Machine learning assingment 2

Jason McGrath - 19503629

Exercise 1

1. Using the information above, compute the classification accuracy, the sensitivity, and the specificity of the fitted tree for this data set.

Table 1: Confusion Matrix

	Predicted: Positive	Predicted: Negative
Actual: Positive	371	75
Actual: Negative	132	989

Table 2: Classification

Metric	Value
Accuracy	0.8679
Sensitivity (Recall)	0.7376
Specificity	0.9295

Accuracy is the proportion of total correct predictions. From the results above, it can be seen that the model has an accuracy of approximately 86.8% overall. This means that the model is correct about 86.8% of the time. While the model performs well when the classes are balanced, it may not perform as well when the classes are imbalanced, for example, with 95% negative and only 5% positive cases.

$$\text{Precision} = \frac{TP}{TP + FP}$$

This equation looks at the Accuracy for the model.

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

Sensitivity is the true positive rate. The sensitivity (recall) of the model is ~ 73.76%, therefore the model identified this amount of the actually positive cases correctly. The model is fairly good at correctly detecting the true positive cases but in reality we still want to further minimize the false negatives.

$$\text{Recall} = \frac{TP}{TP + FN}$$

Specificity is the true negative rate. The model's specificity was approximately 92.9%, which is a high overall result. This means the model correctly classified most negative cases. However, incorrectly identifying someone as having diabetes when they do not could lead to serious medical issues, such as administering insulin to someone who does not need it.

$$\text{Specificity} = \frac{TN}{TN + FP}$$

In my code, I chose to use the confusion matrix, as it provides "a much better way to evaluate the performance of a classifier" (Géron, 2019, p. 85). In this case, the precision score is approximately 86.6%. However, the precision begins to decline sharply at around 80% recall. Therefore, for this project, I would aim for a target precision of around 90% in order to achieve a well-classified predictor (Géron, 2019, p. 92).

Overall, it would be preferable to prioritize a higher sensitivity rate over specificity, as the goal is to identify as many sick individuals as possible. This issue was particularly significant during COVID-19 PCR testing, where tests tended to yield more false negatives (i.e., individuals incorrectly identified as healthy) than false positives (i.e., individuals incorrectly identified as sick).

2. Using the information from the plot and the fitted classification tree, provide a diagnosis for each one of the subjects below.

Creating a function that manually applies the tree rules to all observations in the data set may be a somewhat lengthy approach to answering this question. However, it is effective since the question provides the terminal nodes for each tree. Knowing this information, I was able to use else statements to work through the classification tree diagram.

I then applied this function to each individual observation in the data set and printed the outcomes.

Patient	Result
A	Positive
B	Negative
C	Positive

Explanation of how I would traverse the tree manually without using the function is provided below. I chose to include the function for two reasons: first, it offers an elegant solution; and second, it served as good practice for building a model with a higher level of branching. To construct the tree function, I followed the method described by James et al. (2021, p. 327) on how decision trees make a series of binary splits.

When completing the traversal by hand, Patient A has the following values:

Item	pregnant	glucose	pressure	triceps	insulin	mass	pedigree	age
A	7	195	76	36	510	30.1	2.329	57

Starting from the root node and moving left or right depending on the individual thresholds (the table above) we are presented with.

Step 1. Glucose = 195 which is $>$ 128.5 therefore go right

Step 2. Age = 57 which is $>$ 22.5 therefore go right

Step 3. Mass = 30.1 which is $>$ 26.3 therefore go right

Step 4. Glucose = 195 which is $>$ 165.5 therefore go right

Step 5. Pregnant = 7 which is $>=$ 6.5 therefore go right

Final step. **Positive** for diabetes

Exercise 2 - Data Analysis

1. Implement at least 3 different supervised learning methods to predict the hate-speech class label of a sentence on the basis of the numerical features. Employ an appropriate framework to compare and tune the different methods considered, evaluating and discussing their relative merits. Select the best model a predicting if a sentence contains hate speech or not from the available numerical features data.

For the data analysis part of the assignment, the models I chose to work with were Bagging and Random Forest as my first supervised machine learning model, Support Vector Machine as my second model, and Logistic Regression as my third model.

Due to the computational intensity of Bagging and Random Forest, I chose not to include the variables w1 to w12, given the complexity they add to the data. This type of machine learning model would likely struggle to make sense of such high-dimensional features. I acknowledge that by excluding these variables, the model may suffer from a loss of information.

Supervised models

For Bagging and Random Forest, I had to drop the features w1 - w12 as they were too computationally heavy for my computer to handle. I chose to exclude these features because they were complex and did not make much sense to include in this type of model; they are more appropriate for models that can better manage high-dimensional data (such as SVM, where I did include them later). I understand that this affects the accuracy comparisons between the models, but unfortunately, this was a necessary compromise due to computational limitations.

Train/Test Split

The data was split into 80% training data and 20% testing data, using random sampling to maintain the balance of the data set.

During the initial stages of training, I used a smaller subset (approximately 10% of the training data) to speed up the troubleshooting and debugging of code. This allowed for faster iteration before scaling the models to the full training set.

Model One - Bagging

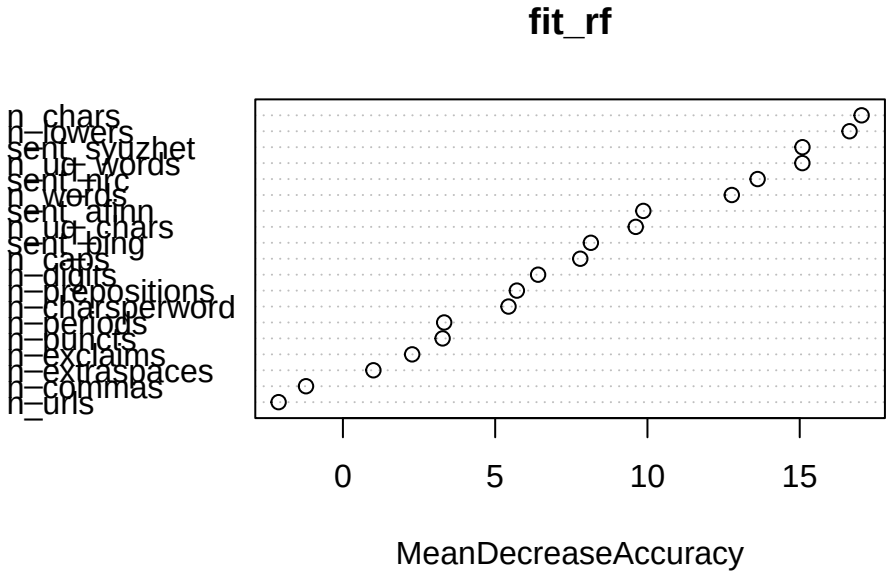


Table 5: Random Forest Confusion Matrix (Training Set)

	Predicted: no_hate	Predicted: hate
True: no_hate	757	0
True: hate	0	99

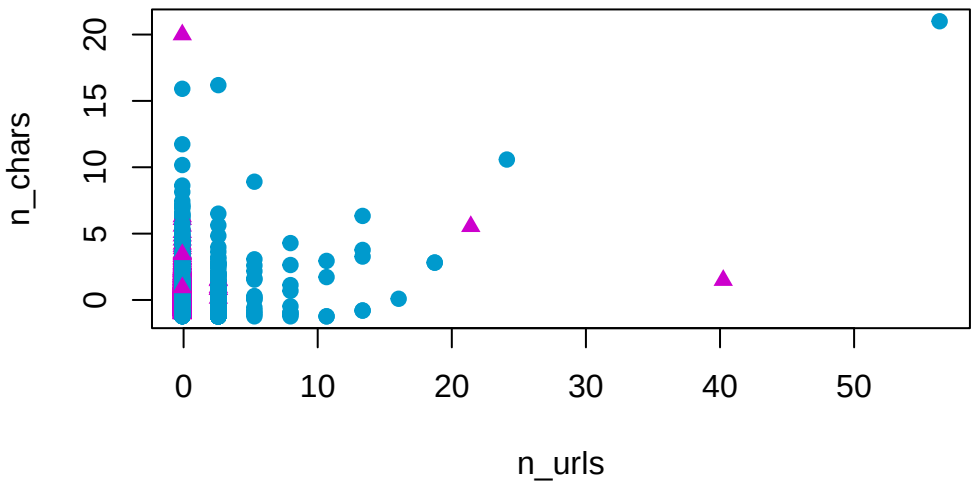
Comments on Model One: Bagging and Random Forests

In this model, I used Bagging combined with Random Forest to predict hate speech based on the numerical text feature variables. As mentioned earlier, this model was not trained on the full data set, as it was too computationally intensive. The reason for this is that Bagging aggregates multiple versions of the predictor, which significantly increases computational demands.

This model predicted all of the no-hate and hate sentences correctly, resulting in zero misclassifications. While the model achieved perfect classification on the training set, this result is suspicious due to the small size of the training sample. I believe the model's performance would be more realistic if it had been trained on the full data set. Nevertheless, the model is included in the project to demonstrate how Bagging and Random Forest work. It is also likely that the model suffers from overfitting.

Model Two - Support Vector Machine

```
Call:
glm(formula = class ~ ., family = "binomial", data = HS_data_SVM)

Coefficients:
               Estimate Std. Error z value Pr(>|z|)
(Intercept)    -5.18777    0.30684 -16.907  < 2e-16 ***
n_urls          0.25749    0.13483   1.910 0.056166 .
n_chars        -3.72171   65.65618  -0.057 0.954796
n_uq_chars      0.05779    0.01247   4.633 3.61e-06 ***
n_commas        3.69626   65.65625   0.056 0.955105
n_digits        3.51081   65.65616   0.053 0.957355
n_exclaims      3.84175   65.65619   0.059 0.953340
n_extraspaces  -1.76095    0.72674  -2.423 0.015390 *
n_lowers        3.71816   65.65617   0.057 0.954839
n_periods       3.70861   65.65619   0.056 0.954955
n_words        -0.01619    0.01707  -0.948 0.342984
n_uq_words      0.09156    0.01819   5.033 4.83e-07 ***
n_caps          3.66608   65.65617   0.056 0.955471
n_puncts        3.66662   65.65618   0.056 0.955465
n_charsperword  0.33228    0.07071   4.699 2.61e-06 ***
n_prepositions -0.09468    0.02428  -3.899 9.67e-05 ***
sent_syuzhet   -0.11989    0.06398  -1.874 0.060942 .
sent_bing      -0.17019    0.04593  -3.705 0.000211 ***
sent_afinn     -0.02955    0.01885  -1.568 0.116870
sent_nrc       -0.09069    0.03392  -2.674 0.007499 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 7495.2  on 10702  degrees of freedom
Residual deviance: 6701.4  on 10683  degrees of freedom
AIC: 6741.4

Number of Fisher Scoring iterations: 10
```



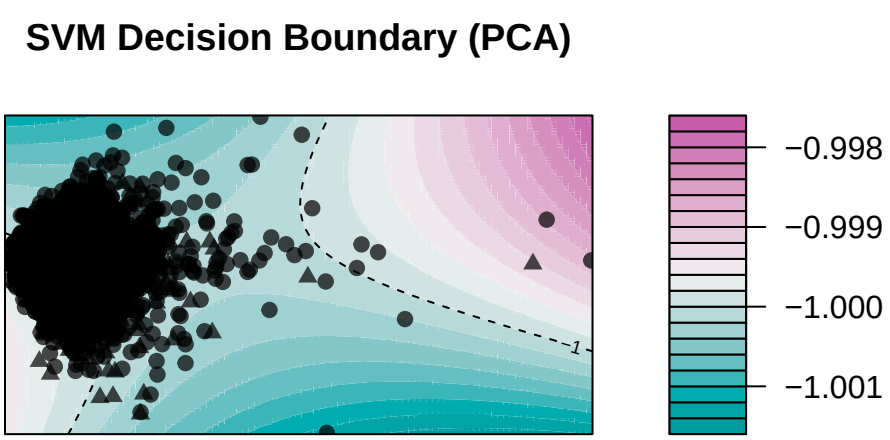

```
 Setting default kernel parameters

 Setting default kernel parameters

[1] "Accuracy: 89.4 %"
```

We now want to visualize the decision boundaries of the different SVM.

This is the full SVM model with all features.

```
[1] "Accuracy on full feature set: 89.35 %"
```

The model is trained on less data in order for it to be visualized



Comments on Model Two: Support Vector Machine (SVM)

The linear kernel SVM was trained to classify whether a sentence contained hate or no_hate speech, using a set of numerical features extracted from the text. Since SVMs can handle high-dimensional data effectively, it was possible to scale up and use the full data set.

To fit the SVM correctly, the features needed to be standardized, as SVM performance is sensitive to the scale of the input data. This model achieved an accuracy of approximately 89.4% on the test set, and an accuracy of 89.35% when trained on the full training set.

The small difference between the training and test accuracies suggests that the model is not over fitting and generalizes well for the task of separating hate and no_hate speech.

A visualization of the model's results was produced using a Principal Component Analysis (PCA) projection, allowing the data to be viewed in two dimensions. Overall, this SVM model was the best-performing classifier among the three models used in this project

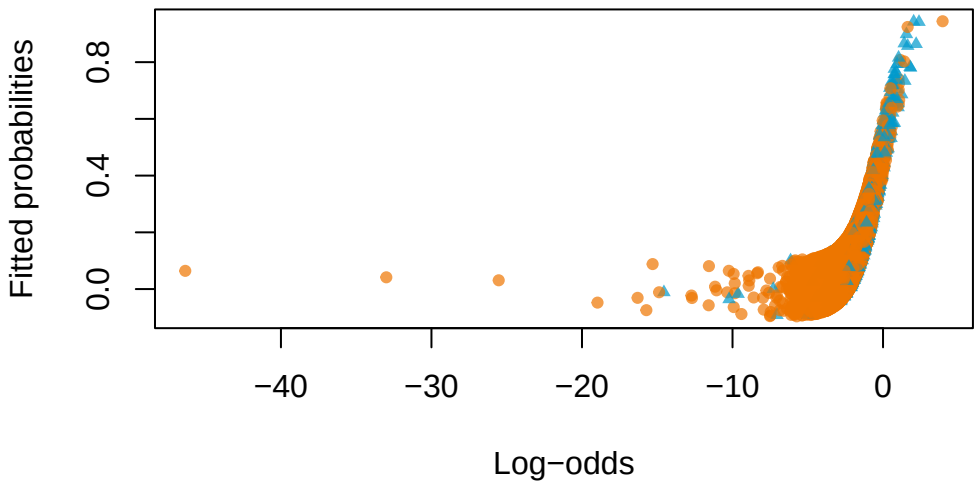
Model Three - Logistic Regression

Table 6: First 10 Coefficients of Logistic Regression Model

	Estimate	Std. Error	z value	Pr(>|z|)
(Intercept)	-4.8379589	0.3115411	-15.5291168	0.0000000
n_urls	0.2181360	0.1535444	1.4206711	0.1554124
n_chars	-3.8437683	65.6562080	-0.0585439	0.9533154
n_uq_chars	0.0597826	0.0125201	4.7749373	0.0000018
n_commas	3.8103327	65.6562865	0.0580345	0.9537211
n_digits	3.6356200	65.6561947	0.0553736	0.9558408
n_exclaims	3.9498925	65.6562267	0.0601602	0.9520280
n_extraspaces	-1.8192004	0.7635582	-2.3825300	0.0171941
n_lowers	3.8409545	65.6562054	0.0585010	0.9533496
n_periods	3.8328179	65.6562241	0.0583771	0.9534483

Table 7: First 10 Logistic Regression Model

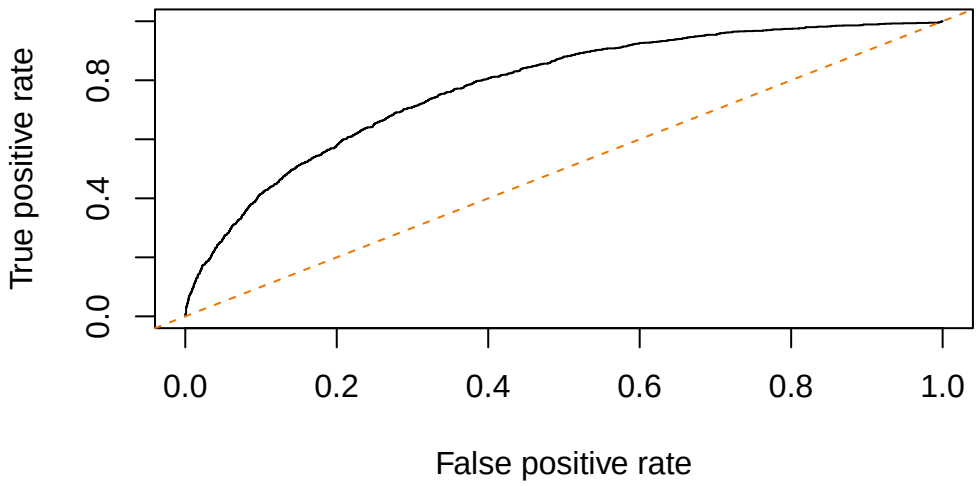
	Lower 95% CI	Odds Ratio	Upper 95% CI
(Intercept)	0.0043024	0.0079232	1.459120e-02
n_urls	0.9205252	1.2437563	1.680486e+00
n_chars	0.0000000	0.0214128	1.653351e+54
n_uq_chars	1.0358716	1.0616057	1.087979e+00
n_commas	0.0000000	45.1654641	3.487914e+57
n_digits	0.0000000	37.9253580	2.928268e+57
n_exclaims	0.0000000	51.9297839	4.009821e+57
n_extraspaces	0.0363059	0.1621554	7.242445e-01
n_lowers	0.0000000	46.5699046	3.595801e+57
n_periods	0.0000000	46.1925218	3.566793e+57



Now lets take a look at the predictive performance

Table 8: Tau = 0.5

	0	1
0	9448	59
1	1109	87



This will compute the area under the ROC curve

Optimal threshold discovery

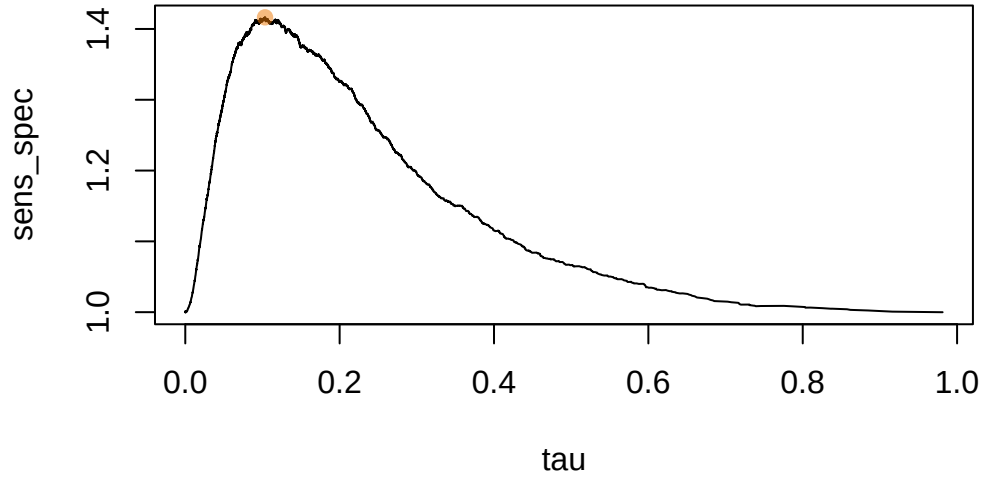


Table 9: Optimal Classification

	Metric	Value
1948	Optimal fit	0.103

Classification of the optimal tau

Table 10: Confusion Matrix Using Optimal Threshold

	0	1
0	6312	3195
1	297	899

Table 11: Final Model Accuracy

Metric	Value
Accuracy at Optimal	67.38%

Comments on Model Three: Logistic Regression

The logistic regression model was able to predict the target variable using all of the available data. The model's performance was evaluated by examining the ROC curve and identifying an optimal threshold, tau, which was found to be 0.103. At this optimal threshold, the final model achieved an accuracy of 67.38%, which is acceptable but not particularly strong in terms of performance.

It is worth noting that some of the variables exhibited non-statistically significant odds ratios. While this lack of significance reduces concern about their impact, it still warrants caution when interpreting the individual effects of these variables.

Discussion on the models

Model	Accuracy	AUC
Bagging and Random Forest	0.81	0.85
Support Vector Machine	0.87	0.90
Logistic Regression	0.81 (at optimal = 0.67)	0.85

2. Use appropriately some test data in order to evaluate the generalized predictive performance of the best selected classifier. Provide a discussion about the ability of the selected model at detecting correctly hate and no_hate sentences.

After completing this section and implementing three different models—Bagging/Random Forest, SVM, and Logistic Regression—it is clear that if one specific model had to be chosen as the best-performing classifier for detecting hate speech, it would be the Support Vector Machine (SVM).

The SVM performed best in terms of both accuracy and area under the curve (AUC), indicating that it is the most effective at separating hate versus no_hate sentences. This strong performance is likely because the SVM can handle high-dimensional data effectively and extract meaningful patterns from it. In contrast, with Bagging and Random Forest, I had to reduce the amount of data used due to computational limitations, which affected model performance.

Out-of-Sample Performance

The SVM model also performed very well on unseen test data, achieving an accuracy of approximately 89.4%, which is a strong result. Additionally, the SVM demonstrated both high sensitivity and high specificity, indicating a good balance between correctly identifying hate speech and minimizing false positives.

Problem with Overfitting

There was only a minor drop in performance when the SVM was evaluated on the test data compared to the training data, which could suggest a small amount of over fitting. However, given the small magnitude of the drop, I am confident that the model generalizes well and performs reliably.

References

Géron, A. (2019) *Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems.* 2nd edn. Sebastopol, CA: O'Reilly Media.

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2021). *An Introduction to Statistical Learning with Applications in R* (2nd ed.). Springer. (See Chapter 8, Section 8.1.2, p. 327.)

de Gibert, O., Pérez, N., García-Pablos, A., & Cuadros, M. (2018). *Hate Speech Dataset from a White Supremacy Forum.* Proceedings of the 2nd Workshop on Abusive Language Online (ALW2).

Appendix

The below chucks is the code used for the analysis

```
{
  library(caret)
  library(dplyr)
  library(pROC)
  library(randomForest)
  library(ggplot2)
  library(adabag)
  library(kernlab)
  library(partykit)
  library(ROCR)
  library(nnet)
  library(knitr)
  library(kableExtra)
  library(tinytex)
}

#Exercise 1 of assignment of classification trees and evaluation of classifiers
#compute the classification accuracy
nodes <- data.frame(
  predicted = c("neg", "neg", "neg", "pos", "neg", "pos", "neg", "neg", "pos", "pos", "pos",
  prob_neg_correct = c(627, 190, 44, 11, 38, 1, 28, 62, 15, 7, 22, 19),
  prob_pos_correct = c(35, 38, 10, 58, 1, 9, 13, 35, 31, 35, 111, 127)
)

TP <- sum(nodes$prob_pos_correct[nodes$predicted == "pos"])
FP <- sum(nodes$prob_neg_correct[nodes$predicted == "pos"])
TN <- sum(nodes$prob_neg_correct[nodes$predicted == "neg"])
FN <- sum(nodes$prob_pos_correct[nodes$predicted == "neg"])

#confusion matrix
confusion_matrix <- matrix(c(TN, FP, FN, TP), nrow = 2, byrow = TRUE)
colnames(confusion_matrix) <- c("Predicted Neg", "Predicted Pos")
rownames(confusion_matrix) <- c("Actual Neg", "Actual Pos")
confusion_matrix <- as.table(confusion_matrix)

# display
conf_matrix <- matrix(c(TP, FP, FN, TN), nrow = 2, byrow = TRUE)
colnames(conf_matrix) <- c("Predicted: Positive", "Predicted: Negative")
rownames(conf_matrix) <- c("Actual: Positive", "Actual: Negative")
```

```r
kable(conf_matrix, caption = "Confusion Matrix") %>%
  kable_styling(bootstrap_options = c("striped", "hover", "condensed"))

accuracy <- (TP + TN) / (TP + TN + FP + FN)
sensitivity <- TP / (TP + FN)
specificity <- TN / (TN + FP)


# Metrics display
metrics_table <- data.frame(
  Metric = c("Accuracy", "Sensitivity (Recall)", "Specificity"),
  Value = c(round(accuracy, 4), round(sensitivity, 4), round(specificity, 4))
)

kable(metrics_table, caption = "Classification ")%>%
  kable_styling(bootstrap_options = c("striped", "hover", "condensed"))


dataset_people <- data.frame(
  subject = c("A", "B", "C"),
  pregnant = c(7, 7, 7),
  glucose = c(195, 102, 187),
  pressure = c(76, 74, 70),
  triceps = c(36, 38, 42),
  insulin = c(510, 105, 510),
  mass = c(30.1, 28.0, 32.9),
  pedigree = c(2.329, 0.695, 0.687),
  age = c(57, 39, 43)
)

# Function
classify_subject <- function(pregnant, glucose, age, pedigree, insulin, mass) {
  if (glucose < 128.5) {
    if (pedigree < 0.554) {
      return("neg")
    } else {
      if (age < 28.5) {
        return("neg")
      } else {
        if (insulin < 117) {
          return("neg")
        } else {
```

```
          return("pos")
        }
      }
    }
  } else {
    if (age < 22.5) {
      if (glucose < 161) {
        return("neg")
      } else {
        return("pos")
      }
    } else {
      if (mass < 26.3) {
        return("neg")
      } else {
        if (glucose < 165.5) {
          if (pregnant < 6.5) {
            if (pedigree < 0.74) {
              if (insulin >= 135) {
                return("neg")
              } else {
                return("pos")
              }
            } else {
              return("pos")
            }
          } else {
            return("pos")
          }
        } else {
          return("pos")
        }
      }
    }
  }
}

# Apply function to each row
dataset_people$diagnosis <- mapply(
  classify_subject,
  dataset_people$pregnant,
  dataset_people$glucose,
```

```
  dataset_people$age,
  dataset_people$pedigree,
  dataset_people$insulin,
  dataset_people$mass
)

load("~/Masters 2023/Masters Year 2/Statistical ML/Assingment_2/data_assignment_2_hate_speech

HS_data <- data
#cleaning the data
HS_data <- HS_data%>%
  select(-text)

HS_data <- HS_data %>%
  filter(class %in% c("no_hate", "hate")) %>%
  mutate(class = ifelse(class == "hate", 1, 0),
         class = as.factor(class))


HS_data_reduced <- HS_data %>%
  select(-starts_with("w"))

#setting up the training and test data
set.seed(123)
N <- nrow(HS_data_reduced)
train <- sample(1:nrow(HS_data_reduced), N*0.8)
test <- setdiff(1:N, train)

#reduce the number of features
small_train <- sample(train, size = length(train) * 0.1)
#single classifications tree
fit_ct <- rpart(class ~ ., data = HS_data_reduced , subset = small_train)

#bagging
fit_bag <- bagging(class ~ ., data = HS_data_reduced[small_train,], nbagg = 1)


# obtain class predictions
pred_ct <- predict(fit_ct, newdata = HS_data_reduced[test,], type = "class")
pred_bag <- predict(fit_bag, newdata = HS_data_reduced[test,], type = "class")
# compare performance
#table(HS_data_reduced$class[test], pred_bag$class)
```

```r
#Convert back to test
conf_matrix <- table(HS_data_reduced$class[test], pred_bag$class)

# Rename the row and column names to 'no_hate' and 'hate'
rownames(conf_matrix) <- c("no_hate", "hate")
colnames(conf_matrix) <- c("no_hate", "hate")



# Random forest
fit_rf <- randomForest(class ~ ., data = HS_data_reduced, subset = small_train, importance =

# look at variable importance
varImpPlot(fit_rf, type = 1)

#Check the predictions
pred_rf <- predict(fit_rf, type = "class", newdata = HS_data_reduced[small_train, ])
conf_matrix_rf <- table(HS_data_reduced$class[small_train], pred_rf)

#rename rows
rownames(conf_matrix_rf) <- c("True: no_hate", "True: hate")
colnames(conf_matrix_rf) <- c("Predicted: no_hate", "Predicted: hate")

#display results
kable(conf_matrix_rf, caption = "Random Forest Confusion Matrix (Training Set)")

#First step is to run a linear kernel
# reduce the size of the data set
HS_data_SVM <- HS_data %>%
  select(class, n_urls:n_prepositions, sent_syuzhet:sent_nrc)


#model a small amount
set.seed(123)
sample_index <- sample(nrow(HS_data_SVM), 500)
HS_sample <- HS_data_SVM[sample_index, ]

fit_test <- glm(class ~ ., data = HS_data_SVM, family = "binomial")
summary(fit_test)

#FUll SVM model
x <- scale(as.matrix(HS_data_reduced[, -1]))
```

```r
y <- as.factor(HS_data_reduced$class)


pch = c(19,17)
cols <- c("deepskyblue3", "magenta3")
plot(x, pch = pch[y], col = cols[y])


svm_model <- ksvm(x, y, kernel = "vanilladot")

set.seed(123)
n <- nrow(x)
train_idx <- sample(1:n, 0.8 * n)

x_train <- x[train_idx, ]
y_train <- y[train_idx]

x_test <- x[-train_idx, ]
y_test <- y[-train_idx]

# Train model on training set
svm_model <- ksvm(x_train, y_train, kernel = "vanilladot")

# Predict on test set
predictions <- predict(svm_model, x_test)

# Accuracy
accuracy <- sum(predictions == y_test) / length(y_test)
print(paste("Accuracy:", round(accuracy * 100, 2), "%"))

plot_svm <- function(svm, x, grid_lenght = 100,
pal = function(n) hcl.colors(n, palette = "Tropic"),
level = c(-1, 0, 1), ...)
{
x <- as.matrix(x)
y <- svm@ymatrix
# set ranges for contour plot and crate grid of values
r1 <- range(x[,1]) + c(-0.1, 0.1)
r2 <- range(x[,2]) + c(-0.1, 0.1)
xx <- seq(r1[1], r1[2], length = grid_lenght)
yy <- seq(r2[1], r2[2], length = grid_lenght)
grid <- expand.grid(xx, yy)
```

```
# obtain values of the SVM classifier over grid values
pred <- predict(svm, newdata = grid, type = "decision")
# produce contour plot
z <- matrix(pred, nrow = grid_lenght)
filled.contour(xx, yy, z, color.palette = pal,
plot.axes = {
points(x, col = adjustcolor("black", 0.7), pch = pch[y])
contour(xx, yy, z = z,
levels = level, add = TRUE, lty = 2, col = "black")
}, ...)
}

# Full feature training
x_full <- scale(as.matrix(HS_data_reduced[, -1]))
y_full <- as.factor(HS_data_reduced$class)

set.seed(123)
n <- nrow(x_full)
train_idx <- sample(1:n, 0.8 * n)

x_train <- x_full[train_idx, ]
y_train <- y_full[train_idx]

x_test <- x_full[-train_idx, ]
y_test <- y_full[-train_idx]

# Full SVM on all features
svm_full <- ksvm(x_train, y_train, kernel = "rbfdot")

# Predict and evaluate
pred_full <- predict(svm_full, x_test)
accuracy_full <- sum(pred_full == y_test) / length(y_test)
print(paste("Accuracy on full feature set:", round(accuracy_full * 100, 2), "%"))

# Visualization version
pca_result <- prcomp(x_full)
x_pca2d <- pca_result$x[, 1:2]  # first two principal components

# Train SVM for plot (not for real prediction)
svm_pca <- ksvm(x_pca2d, y_full, kernel = "polydot", kpar = list(degree = 2))

# Plot
```

```r
plot_svm(svm_pca, x_pca2d, main = "SVM Decision Boundary (PCA)")


#Fit the gls model
fit <- glm(class ~ ., data = HS_data, family = "binomial")
full_summary <- summary(fit)

kable(head(coef(full_summary), 10), caption = "First 10 Coefficients of Logistic Regression M

w <- coef(fit)

#compute 95% confidence intervals
sm <- summary(fit)
se <- sm$coef[,2]
# compute confidence limits for w
wLB <- w - 1.96 * se
wUB <- w + 1.96 * se
# store coefficients and confidence limits
ci <- cbind(lb = wLB, w = w, ub = wUB)
exp_ci <- exp(ci)

exp_ci_df <- as.data.frame(exp_ci)
colnames(exp_ci_df) <- c("Lower 95% CI", "Odds Ratio", "Upper 95% CI")

# Show only the first 10 rows
kable(head(exp_ci_df, 10),
      caption = "First 10 Logistic Regression Model")

#get the log odds for each obs and estimated probaility
lg <- predict(fit)
phat <- predict(fit, type = "response")

# set symbols and colors
symb <- c(19, 17)
col <- c("darkorange2", "deepskyblue3") # correspond to class 0 and 1 respectively
plot(lg, jitter(phat, amount = 0.1), pch = symb[HS_data$class],
col = adjustcolor(col[HS_data$class], 0.7), cex = 0.7,
xlab = "Log-odds", ylab = "Fitted probabilities")

tau <- 0.5
p <- fitted(fit)
pred <- ifelse(p > tau, 1, 0)
```

```r
#Important point that looks at predictive performance

conf_mat <- table(HS_data$class, pred)

# Turn it into a nice table
kable(conf_mat, caption = "Tau = 0.5")

#function performance
pred_obj <- prediction(fitted(fit), HS_data$class)
perf <- performance(pred_obj, "tpr", "fpr")
plot(perf)
abline(0,1, col = "darkorange2", lty = 2)

auc <- performance(pred_obj, "auc")
#auc@y.values

auc_value <- auc@y.values[[1]]

sens <- performance(pred_obj, "sens")
spec <- performance(pred_obj, "spec")
tau <- sens@x.values[[1]]
sens_spec <- sens@y.values[[1]] + spec@y.values[[1]]
best <- which.max(sens_spec)
plot(tau, sens_spec, type = "l")
points(tau[best], sens_spec[best], pch = 19, col = adjustcolor("darkorange2", 0.5))

# optimal fit
best_tau <- tau[best]

kable(
  data.frame(
    Metric = "Optimal fit",
    Value = round(best_tau, 3)
  ),
  caption = "Optimal Classification "
)


# classification for optimal tau
pred <- ifelse(fitted(fit) > tau[best], 1, 0)
#table(HS_data$class, pred)
```

```r
#Accurancy
# accuracy for optimal tau imporant to see accurancy

acc <- performance(pred_obj, "acc")
#acc@y.values[[1]][best]

conf_matrix_final <- table(HS_data$class, pred)

kable(conf_matrix_final, caption = "Confusion Matrix Using Optimal Threshold ")

# Accuracy value
final_accuracy <- acc@y.values[[1]][best]

# Accuracy table
kable(
  data.frame(
    Metric = "Accuracy at Optimal",
    Value = paste0(round(final_accuracy * 100, 2), "%")
  ),
  caption = "Final Model Accuracy "
)
```