CSCI E-106:Assignment 10

Ian Kelk

Contents

```r
# Ensure required packages are installed, then load them
options(repos = c(CRAN = "https://cloud.r-project.org"))

req_pkgs <- c("faraway", "dplyr", "tibble", "tidyr", "ggplot2",
              "knitr", "caret", "ResourceSelection", "tree", "rpart",
              "rpart.plot", "randomForest", "C50", "xgboost", "adabag")

to_install <- setdiff(req_pkgs, rownames(installed.packages()))
if (length(to_install)) {
  install.packages(to_install, dependencies = TRUE)
}

# Load quietly
invisible(lapply(req_pkgs, function(p) {
  suppressPackageStartupMessages(library(p, character.only = TRUE))
}))
```

```r
confmat_kable <- function(cm, caption) {
  cm$table %>%
    as.data.frame() %>%
    as_tibble() %>%
    knitr::kable(caption = caption)
}

confmat_heatmap <- function(cm, title) {
  cm$table %>%
    as.data.frame() %>%
    as_tibble() %>%
    ggplot(aes(x = Reference, y = Prediction, fill = Freq)) +
    geom_tile() +
    geom_text(aes(label = Freq), size = 3) +
    scale_fill_gradient(low = "white", high = "steelblue") +
    coord_equal() +
    labs(
      title = title,
      x = "True class",
      y = "Predicted class"
    ) +
    theme_minimal() +
    theme(
      plot.title   = element_text(hjust = 0.5, size = 10),
      axis.title   = element_text(size = 9),
      axis.text    = element_text(size = 8),
      legend.title = element_text(size = 8),
      legend.text  = element_text(size = 7)
    )
}
```

Due Date: November 21, 2025 at 11:59 pm EST

Instructions

Students should submit their reports on Canvas. The report needs to clearly state what question is being solved, step-by-step walk-through solutions, and final answers clearly indicated. Please solve by hand where appropriate.

Please submit two files: (1) a R Markdown file (.Rmd extension) and (2) a PDF document, word, or html generated using knitr for the .Rmd file submitted in (1) where appropriate. Please, use RStudio Cloud for your solutions.

Problem 1

Use the wbca data(copy and paste the following command into R console:library(faraway);data("wbca"). The dataset wbca comes from a study of breast cancer in Wisconsin. There are 681 cases of potentially cancerous tumors of which 238 are actually malignant. Determining whether a tumor is really malignant is traditionally determined by an invasive surgical procedure. The purpose of this study was to determine whether a new procedure called fine needle aspiration, which draws only a small sample of tissue, could be effective in determining tumor status. Use 70% of the data for train data set and use remaining data (30% of data) for test data set (use set.seed(1023)). (50 points, 10 points each)

a-) Fit a binary regression with Class as the response and the other nine variables as predictors on the train data set. Report the residual deviance and associated degrees of freedom. Can this information be used to determine if this model fits the data? Explain.

b-) Use AIC as the criterion to determine the best subset of variables. Perform Hosmer-Lemeshow Goodness of Fit Test and comment (Use the step function.)

c-) Suppose that a cancer is classified as benign if $p > 0.5$ and malignant if $p < 0.5$. Compute the confusion matrix and accuracy rates if this method is applied to the train data with the reduced model.

d-) Suppose we change the cutoff to 0.9 so that $p < 0.9$ is classified as malignant and $p > 0.9$ as benign. Compute the confusion matrix.

e-) Repeat par c and d on the test data set and comment on the model performance and which cutoff will you choose?

Problem 2

Use train and test data set in problem 1 to answer the questions below.(50 points) a-) Fit a decision tree model to predict Class by using the other nine variables as predictors on the train data set by using xgboost, adaboost, random forest, boosting in library C5.0, tree and rpart libraries and methods. Calculate the confusing matrix for each method and comment on your findings. (30 points)

b-) Calculate the confusing matrixes on the test data set and compare against the logistic regression model confusing matrix. Which model would you choose? (20 points)

We analyze the Wisconsin breast cancer (wbca) data using logistic regression and tree-based classification methods. The goal is to model the probability that a tumor is benign based on nine cytological predictors and to compare logistic regression with several tree and ensemble methods on training and test sets.

AS THIS IS A SECTION NOTEBOOK EXAMPLE, WE WILL NOT BE USING THE `wbca` DATA, AND INSTEAD WILL USE `pima`

Problem 1: Logistic regression on the pima data

We begin by fitting and assessing logistic regression models on the pima data. Throughout, we treat *test* as a binary outcome (negative vs positive), and we interpret model coefficients in terms of log-odds and probabilities. We use a 70/30 train/test split with a fixed random seed for reproducibility.

For this session, we're going to use the `pima` dataset

- 768 observations on 8 predictors and 1 binary response.

Predictors:

- `pregnant` – number of times pregnant
- `glucose` – plasma glucose concentration
- `diastolic` – diastolic blood pressure
- `triceps` – triceps skin fold thickness
- `insulin` – 2-hour serum insulin
- `bmi` – body mass index
- `diabetes` – diabetes pedigree function
- `age` – age in years

Response:

- `test` – indicator of whether the patient shows signs of diabetes (coded 0 if negative, 1 if positive)

Data loading and splitting

In this section we load the `pima` data from the `faraway` package, convert it to a tibble for convenient handling, and recode the response so that `test` is a factor with `negative` and `positive` labels for modeling. We then inspect the basic structure of the data and the class balance, set the random seed, and create a 70/30 train/test split using stratified sampling on `test`.

```r
# Load the pima data
data("pima", package = "faraway")

pima_raw <- pima

# Convert to tibble and recode test as a factor with labels
pima <- pima_raw %>%
  as_tibble() %>%
  mutate(
    test = factor(
      test,
      levels = c(0, 1),
      labels = c("negative", "positive")
    )
  )

# Quick structural summary and outcome distribution
glimpse(pima)
```

```
## Rows: 768
## Columns: 9
## $ pregnant  <int> 6, 1, 8, 1, 0, 5, 3, 10, 2, 8, 4, 10, 10, 1, 5, 7, 0, 7, 1, ~
## $ glucose   <int> 148, 85, 183, 89, 137, 116, 78, 115, 197, 125, 110, 168, 139~
## $ diastolic <int> 72, 66, 64, 66, 40, 74, 50, 0, 70, 96, 92, 74, 80, 60, 72, 0~
## $ triceps   <int> 35, 29, 0, 23, 35, 0, 32, 0, 45, 0, 0, 0, 0, 23, 19, 0, 47, ~
## $ insulin   <int> 0, 0, 0, 94, 168, 0, 88, 0, 543, 0, 0, 0, 0, 846, 175, 0, 23~
## $ bmi       <dbl> 33.6, 26.6, 23.3, 28.1, 43.1, 25.6, 31.0, 35.3, 30.5, 0.0, 3~
## $ diabetes  <dbl> 0.627, 0.351, 0.672, 0.167, 2.288, 0.201, 0.248, 0.134, 0.15~
## $ age       <int> 50, 31, 32, 21, 33, 30, 26, 29, 53, 54, 30, 34, 57, 59, 51, ~
## $ test      <fct> positive, negative, positive, negative, positive, negative, ~
```

```r
test_counts <- pima %>%
  count(test) %>%
  mutate(proportion = n / sum(n))

knitr::kable(
  test_counts,
  caption = "Test outcome counts and proportions in the full pima data"
)
```

Table 1: Test outcome counts and proportions in the full pima data

test	n	proportion
negative	500	0.6510417
positive	268	0.3489583

We now set the random seed, create a 70/30 train/test split stratified by `test`, and report the sizes of the resulting samples.

```r
set.seed(1023)

train_index <- caret::createDataPartition(
  y = pima$test,
  p = 0.7,
  list = FALSE
)

pima_train <- pima[train_index, ] %>% as.data.frame()
pima_test  <- pima[-train_index, ] %>% as.data.frame()

# Training/test sizes and shares (should be approx 0.70 / 0.30)
train_test_sizes <- tibble(
  sample = c("Training", "Test"),
  n      = c(nrow(pima_train), nrow(pima_test))
) |>
  mutate(share = n / nrow(pima))

knitr::kable(
  train_test_sizes,
  digits  = 3,
  caption = "Training and test sample sizes and shares for the pima data"
)
```

Table 2: Training and test sample sizes and shares for the pima data

sample	n	share
Training	538	0.701
Test	230	0.299

```r
# Class proportions: overall vs training vs test
class_props <- rbind(
  overall = prop.table(table(pima$test)),
  train   = prop.table(table(pima_train$test)),
  test    = prop.table(table(pima_test$test))
)

knitr::kable(
  round(class_props, 4),
  caption = "Class proportions for the overall pima data and the training/test splits"
)
```

Table 3: Class proportions for the overall pima data and the training/test splits

	negative	positive
overall	0.6510	0.3490
train	0.6506	0.3494
test	0.6522	0.3478

Conclusion: Overall, about two thirds of the patients in the full data have a negative diabetes test and about one third have a positive test, so the outcome is somewhat imbalanced but not extremely rare for the positive class. The stratified 70/30 split preserves this pattern: both the training set of 538 patients and the test set of 230 patients have roughly the same negative/positive mix, so we are not introducing any major shift in class prevalence between the two samples. This means that performance metrics computed on the test set should be reasonably comparable to those on the training set.

1(a) Full logistic regression and residual deviance

We first fit a full logistic regression model with `test` as the binary response and all eight predictors as covariates. The model is of the form

$$\log\left(\frac{p_i}{1-p_i}\right) = \beta_0 + \beta_1\,\text{pregnant}_i + \beta_2\,\text{glucose}_i + \cdots + \beta_8\,\text{age}_i$$

where p_i is the probability that the ith patient tests positive for diabetes. We then extract the residual deviance and its associated residual degrees of freedom as a basic measure of model fit.

```r
# Fit full logistic regression on the training data
glm_full <- glm(
  test ~ .,
  data   = pima_train,
  family = binomial
)

summary(glm_full)
```

```
##
## Call:
## glm(formula = test ~ ., family = binomial, data = pima_train)
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -9.205200   0.921780  -9.986  < 2e-16 ***
## pregnant     0.101701   0.039049   2.604   0.0092 **
## glucose      0.037789   0.004648   8.130 4.28e-16 ***
## diastolic   -0.016116   0.006695  -2.407   0.0161 *
## triceps      0.002914   0.008539   0.341   0.7329
## insulin     -0.001435   0.001056  -1.359   0.1741
## bmi          0.110172   0.019610   5.618 1.93e-08 ***
## diabetes     0.561049   0.351205   1.597   0.1102
## age          0.022107   0.011156   1.982   0.0475 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 696.28  on 537  degrees of freedom
## Residual deviance: 487.57  on 529  degrees of freedom
## AIC: 505.57
##
## Number of Fisher Scoring iterations: 5
```

```r
# Extract residual deviance and residual df
full_fit_summary <- tibble(
  model             = "Full logistic regression",
  residual_deviance = glm_full$deviance,
  df_residual       = glm_full$df.residual
)

knitr::kable(
  full_fit_summary,
  digits  = 3,
  caption = "Residual deviance and residual df for the full logistic regression model"
)
```

Table 4: Residual deviance and residual df for the full logistic regression model

model	residual_deviance	df_residual
Full logistic regression	487.57	529

Conclusion: Deviance in logistic regression is the GLM analogue of residual sum of squares: it is -2 times the log-likelihood of the fitted model relative to a saturated (perfect-fit) model, so smaller deviance means a model that fits the data better. For the full logistic regression, the residual deviance is about 487.6 on 529 residual degrees of freedom. In a saturated model with a perfect fit we would expect a residual deviance close to 0, whereas for a well-fitting but not overfitted model we often compare the residual deviance to a χ^2 distribution with the same degrees of freedom. Here the residual deviance is somewhat smaller than the degrees of freedom, which does not by itself indicate a lack of fit. However, this single summary number is not enough to conclude that the model is adequate: it does not tell us where any lack of fit might occur, and its interpretation depends on large-sample approximations and the correctness of the binomial assumptions. We therefore need additional diagnostics, such as goodness-of-fit tests and residual plots, to more fully assess how well the model describes the data.

1(b) AIC-based model selection and Hosmer–Lemeshow test

Here we use two complementary summaries: the Akaike Information Criterion (AIC) to balance model fit against complexity, and a Hosmer–Lemeshow goodness-of-fit test to check how well the fitted probabilities agree with the observed outcomes. AIC is -2 times the log-likelihood plus a penalty $2k$ for the number of parameters, so smaller AIC means a model that fits the data well without being needlessly complex. A goodness-of-fit test such as Hosmer–Lemeshow groups observations by predicted risk and compares observed to expected counts; a large p-value means we do not see strong evidence that the model's probabilities are badly miscalibrated.

We now use AIC-based stepwise selection to choose a reduced logistic regression model. Starting from the full model, we apply `step()` with AIC as the criterion, allowing variables to be both dropped and re-added. We then compute fitted probabilities on the training data and use the Hosmer–Lemeshow goodness-of-fit test to assess whether the reduced model appears to fit the data adequately.

```r
# AIC-based stepwise selection starting from the full model
glm_step <- step(
  glm_full,
  direction = "both",
  trace     = FALSE
)

summary(glm_step)
```

```
##
## Call:
## glm(formula = test ~ pregnant + glucose + diastolic + bmi + diabetes +
##     age, family = binomial, data = pima_train)
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -8.976939   0.896109 -10.018  < 2e-16 ***
## pregnant     0.102583   0.038870   2.639  0.00831 **
## glucose      0.035453   0.004247   8.348  < 2e-16 ***
## diastolic   -0.016003   0.006518  -2.455  0.01409 *
## bmi          0.109286   0.018488   5.911  3.4e-09 ***
## diabetes     0.506924   0.347172   1.460  0.14425
## age          0.023237   0.011040   2.105  0.03530 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 696.28  on 537  degrees of freedom
## Residual deviance: 489.46  on 531  degrees of freedom
## AIC: 503.46
##
## Number of Fisher Scoring iterations: 5
```

```r
# Predicted probabilities on the training data from the reduced model
train_prob_glm_step <- predict(
  glm_step,
  newdata = pima_train,
  type    = "response"
)

# Hosmer-Lemeshow goodness-of-fit test (g = 10 groups)
y_train_numeric <- ifelse(pima_train$test == "positive", 1, 0)

hl_test <- ResourceSelection::hoslem.test(
  x = y_train_numeric,
  y = train_prob_glm_step,
  g = 10
)

hl_summary <- tibble(
  statistic = as.numeric(hl_test$statistic),
  df        = as.numeric(hl_test$parameter),
  p_value   = as.numeric(hl_test$p.value)
)

knitr::kable(
  hl_summary,
  digits  = 4,
  caption = "Hosmer-Lemeshow goodness-of-fit test for the reduced logistic model"
)
```

Table 5: Hosmer–Lemeshow goodness-of-fit test for the reduced logistic model

statistic	df	p_value
7.8719	8	0.4461

Conclusion: The AIC-based stepwise procedure retains six predictors in the reduced model: `pregnant`, `glucose`, `diastolic`, `bmi`, `diabetes`, and `age`. Variables such as `triceps` and `insulin` are dropped because they do not provide enough improvement in AIC once the other predictors are in the model. The Hosmer–Lemeshow test for this reduced model yields a test statistic of about 7.87 on 8 degrees of freedom with $p \approx 0.45$. Because this p-value is well above common significance levels, we do not see evidence of serious lack of fit on the training data: the observed and expected counts in the probability groups are reasonably consistent. At the same time, the Hosmer–Lemeshow test has limited power in small samples and depends on the arbitrary choice of grouping, so it should be viewed as one piece of evidence rather than a definitive verdict; it is best interpreted alongside residual plots and other diagnostics.

1(c) Training confusion matrix and accuracy with cutoff 0.5

Using the reduced logistic regression model, we now obtain predicted probabilities for each training observation and classify patients as test-positive or test-negative using a cutoff of 0.5. Specifically, we classify a patient as test-positive if $\hat{p}_i > 0.5$ and test-negative otherwise. We then compute the confusion matrix, along with overall accuracy, sensitivity, and specificity, taking a positive diabetes test as the positive class.

Quick definitions:

Sensitivity and specificity are two sides of how a classifier handles positives vs negatives:

- **Sensitivity** (true positive rate) answers: *"Among the people who truly have the condition/are positive, what fraction did the model correctly flag as positive?"* High sensitivity means few false negatives (you rarely miss real positives).
- **Specificity** (true negative rate) answers: *"Among the people who truly do not have the condition/are negative, what fraction did the model correctly label as negative?"* High specificity means few false positives (you rarely raise false alarms).

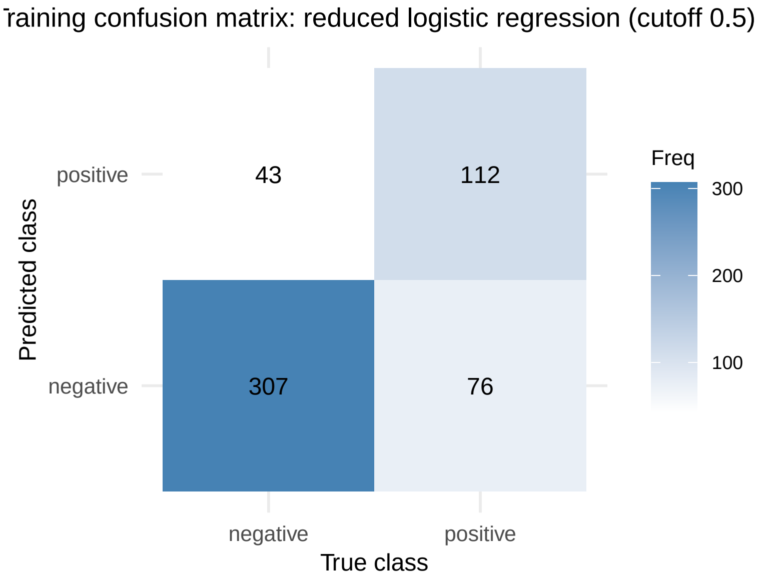
Sensitivity focuses on catching positives, specificity focuses on not wrongly accusing negatives.

```r
# Class predictions for the training set with cutoff 0.5
train_class_cut_05 <- ifelse(
  train_prob_glm_step > 0.5,
  "positive",
  "negative"
) %>%
  factor(levels = levels(pima_train$test))

cm_train_cut_05 <- caret::confusionMatrix(
  data      = train_class_cut_05,
  reference = pima_train$test,
  positive  = "positive"
)

# Confusion matrix: table + heatmap
confmat_kable(
  cm_train_cut_05,
  caption = "Training confusion matrix: reduced logistic regression (cutoff 0.5)"
)
```

Table 6: Training confusion matrix: reduced logistic regression (cutoff 0.5)

Prediction	Reference	Freq
negative	negative	307
positive	negative	43
negative	positive	76
positive	positive	112

```r
confmat_heatmap(
  cm_train_cut_05,
  title = "Training confusion matrix: reduced logistic regression (cutoff 0.5)"
)
```



```r
# Accuracy, sensitivity, and specificity
metrics_train_cut_05 <- tibble(
  metric = c("Accuracy", "Sensitivity (positive)", "Specificity (positive)"),
  value  = c(
    cm_train_cut_05$overall["Accuracy"],
    cm_train_cut_05$byClass["Sensitivity"],
    cm_train_cut_05$byClass["Specificity"]
  )
)

knitr::kable(
  metrics_train_cut_05,
  digits  = 4,
  caption = "Training performance metrics: reduced logistic regression (cutoff 0.5)"
)
```

Table 7: Training performance metrics: reduced logistic regression (cutoff 0.5)

metric	value
Accuracy	0.7788
Sensitivity (positive)	0.5957
Specificity (positive)	0.8771

Conclusion: At the 0.5 cutoff on the training data, the reduced logistic model achieves an accuracy of about 0.78, with sensitivity for positive tests around 0.60 and specificity around 0.88. This means that the model correctly classifies nearly 78% of patients overall, correctly identifies roughly 60% of those with a positive diabetes test, and correctly flags almost 88% of those without diabetes as negative. In practical terms, the model is more conservative on negatives than on positives: it is better at ruling out diabetes than at detecting it. Whether this is acceptable clinically depends on how costly it is to miss a true positive relative to generating a false positive that might trigger additional follow-up testing.

1(d) Training confusion matrix with cutoff 0.9

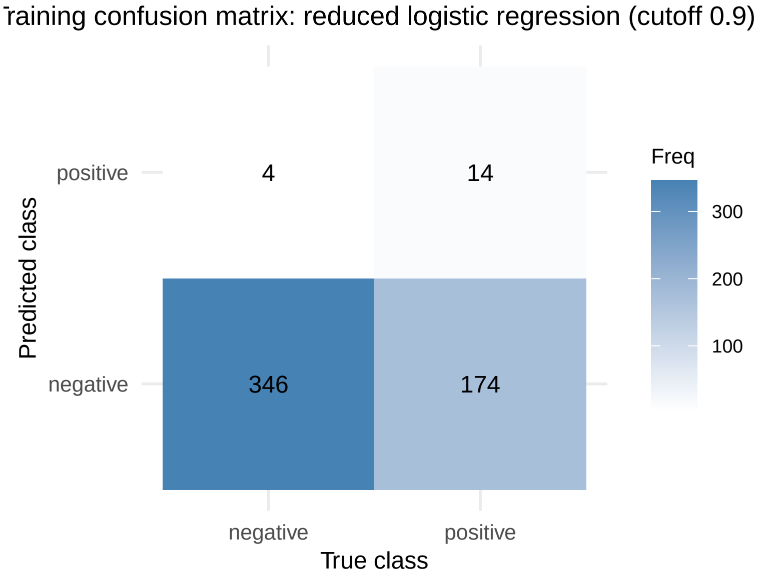
We now tighten the decision rule by increasing the cutoff to 0.9. A patient is classified as test-positive if $\hat{p}_i > 0.9$ and test-negative otherwise. This rule makes it harder to call a patient test-positive, which may reduce false positives at the expense of more false negatives (missed positives).

```r
# Class predictions for the training set with cutoff 0.9
train_class_cut_09 <- ifelse(
  train_prob_glm_step > 0.9,
  "positive",
  "negative"
) %>%
  factor(levels = levels(pima_train$test))

cm_train_cut_09 <- caret::confusionMatrix(
  data      = train_class_cut_09,
  reference = pima_train$test,
  positive  = "positive"
)

# Confusion matrix: table + heatmap
confmat_kable(
  cm_train_cut_09,
  caption = "Training confusion matrix: reduced logistic regression (cutoff 0.9)"
)
```

Table 8: Training confusion matrix: reduced logistic regression (cutoff 0.9)

Prediction	Reference	Freq
negative	negative	346
positive	negative	4
negative	positive	174
positive	positive	14

```r
confmat_heatmap(
  cm_train_cut_09,
  title = "Training confusion matrix: reduced logistic regression (cutoff 0.9)"
)
```



```r
# Accuracy, sensitivity, and specificity
metrics_train_cut_09 <- tibble(
  metric = c("Accuracy", "Sensitivity (positive)", "Specificity (positive)"),
  value  = c(
    cm_train_cut_09$overall["Accuracy"],
    cm_train_cut_09$byClass["Sensitivity"],
    cm_train_cut_09$byClass["Specificity"]
  )
)

knitr::kable(
  metrics_train_cut_09,
  digits  = 4,
  caption = "Training performance metrics: reduced logistic regression (cutoff 0.9)"
)
```

Table 9: Training performance metrics: reduced logistic regression (cutoff 0.9)

metric	value
Accuracy	0.6691
Sensitivity (positive)	0.0745
Specificity (positive)	0.9886

Conclusion: When we increase the cutoff to 0.9, the training accuracy drops to about 0.67, sensitivity for positives falls dramatically to roughly 0.07, and specificity increases to about 0.99. This behavior matches our expectations: by requiring very high predicted probability before calling a case positive, the model becomes extremely reluctant to predict a positive test. Almost all negative patients are correctly classified (very high specificity), but the vast majority of true positives are now missed (very low sensitivity). On the training data alone, this more stringent cutoff offers little advantage: it reduces false positives, but at the cost of rejecting almost all truly positive cases, which is typically unacceptable for a screening tool where missing patients with diabetes can have serious consequences.

1(e) Test-set performance and choice of cutoff

Finally, we evaluate the reduced logistic regression model on the held-out test set using both cutoffs 0.5 and 0.9. We compute predicted probabilities for each test observation, classify patients at each cutoff, and then construct confusion matrices and performance metrics. We then compare test performance between the two cutoffs and decide which threshold is more appropriate overall.

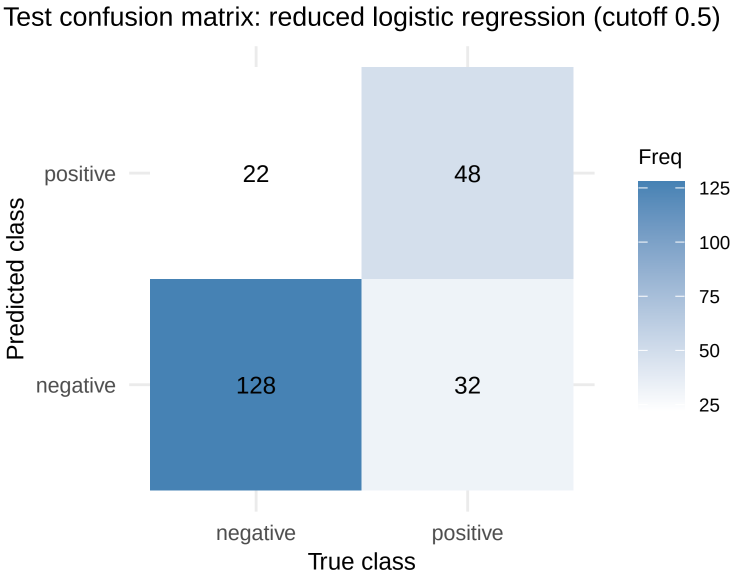
```
# Predicted probabilities on the test data from the reduced logistic model
test_prob_glm_step <- predict(
  glm_step,
  newdata = pima_test,
  type    = "response"
)

# Class predictions on the test set, cutoff 0.5
test_class_cut_05 <- ifelse(
  test_prob_glm_step > 0.5,
  "positive",
  "negative"
) %>%
  factor(levels = levels(pima_test$test))

cm_test_cut_05 <- caret::confusionMatrix(
  data      = test_class_cut_05,
  reference = pima_test$test,
  positive  = "positive"
)

# Confusion matrix and performance metrics for cutoff 0.5
confmat_kable(
  cm_test_cut_05,
  caption = "Test confusion matrix: reduced logistic regression (cutoff 0.5)"
)
```

Table 10: Test confusion matrix: reduced logistic regression (cutoff 0.5)

Prediction	Reference	Freq
negative	negative	128
positive	negative	22
negative	positive	32

Prediction	Reference	Freq
positive	positive	48

```r
confmat_heatmap(
  cm_test_cut_05,
  title = "Test confusion matrix: reduced logistic regression (cutoff 0.5)"
)
```



```r
metrics_test_cut_05 <- tibble(
  metric = c("Accuracy", "Sensitivity (positive)", "Specificity (positive)"),
  value  = c(
    cm_test_cut_05$overall["Accuracy"],
    cm_test_cut_05$byClass["Sensitivity"],
    cm_test_cut_05$byClass["Specificity"]
  )
)

knitr::kable(
  metrics_test_cut_05,
  digits  = 4,
  caption = "Test performance metrics: reduced logistic regression (cutoff 0.5)"
)
```

Table 11: Test performance metrics: reduced logistic regression (cutoff 0.5)

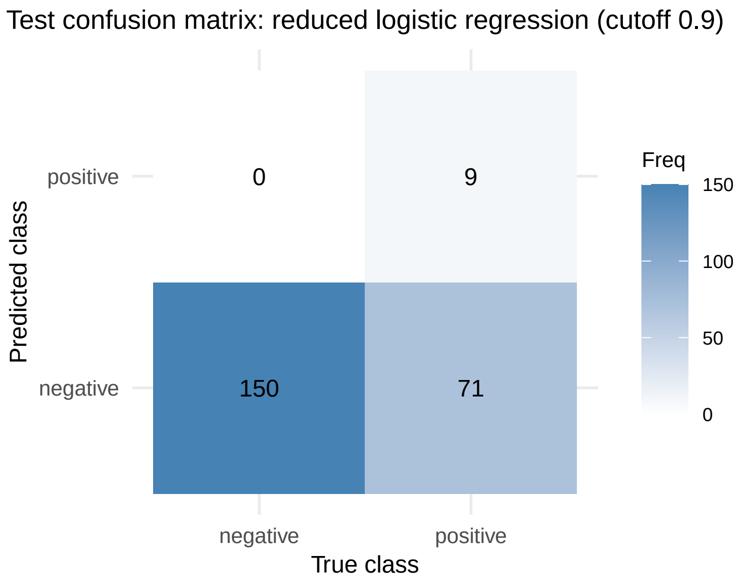
metric	value
Accuracy	0.7652
Sensitivity (positive)	0.6000
Specificity (positive)	0.8533

```r
# Class predictions on the test set, cutoff 0.9
test_class_cut_09 <- ifelse(
  test_prob_glm_step > 0.9,
  "positive",
  "negative"
) %>%
  factor(levels = levels(pima_test$test))

cm_test_cut_09 <- caret::confusionMatrix(
  data      = test_class_cut_09,
  reference = pima_test$test,
  positive  = "positive"
)

# Confusion matrix and performance metrics for cutoff 0.9
confmat_kable(
  cm_test_cut_09,
  caption = "Test confusion matrix: reduced logistic regression (cutoff 0.9)"
)
```

Table 12: Test confusion matrix: reduced logistic regression (cutoff 0.9)

Prediction	Reference	Freq
negative	negative	150
positive	negative	0
negative	positive	71
positive	positive	9

```r
confmat_heatmap(
  cm_test_cut_09,
  title = "Test confusion matrix: reduced logistic regression (cutoff 0.9)"
)
```



```r
metrics_test_cut_09 <- tibble(
  metric = c("Accuracy", "Sensitivity (positive)", "Specificity (positive)"),
  value  = c(
    cm_test_cut_09$overall["Accuracy"],
    cm_test_cut_09$byClass["Sensitivity"],
    cm_test_cut_09$byClass["Specificity"]
  )
)

knitr::kable(
  metrics_test_cut_09,
  digits  = 4,
  caption = "Test performance metrics: reduced logistic regression (cutoff 0.9)"
)
```

Table 13: Test performance metrics: reduced logistic regression (cutoff 0.9)

metric	value
Accuracy	0.6913
Sensitivity (positive)	0.1125
Specificity (positive)	1.0000

Conclusion: On the test set, the reduced logistic model with cutoff 0.5 attains accuracy around 0.77, sensitivity about 0.60, and specificity about 0.85. These values are fairly close to the corresponding training metrics, suggesting that the model generalizes reasonably well and is not heavily overfitting. When we raise the cutoff to 0.9, test accuracy falls to about 0.69, sensitivity collapses to roughly 0.11, and specificity reaches 1.00, meaning that the model never predicts a positive test unless it is extremely certain. In practice, this stricter rule eliminates false positives but misses almost all true positives, which is undesirable in a screening context where undiagnosed diabetes may lead to substantial harm. Balancing these considerations, we would prefer the cutoff of 0.5 on the test data: it maintains higher overall accuracy and, more importantly, much higher sensitivity, while still keeping specificity reasonably high.

Problem 2: Tree-based and ensemble models

We now compare the reduced logistic regression model from Problem 1 with a collection of tree-based and ensemble methods, using the same training and test splits. We fit each method to predict `test` from the eight predictors on the training data, compute confusion matrices on the training and test sets, and then compare their performance to decide on a champion model.

2(a) Training confusion matrices for tree-based and ensemble methods

In this part we fit six different models on the training data:

- A classification tree using `tree::tree`
- A classification tree using `rpart::rpart`
- A random forest using `randomForest`
- A boosting model using `C5.0` from the `C50` package
- An AdaBoost model using `adabag`
- A gradient boosting model using `xgboost`

We then compute training confusion matrices for each model.

```r
# Convenience objects for predictors and response
pima_train_x <- pima_train %>% select(-test)
pima_train_y <- pima_train$test

pima_test_x  <- pima_test %>% select(-test)
pima_test_y  <- pima_test$test
```

Tree model (tree package)

We first fit a classification tree using the `tree` package with `test` as the response and all eight predictors as covariates, then compute the training confusion matrix.

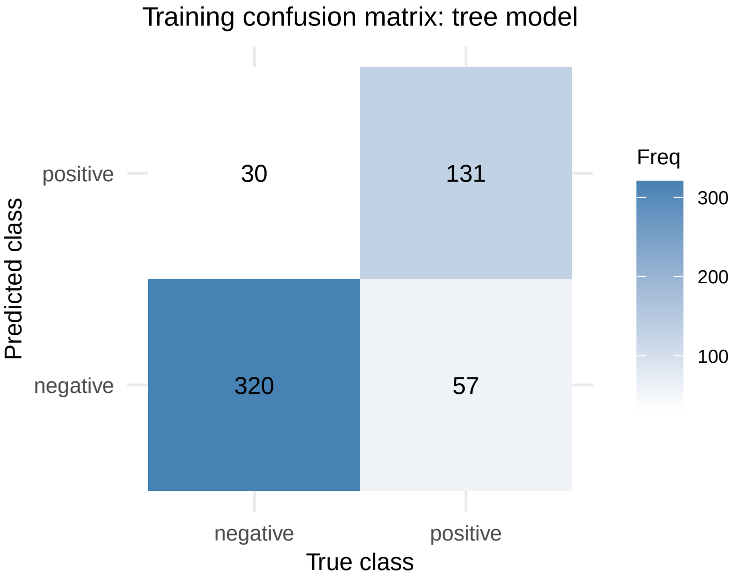
```r
set.seed(1023)

tree_model <- tree::tree(
  test ~ .,
  data = pima_train
)

# Training predictions and confusion matrix
train_pred_tree <- predict(
  tree_model,
  newdata = pima_train,
  type    = "class"
)
```

```r
cm_train_tree <- caret::confusionMatrix(
  data      = train_pred_tree,
  reference = pima_train_y,
  positive  = "positive"
)

# Confusion matrix: table + heatmap
confmat_kable(
  cm_train_tree,
  caption = "Training confusion matrix: tree model"
)
```

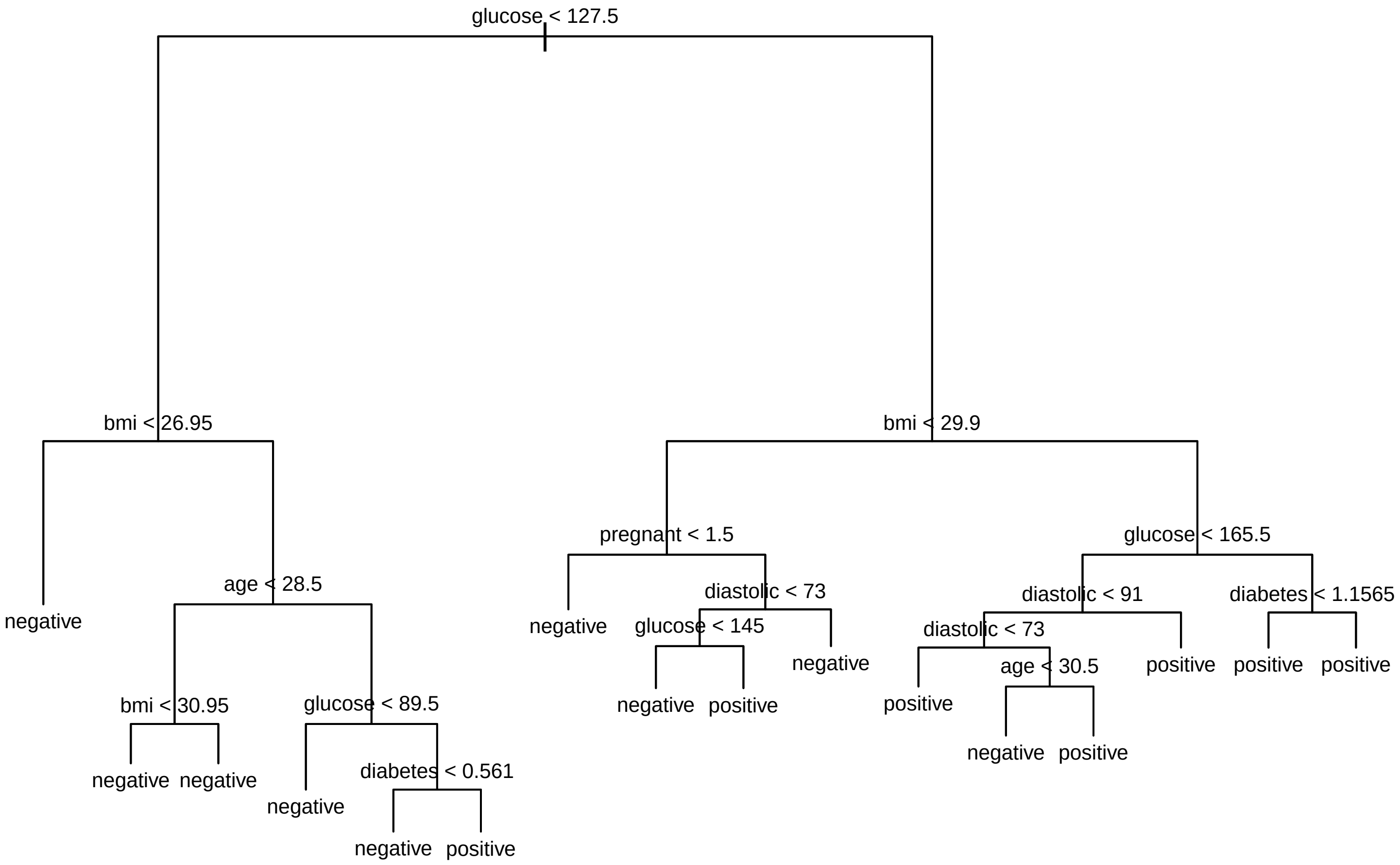
Table 14: Training confusion matrix: tree model

Prediction	Reference	Freq
negative	negative	320
positive	negative	30
negative	positive	57
positive	positive	131

```r
confmat_heatmap(
  cm_train_tree,
  title = "Training confusion matrix: tree model"
)
```



```r
# Plot the tree from the `tree` package with smaller labels
par(mar = c(1, 1, 1, 1))
plot(tree_model)
text(tree_model, pretty = 0, cex = 0.7)
```



Tree model (rpart package)

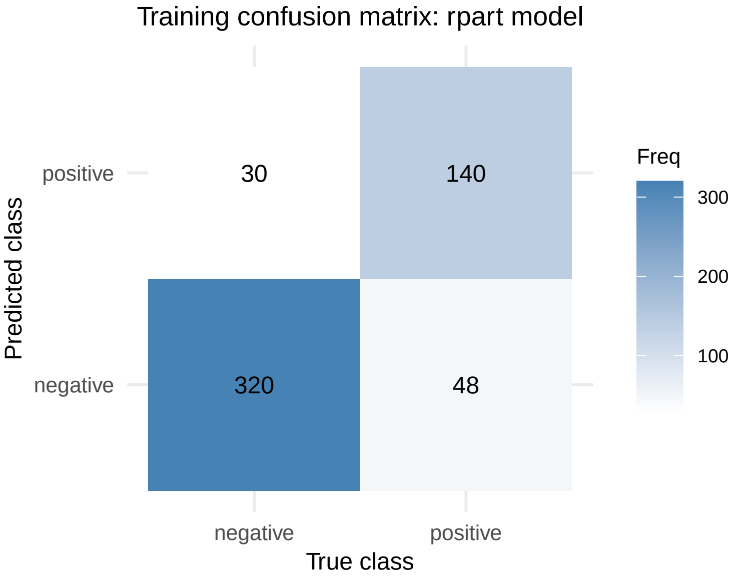
We next fit a classification tree using `rpart` with the Gini impurity criterion and compute the training confusion matrix.

```r
set.seed(1023)

rpart_model <- rpart::rpart(
  test ~ .,
  data   = pima_train,
  method = "class"
)

# Training predictions and confusion matrix
train_pred_rpart <- predict(
  rpart_model,
  newdata = pima_train,
  type    = "class"
)

cm_train_rpart <- caret::confusionMatrix(
  data      = train_pred_rpart,
  reference = pima_train_y,
  positive  = "positive"
)

# Confusion matrix: table + heatmap
confmat_kable(
  cm_train_rpart,
  caption = "Training confusion matrix: rpart model"
)
```

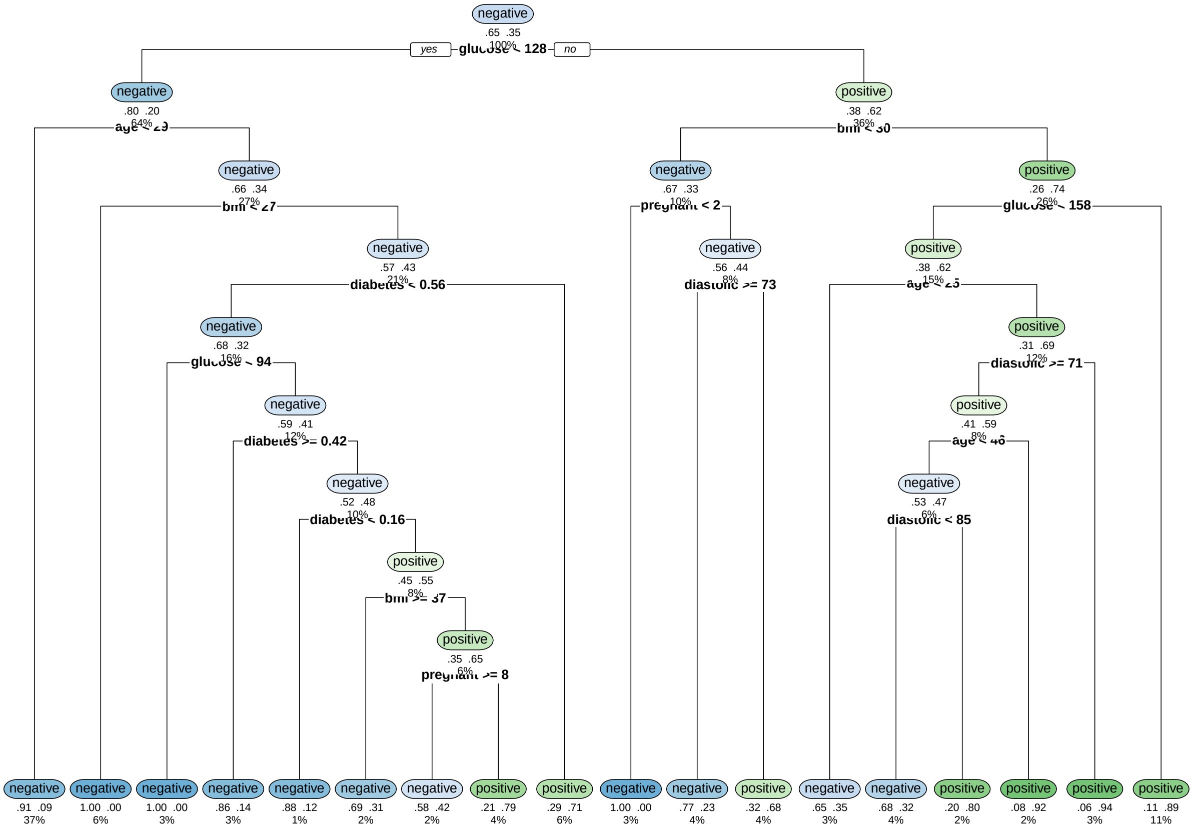
Table 15: Training confusion matrix: rpart model

Prediction	Reference	Freq
negative	negative	320
positive	negative	30
negative	positive	48
positive	positive	140

```r
confmat_heatmap(
  cm_train_rpart,
  title = "Training confusion matrix: rpart model"
)
```



```r
# Plot the rpart tree using rpart.plot with larger text
rpart.plot::rpart.plot(
  rpart_model,
  type          = 2,    # label all nodes
  extra         = 104,  # show prob + class
  under         = TRUE,
  fallen.leaves = TRUE,
  cex           = 0.4
)
```



Random forest model

We now fit a random forest classifier using the `randomForest` package and compute its training confusion matrix.

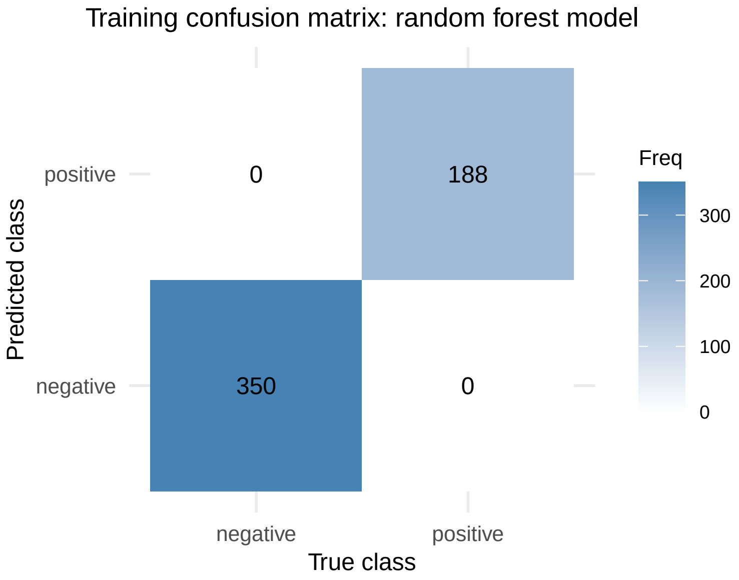
```r
set.seed(1023)

rf_model <- randomForest::randomForest(
  test ~ .,
  data = pima_train
)

# Training predictions and confusion matrix
train_pred_rf <- predict(
  rf_model,
  newdata = pima_train
)

cm_train_rf <- caret::confusionMatrix(
  data      = train_pred_rf,
  reference = pima_train_y,
  positive  = "positive"
)

# Confusion matrix: table + heatmap
confmat_kable(
  cm_train_rf,
  caption = "Training confusion matrix: random forest model"
)
```

Table 16: Training confusion matrix: random forest model

Prediction	Reference	Freq
negative	negative	350
positive	negative	0

Prediction	Reference	Freq
negative	positive	0
positive	positive	188

```r
confmat_heatmap(
  cm_train_rf,
  title = "Training confusion matrix: random forest model"
)
```



Boosting with C5.0

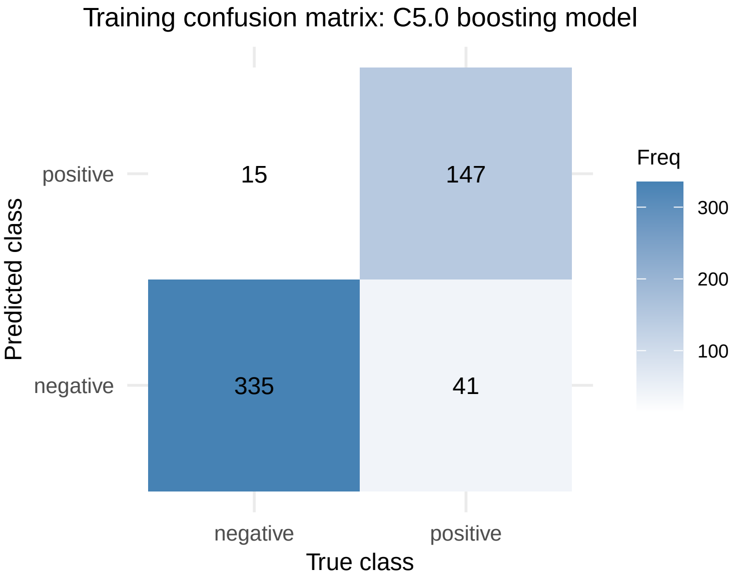
We fit a boosting model using `C5.0` from the `C50` package, treating `test` as a factor response, and compute the training confusion matrix.

```r
set.seed(1023)

c50_model <- C50::C5.0(
  x = pima_train_x,
  y = pima_train_y
)

# Training predictions and confusion matrix
train_pred_c50 <- predict(
  c50_model,
  newdata = pima_train_x
)

cm_train_c50 <- caret::confusionMatrix(
  data      = train_pred_c50,
  reference = pima_train_y,
  positive  = "positive"
)

# Confusion matrix: table + heatmap
confmat_kable(
  cm_train_c50,
  caption = "Training confusion matrix: C5.0 boosting model"
)
```

Table 17: Training confusion matrix: C5.0 boosting model

Prediction	Reference	Freq
negative	negative	335
positive	negative	15
negative	positive	41
positive	positive	147

```r
confmat_heatmap(
  cm_train_c50,
  title = "Training confusion matrix: C5.0 boosting model"
)
```



AdaBoost model (adabag)

We next fit an AdaBoost model using the `adabag` package with `test` as the response and all eight predictors as covariates, then compute the training confusion matrix.

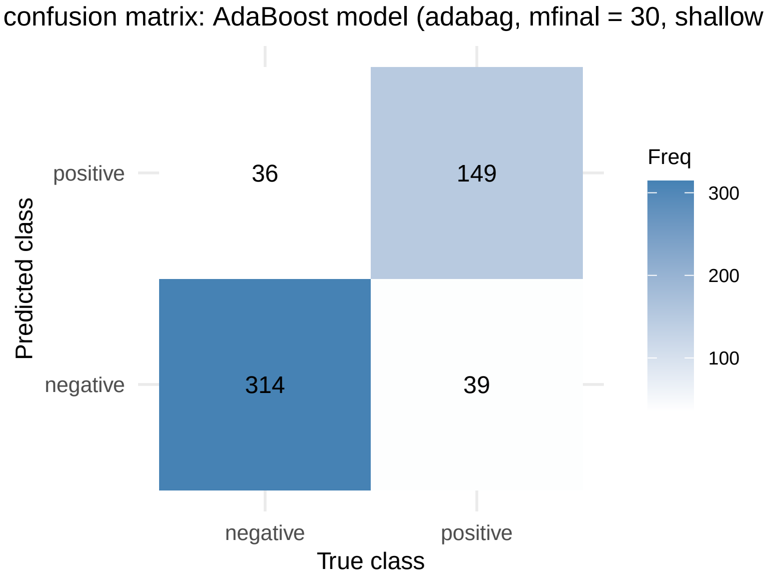
```r
set.seed(1023)

# AdaBoost model using adabag with a bit more complexity
ada_model <- adabag::boosting(
  test   ~ .,
  data   = pima_train,
  mfinal = 30,  # more boosting iterations for a stronger ensemble
  control = rpart::rpart.control(
    maxdepth = 3,    # still relatively shallow trees
    minsplit = 20
  )
)

# Training predictions and confusion matrix
ada_train_pred <- predict(
  ada_model,
  newdata = pima_train
)

train_pred_ada <- ada_train_pred$class

# Make sure both are factors with identical levels
pima_train_y   <- factor(pima_train_y)  # just to be sure
```

```
train_pred_ada <- factor(train_pred_ada, levels = levels(pima_train_y))

cm_train_ada <- caret::confusionMatrix(
  data      = train_pred_ada,
  reference = pima_train_y,
  positive  = "positive"
)

# Confusion matrix: table + heatmap
confmat_kable(
  cm_train_ada,
  caption = "Training confusion matrix: AdaBoost model (adabag, mfinal = 30, shallow trees)"
)
```

Table 18: Training confusion matrix: AdaBoost model (adabag, mfinal = 30, shallow trees)

Prediction	Reference	Freq
negative	negative	314
positive	negative	36
negative	positive	39
positive	positive	149

```
confmat_heatmap(
  cm_train_ada,
  title = "Training confusion matrix: AdaBoost model (adabag, mfinal = 30, shallow trees)"
)
```



XGBoost model

Finally, we fit a gradient boosting model using `xgboost`. We first create a model matrix of predictors and a numeric response coded as 1 for a positive test and 0 for a negative test, then fit a binary logistic model and classify using a probability cutoff of 0.5.

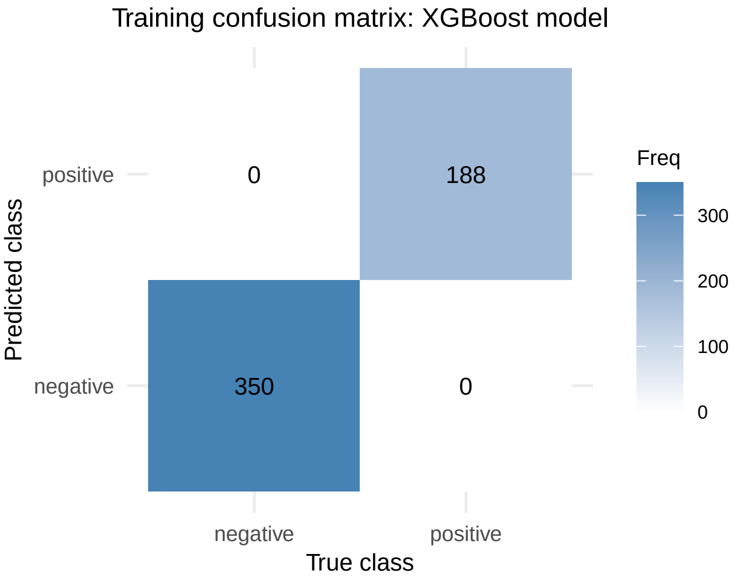
```
set.seed(1023)

# Model matrix for xgboost (no intercept column)
xgb_train_x <- model.matrix(
  test ~ . - 1,
  data = pima_train
)
```

```r
xgb_train_y <- as.numeric(pima_train_y == "positive")

xgb_model <- xgboost::xgboost(
  data      = xgb_train_x,
  label     = xgb_train_y,
  objective = "binary:logistic",
  nrounds   = 100,
  verbose   = 0
)

# Training predictions and confusion matrix
xgb_train_prob <- predict(
  xgb_model,
  newdata = xgb_train_x
)

train_pred_xgb <- ifelse(
  xgb_train_prob > 0.5,
  "positive",
  "negative"
) %>%
  factor(levels = levels(pima_train_y))

cm_train_xgb <- caret::confusionMatrix(
  data      = train_pred_xgb,
  reference = pima_train_y,
  positive  = "positive"
)

# Confusion matrix: table + heatmap
confmat_kable(
  cm_train_xgb,
  caption = "Training confusion matrix: XGBoost model"
)
```

Table 19: Training confusion matrix: XGBoost model

Prediction	Reference	Freq
negative	negative	350
positive	negative	0
negative	positive	0
positive	positive	188

```r
confmat_heatmap(
  cm_train_xgb,
  title = "Training confusion matrix: XGBoost model"
)
```



Conclusion: Across the six tree-based and ensemble methods, training performance varies substantially. The single-tree models (`tree` and `rpart`) already outperform the reduced logistic model on the training set, with higher accuracy and better balance between sensitivity and specificity, though they still make nontrivial errors on both positive and negative cases. The C5.0 and AdaBoost models improve training accuracy further and reduce misclassification rates for positives, indicating that boosting is successfully capturing additional nonlinear structure in the predictors. In contrast, the random forest and XGBoost models achieve essentially perfect performance on the training data, correctly classifying all observations. While this looks impressive, such near-perfect training accuracy is a classic sign of potential overfitting: highly flexible ensembles can memorize the training set, especially with many trees or boosting iterations. This makes it especially important to compare these models on the test set before declaring them superior to the simpler logistic and single-tree models.

2(b) Test-set confusion matrices and model comparison

We now evaluate each of the tree-based and ensemble models on the held-out test set. For each method, we compute predicted classes on `pima_test`, form the confusion matrix, and compare the test performance to that of the reduced logistic regression model from Problem 1 (with the chosen cutoff). We then choose a champion model based on test-set performance and interpretability.

Tree model (tree package) on the test set

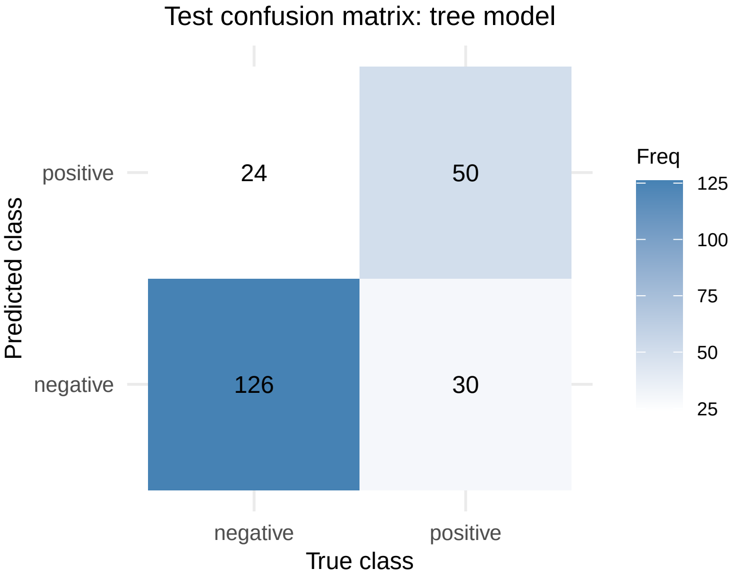
```r
# Test predictions and confusion matrix: tree model
test_pred_tree <- predict(
  tree_model,
  newdata = pima_test,
  type    = "class"
)

cm_test_tree <- caret::confusionMatrix(
  data      = test_pred_tree,
  reference = pima_test_y,
  positive  = "positive"
)

confmat_kable(
  cm_test_tree,
  caption = "Test confusion matrix: tree model"
)
```

Table 20: Test confusion matrix: tree model

Prediction	Reference	Freq
negative	negative	126

Prediction	Reference	Freq
positive	negative	24
negative	positive	30
positive	positive	50

```
confmat_heatmap(
  cm_test_tree,
  title = "Test confusion matrix: tree model"
)
```



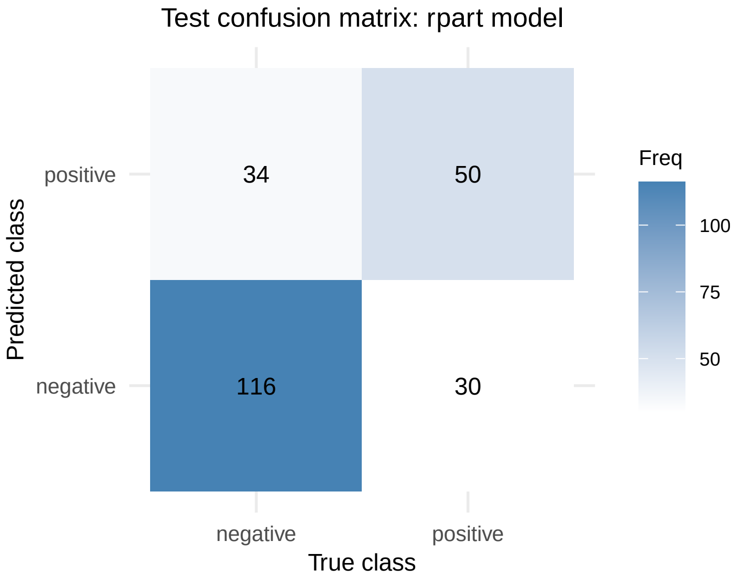
Tree model (rpart package) on the test set

```
# Test predictions and confusion matrix: rpart model
test_pred_rpart <- predict(
  rpart_model,
  newdata = pima_test,
  type    = "class"
)

cm_test_rpart <- caret::confusionMatrix(
  data      = test_pred_rpart,
  reference = pima_test_y,
  positive  = "positive"
)

confmat_kable(
  cm_test_rpart,
  caption = "Test confusion matrix: rpart model"
)
```

Table 21: Test confusion matrix: rpart model

Prediction	Reference	Freq
negative	negative	116
positive	negative	34
negative	positive	30
positive	positive	50

```r
confmat_heatmap(
  cm_test_rpart,
  title = "Test confusion matrix: rpart model"
)
```



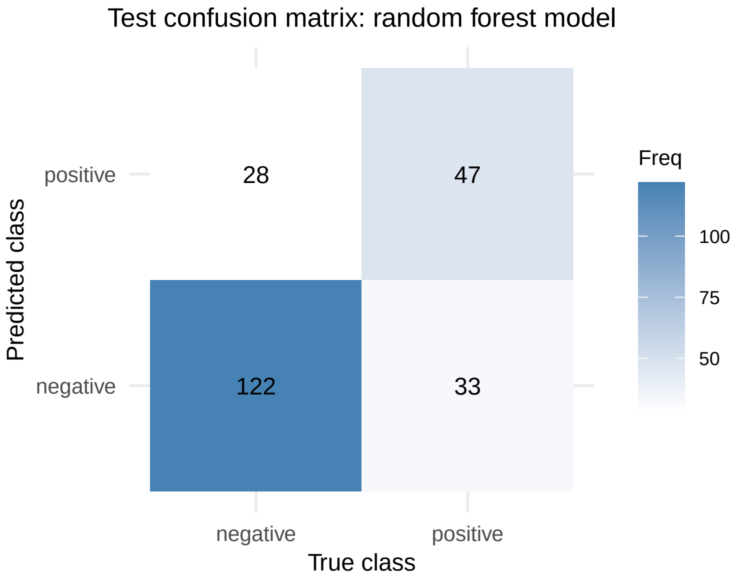
Random forest model on the test set

```r
# Test predictions and confusion matrix: random forest model
test_pred_rf <- predict(
  rf_model,
  newdata = pima_test
)

cm_test_rf <- caret::confusionMatrix(
  data      = test_pred_rf,
  reference = pima_test_y,
  positive  = "positive"
)

confmat_kable(
  cm_test_rf,
  caption = "Test confusion matrix: random forest model"
)
```

Table 22: Test confusion matrix: random forest model

Prediction	Reference	Freq
negative	negative	122
positive	negative	28
negative	positive	33
positive	positive	47

```r
confmat_heatmap(
  cm_test_rf,
  title = "Test confusion matrix: random forest model"
)
```



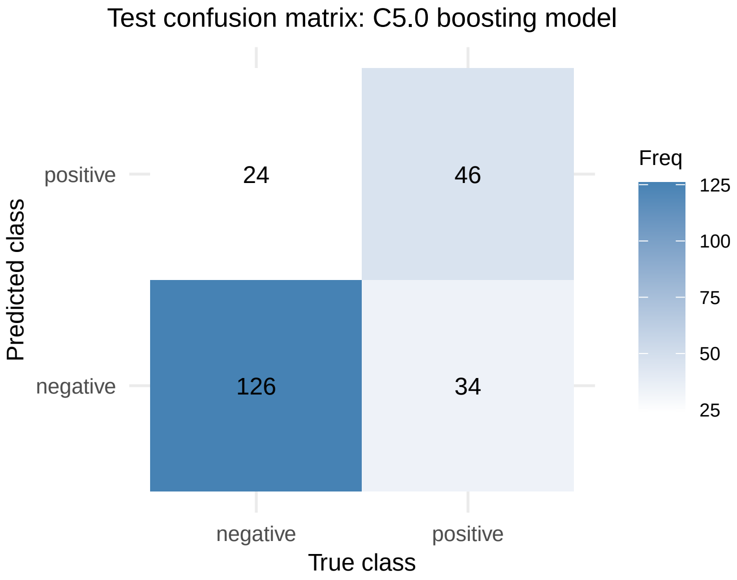
C5.0 boosting model on the test set

```
# Test predictions and confusion matrix: C5.0 boosting model
test_pred_c50 <- predict(
  c50_model,
  newdata = pima_test_x
)

cm_test_c50 <- caret::confusionMatrix(
  data      = test_pred_c50,
  reference = pima_test_y,
  positive  = "positive"
)

confmat_kable(
  cm_test_c50,
  caption = "Test confusion matrix: C5.0 boosting model"
)
```

Table 23: Test confusion matrix: C5.0 boosting model

Prediction	Reference	Freq
negative	negative	126
positive	negative	24
negative	positive	34
positive	positive	46

```
confmat_heatmap(
  cm_test_c50,
  title = "Test confusion matrix: C5.0 boosting model"
)
```



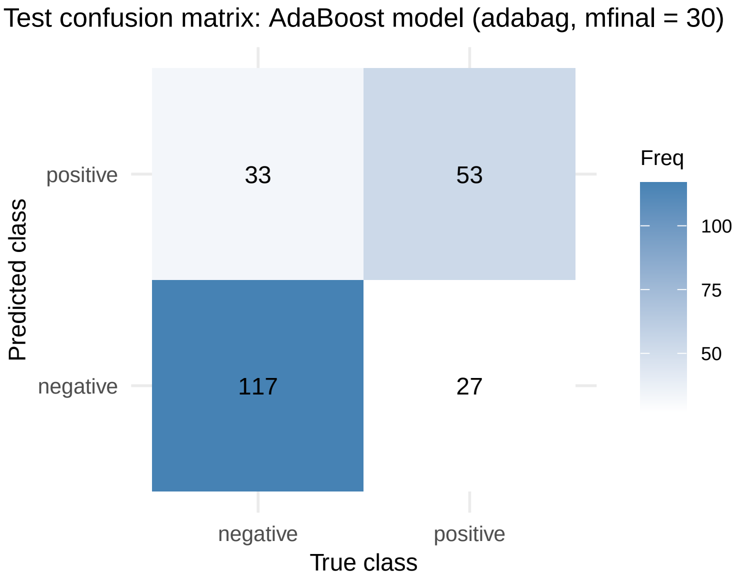
AdaBoost model (adabag) on the test set

```
# Test predictions and confusion matrix: AdaBoost model (adabag)
ada_test_pred <- predict(
  ada_model,
  newdata = pima_test
)

test_pred_ada <- ada_test_pred$class
test_pred_ada <- factor(test_pred_ada, levels = levels(pima_test_y))

cm_test_ada <- caret::confusionMatrix(
  data      = test_pred_ada,
  reference = pima_test_y,
  positive  = "positive"
)

confmat_kable(
  cm_test_ada,
  caption = "Test confusion matrix: AdaBoost model (adabag, mfinal = 30)"
)
```

Table 24: Test confusion matrix: AdaBoost model (adabag, mfinal = 30)

Prediction	Reference	Freq
negative	negative	117
positive	negative	33
negative	positive	27
positive	positive	53

```
confmat_heatmap(
  cm_test_ada,
  title = "Test confusion matrix: AdaBoost model (adabag, mfinal = 30)"
)
```



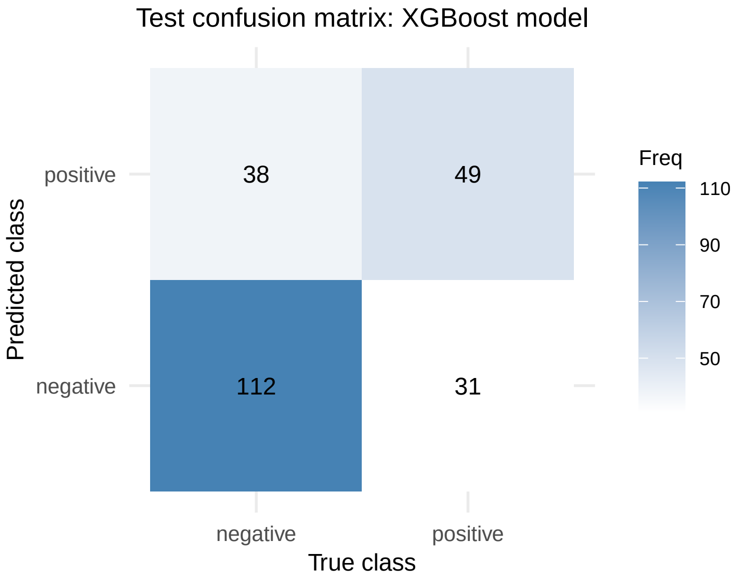
XGBoost model on the test set

```r
# Test predictions and confusion matrix: XGBoost model
xgb_test_x <- model.matrix(
  test ~ . - 1,
  data = pima_test
)

xgb_test_prob <- predict(
  xgb_model,
  newdata = xgb_test_x
)

test_pred_xgb <- ifelse(
  xgb_test_prob > 0.5,
  "positive",
  "negative"
) %>%
  factor(levels = levels(pima_test_y))

cm_test_xgb <- caret::confusionMatrix(
  data      = test_pred_xgb,
  reference = pima_test_y,
  positive  = "positive"
)

confmat_kable(
  cm_test_xgb,
  caption = "Test confusion matrix: XGBoost model"
)
```

Table 25: Test confusion matrix: XGBoost model

Prediction	Reference	Freq
negative	negative	112
positive	negative	38
negative	positive	31
positive	positive	49

```
confmat_heatmap(
  cm_test_xgb,
  title = "Test confusion matrix: XGBoost model"
)
```



Summary of test-set performance across methods

To help compare all methods, including logistic regression, we assemble a summary table of test accuracy, sensitivity, and specificity for positive tests. For logistic regression, we include both cutoffs 0.5 and 0.9; for the tree-based methods, we use the default cutoff implied by their predicted classes.

```
test_performance_summary <- tibble(
  model = c(
    "Logistic (cutoff 0.5)",
    "Logistic (cutoff 0.9)",
    "Tree",
    "rpart",
    "Random forest",
    "C5.0",
    "AdaBoost",
    "XGBoost"
  ),
  accuracy = c(
    cm_test_cut_05$overall["Accuracy"],
    cm_test_cut_09$overall["Accuracy"],
    cm_test_tree$overall["Accuracy"],
    cm_test_rpart$overall["Accuracy"],
    cm_test_rf$overall["Accuracy"],
    cm_test_c50$overall["Accuracy"],
    cm_test_ada$overall["Accuracy"],
    cm_test_xgb$overall["Accuracy"]
  ),
  sensitivity_positive = c(
    cm_test_cut_05$byClass["Sensitivity"],
    cm_test_cut_09$byClass["Sensitivity"],
    cm_test_tree$byClass["Sensitivity"],
    cm_test_rpart$byClass["Sensitivity"],
    cm_test_rf$byClass["Sensitivity"],
    cm_test_c50$byClass["Sensitivity"],
    cm_test_ada$byClass["Sensitivity"],
    cm_test_xgb$byClass["Sensitivity"]
```

```r
  ),
  specificity_positive = c(
    cm_test_cut_05$byClass["Specificity"],
    cm_test_cut_09$byClass["Specificity"],
    cm_test_tree$byClass["Specificity"],
    cm_test_rpart$byClass["Specificity"],
    cm_test_rf$byClass["Specificity"],
    cm_test_c50$byClass["Specificity"],
    cm_test_ada$byClass["Specificity"],
    cm_test_xgb$byClass["Specificity"]
  )
)

knitr::kable(
  test_performance_summary,
  digits  = 4,
  caption = "Test-set accuracy, sensitivity, and specificity for positive test cases across all models"
)
```

Table 26: Test-set accuracy, sensitivity, and specificity for positive test cases across all models

model	accuracy	sensitivity_positive	specificity_positive
Logistic (cutoff 0.5)	0.7652	0.6000	0.8533
Logistic (cutoff 0.9)	0.6913	0.1125	1.0000
Tree	0.7652	0.6250	0.8400
rpart	0.7217	0.6250	0.7733
Random forest	0.7348	0.5875	0.8133
C5.0	0.7478	0.5750	0.8400
AdaBoost	0.7391	0.6625	0.7800
XGBoost	0.7000	0.6125	0.7467

Conclusion: Looking at the test-set summary, the reduced logistic regression with cutoff 0.5 and the simple `tree` model achieve the highest accuracy (both around 0.77) with fairly similar sensitivity and specificity profiles. Logistic regression with cutoff 0.9 performs noticeably worse overall: although it attains perfect specificity, its sensitivity for positive tests is extremely low, so it is not a good choice for screening. Among the tree-based and ensemble methods, C5.0 and AdaBoost come closest to the logistic model in terms of accuracy, with AdaBoost offering the highest test-set sensitivity for positive cases but somewhat lower specificity. Random forest and XGBoost, despite their perfect training performance, do not dominate on the test set: their accuracies are only moderate and their sensitivity/specificity balance is comparable to, or slightly worse than, simpler models, which is consistent with mild overfitting.

Taking both predictive performance and interpretability into account, we would select the reduced logistic regression model with cutoff 0.5 as the champion. It delivers top-tier test accuracy, a reasonable trade-off between sensitivity and specificity for positive diabetes tests, and easily interpretable coefficients that quantify how each predictor affects the log-odds of a positive test. If we wanted slightly higher sensitivity and were willing to sacrifice some specificity and interpretability, the AdaBoost model could be considered as an alternative, but for most purposes the logistic regression model provides the clearest and most stable choice.

Some notes about adaboost:

THE BELOW METRICS WERE COMPUTED WHILE I WAS FIGHTING WITH ADABOOST. THEY ARE INCLUDED FOR TEACHING PURPOSES. DO NOT HARDCODE METRICS LIKE THIS IN GENERAL, I ONLY DID IT BECAUSE I DON'T WANT TO RUN ADABOOST THREE TIMES!

- A shallow AdaBoost model (`mfinal = 10`, `maxdepth = 2`, `minsplit = 20`) gives the weakest ensemble: lower test accuracy and only moderate sensitivity and specificity.
- A moderately complex AdaBoost model (`mfinal = 30`, `maxdepth = 3`, `minsplit = 20`) improves test accuracy and sensitivity, with a slight gain in specificity. This setting strikes a good balance between catching positives and avoiding false alarms.
- The default AdaBoost settings (`mfinal = 100`, `maxdepth = 30`, `minsplit = 20`, via `rpart.control()` defaults) yield the highest test accuracy and the highest specificity, but sensitivity drops back relative to the moderate configuration.

The model becomes more conservative about predicting a positive test.

```
ada_config_summary <- tibble(
  config = c("Shallow", "Moderate", "Default"),
  mfinal = c(10, 30, 100),
  maxdepth = c(2, 3, 30),
  minsplit = c(20, 20, 20),
  accuracy = c(0.7174, 0.7391, 0.7435),
  sensitivity_positive = c(0.6250, 0.6625, 0.6250),
  specificity_positive = c(0.7667, 0.7800, 0.8067)
)

knitr::kable(
  ada_config_summary,
  digits  = 4,
  caption = "Test-set performance for three AdaBoost configurations on the pima data"
)
```

Table 27: Test-set performance for three AdaBoost configurations on the pima data

config	mfinal	maxdepth	minsplit	accuracy	sensitivity_positive	specificity_positive
Shallow	10	2	20	0.7174	0.6250	0.7667
Moderate	30	3	20	0.7391	0.6625	0.7800
Default	100	30	20	0.7435	0.6250	0.8067

The shallow configuration yields the lowest accuracy and is dominated by the moderate and default settings. Between the moderate and default AdaBoost models, the default configuration slightly improves accuracy and specificity but at the cost of lower sensitivity for positive tests. If our priority is to detect as many true positives as possible in a screening setting, the moderate configuration (`mfinal = 30`, `maxdepth = 3`, `minsplit = 20`) is a more appealing trade-off; if we instead emphasize avoiding false positives and maximizing overall accuracy, the default configuration is preferable.

```
## Total time to run: 6.62 secs
```