Submitted to the Annals of the Department of Statistics and Operations Research

PREDICTING DIABETES WITH GRADIENT BOOSTING MACHINES

By Fuat Sarp Olcay

University of Vienna

Diabetes is a chronic disease that occurs either when the pancreas does not produce enough insulin or when the body cannot effectively use the insulin it produces. In 2014, 8.5% of adults aged 18 years and older had diabetes. In 2019, diabetes was the direct cause of 1.5 million deaths and 48% of all deaths due to diabetes occurred before the age of 70 years.

In this article I will apply the Gradient Boosting Machines algorithm on a publicly accessible data set in order to build a classification model for predicting diabetes based on 8 different diagnostic and characteristic parameter values. In addition, I will present suitable criteria to evaluate the implemented model.

1. Introduction.

Ensemble Methods are methods that combine many model predictions. Common types of ensemble methods are Bayes Optimal Classification, Bootstrap Aggregating, Random Forests and Boosting. Boosting involves incrementally building an ensemble by training each new model instance to emphasize the training instances that previous models misclassified. Whereas random forests build an ensemble of deep, independent trees, GBMs(Gradient Boosting Machines) build an ensemble of shallow and weak successive trees with each tree learning and improving on the previous.

1.1. *The Data Set.*

This paper uses the Kaggle Pima Indians Diabetes Data set (UCI Machine Learning), which was obtained from National Institute of Diabetes and Digestive and Kidney Diseases. It consists of 768 observations, 8 diagnostic and characteristic quantitative values and one binary variable, which indicates if the patient has diabetes(=1) or not (=0). The observed patients are females, who are at least 21 years old. The Pima are a group of Native Americans living in an area consisting of what is now central and southern Arizona, as well as northwestern Mexico in the states of Sonora and Chihuahua (Smith J.W. Everhart J.E. Dickson W.C. Knowler, 1988).

1.2. *Boosting.*

The idea of boosting comes from whether a weak learner can be modified to become better (Kearns and Vazirani, 1994). One of the first implications of the boosting algorithm was the "AdaBoost" algorithm Freund and Schapire (1997). The algorithm takes as input a training set $x_1; y_1, ..., x_n; y_n$, where each x_i belongs to some *domain* or *instance space* X, and each label y_i is in some label set Y. AdaBoost calls a given *weak* or *base learning algorithm* repeatedly in a series of rounds $t = 1, \ldots, T$. One of the main ideas of the algorithm is to maintain a distribution or set of weights over the training set. Initially, all weights are set equally, but on each round, the weights of incorrectly classified examples are increased so that the weak learner is forced to focus on the hard examples in the training set.

The technique of Boosting uses various loss functions. In case of Adaptive Boosting or AdaBoost, it minimises the exponential loss function that can make the algorithm sensitive to outliers. With Gradient Boosting, any differentiable loss function can be utilised. Gradient Boosting algorithm is more robust to outliers than AdaBoost Choudhury (2021).

1.3. *Gradient Boosting Algorithm.* In the following I follow the articles Friedman (2002) and Friedman (2000). Gradient Boosting constructs additive regression models by sequentially fitting a sample parameterized function i.e "base learner" to current "pseudo"-residuals by least-squares at each iteraton. The pseudo-residuals are the gradient of the loss functional being minimized, with respect to the model values at each training data point, evaluated at the current step.

In the function estimation problem one has a system consisting of a random "output" variable y and a set of random "input" variables $\mathbf{X} = x_1, ..., x_n$. Given a "training" sample of known $(y, \mathbf{x})$ values, the goal is to find a function $F^*(\mathbf{x})$ that maps $\mathbf{x}$ to y, such that over the joint distribution of all $(y,\mathbf{x})$-values, the expected value of some specified loss function $\Psi(y, F(\mathbf{x}))$ is minimized

$$F^*(\mathbf{x}) = \arg\min_{F(\boldsymbol{x})} \mathbb{E}_{y,\mathbf{x}}(\Psi(y, F(\mathbf{x}))). \tag{1.1}$$

Boosting approximates $F^*(\mathbf{x})$ by an "additive" expansion of the form

$$F(\mathbf{x}) = \sum_{m=0}^{M} \beta_m h(\mathbf{x}; \mathbf{a}_m), \tag{1.2}$$

where the functions $h(\mathbf{x;a})$ ("base learner") are usually chosen to be simple functions of $\mathbf{x}$ with parameters $\mathbf{a} = a_1, a_2, ...$ The expansion coefficients β_m and the parameters $\mathbf{a}_m$ are jointly fit to the training data in a forward "stage-wise" manner. One starts with an initial guess $F_0(\mathbf{x})$, and then for m = 1,2,...,M

$$(\beta_m, \mathbf{a}_m) = \arg\min_{\beta,\mathbf{a}} \sum_{i=1}^{N} \Psi(y_i, F_{m-1}(\mathbf{x}_i) + \beta h(\mathbf{x}_i; \mathbf{a})), \tag{1.3}$$

and

$$F_m(\mathbf{x}) = F_{m-1}(\mathbf{x} + \beta_m h(\mathbf{x}; \mathbf{a}_m). \tag{1.4}$$

Gradient boosting (Friedman 1999) approximately solves (1.3) for arbitrary loss functions $\Psi(y, F(\mathbf{x})$ with a two step procedure. First, the function $h(\mathbf{x}; \mathbf{a})$ is fit by least-squares

$$\mathbf{a}_m = \arg\min_{a,\rho} \sum_{i=1}^{N} [\tilde{y}_{im} - \rho h(\mathbf{x}_i; \mathbf{a}]^2 \tag{1.5}$$

to the current "pseudo"-residuals

$$\tilde{y}_{im} = -\left[\frac{\partial \Psi(y_i, F(\mathbf{x}_i))}{\partial F(\mathbf{x}_i)}\right]_{F(\mathbf{x})=F_{m-1}(\mathbf{x})} \tag{1.6}$$

Then, given $h(\mathbf{x};\mathbf{a}_m)$, the optimal value of the coefficient β_m is determined

$$\beta_m = \arg\min_{\beta} \sum_{i=1}^{N} \Psi(y_i, F_{m-1}(\mathbf{x}_i) + \beta h(\mathbf{x}_i;\mathbf{a}_m)). \tag{1.7}$$

This strategy replaces a potentially difficult function optimization problem (1.3) by one based on least-squares (1.5), followed by a single parameter optimization (1.7) based on the general loss criterion Ψ. Gradient tree boosting specializes this approach to the case where the base learner $h(\mathbf{x};\mathbf{a})$ is an L-terminal node regression tree. At each iteration m, a regression tree partitions the $\mathbf{x}$-space into L-disjoint regions $(R_{lm})_{l=1}^{L}$ and predicts a separate constant value in each one

$$h(\mathbf{x};(R_{lm})_{l=1}^{L}) = \sum_{l=1}^{L} \bar{y}_{lm} \mathbb{1}(x \in R_{lm}). \tag{1.8}$$

Here $\bar{y}$ is the mean of (1.6) in each region R_{lm}. The parameters of this base learner are the splitting variables and corresponding split points defining the tree, which then define the corresponding regions $(R_{lm})_1^L$ of the partition at the m-th iteration. These are induced in a top-down "best-first" manner using a least-squares splitting criterion (Friedman, Hastie, Tibshirani 1998). With regression trees, (1.7) can be solved separately within each R_{lm}, defined by the corresponding terminal node l of the m-th tree. Because the tree (1.8) predicts a constant value $\bar{y}_{lm}$ within each region R_{lm}, the solution to (1.7) reduces to an estimate based on the loss function Ψ.

$$\gamma_{lm} = \arg\min_{\gamma} \sum_{\mathbf{x}_i \in R_{lm}} \Psi(y_i, F_{m-1}(\mathbf{x}_i) + \gamma). \tag{1.9}$$

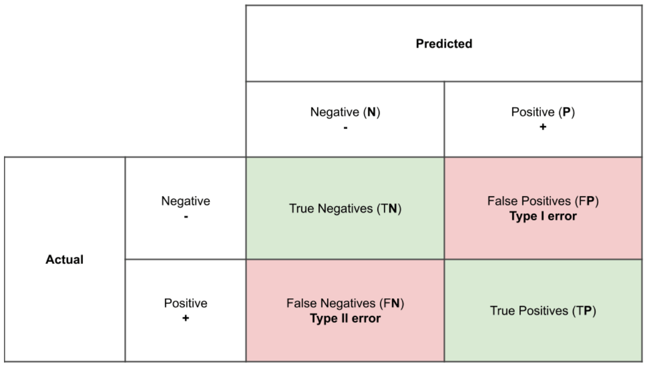
The approximation $F_{m-1}(\mathbf{x})$ is then updated in each corresponding region

$$F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + v \cdot \gamma_{lm} \mathbb{1}(x \in R_{lm}). \tag{1.10}$$

Where the parameter v $(0 < v < 1)$ i.e "shrinkage", controls the learning rate of the procedure.

1.4. *Methods for evaluation.* The quality of models with a binary response variable can be measured by several methods. Under this section, I will be explaining the criteria, which was used to assess the built model. For these evaluation methods, the data set was split into 2 random fractions with 80% and 20% of the data(cf. Dobbin and Simon (2011)). These 2 fractions of the data set are called "training" and "test" sets. With the training-set the model was built and the test-set was used in order to evaluate the model. During the construction of the model, the test-set was excluded so that the evaluation criteria would be more accurate.

1.4.1. *Confusion Matrix.* A confusion matrix is a summary of prediction results on a classification problem (cf. Ting (2017)). It shows the performance of a model based on the so called ”test-set”. From the values of a confusion matrix, several performance indicators can be derived.



		Predicted	
		Negative (**N**) -	Positive (**P**) +
Actual	Negative -	True Negatives (T**N**)	False Positives (F**P**) **Type I error**
	Positive +	False Negatives (F**N**) **Type II error**	True Positives (T**P**)

FIG 1. *An example confusion matrix for binary classification outcomes.*

Sensivity or the so-called true positive rate **(TPR)** or recall, indicates the rate of positive classified data points that were actually positive, across all actually positive data points. It can be calculated with the expression

$$Sensitivity = \frac{TP}{TP + FN} \tag{1.11}$$

Specificity or the so-called true negative rate **(TNR)** indicates the rate of negative classified data points that were actually negative, across all actually negative data points. It can be calculated with the formula

$$Specificity = \frac{TN}{TN + FP} \tag{1.12}$$

False positive rate indicates the rate of the incorrectly positive classified data points across all data that was negative in reality. It is calculated as

$$FPR = \frac{FP}{FP + TN} \tag{1.13}$$

False negative rate represents the rate of the incorrectly negative classified data points across all data that was positive in reality. It can be calculated as

$$FNR = \frac{FN}{FN + TP} \tag{1.14}$$

Accuracy describes the rate of all correctly classified data points across all data set. It can be calculated with the expression

$$Accuracy = \frac{TN + TP}{FP + FN + TP + TN} \tag{1.15}$$

1.4.2. *The ROC curve.* The receiver operating characteristics curve illustrates the relationship between true positive rate (TPR) or sensitivity and the false positive rate (FPR) of a classification model. On the x-axis of the plot the FPR is displayed and on the y-axis the TPR. It can be thought of as a plot of the power as a function of the Type 1 Error of the classification. An ROC curve plots TPR and FPR at different classification thresholds. Lowering the threshold classifies more items as positive, thus increasing both false positives and true positives. In general a point is better than another point if its sensitivity is higher while its FPR is also smaller. The diagonal line on the plot illustrates totally randomized classification results. If a point is below this line, then it indicates that the classification result is worse than randomly picking a class Florkowski (2008).

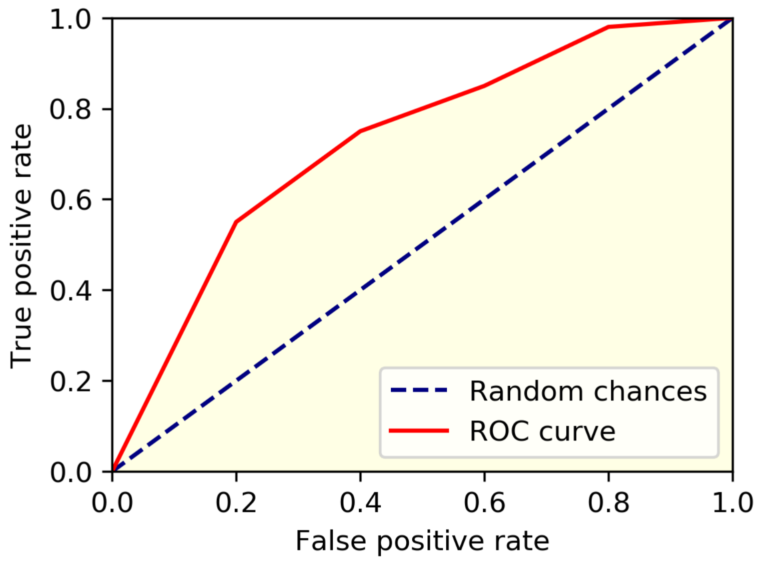


FIG 2. *An example of ROC Curve.*

From the ROC curve the area under the curve, **AUC**, can be derived. AUC represents the area under the ROC curve, which graphically and mathematically illustrates the accuracy of a classification model in successfully distinguishing different classes from each other. A perfect classification would result in the maximum AUC value of 1, while an AUC value lower than 0.5 would correspond to the are under the horizontal 45 degree line, which describes that the model is worse than random classification of classes. AUC is defined between 0 and 1, and a higher AUC value would indicate better classification performance.

2. Methods. The software R was used for all computations and graphics(version 4.2.1).

2.1. *Splitting the data.* Before the data was splitted, I wrote set.seed(123) for better reproducibility. With the commands `initial_split()`, `training()` and `testing()` from the package `rsample` the data was splitted into 80% and 20% test and training fractions. The training set contains 614 observations and the test set 154 observations of the 9 variables of the data set.

2.2. *GBM.* Next the **hyperparameter grid** was created by using the command `expand.grid()` from the package *GBM.* The hyperparameter grid serves for finding the most optimal parameters for the GBM fit. The parameters of the hyperparameter grid and the gbm model fit are **shrinkage**, which is the shrinkage parameter applied to each tree in the expansion. It serves for the so-called "learning rate" of the model fit. **Interaction.depth**, specifies the maximum depth of each tree. **Minnobsinnode** specifies the minimum number of observations in the terminal nodes of the trees. **Bag.fraction** specifies the fraction of the training set observations randomly selected to propose the next tree in the expansion. The parameters from the **hyperparameter grid** are being fit with a `for-loop` to a model and the parameters that produce the minimal *Root Mean Square Error* indicate that those are the best parameters for the optimal model.

Then with the tuned parameters from the `hyperparameter grid`, the GBM model was fit with the command `gbm()` from the package *gbm.* The aim of the fitted model is to classify whether a patient has diabetes or not. Since the target variable is binary, the family was set to "binomial".

2.3. *Prediction.* The next step was to make a prediction on the test set. With the commands `predict.gbm()` from the package *gbm* and `prediction()` from the package *ROCR*, two separate predictions were created on the test set, in order to be able to measure the model performance.

2.4. *Confusion Matrix.* Then in order to be able to build a confusion matrix of the model, I needed an optimal cutoff point, which was obtained by the command `optimalCutoff()` from the package *InformationValue.* With the command `ifelse()` and the obtained *cutoff point* a vector of class predictions was created whether the predicted probability was greater or less than the *cutoff point.* If it was greater, it would be classified as 1(=diabetes) and vice versa as 0(=no diabetes). Then with the command `table()` the confusion matrix was produced.

2.5. *Model performance.* The next step is to evaluate the gradient boosting model. From the object that was created by the command `optimalCutoff()` the TPR, FPR, specificity, missclassification error and accuracy of the model were delivered. Then with the command `performance()` from the package *ROCR* the *Precision/Recall Curve*, *Sensitivity/Specificity Curve*, the *AUC* value and the *ROC Curve* were plotted.

3. Results. The first expansion of the **hyperparameter grid** includes a total of 256 rows, which indicates that the model will be fitted 256 times to a GBM model including every single combination of the parameters. The first 5 rows from the results of the first round of the hyperparameter grid are displayed in the table.

shrinkage	interaction.depth	minnobsinnode	bag.fraction	optimal_trees	min_RMSE
0.3	1	15	0.4	43	0.9673
0.1	1	20	0.4	115	0.9739
0.05	1	5	0.8	294	0.9750
0.1	1	10	0.4	70	0.9757
0.05	1	15	0.4	119	0.9758

Since the resulting RMSE of the parameters are too close to each other, I decided to adjust the hyperparameter grid according to the resulting tuned parameters. The second expansion of the grid contains 256 rows, indicating that the model will be fitted with a `for-loop` 256 times again. The resulting parameters and their *RMSE* are displayed in the table.

shrinkage	interaction.depth	minnobsinnode	bag.fraction	optimal_trees	min_RMSE
0.3	1	15	0.2	43	0.9673
0.2	1	20	0.2	115	0.9739
0.1	1	5	0.6	294	0.9750
0.2	1	10	0.2	70	0.9757
0.1	1	15	0.2	119	0.9758

Although the parameters of the hyperparameter grid were modified, the resulting RMSE values did not change drastically. So the selected parameters are the ones in the first row of the table. Next, the GBM model was fit. GBM gives values to the parameters based on their relative influence on the target variable. The resulting relative influences are displayed in the table.

Variable	Relative Influence
Glucose	32.9562
Diabetes Pedigree Function	17.4596
BMI	13.8226
Age	13.1364
Insulin	9.0108
Pregnancies	6.2609
Skin Thickness	5.9127
Blood Pressure	1.4406

Next, with the help of the command `optimalCutoff()`, the optimal *cutoff point* for the predictions was calculated. This *cutoff point* should minimize the misclassification error, so I set optimizeFor= ”misclasserror”. The resulting confusion matrix is displayed in the figure 3.

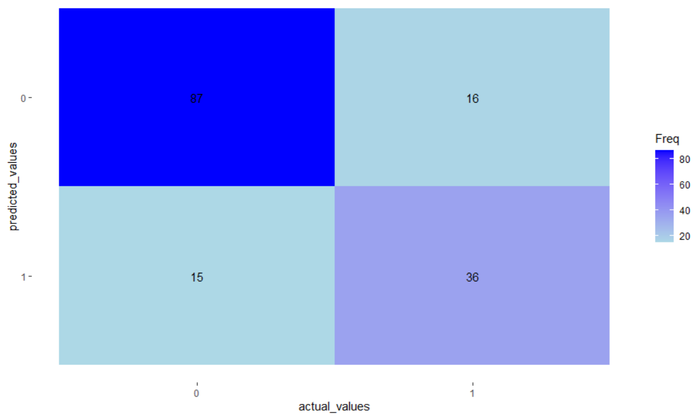


Fig 3. *The resulting confusion matrix.*

The TPR of the model is at 69.23%, with the FPR at 14.7%, specificity at 85.29% and accuracy at 79.87%. The AUC of the model is at 82.37%.

4. Discussion.

The aim of this work is to explain the Gradient Boosting Machines algorithm and demonstrate it on the real life diabetes data set. The model had a slightly over average performance predicting diabetes, with AUC being over 80%. The variable with the highest importance according to the GBM's variable selection is glucose, being the most important variable by far. The diabetes pedigree function, which is a function which scores likelihood of diabetes based on family history, is the second most important criteria to have diabetes based on the model estimate.

From the confusion matrix in figure 3 the conclusions can be delivered that the model is most successful at classifying the patients that don't have diabetes(=0). Sensitivity is at 70.59% and specificity is at 85.29%. The resulting ROC-curve is displayed in the figure 4.

One can drive s a main conclusion that GBM algorithm shows an over average performance at modeling and classifying the target variable although the hyper-parameters are being tuned twice by the *hyperparameter grid*. But the variable selection aspect of the GBM algorithm is a valuable feature, which can be helpful to understand unknown datasets better. However, considering that the AUC is at 82.37%, it still remains debatable if it is enough for such a classification.

References.

CHOUDHURY, A. (2021). AdaBoost vs gradient boosting: A comparison of leading boosting algorithms. https://analyticsindiamag.com/adaboost-vs-gradient-boosting-a-comparison-of-leading-boosting-algorithms/. Accessed: 2022-6-25.

DOBBIN, K. K. and SIMON, R. M. (2011). Optimally splitting cases for training and testing high dimensional classifiers. *BMC Med. Genomics* **4** 31.

FLORKOWSKI, C. M. (2008). Sensitivity, specificity, receiver-operating characteristic (ROC) curves and likelihood ratios: communicating the performance of diagnostic tests. *Clin. Biochem. Rev.* **29 Suppl 1** S83–7.

FREUND, Y. and SCHAPIRE, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *J. Comput. Syst. Sci.* **55** 119–139.

FRIEDMAN, J. (2000). Greedy Function Approximation: A Gradient Boosting Machine. *The Annals of Statistics* **29**.

FRIEDMAN, J. H. (2002). Stochastic Gradient Boosting. *Computational statistics and data analysis* **38** 367–378.

KEARNS, M. J. and VAZIRANI, U. V. (1994). *An introduction to computational learning theory*. MIT Press, London, England.

UCI MACHINE LEARNING Pima Indians Diabetes Database.

SMITH J. W. EVERHART J. E. DICKSON W. C. KNOWLER, W. C. . J. R. S. (1988). Using the ADAP learning algorithm to forecast the onset of diabetes mellitus. *IEEE Computer Society Press.* 261–265.

TING, K. M. (2017). *Confusion Matrix* In *Encyclopedia of Machine Learning and Data Mining* 260–260. Springer US, Boston, MA.

APPENDIX A: FIGURES AND TABLES

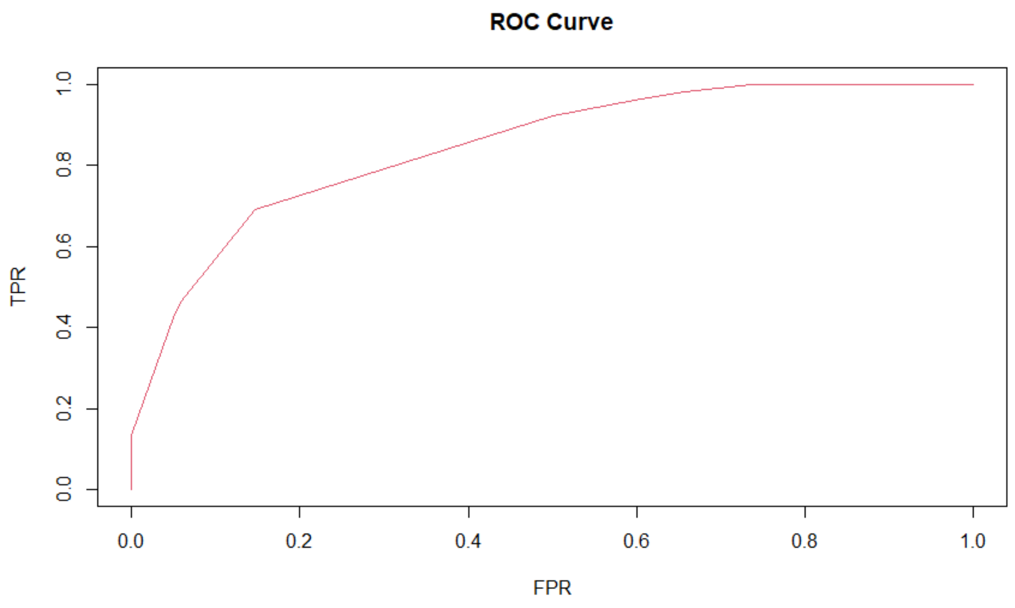


FIG 4. *The resulting ROC-curve on the test set. FPR is plotted against TPR.*

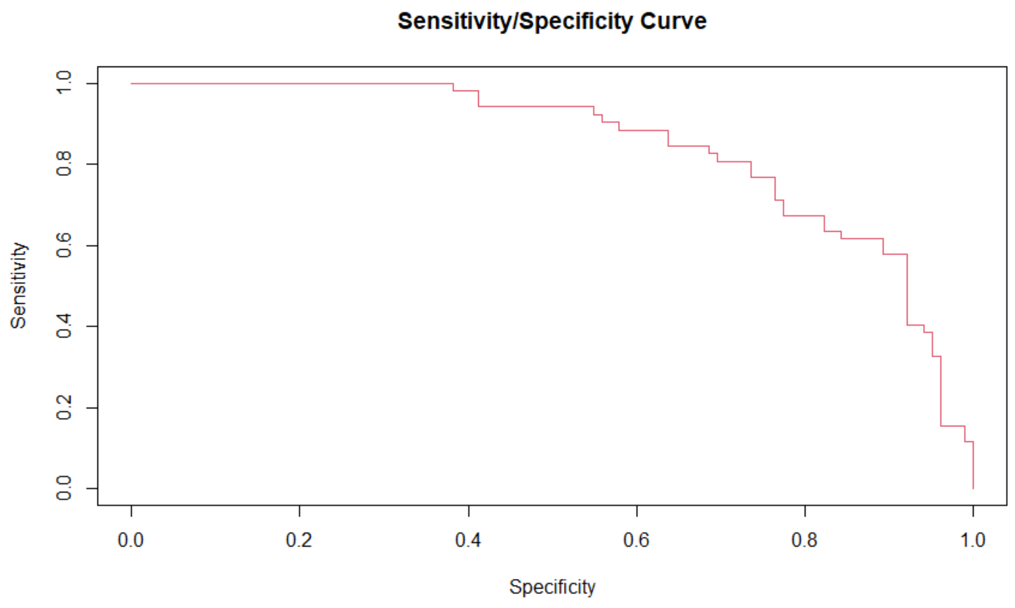


Fig 5. *The resulting sensitivity/specificity curve.*

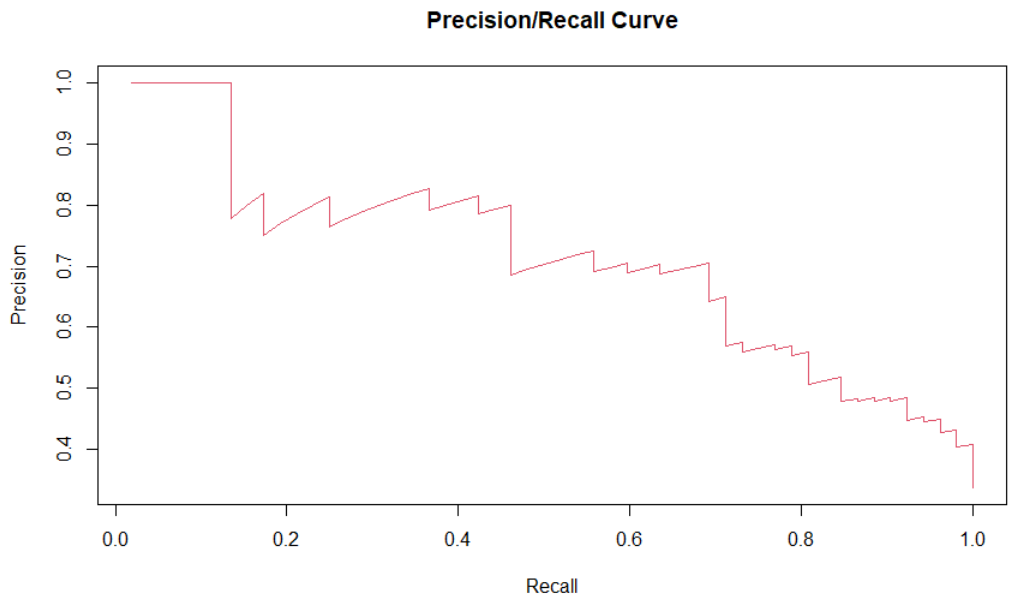


Fig 6. *The resulting precision/recall curve.*

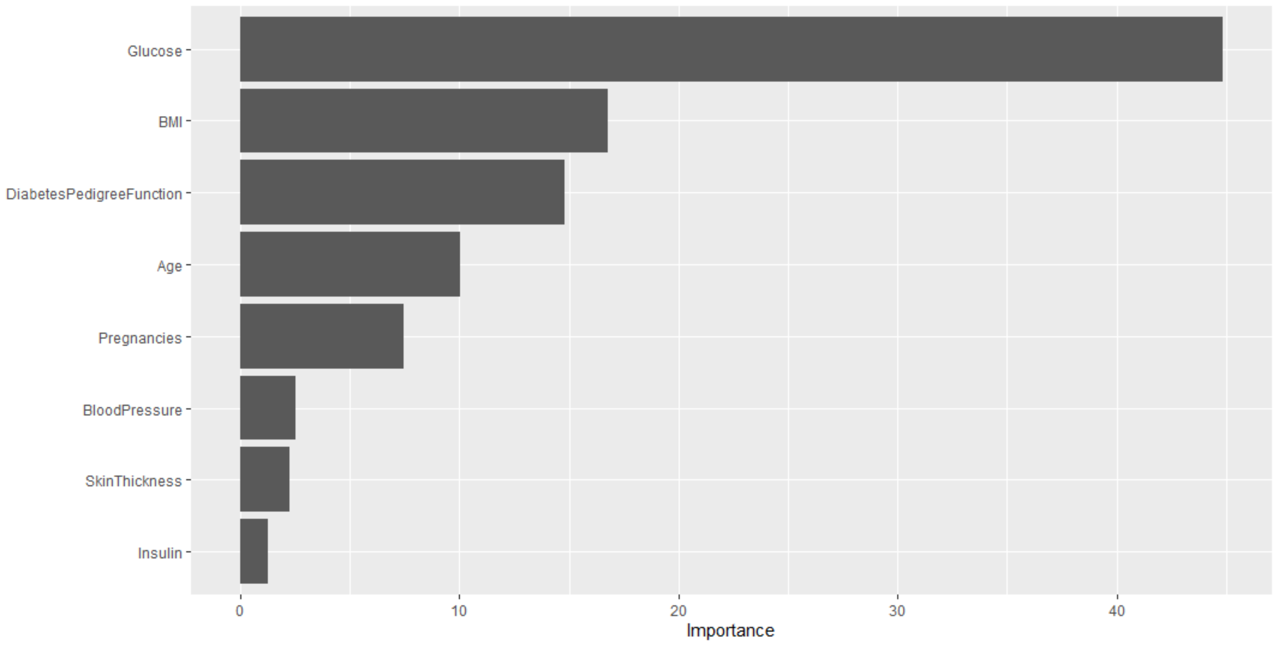


Fig 7. *The resulting relative importance plot of the variables.*

```
library(gbm)
library(dplyr)
library(vip)
library(pROC)
library(caret)
library(yardstick)
library(ggplot2)
library(InformationValue)
library(ROCR)
library(readr)
library(rsample)
```

Import and split the data

```
diabetes <- read_csv("diabetes.csv")
set.seed(123)                                  # for better reproducibility
diabetes_split <- initial_split(diabetes, prop = 0.8) #Split the data into 80% - 20%
diab_train <- training(diabetes_split)    # train_set
diab_test <- testing(diabetes_split)      #test_set
```

Data overview

```
head(diabetes, 3)
```

```
## # A tibble: 3 x 9
##   Pregnancies Glucose BloodPressure SkinThickness Insulin   BMI DiabetesPedigre~
##         <dbl>   <dbl>         <dbl>         <dbl>   <dbl> <dbl>            <dbl>
## 1           6     148            72            35       0  33.6            0.627
## 2           1      85            66            29       0  26.6            0.351
## 3           8     183            64             0       0  23.3            0.672
## # ... with 2 more variables: Age <dbl>, Outcome <dbl>
```

Hyperparameter Grid

```
hyper_grid <- expand.grid(
  shrinkage = c(.01, .05 , .1, .3 ),
  interaction.depth = c(1, 3, 5, 7),
  n.minobsinnode = c(5, 10, 15, 20),
  bag.fraction = c(.4 ,.6, .8, 1),
  optimal_trees = 0,                # a place to put results
  min_RMSE = 0                      # a place to put results
)
```

Total number of combinations

```
nrow(hyper_grid)
```

```
## [1] 256
```

```
# randomize data
random_index <- sample(1:nrow(diab_train), nrow(diab_train))
random_diab_train <- diab_train[random_index, ]
```

Grid Search

```
for(i in 1:nrow(hyper_grid)) {

  # reproducibility
  set.seed(123)

  # train model
  gbm.tune <- gbm(
    formula = Outcome ~ .,
    distribution = "bernoulli",
    data = random_diab_train,
    n.trees = 5000,
    interaction.depth = hyper_grid$interaction.depth[i],
    shrinkage = hyper_grid$shrinkage[i],
    n.minobsinnode = hyper_grid$n.minobsinnode[i],
    bag.fraction = hyper_grid$bag.fraction[i],
    train.fraction = .75,
    n.cores = NULL, # will use all cores by default
    verbose = FALSE
  )

  # add min training error and trees to grid
  hyper_grid$optimal_trees[i] <- which.min(gbm.tune$valid.error)
  hyper_grid$min_RMSE[i] <- sqrt(min(gbm.tune$valid.error))
}


hyper_grid %>% dplyr::arrange(min_RMSE) %>% head(10) #Extract the first 10 best combinations.
```

Adjusted Hyperparameter grid

```
hyper_grid2 <- expand.grid(
  shrinkage = c(.2, .05 , .1, .3 ),
  interaction.depth = c(1, 3, 5, 7),
  n.minobsinnode = c(5, 10, 15, 20),
  bag.fraction = c(.2, .4 ,.6, .8),
  optimal_trees = 0,               # a place to put results
  min_RMSE = 0                     # a place to put results
)
```

Total number of combinations

```
nrow(hyper_grid2)
```

```
## [1] 256
```

Adjusted Grid search

```r
for(i in 1:nrow(hyper_grid2)) {

  # reproducibility
  set.seed(123)

  # train model
  gbm.tune <- gbm(
    formula = Outcome ~ .,
    distribution = "bernoulli",
    data = random_diab_train,
    n.trees = 2000,
    interaction.depth = hyper_grid$interaction.depth[i],
    shrinkage = hyper_grid$shrinkage[i],
    n.minobsinnode = hyper_grid$n.minobsinnode[i],
    bag.fraction = hyper_grid$bag.fraction[i],
    train.fraction = .75,
    n.cores = NULL, # will use all cores by default
    verbose = FALSE
  )

  # add min training error and trees to grid
  hyper_grid2$optimal_trees[i] <- which.min(gbm.tune$valid.error)
  hyper_grid2$min_RMSE[i] <- sqrt(min(gbm.tune$valid.error))
}

hyper_grid2 %>% dplyr::arrange(min_RMSE) %>%head(10)
```

Final model

```r
gbm.fit.final <- gbm(
  formula = Outcome ~ .,
  distribution = "bernoulli",
  data = diab_train,
  n.trees = 43,
  interaction.depth = 1,
  shrinkage = 0.3,
  n.minobsinnode = 15,
  bag.fraction = .2,
  train.fraction = 1,
  n.cores = NULL, # will use all cores by default
  verbose = FALSE
)
```

Relative influence plot

```r
vip::vip(gbm.fit.final, title = "Diabetes")
```

Predictions

```r
pred_gbm <- predict.gbm(object = gbm.fit.final,
                        newdata = diab_test,
                        n.trees = 35,
                        type = "response")
predigbm <- prediction(pred_gbm, diab_test$Outcome)
```

Get the optimal Cutoff

```r
cutoff <- optimalCutoff(diab_test$Outcome, pred_gbm, optimiseFor = "misclasserror",
                        returnDiagnostics = TRUE)
```

Confusion matrix

```r
predicted_values <- ifelse(pred_gbm > cutoff$optimalCutoff, 1, 0)
actual_values <- diab_test$Outcome
cm <- table(predicted_values, actual_values)
cm
sum(diag(cm)) / sum(cm)
```

Visualizing

```r
cm1 <- conf_mat(cm, actual_values, predicted_values)
autoplot(cm1, type = "heatmap") +
  scale_fill_gradient(low = "lightblue", high = "blue") + theme(legend.position = "right")
```

Performance

```r
cutoff$TPR # True positive rate
cutoff$FPR # False positive rate
cutoff$Specificity # Specificity
cutoff$misclassificationError # Misclassification Error
1 - cutoff$misclassificationError #Accuracy
```

Precision/Recall curve

```r
perfprec <- performance(predigbm, "prec", "rec")
plot(perfprec, col=2, main="Precision/Recall Curve")
```

Sensitivity/Specificity curve

```r
perfsens <- performance(predigbm, "sens", "spec")
plot(perfsens, col=2, main="Sensitivity/Specificity Curve")
```

AUC and ROC

```r
performance(predigbm,"auc")@y.values
plot(performance(predigbm, "rch"), xlab="FPR", ylab="TPR", col=2,
     main="ROC Curve")
```