.7973, 0.846)
##     No Information Rate : 0.9572
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.1348
```

```
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.8371
##             Specificity : 0.5000
##          Pos Pred Value : 0.9740
##          Neg Pred Value : 0.1207
##              Prevalence : 0.9572
##          Detection Rate : 0.8012
##    Detection Prevalence : 0.8226
##       Balanced Accuracy : 0.6685
##
##        'Positive' Class : 0
##
```

```
cat("\nAUC:", round(auc_value, 4), "\n")
```

```
##
## AUC: 0.8285
```

The XGBoost model shows good predictive capability, achieving an AUC of 0.829, slightly surpassing the SVM model's AUC of 0.809. This high AUC reflects the model's effectiveness in distinguishing between stroke and non-stroke cases. The model also achieved an accuracy of 82.26% on the test set (95% CI: 79.73% - 84.6%), alongside high sensitivity of 83.71%, but a moderate specificity of 50%, indicating a strong ability to identify stroke cases but some difficulty in excluding non-stroke cases accurately.

During training, the model converged well, reaching optimal performance at the 12th iteration with a test AUC of 0.828, where early stopping helped prevent overfitting. Despite this, a gap remained between the train (0.954) and test (0.828) AUCs, hinting at mild overfitting. The model's ability to correctly identify non-stroke cases is high (positive predictive value of 97.4%), but it faces challenges with stroke case predictions, as shown by a lower negative predictive value of 12.07%. This reflects the difficulties posed by the imbalanced dataset, where 95.72% of cases are non-stroke.

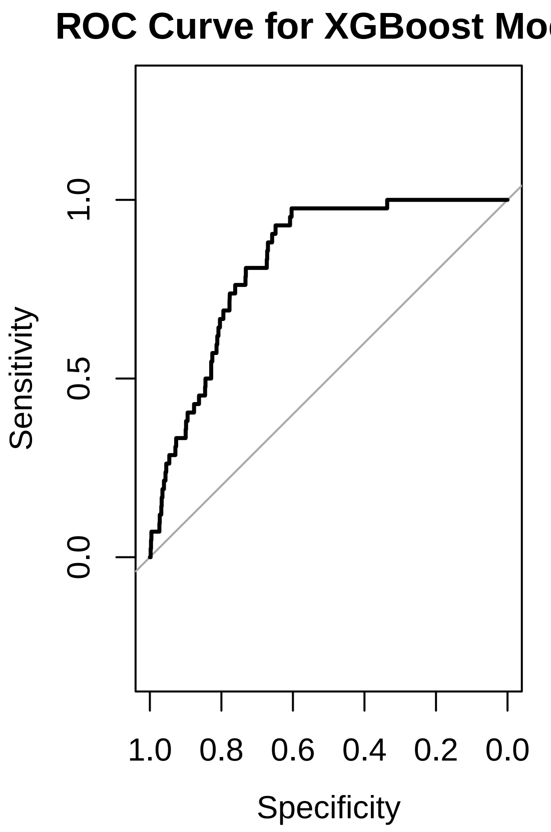
The model's balanced accuracy of 66.85% and a Kappa score of 0.1348 indicate moderate overall performance once class imbalance is taken into account. Training dynamics reveal rapid improvement from an initial train AUC of 0.884 and test AUC of 0.772 to optimal performance. When comparing models, XGBoost outperformed SVM in both accuracy and AUC, suggesting it may be a more reliable tool for stroke prediction. Nonetheless, the class imbalance remains a challenge.

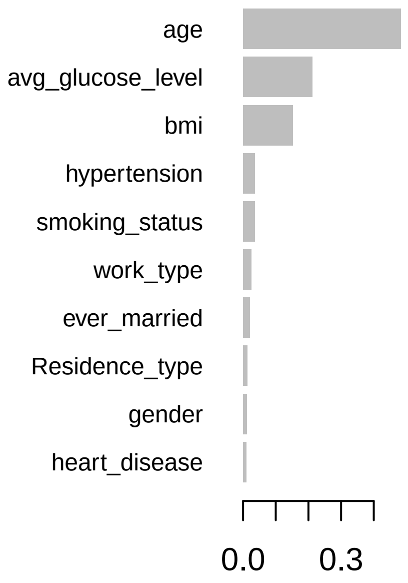
```
# Feature importance analysis
importance_matrix <- xgb.importance(feature_names = features, model = xgb_model)
print(importance_matrix)
```

```
##               Feature        Gain       Cover  Frequency
##                <char>       <num>       <num>      <num>
##  1:               age 0.482922665 0.337150876 0.16690042
##  2: avg_glucose_level 0.212653021 0.275330701 0.35343619
##  3:               bmi 0.152780602 0.209223607 0.25666199
##  4:      hypertension 0.035679552 0.055658009 0.02384292
##  5:    smoking_status 0.035531148 0.043448897 0.06732118
##  6:         work_type 0.025467967 0.032078220 0.04628331
##  7:      ever_married 0.020635472 0.017940013 0.01963534
##  8:    Residence_type 0.013509676 0.007178726 0.03085554
##  9:            gender 0.010823160 0.005346709 0.02244039
## 10:     heart_disease 0.009996738 0.016644241 0.01262272
```

Age emerges as the dominant predictor with the highest gain (48.29%), indicating it contributes most to the model's performance, followed by average glucose level (21.27%) and BMI (15.28%). These top three features account for approximately 85% of the model's predictive power. The secondary tier of predictors includes hypertension and smoking status (both around 3.5% gain), while features like gender and heart disease show minimal impact (around 1% gain). Interestingly, the frequency metric shows a different pattern, with average glucose level being used most often in tree decisions (35.34%), followed by BMI (25.67%) and age (16.69%), suggesting that while age has the highest impact per use, glucose levels and BMI are more frequently utilized in making predictions.

```r
# Visualizations
par(mfrow = c(1, 2))
# ROC Curve
plot(roc_curve, main = "ROC Curve for XGBoost Model")
# Feature Importance Plot
xgb.plot.importance(importance_matrix)
```





The ROC curve on the left demonstrates strong discriminative performance, with the model's curve sharply rising above the diagonal reference line and achieving an AUC of 0.829. This high AUC suggests the model's solid ability to differentiate between stroke and non-stroke cases.

On the right, the feature importance plot illustrates a clear ranking of predictor variables. Age stands out with the highest importance score (around 0.4 gain), establishing it as the primary driver in stroke risk prediction. This is followed by average glucose level (approximately 0.2 gain) and BMI (around 0.15 gain). After these top three predictors, there's a marked decrease in importance, as other features—such as hypertension, smoking status, work type, marital status, residence type, gender, and heart disease—each contribute minimally, with importance scores below 0.05.

This hierarchy in feature importance suggests that a streamlined model focusing on age, average glucose level, and BMI might retain most of the predictive power, potentially simplifying the model while maintaining effectiveness.

```r
# Split data
set.seed(23)
train_index <- createDataPartition(stroke_data_xgb$stroke, p = 0.8, list = FALSE)
train_data <- stroke_data_xgb[train_index, ]
test_data <- stroke_data_xgb[-train_index, ]

features <- c("gender", "age", "hypertension", "heart_disease", "ever_married",
             "work_type", "Residence_type", "avg_glucose_level", "bmi", "smoking_status")

# Function to generate random parameters
generate_params <- function() {
  list(
    nrounds = sample(c(100, 150, 200), 1),
    max_depth = sample(3:6, 1),
    eta = runif(1, 0.01, 0.1),
    gamma = runif(1, 0, 0.3),
    colsample_bytree = runif(1, 0.6, 1.0),
    min_child_weight = sample(1:5, 1),
    subsample = runif(1, 0.6, 1.0)
  )
}

# Number of random parameter sets to try
n_iterations <- 200

# Calculate class weights
pos_weight <- sum(train_data$stroke == 0) / sum(train_data$stroke == 1)

# Perform random search with cross-validation
set.seed(23)
cv_results <- list()
best_score <- 0
best_params <- NULL

for (i in 1:n_iterations) {
  params <- generate_params()

  # Create DMatrix for current fold
  dtrain <- xgb.DMatrix(
    data = as.matrix(train_data[, features]),
    label = train_data$stroke
  )

  # Cross-validation for current parameter set
  cv <- xgb.cv(
    params = c(params,
              list(objective = "binary:logistic",
                   eval_metric = "auc",
                   scale_pos_weight = pos_weight)),
    data = dtrain,
    nrounds = params$nrounds,
    nfold = 5,
    early_stopping_rounds = 10,
    verbose = 0
  )

  # Extract best score
  best_iter <- cv$best_iteration
  mean_auc <- max(cv$evaluation_log$test_auc_mean)
  sd_auc <- cv$evaluation_log$test_auc_std[which.max(cv$evaluation_log$test_auc_mean)]

  # Store results
  cv_results[[i]] <- list(
    params = params,
    mean_auc = mean_auc,
    sd_auc = sd_auc,
    best_iter = best_iter
  )

  # Update best parameters if necessary
  if (mean_auc > best_score) {
    best_score <- mean_auc
    best_params <- params
    best_params$nrounds <- best_iter
  }
}  # Close the for loop

# Train final model with best parameters
final_model <- xgb.train(
  params = c(best_params,
            list(objective = "binary:logistic",
                 eval_metric = "auc",
                 scale_pos_weight = pos_weight)),
  data = xgb.DMatrix(data = as.matrix(train_data[, features]),
                     label = train_data$stroke),
  nrounds = best_params$nrounds
)
```

```
## [15:56:50] WARNING: src/learner.cc:767:
## Parameters: { "nrounds" } are not used.
```

```r
# Evaluate final model
test_pred <- predict(final_model, as.matrix(test_data[, features]))
test_roc <- roc(test_data$stroke, test_pred)
```

```
## Setting levels: control = 0, case = 1

## Setting direction: controls < cases
```

```r
test_auc <- auc(test_roc)

# Print results
cat("\nBest Parameters:\n")
```

```
##
## Best Parameters:
```

```r
print(best_params)
```

```
## $nrounds
## [1] 23
##
## $max_depth
## [1] 3
##
## $eta
## [1] 0.08442434
##
## $gamma
## [1] 0.2455268
##
## $colsample_bytree
## [1] 0.77001
##
## $min_child_weight
## [1] 1
##
## $subsample
## [1] 0.6875576
```

```r
cat("\nTest AUC:", round(test_auc, 4), "\n")
```

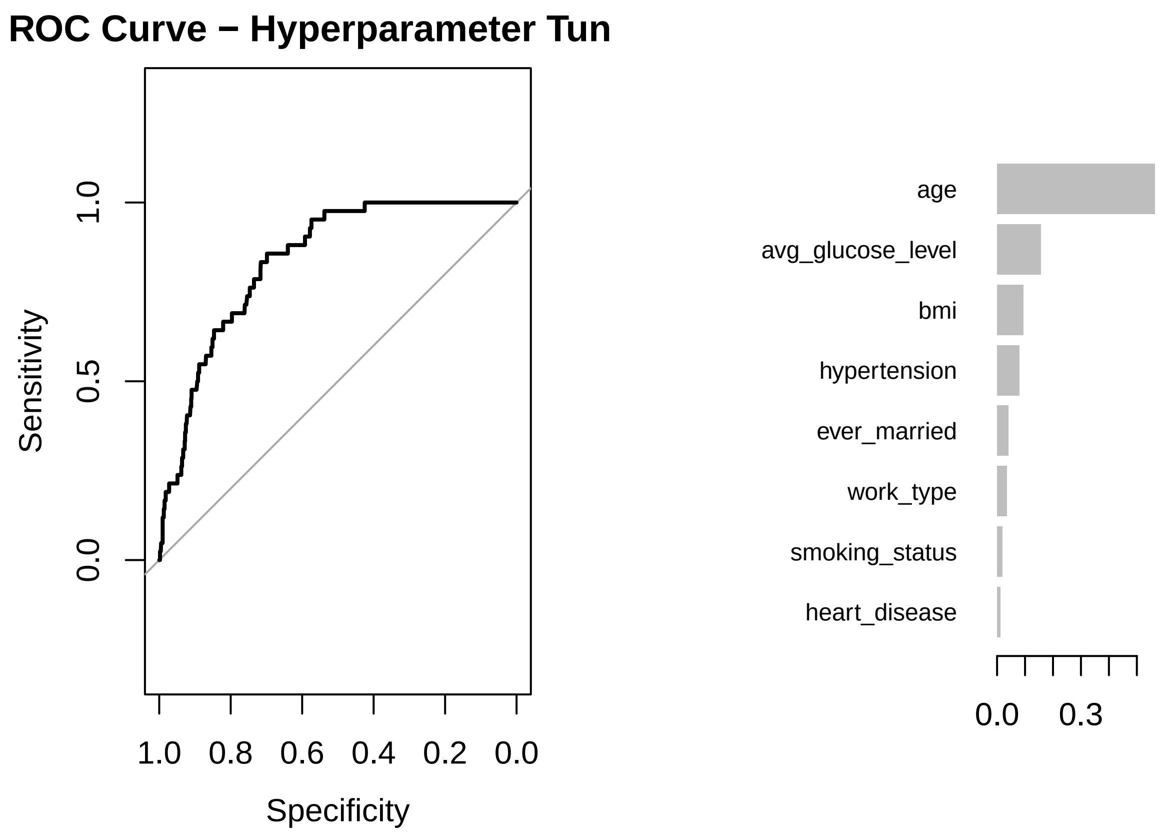
```
##
## Test AUC: 0.8398
```

We can se that this XGBoost implementation demonstrates a more sophisticated approach through systematic hyperparameter tuning using random search cross-validation, resulting in an improved AUC of 0.8398 compared to the first version's 0.8285. This optimization process explored 200 different parameter combinations, ultimately finding optimal settings including 23 rounds of boosting, a tree depth of 3, a learning rate of 0.084, and various other fine-tuned parameters that control model complexity and sampling. The modest 1.1% improvement in performance suggests that while the optimization was successful, we might be approaching the inherent predictive limit of this dataset given the available features. The tuned model's parameters indicate a preference for a relatively conservative approach, balancing model complexity with generalization ability through moderate sampling rates (colsample_bytree = 0.77, subsample = 0.688) and controlled tree growth (max_depth = 3), which helps prevent overfitting while maintaining predictive performance.

```r
# Feature importance
importance_matrix <- xgb.importance(feature_names = features, model = final_model)
print(importance_matrix)
```

```
##              Feature       Gain       Cover   Frequency
##               <char>      <num>       <num>       <num>
## 1:               age 0.56438041 0.415661792 0.288590604
## 2: avg_glucose_level 0.15742324 0.212492846 0.261744966
## 3:               bmi 0.09401344 0.150584317 0.234899329
## 4:      hypertension 0.07978487 0.079067155 0.046979866
## 5:      ever_married 0.03972432 0.029337277 0.013422819
## 6:         work_type 0.03401899 0.063276208 0.093959732
## 7:    smoking_status 0.01849881 0.039780042 0.053691275
## 8:     heart_disease 0.01215592 0.009800364 0.006711409
```

```
# Plot ROC curve and feature importance
par(mfrow = c(1, 2))
plot(test_roc, main = "ROC Curve - Hyperparameter Tuning")
xgb.plot.importance(importance_matrix)
```



This hyperparameter-tuned model demonstrates notable shifts in feature importance while achieving a slightly improved AUC of 0.8398 (compared to 0.8285). Age has become an even more dominant predictor, increasing from 48% to 56% gain, while both average glucose level and BMI showed decreased importance (dropping to 16% and 9% respectively, from their previous 21% and 15%). This redistribution of feature importance suggests that the optimized model has found a more efficient way to utilize age as the primary predictor, while also giving slightly more weight to previously underutilized features like hypertension (increasing to 8%). These changes, resulting from the systematic hyperparameter tuning process, indicate a more refined model that better leverages the predictive power of each feature, particularly emphasizing the crucial role of age in stroke prediction.

H2O models with autoh2o

Split the data to get a 80% for training and 20% for testing.#flo

```r
h2o.init()
```

```
##  Connection successful!
##
## R is connected to the H2O cluster:
##     H2O cluster uptime:         4 days 15 hours
##     H2O cluster timezone:       Europe/Madrid
##     H2O data parsing timezone:  UTC
##     H2O cluster version:        3.44.0.3
##     H2O cluster version age:    10 months and 10 days
##     H2O cluster name:           H2O_started_from_R_soroush_yeq133
##     H2O cluster total nodes:    1
##     H2O cluster total memory:   3.64 GB
##     H2O cluster total cores:    10
##     H2O cluster allowed cores:  10
##     H2O cluster healthy:        TRUE
##     H2O Connection ip:          localhost
##     H2O Connection port:        54321
##     H2O Connection proxy:       NA
##     H2O Internal Security:      FALSE
##     R Version:                  R version 4.4.1 (2024-06-14)
```

```r
stroke_h2o=as.h2o(stroke_data)
```

```
##   |                                                                              |
```

```r
split_data = h2o.splitFrame(data=stroke_h2o,ratios=0.8,seed=23)
train_h2o = split_data[[1]]
test_h2o = split_data[[2]]
predictor = c("gender","age","hypertension","heart_disease","ever_married","work_type","Residence_type"
aml = h2o.automl(x = predictor,y="stroke",training_frame=train_h2o,max_models=10,keep_cross_validation_
```

```
##   |                                                                              |
## 15:56:50.852: AutoML: XGBoost is not available; skipping it.  |
```

```r
lb <- aml@leaderboard
print(lb, n = nrow(lb),extra_columns="ALL")
```

```
##                                                  model_id       auc   logloss
## 1     StackedEnsemble_AllModels_1_AutoML_5_20241031_155650 0.8321991 0.1610895
## 2  StackedEnsemble_BestOfFamily_1_AutoML_5_20241031_155650 0.8312252 0.1612843
## 3                          GLM_1_AutoML_5_20241031_155650 0.8295080 0.1619212
## 4                          GBM_1_AutoML_5_20241031_155650 0.8268580 0.1678332
## 5                          XRT_1_AutoML_5_20241031_155650 0.8232607 0.1928370
## 6                          GBM_2_AutoML_5_20241031_155650 0.8148693 0.1841245
## 7                          GBM_5_AutoML_5_20241031_155650 0.8143710 0.1762200
## 8                          GBM_3_AutoML_5_20241031_155650 0.8073816 0.1885697
## 9                          GBM_4_AutoML_5_20241031_155650 0.8028584 0.2025739
## 10                         DRF_1_AutoML_5_20241031_155650 0.7935992 0.3196497
## 11                GBM_grid_1_AutoML_5_20241031_155650_model_1 0.7633213 0.2102651
## 12                 DeepLearning_1_AutoML_5_20241031_155650 0.6882209 4.3946837
##          aucpr mean_per_class_error      rmse        mse
## 1  0.16813258            0.3170083 0.2084177 0.04343796
## 2  0.16607856            0.2663762 0.2084538 0.04345299
## 3  0.17453868            0.2979178 0.2084133 0.04343609
## 4  0.17248534            0.3021018 0.2098526 0.04403812
## 5  0.16164737            0.3072104 0.2160264 0.04666741
## 6  0.17266084            0.2816206 0.2121988 0.04502834
## 7  0.17390066            0.3081274 0.2112391 0.04462197
## 8  0.16222486            0.3506556 0.2134083 0.04554312
## 9  0.14615829            0.3145649 0.2160072 0.04665909
## 10 0.13639058            0.3258002 0.2195888 0.04821924
## 11 0.12781291            0.3624504 0.2171430 0.04715110
## 12 0.08860055            0.3338708 0.8444236 0.71305118
##
## [12 rows x 7 columns]
```

We are going to take as a initial point to create models the best 3 algorithm. In this case we will be comparing the Grading bosting machine, Generalized linear models and the eXtremely Randomized Trees.

```
gbm_model = h2o.gbm(x = predictor,y = "stroke",training_frame = train_h2o,keep_cross_validation_predicti
```

Gradieng Bosting Machine

```
##   |                                                                              |
```

The performance of this model is the follow.

```
gbm_perform = h2o.performance(gbm_model,newdata = test_h2o)
print(gbm_perform)
```

```
## H2OBinomialMetrics: gbm
##
## MSE:  0.03861112
## RMSE:  0.1964971
## LogLoss:  0.1405456
## Mean Per-Class Error:  0.3149004
## AUC:  0.8777287
## AUCPR:  0.2897502
## Gini:  0.7554574
## R^2:  0.1151304
##
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
##          0  1    Error      Rate
## 0      898 62 0.064583   =62/960
## 1       26 20 0.565217    =26/46
## Totals 924 82 0.087475  =88/1006
##
## Maximum Metrics: Maximum metrics at their respective thresholds
##                           metric threshold      value idx
## 1                         max f1  0.145380   0.312500  73
## 2                        max f2  0.055600   0.459057 178
## 3                  max f0point5  0.595774   0.405405   6
## 4                  max accuracy  0.595774   0.959245   6
## 5                 max precision  0.746927   1.000000   0
## 6                    max recall  0.015685   1.000000 312
## 7               max specificity  0.746927   1.000000   0
## 8              max absolute_mcc  0.595774   0.325158   6
## 9    max min_per_class_accuracy  0.055600   0.804348 178
## 10 max mean_per_class_accuracy  0.027221   0.820380 260
## 11                      max tns  0.746927 960.000000   0
## 12                      max fns  0.746927  45.000000   0
## 13                      max fps  0.002404 960.000000 399
## 14                      max tps  0.015685  46.000000 312
## 15                      max tnr  0.746927   1.000000   0
## 16                      max fnr  0.746927   0.978261   0
## 17                      max fpr  0.002404   1.000000 399
## 18                      max tpr  0.015685   1.000000 312
##
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/
```

```r
glm_model = h2o.glm(x = predictor, y = "stroke", training_frame = train_h2o,keep_cross_validation_predi
```

Generalized Linear Models

```
##   |                                                                              |
```

The performance of this model is the follow.

```r
glm_perform = h2o.performance(glm_model,newdata = test_h2o)
print(glm_perform)
```

```
## H2OBinomialMetrics: glm
##
## MSE:  0.03901051
## RMSE:  0.1975108
## LogLoss:  0.1419672
## Mean Per-Class Error:  0.2157156
## AUC:  0.8741848
## AUCPR:  0.2108722
## Gini:  0.7483696
## R^2:  0.1059773
## Residual Deviance:  285.6381
## AIC:  329.6381
##
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
##          0   1    Error       Rate
## 0      838 122 0.127083   =122/960
## 1       14  32 0.304348     =14/46
## Totals 852 154 0.135189  =136/1006
##
```

```
## Maximum Metrics: Maximum metrics at their respective thresholds
##                         metric threshold      value idx
## 1                       max f1  0.110358   0.320000 126
## 2                       max f2  0.092543   0.492021 153
## 3                 max f0point5  0.308445   0.265957  11
## 4                 max accuracy  0.349515   0.955268   2
## 5                max precision  0.349515   0.666667   2
## 6                   max recall  0.020681   1.000000 301
## 7              max specificity  0.356583   0.998958   0
## 8             max absolute_mcc  0.092543   0.341734 153
## 9   max min_per_class_accuracy  0.092543   0.804348 153
## 10 max mean_per_class_accuracy  0.092543   0.821445 153
## 11                     max tns  0.356583 959.000000   0
## 12                     max fns  0.356583  46.000000   0
## 13                     max fps  0.001465 960.000000 399
## 14                     max tps  0.020681  46.000000 301
## 15                     max tnr  0.356583   0.998958   0
## 16                     max fnr  0.356583   1.000000   0
## 17                     max fpr  0.001465   1.000000 399
## 18                     max tpr  0.020681   1.000000 301
##
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/
```

Extremely Randomized Trees This model is a type of random Forest who takes as many trees as predictors you have.

```r
xrt_model = h2o.randomForest(x = predictor, y = "stroke",training_frame = train_h2o,keep_cross_validati
```

```
##   |                                                                              |
```

The performance of this model is the follow.

```r
xrt_perform = h2o.performance(xrt_model,newdata = test_h2o)
print(xrt_perform)
```

```
## H2OBinomialMetrics: drf
##
## MSE:  0.04068915
## RMSE:  0.2017155
## LogLoss:  0.2363819
## Mean Per-Class Error:  0.2720788
## AUC:  0.8395833
## AUCPR:  0.2731995
## Gini:  0.6791667
## R^2:  0.06750706
##
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
##          0   1    Error       Rate
## 0      855 105 0.109375   =105/960
## 1       20  26 0.434783     =20/46
## Totals 875 131 0.124254  =125/1006
##
```

```
## Maximum Metrics: Maximum metrics at their respective thresholds
##                         metric threshold      value idx
## 1                       max f1  0.160000   0.293785  40
## 2                       max f2  0.120000   0.423729  43
## 3                 max f0point5  0.480000   0.368852  14
## 4                 max accuracy  0.788750   0.957256   4
## 5                max precision  0.936442   1.000000   0
## 6                   max recall  0.000000   1.000000  61
## 7              max specificity  0.936442   1.000000   0
## 8             max absolute_mcc  0.480000   0.284252  14
## 9   max min_per_class_accuracy  0.060000   0.733333  53
## 10 max mean_per_class_accuracy  0.060000   0.779710  53
## 11                     max tns  0.936442 960.000000   0
## 12                     max fns  0.936442  45.000000   0
## 13                     max fps  0.000000 960.000000  61
## 14                     max tps  0.000000  46.000000  61
## 15                     max tnr  0.936442   1.000000   0
## 16                     max fnr  0.936442   0.978261   0
## 17                     max fpr  0.000000   1.000000  61
## 18                     max tpr  0.000000   1.000000  61
##
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/
```

When comparing the three models, we observe that each achieves high accuracy, above 80% on the test data. However, considering the variable we aim to predict—the probability of having a stroke—we want to minimize the number of high-risk individuals predicted as low-risk. This means reducing false negatives. The model that best achieves this is the GLM, which still maintains a high accuracy of 0.874.

Ensemble model We will create an ensemble model with the 3 models that we just created.

```r
base_models=list(gbm_model@model_id,xrt_model@model_id,glm_model@model_id)

ensemble_model=h2o.stackedEnsemble(x=predictor,y="stroke", training_frame=train_h2o, base_models=base_mo
```

```
##   |                                                                               |
```

Now we can check the perfomance of the new model.

```r
perf_ensemble=h2o.performance(ensemble_model,newdata=test_h2o)
print(perf_ensemble)
```

```
## H2OBinomialMetrics: stackedensemble
##
## MSE:  0.03834555
## RMSE:  0.1958202
## LogLoss:  0.1475955
## Mean Per-Class Error:  0.1934556
## AUC:  0.8840127
## AUCPR:  0.2631302
## Gini:  0.7680254
##
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
```

```
##          0   1    Error       Rate
## 0      839 121 0.126042   =121/960
## 1       12  34 0.260870     =12/46
## Totals 851 155 0.132207  =133/1006
##
## Maximum Metrics: Maximum metrics at their respective thresholds
##                         metric threshold      value idx
## 1                       max f1  0.063174   0.338308 143
## 2                       max f2  0.063174   0.501475 143
## 3                 max f0point5  0.365050   0.377358  14
## 4                 max accuracy  0.675810   0.955268   0
## 5                max precision  0.675810   1.000000   0
## 6                   max recall  0.025691   1.000000 326
## 7              max specificity  0.675810   1.000000   0
## 8             max absolute_mcc  0.063174   0.354738 143
## 9   max min_per_class_accuracy  0.046656   0.809375 195
## 10 max mean_per_class_accuracy  0.054770   0.823528 169
## 11                     max tns  0.675810 960.000000   0
## 12                     max fns  0.675810  45.000000   0
## 13                     max fps  0.021331 960.000000 399
## 14                     max tps  0.025691  46.000000 326
## 15                     max tnr  0.675810   1.000000   0
## 16                     max fnr  0.675810   0.978261   0
## 17                     max fpr  0.021331   1.000000 399
## 18                     max tpr  0.025691   1.000000 326
##
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/
```

The ensemble model increased accuracy by 1%, reaching a value of 88%. False positives were also reduced, so overall, the ensemble model improved performance and is better suited for the data.