Final Project PSTAT 131

Gabriel Ambrose and Phoebe Greathouse

2025-03-19

Project Introduction

In this project, we will run three different models on a heart attack binary prediction dataset to attempt to predict high-risk patients based on factors such as Age, Gender, and Cholesterol.

Data Exploration

Data Description

We have found a KAGGLE dataset, "Heart Attack Risk Prediction Cleaned Dataset" (Kalwar 2022), consisting of 25 independent variables and two response variables. Our first step was to remove the redundant response variable labelled in the dataset as "Heart Attack Risk (Text)". This initial data cleaning step allows us to properly examine our independent variables to determine which ones have the strongest influence on our response. We will perform a Principle Component Analysis to determine the number of Principle Components needed to describe 90% of the variation in our data. We will also use this to choose the 4 most influential variables for our response "Heart Attack Risk (Binary)" to perform KNN.

Numeric values in our imported dataset are normalized for better model performance and compatibility of applied algorithms for statistical analysis. This takes care of feature scaling for us (data is already normalized), so we can move on to our Principle Component Analysis without worrying.

Furthermore, 24 out of 25 variables were read in as numeric, while our "Gender" variable was a character variable. In our cleaning process, we coded "Male" observations as 0 and "Female" observations as 1 to properly run PCA on our data.
We have also removed any missing values from the dataset by using an "na.omit" function on the entire dataset.

Because each observation is found using health details for individual people, each observation will inherently be independent and identically distributed. This gets one of our assumptions out of the way when we are performing our model fittings.

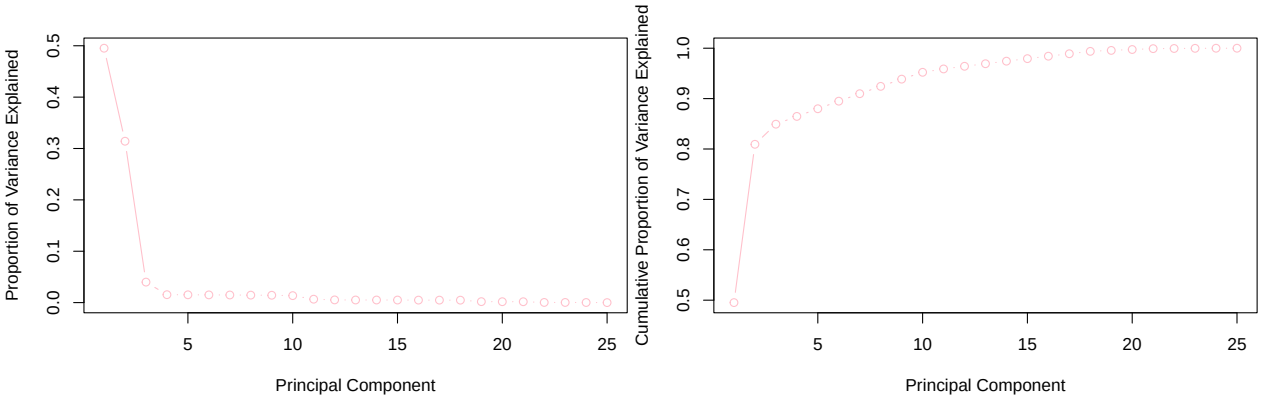
Exploratory Analysis

```
# import csv file from kaggle
library(readr)
heart <- read_csv("heart-attack-risk-prediction-dataset.csv")

# PCA to determine variables with the highest loadings
library(dplyr)
heart <- na.omit(heart)
heart <- heart %>%
  mutate(Gender = ifelse(Gender == "Male", 0, 1))
# remove text response variable
```

```
heart <- heart[,-24]
# PCA
heart_pca <- prcomp(heart[,-20])
# PVE and cumulative PVE
pve <- heart_pca$sdev^2 /sum(heart_pca$sdev^2)
cum_pve <- cumsum(pve)

# plot PC vs PVE and PC vs CPVE
plot(1:length(pve), pve, type = "b", col = "pink", xlab = "Principal Component",
     ylab = "Proportion of Variance Explained")
plot(1:length(cum_pve), cum_pve, type = "b", col = "pink",
     xlab = "Principal Component",
     ylab = "Cumulative Proportion of Variance Explained")

# how many PCs do we need in order to explain 90% of the total variation in the data?
PC_needed <- which.min(abs(cum_pve-0.9))
PC_needed
```

Dimension Reduction

```
## [1] 6
```



It will take 6 principle components to explain 90% of the total variation in the data.

```
# find 4 factors with the most influence on pc1 to use in KNN
pc1 <- heart_pca$rotation[,1]
sorted_loadings <- sort(abs(pc1), decreasing = TRUE)
knitr::kable(head(sorted_loadings, 4))
```

Top 4 PCs to be used in KNN

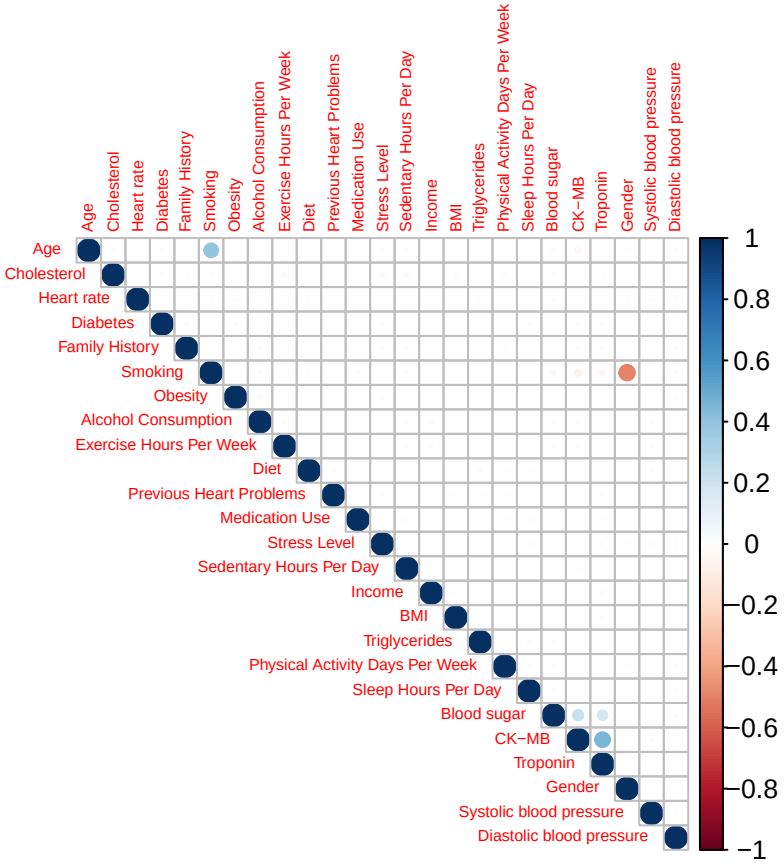
	x
Stress Level	0.9998699
Physical Activity Days Per Week	0.0141309
Obesity	0.0033282
Family History	0.0029951

Correlated Variables Graph

```
library(corrplot)
```

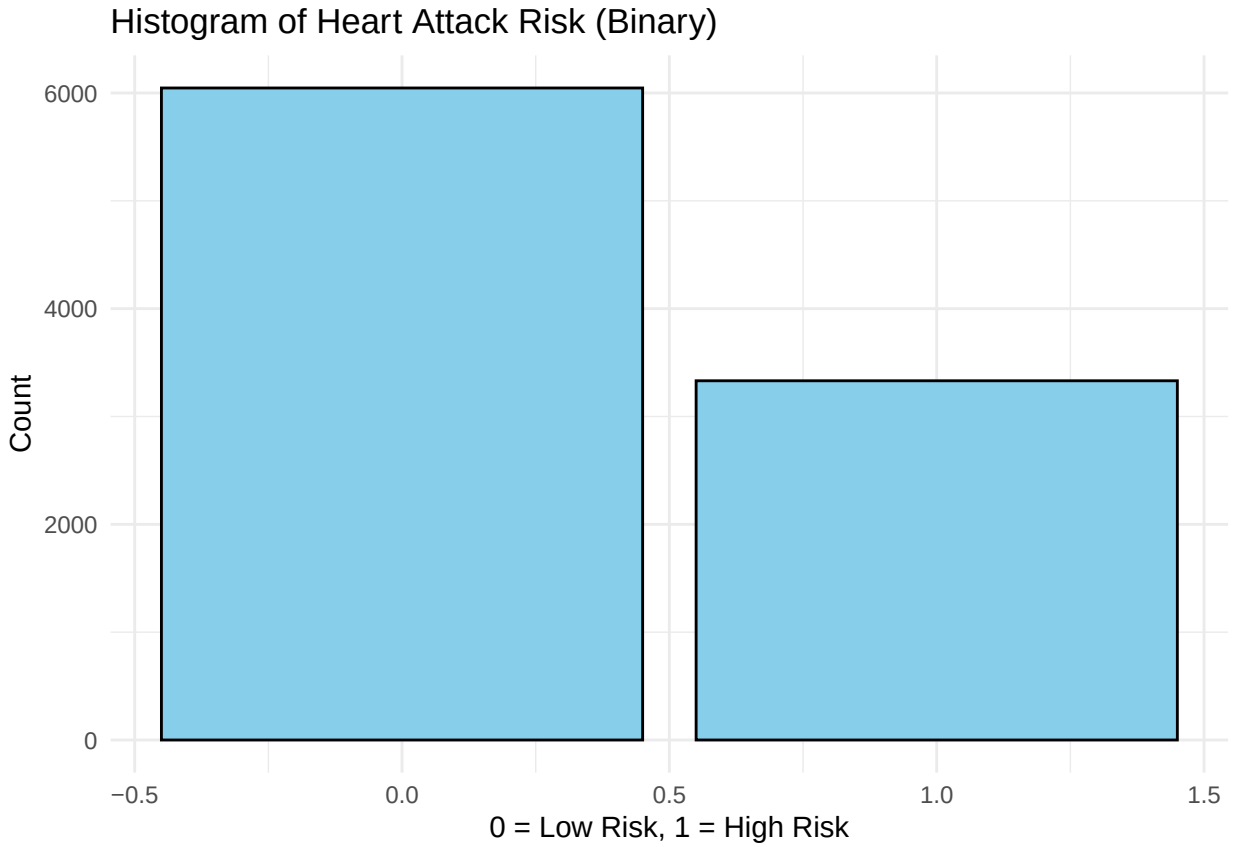
```
## corrplot 0.95 loaded
```

```
# check for highly correlated variables
corr_matrix <- cor(heart[,-20], use = "everything", method = "spearman")
corrplot(corr_matrix, type = "upper", tl.cex = 0.5)
```



Visualization

```
library(ggplot2)
ggplot(data = heart, aes(x = heart$`Heart Attack Risk (Binary)`)) +
  geom_histogram(binwidth = 1, fill = "skyblue",
                 color = "black", stat = "count") +
  labs(title = "Histogram of Heart Attack Risk (Binary)",
       x = "0 = Low Risk, 1 = High Risk", y = "Count") +
  theme_minimal()
```



We have made a histogram displaying our response variable to visualize the distribution of "Heart Attack Risk (Binary)". This shows us that 65.48% of our data corresponds to people with low risk of a heart attack, and 34.52% of our observations in this data set show a high risk of suffering a heart attack.

Statistical Methods

Problem Formulation and Discussion

Our goal for this project is to predict whether an individual is at high risk of a heart attack (in a binary prediction for our response outcome) based on our 25 variables. To do this, we have already performed PCA to determine the most influential variables and to allow us to run K-Nearest-Neighbors method to predict an observation's value at a specific point. Now, we simply need to use our three fitting methods and determine the best prediction model for our response.

We have chosen to test three methods of fitting a model:
1. Logistic Regression
2. Boosting
3. K-Nearest-Neighbors Classification

We will calculate test error rate and AUC evaluation of the ROC curve for model evaluation after fitting. We will also use K-Fold Cross-Validation to ensure we don't overfit our training dataset. Using this method of CV, we can compute our test error rates, allowing us to use the entire dataset to train our models and test using cross validation.

Models

Split Training and Testing Data - Preliminary

```
set.seed(444)
# make response column a factor
heart$`Heart Attack Risk (Binary)` <- as.factor(heart$`Heart Attack Risk (Binary)`)

# split training and testing data
train <- sample(nrow(heart), 7000)
train_heart <- heart[train,]
test_heart <- heart[-train,]
```

1. Logistic Regression

```
set.seed(888)
# run logistic regression function
heart_log <- glm(train_heart$`Heart Attack Risk (Binary)`~.,
                 family = binomial, data = train_heart)

# calculate predicted values
heart_pred_log <- predict(heart_log, newdata = test_heart, type = "response")

# find threshold value for binary prediction
# using the average of our min and max prediction values
min1 <- min(heart_pred_log)
max1 <- max(heart_pred_log)
mean1 <- mean(c(min1, max1))

# create binary prediction matrix
heart_class_log <- ifelse(heart_pred_log > mean1, 1, 0)
heart_class_log <- as.factor(heart_class_log)
table(heart_class_log)
```

```
## heart_class_log
##    0    1
## 1101 1276
```

```
# confusion matrix
conf_matrix_log <- table(Predicted = heart_class_log,
                         True = test_heart$`Heart Attack Risk (Binary)`)

# test error rate
log_error <- 1 - sum(diag(conf_matrix_log)/sum(conf_matrix_log))
```

This confusion matrix calculated from our glm() logistic regression model . Because of what we are predicting and the implications of our model on health and life expectancy, would want a higher false positive rate over a false negative rate (we prefer to predict a heart attack when the patient is in fact low-risk than failing to predict one when they are high risk). This is probably due to our threshold rate calculated by finding the midpoint of our maximum and minimum prediction points. Because our classes are not balanced and we have no values over the typical threshold value of 0.5, we need to use a lower threshold rate but could play around with it to produce a better confusion matrix.

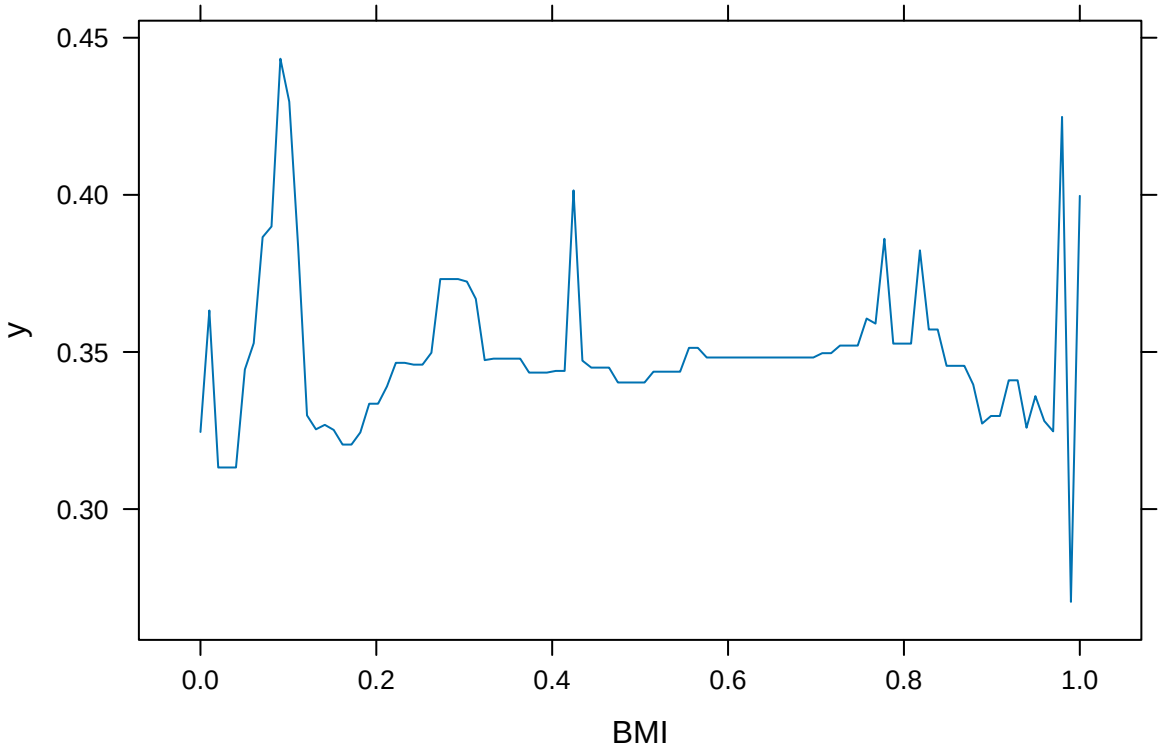
A key assumption in logistic regression is that the predictors are not correlated with each other, which we can reasonably assume since we only found a few weak correlations in our corrplot.

2. Boosting

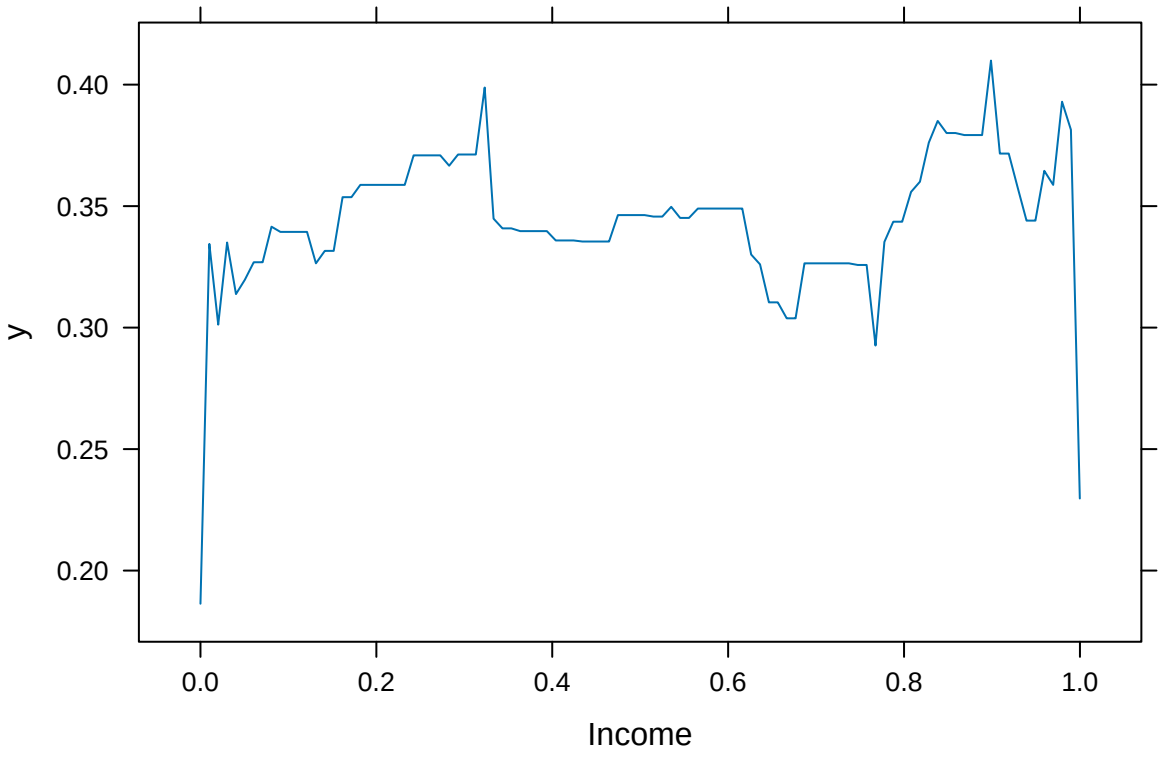
```
set.seed(777)
library(gbm)

# boosting model
boost_heart <- gbm(ifelse(`Heart Attack Risk (Binary)`== "1", 1,0) ~ .,
                   data = train_heart, distribution = "bernoulli",
                   n.trees = 500, interaction.depth = 2)
# summarize to show most important variable
summary(boost_heart, plotit = FALSE)
```

```
##                                                                 var    rel.inf
## BMI                                                             BMI 11.4078235
## `Sedentary Hours Per Day`                 `Sedentary Hours Per Day` 11.0483403
## Income                                                       Income 10.6318047
## `Exercise Hours Per Week`                 `Exercise Hours Per Week`  9.7674804
## Cholesterol                                           Cholesterol  8.2742487
## Triglycerides                                       Triglycerides  8.0419697
## `Systolic blood pressure`                 `Systolic blood pressure`  6.8089370
## Age                                                           Age  6.5730937
## `Heart rate`                                         `Heart rate`  6.0638802
## `Diastolic blood pressure`               `Diastolic blood pressure`  4.1834360
## `Blood sugar`                                       `Blood sugar`  3.0174067
## Troponin                                                 Troponin  2.9815804
## `Physical Activity Days Per Week` `Physical Activity Days Per Week`  2.1062741
## `CK-MB`                                                   `CK-MB`  1.9668629
## `Stress Level`                                     `Stress Level`  1.7514720
## `Sleep Hours Per Day`                       `Sleep Hours Per Day`  1.6144972
## Diet                                                         Diet  0.6276820
## Diabetes                                                 Diabetes  0.5458927
## Obesity                                                   Obesity  0.5382280
## `Alcohol Consumption`                     `Alcohol Consumption`  0.5075231
## Gender                                                     Gender  0.4568478
## `Medication Use`                                 `Medication Use`  0.3485900
## Smoking                                                   Smoking  0.2925613
## `Previous Heart Problems`                 `Previous Heart Problems`  0.2797438
## `Family History`                                   `Family History`  0.1638238
```

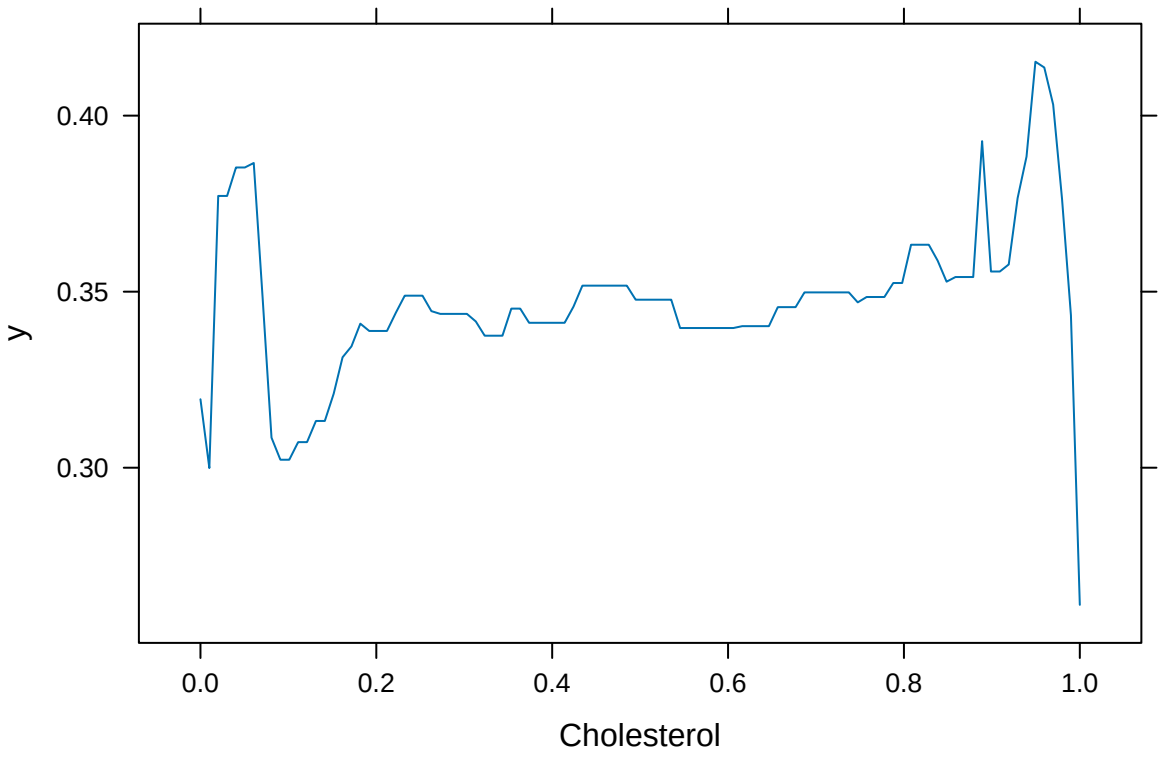
```
plot(boost_heart, i = "BMI", type = "response")
```




```
plot(boost_heart, i = "Income", type = "response")
```




```
plot(boost_heart, i = "Cholesterol", type = "response")
```




```
# prediction binary
boost_pred <- predict(boost_heart, test_heart, type = "response")
boost_pred <- ifelse(boost_pred > 0.5, 1, 0)
# confusion matrix
boost_confusion <-
  table(actual=ifelse(test_heart$`Heart Attack Risk (Binary)` == "1", 1, 0),
        predicted = boost_pred)
boost_confusion
```

```
##        predicted
## actual    0    1
##      0 1435   57
##      1  834   51
```

```
# test error rate
boost_error <- 1 - sum(diag(boost_confusion)/sum(boost_confusion))
```

3. KNN Classification

```
library(class)
library(caret)
set.seed(999)

# we are using our testing and training splits from above the logistic regression chunk
# set our train_x and train_y
train_X <- train_heart[, -20]
train_Y <- train_heart$`Heart Attack Risk (Binary)`
# set our test_x and test_y
test_X <- test_heart[, -20]
test_Y <- test_heart$`Heart Attack Risk (Binary)`
```

```
# model using highest 4 variables with highest loadings from PCA
knn_model_PCA <- train(`Heart Attack Risk (Binary)`~ `Stress Level` +
                       `Physical Activity Days Per Week` + Obesity +
                       `Family History`, method = "knn", data = train_heart)
# model using 4 noncorrelated variables with
# highest relative influence (from boosting model)
knn_model_boost <- train(`Heart Attack Risk (Binary)` ~
                           `BMI`+ `Sedentary Hours Per Day` +
                           `Income` + `Cholesterol`,
                         method = "knn", data = train_heart)

# prediction binary
knn_pred_PCA <- predict(knn_model_PCA, test_heart, type = "raw")
knn_pred_boost <- predict(knn_model_boost, test_heart, type = "raw")
# confusion matrix
knn_confusion_PCA <- table(actual = test_Y, predicted = knn_pred_PCA)
knn_confusion_boost <- table(actual = test_Y, predicted = knn_pred_boost)
# test error rate
knn_error_PCA <- 1 - sum(diag(knn_confusion_PCA)/sum(knn_confusion_PCA))
knn_error_boost <- 1 - sum(diag(knn_confusion_boost)/sum(knn_confusion_boost))
```

Evaluation of Fitted Models

1. Logistic Regression

```
# confusion matrix
conf_matrix_log
```

```
##           True
## Predicted   0   1
##         0 672 429
##         1 820 456
```

```
# test error rate
log_error
```

```
## [1] 0.5254523
```

```
# area under the curve
library(pROC)
```

```
## Type 'citation("pROC")' for a citation.
```

```
##
## Attaching package: 'pROC'
```

```
## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```
log_roc <- roc(test_heart$`Heart Attack Risk (Binary)`, heart_pred_log)
```

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

```
## Setting direction: controls > cases
```

```
auc(log_roc)
```

```
## Area under the curve: 0.5128
```

2. KNN

```
# confusion matrix
knn_confusion_PCA
```

```
##       predicted
## actual    0    1
##      0 1359  133
##      1  806   79
```

```
knn_confusion_boost
```

```
##       predicted
## actual    0    1
##      0 1244  248
##      1  719  166
```

```
# test error rate
knn_error_PCA
```

```
## [1] 0.3950358
```

```
knn_error_boost
```

```
## [1] 0.4068153
```

```
# area under the curve
library(pROC)
knn_pca_roc <- roc(test_heart$`Heart Attack Risk (Binary)`, as.numeric(knn_pred_PCA))
```

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

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

```
knn_boost_roc <- roc(test_heart$`Heart Attack Risk (Binary)`, as.numeric(knn_pred_boost))
```

```
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
```

```
auc(knn_pca_roc)
```

```
## Area under the curve: 0.5001
```

```
auc(knn_boost_roc)
```

```
## Area under the curve: 0.5107
```

3. Boosting

```
# confusion matrix
boost_confusion
```

```
##       predicted
## actual    0    1
##      0 1435   57
##      1  834   51
```

```r
# test error rate
boost_error
```

```
## [1] 0.3748422
```

```r
# area under the curve
boost_roc <- roc(test_heart$`Heart Attack Risk (Binary)`, boost_pred)
```

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

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

```r
auc(boost_roc)
```

```
## Area under the curve: 0.5097
```

Conclusion

Results

As we can see from our different test error rates of 0.5254523, 0.3950358, 0.4068153, and 0.3748422, boosting returns the highest accuracy at 37.48% test error.

Furthermore, after running the `auc()` function on our ROC calculations for each method, we find that the KNN method using our PCA principal components returns the lowest area at 0.5001. This is barely below our boosting AUC of 0.5097, which is second best.

Improvements

Because of our comparison analysis on the three models, we have chosen to use a boosting model to predict whether or not an individual is at risk of having a heart attack.

Our model is very good at predicting when a patient is low-risk, and most of the error comes from incorrectly predicting a low-risk patient when they are in fact high-risk.

This presents a problem in the real world, as we would much prefer to incorrectly predict a heart attack when the person is not at risk. Moving forward, we could adjust some of our binary threshold values to produce a confusion matrix where our predicted outcome favors the 1, or "success", outcome of classifying the individual as "high-risk".

Model-Specific Drawbacks and Improvements

1. Logistic Regression
Test Error Rate: 52.55%
AUC: 0.5128

Drawbacks:

- The logistic regression model had the highest error rate among the three models, misclassifying over half of the test observations.
- It assumes linearity in the log-odds, which may not adequately capture nonlinear relationships present in health-related data.
- Multicollinearity among predictors (e.g., highly correlated health metrics like blood pressure and cholesterol) may have affected coefficient stability and interpretability.
- The default binary classification threshold was set as the mean of the min and max predicted values, which is not optimal, especially in the presence of class imbalance.

Potential Improvements:

- Use regularization techniques such as Lasso (L1) or Ridge (L2) regression to mitigate overfitting and reduce multicollinearity effects.
- Apply feature selection or dimensionality reduction more rigorously before modeling.
- Experiment with different threshold values to optimize for recall/sensitivity, which is critical in health predictions (prioritizing high-risk detection over low-risk misclassification).
- Consider interaction terms or nonlinear transformations to improve fit.

2. Boosting
Test Error Rate: 37.48%
AUC: 0.5097

Drawbacks:

- While Boosting had the lowest error rate, it still struggled to classify high-risk patients accurately, as indicated by a high false negative rate.
- The model may be overfitting the training data or underfitting the signal from minority class cases (high-risk patients).
- Boosting is more complex and computationally intensive, and interpreting its results (like variable importance) can be less transparent than simpler models.

Potential Improvements:

- Use stratified sampling or SMOTE (Synthetic Minority Oversampling Technique) to balance the classes and help the model learn better from underrepresented high-risk cases.
- Perform hyperparameter tuning (e.g., number of trees, learning rate, depth) using grid search or cross-validation to improve generalization.
- Evaluate alternative boosting algorithms like XGBoost or LightGBM, which are optimized for speed and performance.
- Incorporate cost-sensitive learning to penalize false negatives more heavily in the loss function.

3. K-Nearest Neighbors (KNN)
Test Error Rates:
- PCA-based KNN: 39.50%
- Boosting variable KNN: 40.68%

AUCs:
- PCA-based KNN: 0.5001
- Boosting variable KNN: 0.5107

Drawbacks:

- KNN is a lazy learner and does not generalize from training data, making it sensitive to noise and irrelevant features.
- It assumes all features contribute equally, which may not be valid, especially in health datasets with differing levels of importance across variables.
- Model performance can degrade in high-dimensional settings due to the curse of dimensionality, even though PCA was used to reduce features.
- KNN is computationally expensive at prediction time since it compares test points to all training data.

Potential Improvements:

- Use feature scaling or weighting to emphasize more influential predictors.
- Apply more sophisticated dimensionality reduction (beyond PCA), like t-SNE or autoencoders, to capture nonlinear structures.
- Experiment with different values of `k` and distance metrics (e.g., Manhattan vs. Euclidean) through cross-validation.
- Use ensemble KNN or hybrid methods that combine KNN with tree-based learners to enhance prediction.

General Improvements Across Models

- **Threshold Optimization:** Adjust thresholds to favor recall (catching more high-risk patients) instead of accuracy, given the life-critical nature of the prediction.
- **Class Imbalance Handling:** Since only ~35% of the cases are high-risk, resampling methods (oversampling, undersampling, SMOTE) or weighted loss functions could yield better sensitivity.
- **Feature Engineering:** Additional domain knowledge or medical context could help generate more meaningful features or transform current variables for better representation.
- **Model Ensemble:** Combine models (e.g., logistic regression + boosting) using ensemble methods or stacking to improve performance across different data patterns.

References

- Ali Kalwar. (2022). *Heart Attack Risk Prediction Cleaned Dataset*. Kaggle. https://www.kaggle.com/datasets/alikalwar/heart-attack-risk-prediction-cleaned-dataset