Stroke Prediction Analysis

Het Thakkar, Ali Hamza Abidi Syed, Khalyl Smith,
David Oloyede, Justin Wang, & Mohammad Raihan Kapadia

Spring 2021

Introduction

According to the World Health Organization (WHO), strokes are the 2nd leading cause of death in the world, which is responsible for approximately 11% of total deaths. In this paper, we'd like to find a better understanding to what factors cause strokes to occur in so many people every year. The dataset we will be using can be found here with the following variables for our analysis (excluding `id`):

- id: unique identifier
- gender: "Male", "Female" or "Other"
- age: age of the patient
- hypertension: 0 if the patient doesn't have hypertension, 1 if the patient has hypertension
- heart_disease: 0 if the patient doesn't have any heart diseases, 1 if the patient has had a heart disease
- ever_married: "No" or "Yes"
- work_type: "children", "Govt_jov", "Never_worked","Private" or "Self-employed"
- Residence_type: "Rural" or "Urban"
- avg_glucose_level: average glucose level in blood
- bmi: body mass index
- smoking_status: "formerly smoked", "never smoked", "smokes" or "Unknown"
- stroke: 1 if the patient had a stroke or 0 if not

We will be using `stroke` as our response variable and all other variables as our predictors to conduct our analysis. Since `stroke` is a qualitative variable, we'll use logistic regression and a classification tree model. We will also use a random forest to improve our prediction on our decision tree as well as cross-validation for the logistic regression and classification tree.

Logistic Regression (Het Thakkar, Ali Hamza Abidi Syed)

For the Logistic Regression model, we will use `stroke` as our response variable, and all other variables (excluding `id`) that is `gender`, `age`, `hypertension`, `heart_disease`, `ever_married`, `work_type`, `Residence_type`, `avg_glucose_level`, `bmi`, `smoking_status` as our predictors. The formula we will use for the Logistic Regression is displayed below:

Here $p = P(stroke = 1)$, so

$$p = \frac{exp^{\beta_0+\beta_1*gender+\beta_2*age+\beta_3*hypertension+\beta_4*heart\_disease+\beta_5*ever\_married+\beta_6*work\_type+\beta_7*Residence\_type+\beta_8*avg\_glucose\_l}}{1+exp^{\beta_0+\beta_1*gender+\beta_2*age+\beta_3*hypertension+\beta_4*heart\_disease+\beta_5*ever\_married+\beta_6*work\_type+\beta_7*Residence\_type+\beta_8*avg\_glucose}}$$

Before we create the model, notice how there is an imbalance within our response variable:

```
summary(base_stroke$stroke)
```

```
##    0    1
## 4700  209
```

As we can see, there are more cases of patients who have not had any strokes than patients who've had strokes in the past. Fitting a tree with this unbalanced data will produce undesirable results where the prediction model will bias towards the most common class. To rectify this, we need to either oversample or undersample our data. Both approaches involve balancing the instances of both classes, but oversampling produces more instances of the uncommon class while undersampling selects a random subset out of the common class to match the number of the uncommon class. We'll perform undersampling for this model and we can utilize the `caret` library to achieve this.

```
library(caret)
set.seed(5)
base_stroke = downSample(base_stroke[,-c(11)], base_stroke$stroke, list = FALSE, yname = "stroke")
```

Now we can split our data into a training set and testing set with an 80-20 split to get this model:

```
set.seed(5)
sample<-sample.int(n = nrow(base_stroke), size = floor(0.80*nrow(base_stroke)))
base_training<-base_stroke[sample,]
base_test<-base_stroke[-sample,]
```

The R-code below estimates a logistic regression model using the glm (generalized linear model) function.

```
train.stroke = glm(stroke~.,data = base_training, family = "binomial")
summary(train.stroke)
```

```
##
## Call:
## glm(formula = stroke ~ ., family = "binomial", data = base_training)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.5855  -0.5901   0.2574   0.6431   2.3183
##
## Coefficients:
##                             Estimate Std. Error z value Pr(>|z|)
## (Intercept)               -4.3495321  1.2591833  -3.454 0.000552 ***
## genderMale                -0.2396039  0.3065408  -0.782 0.434427
## age                        0.0878043  0.0122866   7.146 8.91e-13 ***
## hypertension               0.7079855  0.3972697   1.782 0.074728 .
## heart_disease              0.5416285  0.5591753   0.969 0.332735
## ever_marriedYes           -0.1357318  0.5168934  -0.263 0.792865
## work_typeGovt_job         -2.1219390  1.3574799  -1.563 0.118018
## work_typePrivate          -0.7708797  1.2981135  -0.594 0.552615
## work_typeSelf-employed    -1.0870960  1.3701432  -0.793 0.427534
## Residence_typeUrban        0.0402322  0.2981657   0.135 0.892665
## avg_glucose_level         -0.0002429  0.0029880  -0.081 0.935217
## bmi                        0.0223694  0.0248038   0.902 0.367134
## smoking_statusnever smoked -0.4535981  0.3800896  -1.193 0.232714
## smoking_statussmokes         0.2634523  0.4546552   0.579 0.562282
## smoking_statusUnknown       -0.1736277  0.4936422  -0.352 0.725042
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 462.97  on 333  degrees of freedom
## Residual deviance: 289.25  on 319  degrees of freedom
## AIC: 319.25
##
## Number of Fisher Scoring iterations: 6
```

```
library(MASS)
step.model<-stepAIC(train.stroke, trace = FALSE)
coef(step.model)
```

```
##           (Intercept)                   age          hypertension
##           -4.20070524            0.08659701            0.72295040
##      work_typeGovt_job      work_typePrivate work_typeSelf-employed
##           -1.89374161           -0.47509272           -0.81929291
```

It looks like `age, hypertension, and work_type` are the most important variables in our model.

Now let's evaluate the performance of our model by using the testing set of our data and calculating the test error rate and accuracy rate.

```
prediction = predict.glm(train.stroke, newdata = base_test, type = "response")
had.stroke = prediction > 0.5

confusionMatrix<-table(base_test$stroke, had.stroke)
confusionMatrix
```

```
##    had.stroke
##     FALSE TRUE
##   0    31   13
##   1     8   32
```

```
testerror<-(confusionMatrix[2] + confusionMatrix[3])/(sum(confusionMatrix))
testerror
```

```
## [1] 0.25
```

The test error rate is 0.25 = 25% and conversely an accuracy rate of 0.75 = 75%. With this accuracy rate, our tree performs well in predicting if a patient may be in risk of getting a stroke. Let's repeat this process ten times with different subsets of training and testing data and calculate the mean of each test prediction errors. After iterating ten times, we receive the following test error rates and their corresponding mean:

```
testerrors
```

```
##  [1] 0.2380952 0.2857143 0.2261905 0.3095238 0.2619048 0.2857143 0.2976190
##  [8] 0.2857143 0.2619048 0.2738095
```

```
mean(testerrors)
```

```
## [1] 0.272619
```

This mean of our test error rates is $0.272619 \approx 27.2\%$ which is just slightly lower than the one we received when we performed our first sampling (28.57%) which supports the claim that our data and model performs well in predicting if a patient is in risk of receiving a stroke.

Next let's study the model without dividing the data into test and training error for better interpretation.

```
total.stroke = glm(stroke~.,data = base_stroke, family = "binomial")
step.model<-stepAIC(total.stroke, trace = FALSE)
coef(step.model)
```

```
##           (Intercept)            genderMale                   age
##           -5.04232165           -0.43245153            0.08573537
##          hypertension      work_typeGovt_job      work_typePrivate
##            0.56415211           -1.61350484           -0.93775340
## work_typeSelf-employed                   bmi
##           -1.53990114            0.04624558
```

Interpretation: We can see from the fitting of the whole data and predicting stroke that the most important thing in predicting strokes is the patient's age and the glucose level come next if we look at the t-statistics. Now to determine which predictors are more important than the other we use the step function and use the AIC metric to see that the most important or significant predictors are: `age, genderMale, hypertension, work_type, and bmi` which is somewhat consistent with the training data we fit earlier.

Cross-Validating Regression Model (Justin Wang, Mohammad Raihan Kapadia)

We'll use K-Fold Cross Validation to find the validation error rate for our model. We considered Leave-One-Out Cross-Validation but that algorithm is too demanding for the size of our data even through downsampling. Let K = 10, we'll calculate the mean CV error over ten repetitions

```
library(boot)
set.seed(5)
cv.errors = NA
cost = function(stroke, pi = 0) mean(abs(stroke - pi) > 0.5)

for(i in 1:10) {
  glm.fit = glm(stroke~., family = "binomial", data = base_stroke)
  cv.errors[i] = cv.glm(base_stroke, glm.fit, cost, K = 10)$delta[1]
}
cv.errors
```

```
##  [1] 0.2177033 0.2296651 0.2272727 0.2344498 0.2368421 0.2344498 0.2368421
##  [8] 0.2344498 0.2296651 0.2368421
```

```
mean(cv.errors)
```

```
## [1] 0.2318182
```

The result of these repetitions gives us a mean validation error rate of $0.2318182 \approx 23.18\%$ which suggests that this model is a sufficient fit for our data.

Classification Tree (Khalyl Smith)

For the classification tree, we'll use `stroke` as our response variable and all other variables (excluding for `id`) as our predictors. This is the formula we will use for the classification tree:

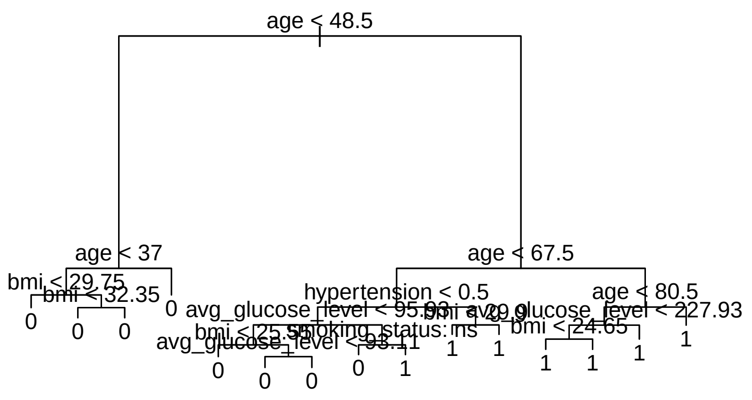
$$\hat{stroke} \sim gender + age + hypertension + heart\_disease + ever\_married +$$
$$work\_type + Residence\_type + avg\_glucose\_level + bmi + smoking\_status$$

Just like we did for the logistic regression model, we need to undersample our data to retrieve desirable results.

```
library(caret)
set.seed(5)
stroke.data2 = downSample(stroke.data[,-c(11)], stroke.data$stroke, list = FALSE, yname = "stroke")
```

Now we can split our data into a training set and testing set with an 80-20 split to get this tree:

```
train = sample(nrow(stroke.data2), round(nrow(stroke.data2)*.80))
test.stroke = stroke.data2[-train,]
tree.stroke = tree::tree(stroke~., data = stroke.data2, subset = train)
```



```
##
## Classification tree:
## tree::tree(formula = stroke ~ ., data = stroke.data2, subset = train)
## Variables actually used in tree construction:
## [1] "age"               "bmi"               "hypertension"
## [4] "avg_glucose_level" "smoking_status"
## Number of terminal nodes:  15
## Residual mean deviance:  0.7606 = 242.6 / 319
## Misclassification error rate: 0.1796 = 60 / 334
```

For our undersampled tree, it looks like `age`, `bmi`, `hypertension`, `avg_glucose_level`, and `smoking_status` are present in this tree. We have a residual mean deviance of `0.7606` and a misclassification error rate of

17.96%. With 15 terminal nodes, we should prune the tree using cross-validation which we will do in the cross-validation section of this report. Although in the case of cross-validation, we will need to account for the new undersampled data. But for now, let's evaluate the performance of our tree by using the testing set of our data and calculating the accuracy rate.

```
tree.pred = predict(tree.stroke, test.stroke, type = 'class')
table(tree.pred, test.stroke$stroke)
```

```
##
## tree.pred  0  1
##         0 37  7
##         1 12 28
```

The accuracy rate is 0.7738095 ≈ 77.4% and conversely a test error rate of 0.2261905 ≈ 22.62%. With this accuracy rate, our tree performs fairly well in predicting if a patient may be in risk of getting a stroke. Let's repeat this process ten times with different subsets of training and testing data and calculate the mean of each test prediction errors.

After iterating ten times, we receive the following test error rates and their corresponding mean:

```
test.errors
```

```
##  [1] 0.2857143 0.2619048 0.2976190 0.2738095 0.3095238 0.2023810 0.2619048
##  [8] 0.3333333 0.2857143 0.2380952
```

```
mean(test.errors)
```

```
## [1] 0.275
```

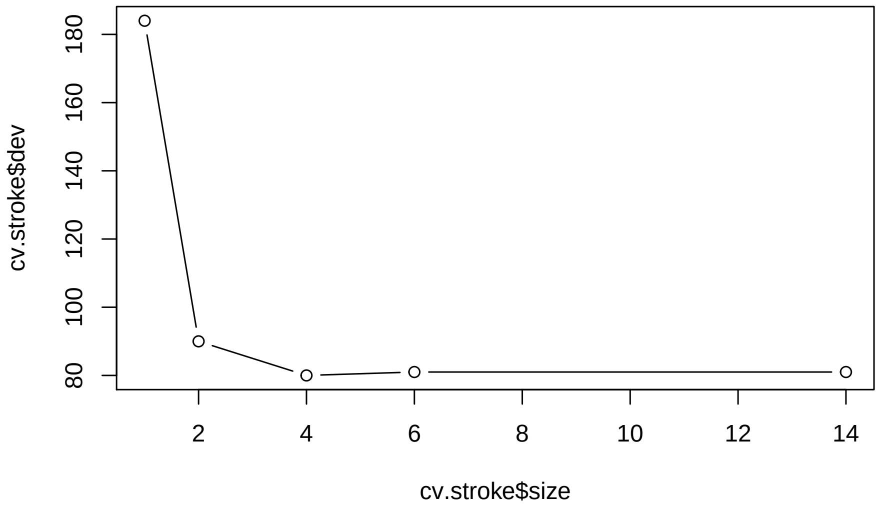
This mean of our test error rates is slightly higher than the one we received when we performed our first sampling (22.62%) which supports the claim that our data and model performs well in predicting if a patient is in risk of receiving a stroke. However, we still need to consider pruning for better results.

Compared to the logistic regression model's mean of test error rates, 25%, our tree model performs slightly better than the logistic regression approach.

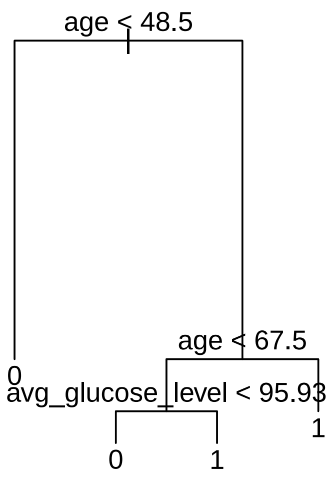
Pruning Classification Tree (Justin Wang, Mohammad Raihan Kapadia)

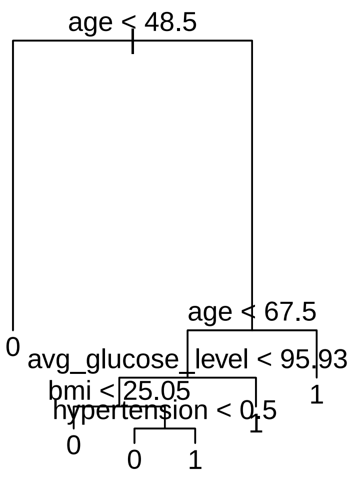
We need to prune our classification tree since it's not viable to interpret a tree with 15 terminal nodes.

```
set.seed(3)
library(tree)
cv.stroke = cv.tree(tree.stroke, FUN = prune.misclass)
plot(cv.stroke$size, cv.stroke$dev, type = "b")
```



It looks like four nodes can lead us to better results, but maybe six nodes is more efficient.





Both are good trees, but we'll choose six nodes because it considers two more variables for predicting `stroke` as opposed to four nodes only using `age` and `avg_glucose_level`.

```
summary(prune6)
```

```
##
## Classification tree:
## snip.tree(tree = tree.stroke, nodes = c(2L, 7L, 13L))
## Variables actually used in tree construction:
## [1] "age"               "avg_glucose_level" "bmi"
## [4] "hypertension"
## Number of terminal nodes:  6
## Residual mean deviance:  0.9242 = 303.1 / 328
## Misclassification error rate: 0.1856 = 62 / 334
```

```
prune.pred = predict(prune6, test.stroke, type = "class")
(prune.table = table(prune.pred, test.stroke$stroke))
```

```
##
## prune.pred  0  1
##          0 32  6
##          1 14 32
```

```
(test.error = (prune.table[2] + prune.table[3])/(sum(prune.table)))
```

```
## [1] 0.2380952
```

The test error rate is $0.2380952 \approx 23.8\%$, which is a little higher than the test error rate of the first tree. This small increase in the test error rate is not a detriment to our data. We can predict `stroke` better using this pruned tree over our original tree.

Random Forest (David Oloyede)

Since it is a classification task and we are predicting the response for stroke, we are using random forests. An advantage of random forests is that it allows for a reduction in variance, and thus lower test error, compared to both single decision trees and tree bagging as well as handling categorical, continuous, and non-linear parameters efficiently without need for scaling. First we must split data. The data is split into training set and testing set with an 80-20 split.

```
library(caTools)
set.seed(10)
div<-sample.split(Y = base_stroke$stroke,SplitRatio = 0.80)
base_training<-subset(base_stroke,subset = div == TRUE)
base_test<-subset(base_stroke,subset = div == FALSE)
```

For the random forest model, we'll use `stroke` as our response variable and all other variables (excluding for `id`) as our predictors. This is the formula we will use for the model:

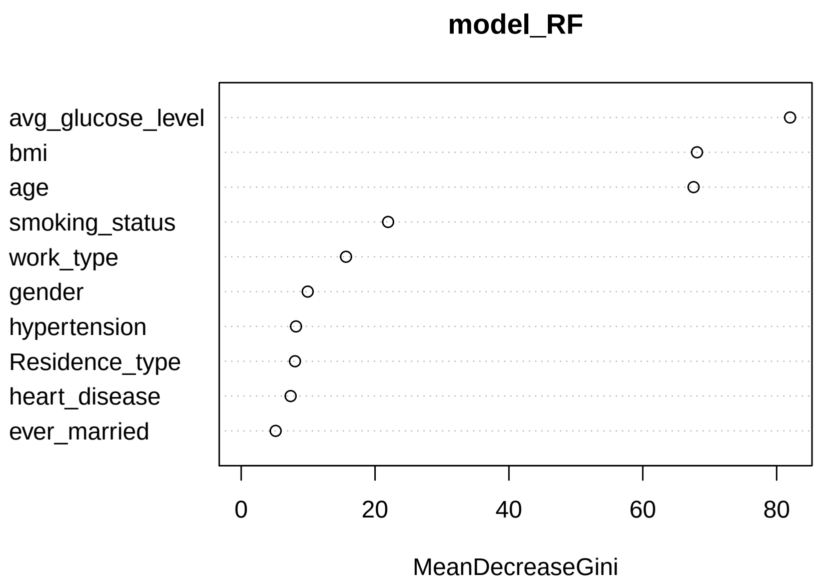
$$\begin{aligned}&\hat{stroke} \sim gender + age + hypertension + heart\_disease + ever\_married+\\&work\_type + Residence\_type + avg\_glucose\_level + bmi + smoking\_status\end{aligned}$$

A longer training period and greater complexity in interpretation as a result of multiple trees is the cost of having all the advantages of the model explained previously. This model will consist of 20 trees.

```
library(randomForest)
set.seed(1)
(model_RF<-randomForest(formula = stroke ~.,data = base_training,ntree =20))
```

```
##
## Call:
##  randomForest(formula = stroke ~ ., data = base_training, ntree = 20)
##                Type of random forest: classification
##                      Number of trees: 20
## No. of variables tried at each split: 3
##
##         OOB estimate of  error rate: 4.99%
## Confusion matrix:
##            No stroke Had stroke class.error
## No stroke       3726         33 0.008778931
## Had stroke       163          4 0.976047904
```

```
varImpPlot(model_RF)
```



Our model has and error rate estimate of 4.99% and it looks like the most important models here are `avg_glucose_level, bmi, and age`, which is much different from the important variables from the training models earlier.

```
prediction<-predict(model_RF,newdata = base_test[,-12])

(confusionMatrix<-table(base_test$stroke,prediction))
```

```
##             prediction
##              No stroke Had stroke
##   No stroke        938          2
##   Had stroke        42          0
```

```r
(accuracy.rate<-(confusionMatrix[1] + confusionMatrix[4])/ sum(confusionMatrix))
```

```
## [1] 0.9551935
```

With an accuracy rate of 95.5% we can see that the model is not very good however of predicting risk of having a stroke with a bias of not having a stroke. Even though random forests model are not as computationally expensive as a lot of others and can reduce high variance. Let's try splitting the data again ten times to retrieve the mean of the test error rates.

```r
test.errors
```

```
##  [1] 0.04276986 0.04276986 0.04378819 0.04480652 0.04582485 0.04276986
##  [7] 0.04378819 0.04989817 0.04276986 0.04480652
```

```r
mean(test.errors)
```

```
## [1] 0.04439919
```

With a test error rate average of 4.43% and conversely an average accuracy rate of 95.57%, our random forest model performs poorly with the heavy imbalance of the `stroke` classification. To properly analyze this data, we do in fact have to consider oversampling or undersampling the data.

Conclusion

Our models perform relatively well in predicting what factors can lead to a patient having a stroke. The random forest model struggles to predict accurately due to the large imbalance in our dataset. In terms of predictive performance, the logistic regression model outperforms the classification tree for this problem, especially after performing cross-validation. For interpretation, the classification tree is very efficient in interpreting our data. With the tree, we can see what combination of variables and values can potentially lead to a patient having a stroke.