Classification

Exercise 01

In this exercise, we apply logistic regression to breast cancer data. The problem entails classifying cancer as malignant or benign based on biopsy samples. Consider the code below and answer the following questions, considering malignant as the "positive" class:

(a) Explain what the recall performance metric is and manually calculate this value according to the results presented in the confusion matrix.
(b) Explain what the precision performance metric is and manually calculate this value according to the results presented in the confusion matrix.
(c) Manually calculate the sensitivity and specificity performance metrics using the numbers from the confusion matrix.
(d) Check your results with the recall, precision, sensitivity, and specificity functions of the Yardstick package.

```r
library(yardstick)
library(mlbench)
library(tidymodels)

# Load the data
data("BreastCancer")
bc_df <- as_tibble(BreastCancer) %>%
na.omit()

# Preprocessing recipe
rec <- recipe(Class ~ ., bc_df) %>%
step_rm(Id) %>%
# Turn ordinal categorical features into numerical values
step_ordinalscore(Cl.thickness,
  Cell.size,
  Cell.shape,
  Marg.adhesion,
  Epith.c.size) %>%
# Turn nominal categorical features (except for the target) into dummy variables
step_dummy(all_nominal(),-all_outcomes()) %>%
# Remove Mitoses 6
step_rm(Mitoses_X6) %>%
# Center and scale the data
step_center(all_predictors()) %>%
step_scale(all_predictors())
# Create a regularized Logistic Regression model
lr.model <- logistic_reg(penalty = 0, mixture = NULL
) %>%
set_engine("glmnet")
# Split the dataset into training and testing sets
```

```r
set.seed(123)
split <- initial_split(bc_df, prop = 0.70)
train <- training(split)
test <- testing(split)
# Prepare the recipe with training data
rec.prep <- rec %>% prep(train)
# Retrieve training and testing data from the recipe
train.prep <- juice(rec.prep)
test.prep <- bake(rec.prep, test)
# Fit the model on preprocessed training data
lr.fit <- lr.model %>% fit(Class ~ ., train.prep)
# Make predictions for preprocessed testing data
test.pred <- test.prep %>%
bind_cols(lr.fit %>% predict(new_data = test.prep))
# Calculate the confusion matrix
conf_mat(test.pred,Class,.pred_class)$table
```

```
##             Truth
## Prediction  benign malignant
##   benign       136         5
##   malignant      3        61
```

```r
# Calculate performance metrics using Yardstick
recall(test.pred,Class,.pred_class, event_level = "second")
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 recall  binary         0.924
```

```r
precision(test.pred,Class,.pred_class, event_level = "second")
```

```
## # A tibble: 1 x 3
##   .metric   .estimator .estimate
##   <chr>     <chr>          <dbl>
## 1 precision binary         0.953
```

```r
sensitivity(test.pred,Class,.pred_class, event_level = "second")
```

```
## # A tibble: 1 x 3
##   .metric     .estimator .estimate
##   <chr>       <chr>          <dbl>
## 1 sensitivity binary         0.924
```

```r
specificity(test.pred,Class,.pred_class, event_level = "second")
```

```
## # A tibble: 1 x 3
##   .metric     .estimator .estimate
##   <chr>       <chr>          <dbl>
## 1 specificity binary         0.978
```

a) Recall is the ratio of true positives to the total of observations in the positive class. In other words, it identifies the proportion of true positives that were correctly identified. It should be used when identifying false negatives is more crucial than identifying false positives - in health or security-critical applications, for instance. It can be calculated as the ratio (TRUE POSITIVES) / (TRUE POSITIVES + FALSE NEGATIVES). In this case, we have 61 / (61 + 5), which is equivalent to 0.924242.

b) Precision is the ratio of true positives to the total of observations classified as positive. In other words, it identifies the proportion of correctly classified positives. It is generally used in situations where identifying false positives is more crucial than identifying false negatives. It can be calculated as the ratio (TRUE POSITIVES) / (TRUE POSITIVES + FALSE POSITIVES). In this case, we have 61 / (61 + 3), which yields a value of 0.953125.

c) Sensitivity: ratio of true positives to the total of observations in the positive class. Calculating manually, we obtain 61 / (61 + 5) = 0.924242. Specificity: ratio of true negatives to the total of observations in the negative class. Calculating manually, we obtain 136 / (136 + 3) = 0.9784173.

d) In the code.

Exercise 02

Continuing with the previous exercise, adjust the parameters for the SVM and Logistic Regression models based on the area under the ROC curve (AUC). For Logistic Regression, we will tune the mixture and penalty parameters to perform a regularized regression. For SVM, we will use an exponential kernel (or rbf) and tune the kernel parameter σ, as well as the soft margin penalty factor C.

```r
## Logistic Regression

# Create a Logistic Regression model with parameters to be tuned
lr.model <- logistic_reg(penalty = tune(),
mixture = tune()
) %>% set_engine("glmnet")

# Set up the grid
grd <- grid_max_entropy(penalty(),
mixture(),
size = 10)

# 10-fold cross validation
folds <- vfold_cv(bc_df,v=10)

# Store the AUC and accuracy score
metrs <- metric_set(roc_auc, accuracy)

# Tune the parameters
tune.res <- tune_grid( lr.model,
rec,
resamples = folds,
grid = grd,
metrics = metrs
)

# Collect metrics
tune.res %>% unnest(.metrics)
```

```
## # A tibble: 200 x 9
##    splits           id     penalty mixture .metric  .estimator
##    <list>           <chr>    <dbl>   <dbl> <chr>    <chr>
##  1 <split [614/69]> Fold01 5.86e- 2  0.0362 accuracy binary
##  2 <split [614/69]> Fold01 5.86e- 2  0.0362 roc_auc  binary
##  3 <split [614/69]> Fold01 2.92e- 6  0.0551 accuracy binary
##  4 <split [614/69]> Fold01 2.92e- 6  0.0551 roc_auc  binary
##  5 <split [614/69]> Fold01 2.89e-10  0.183  accuracy binary
##  6 <split [614/69]> Fold01 2.89e-10  0.183  roc_auc  binary
##  7 <split [614/69]> Fold01 8.35e- 7  0.422  accuracy binary
##  8 <split [614/69]> Fold01 8.35e- 7  0.422  roc_auc  binary
##  9 <split [614/69]> Fold01 6.11e- 1  0.491  accuracy binary
## 10 <split [614/69]> Fold01 6.11e- 1  0.491  roc_auc  binary
## # i 190 more rows
## # i 3 more variables: .estimate <dbl>, .config <chr>, .notes <list>
```

```
tune.res %>% collect_metrics()
```

```
## # A tibble: 20 x 8
##     penalty mixture .metric  .estimator  mean     n std_err .config
##       <dbl>   <dbl> <chr>    <chr>      <dbl> <int>   <dbl> <chr>
##  1 5.86e- 2  0.0362 accuracy binary     0.962    10 0.00617 Preproces~
##  2 5.86e- 2  0.0362 roc_auc  binary     0.993    10 0.00215 Preproces~
##  3 2.92e- 6  0.0551 accuracy binary     0.955    10 0.00592 Preproces~
##  4 2.92e- 6  0.0551 roc_auc  binary     0.987    10 0.00333 Preproces~
##  5 2.89e-10  0.183  accuracy binary     0.952    10 0.00652 Preproces~
##  6 2.89e-10  0.183  roc_auc  binary     0.985    10 0.00366 Preproces~
##  7 8.35e- 7  0.422  accuracy binary     0.950    10 0.00694 Preproces~
##  8 8.35e- 7  0.422  roc_auc  binary     0.984    10 0.00422 Preproces~
##  9 6.11e- 1  0.491  accuracy binary     0.650    10 0.0161  Preproces~
## 10 6.11e- 1  0.491  roc_auc  binary     0.988    10 0.00306 Preproces~
## 11 1.49e- 4  0.604  accuracy binary     0.952    10 0.00652 Preproces~
## 12 1.49e- 4  0.604  roc_auc  binary     0.984    10 0.00413 Preproces~
## 13 4.41e- 9  0.609  accuracy binary     0.950    10 0.00694 Preproces~
## 14 4.41e- 9  0.609  roc_auc  binary     0.983    10 0.00428 Preproces~
## 15 6.45e- 7  0.942  accuracy binary     0.952    10 0.00690 Preproces~
## 16 6.45e- 7  0.942  roc_auc  binary     0.982    10 0.00492 Preproces~
## 17 2.37e- 2  0.948  accuracy binary     0.955    10 0.00701 Preproces~
## 18 2.37e- 2  0.948  roc_auc  binary     0.992    10 0.00183 Preproces~
## 19 3.30e-10  0.961  accuracy binary     0.952    10 0.00690 Preproces~
## 20 3.30e-10  0.961  roc_auc  binary     0.982    10 0.00491 Preproces~
```

```
# Retrieve the mean value for each metric
tune.res %>%
unnest(.metrics) %>%
group_by(penalty,mixture,.metric) %>%
summarise( mean_estimate = mean(.estimate))
```

```
## # A tibble: 20 x 4
## # Groups:   penalty, mixture [10]
##     penalty mixture .metric  mean_estimate
##       <dbl>   <dbl> <chr>            <dbl>
##  1 2.89e-10  0.183  accuracy         0.952
##  2 2.89e-10  0.183  roc_auc         0.985
##  3 3.30e-10  0.961  accuracy        0.952
##  4 3.30e-10  0.961  roc_auc         0.982
##  5 4.41e- 9  0.609  accuracy        0.950
##  6 4.41e- 9  0.609  roc_auc         0.983
##  7 6.45e- 7  0.942  accuracy        0.952
##  8 6.45e- 7  0.942  roc_auc         0.982
##  9 8.35e- 7  0.422  accuracy        0.950
## 10 8.35e- 7  0.422  roc_auc         0.984
## 11 2.92e- 6  0.0551 accuracy        0.955
## 12 2.92e- 6  0.0551 roc_auc         0.987
## 13 1.49e- 4  0.604  accuracy        0.952
## 14 1.49e- 4  0.604  roc_auc         0.984
## 15 2.37e- 2  0.948  accuracy        0.955
## 16 2.37e- 2  0.948  roc_auc         0.992
## 17 5.86e- 2  0.0362 accuracy        0.962
## 18 5.86e- 2  0.0362 roc_auc         0.993
## 19 6.11e- 1  0.491  accuracy        0.650
## 20 6.11e- 1  0.491  roc_auc         0.988
```

```
# Select the top 3 models based on the AUC
best_log_models = show_best(tune.res, metric = "roc_auc", n = 3)
print(best_log_models)
```

```
## # A tibble: 3 x 8
##   penalty mixture .metric .estimator  mean     n std_err .config
##     <dbl>   <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1  0.0586  0.0362 roc_auc binary     0.993    10 0.00215 Preprocessor~
## 2  0.0237  0.948  roc_auc binary     0.992    10 0.00183 Preprocessor~
## 3  0.611   0.491  roc_auc binary     0.988    10 0.00306 Preprocessor~
```

```
## SVM
library(kernlab)

# Create a SVM model with parameters to be tuned
svm.model <- svm_rbf(mode = "classification",
                     cost = tune(),
                     rbf_sigma = tune()) %>%
                     set_engine("kernlab")



# Set up the grid
grd <- grid_max_entropy(cost(),
  rbf_sigma(),
  size = 10)

# 10-fold cross validation
folds <- vfold_cv(bc_df,v=10)

# Store the AUC and accuracy score
metrs <- metric_set(roc_auc,accuracy)

# Tune the parameters
```

```r
tune.res <- tune_grid(
svm.model,
rec,
resamples = folds,
grid = grd,
metrics = metrs
)

# Collect metrics
tune.res %>% unnest(.metrics)
```

```
## # A tibble: 200 x 9
##    splits            id          cost rbf_sigma .metric  .estimator
##    <list>            <chr>      <dbl>     <dbl> <chr>    <chr>
##  1 <split [614/69]> Fold01  0.00140   3.29e- 4 accuracy binary
##  2 <split [614/69]> Fold01  0.00140   3.29e- 4 roc_auc  binary
##  3 <split [614/69]> Fold01  0.000989  2.12e-10 accuracy binary
##  4 <split [614/69]> Fold01  0.000989  2.12e-10 roc_auc  binary
##  5 <split [614/69]> Fold01  0.00743   1.36e- 7 accuracy binary
##  6 <split [614/69]> Fold01  0.00743   1.36e- 7 roc_auc  binary
##  7 <split [614/69]> Fold01 22.7       4.45e- 9 accuracy binary
##  8 <split [614/69]> Fold01 22.7       4.45e- 9 roc_auc  binary
##  9 <split [614/69]> Fold01  0.382     5.95e- 6 accuracy binary
## 10 <split [614/69]> Fold01  0.382     5.95e- 6 roc_auc  binary
## # i 190 more rows
## # i 3 more variables: .estimate <dbl>, .config <chr>, .notes <list>
```

```r
tune.res %>% collect_metrics()
```

```
## # A tibble: 20 x 8
##         cost rbf_sigma .metric  .estimator  mean     n std_err .config
##        <dbl>     <dbl> <chr>    <chr>      <dbl> <int>   <dbl> <chr>
##  1  0.00140   3.29e- 4 accuracy binary     0.650    10 0.0255  Prepro~
##  2  0.00140   3.29e- 4 roc_auc  binary     0.993    10 0.00233 Prepro~
##  3  0.000989  2.12e-10 accuracy binary     0.650    10 0.0255  Prepro~
##  4  0.000989  2.12e-10 roc_auc  binary     0.993    10 0.00227 Prepro~
##  5  0.00743   1.36e- 7 accuracy binary     0.650    10 0.0255  Prepro~
##  6  0.00743   1.36e- 7 roc_auc  binary     0.993    10 0.00238 Prepro~
##  7 22.7       4.45e- 9 accuracy binary     0.650    10 0.0255  Prepro~
##  8 22.7       4.45e- 9 roc_auc  binary     0.993    10 0.00235 Prepro~
##  9  0.382     5.95e- 6 accuracy binary     0.650    10 0.0255  Prepro~
## 10  0.382     5.95e- 6 roc_auc  binary     0.994    10 0.00231 Prepro~
## 11  7.33      5.12e- 1 accuracy binary     0.939    10 0.00611 Prepro~
## 12  7.33      5.12e- 1 roc_auc  binary     0.976    10 0.00355 Prepro~
## 13 31.7       7.43e- 5 accuracy binary     0.968    10 0.00782 Prepro~
## 14 31.7       7.43e- 5 roc_auc  binary     0.993    10 0.00306 Prepro~
## 15  0.00194   6.17e- 1 accuracy binary     0.650    10 0.0255  Prepro~
## 16  0.00194   6.17e- 1 roc_auc  binary     0.978    10 0.00360 Prepro~
## 17  0.102     5.75e- 2 accuracy binary     0.917    10 0.0109  Prepro~
## 18  0.102     5.75e- 2 roc_auc  binary     0.988    10 0.00304 Prepro~
## 19  0.203     4.39e-10 accuracy binary     0.650    10 0.0255  Prepro~
## 20  0.203     4.39e-10 roc_auc  binary     0.993    10 0.00238 Prepro~
```

```
# Retrieve the mean value for each metric
tune.res %>%
unnest(.metrics) %>%
group_by(rbf_sigma, cost, .metric) %>%
summarise( mean_estimate = mean(.estimate))
```

```
## # A tibble: 20 x 4
## # Groups:   rbf_sigma, cost [10]
##    rbf_sigma      cost .metric  mean_estimate
##        <dbl>     <dbl> <chr>            <dbl>
##  1  2.12e-10  0.000989 accuracy         0.650
##  2  2.12e-10  0.000989 roc_auc          0.993
##  3  4.39e-10  0.203    accuracy         0.650
##  4  4.39e-10  0.203    roc_auc          0.993
##  5  4.45e- 9 22.7      accuracy         0.650
##  6  4.45e- 9 22.7      roc_auc          0.993
##  7  1.36e- 7  0.00743  accuracy         0.650
##  8  1.36e- 7  0.00743  roc_auc          0.993
##  9  5.95e- 6  0.382    accuracy         0.650
## 10  5.95e- 6  0.382    roc_auc          0.994
## 11  7.43e- 5 31.7      accuracy         0.968
## 12  7.43e- 5 31.7      roc_auc          0.993
## 13  3.29e- 4  0.00140  accuracy         0.650
## 14  3.29e- 4  0.00140  roc_auc          0.993
## 15  5.75e- 2  0.102    accuracy         0.917
## 16  5.75e- 2  0.102    roc_auc          0.988
## 17  5.12e- 1  7.33     accuracy         0.939
## 18  5.12e- 1  7.33     roc_auc          0.976
## 19  6.17e- 1  0.00194  accuracy         0.650
## 20  6.17e- 1  0.00194  roc_auc          0.978
```

```
# Select the top 3 models based on the AUC
best_svm_models <- show_best(tune.res, metric = "roc_auc", n = 3)
print(best_svm_models)
```

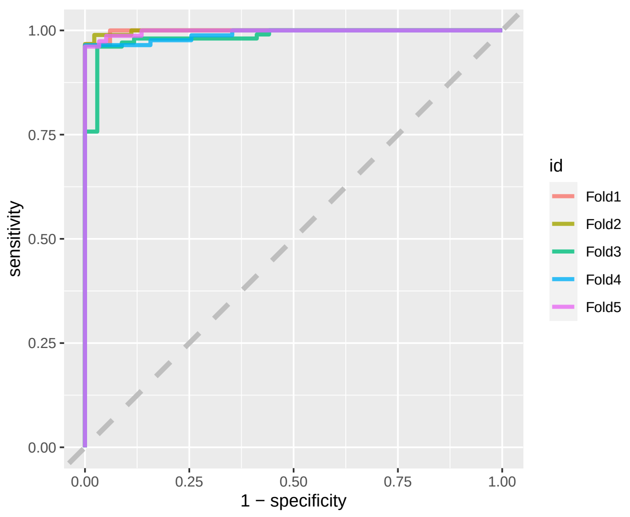
```
## # A tibble: 3 x 8
##       cost    rbf_sigma .metric .estimator  mean     n std_err .config
##      <dbl>        <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1  0.382        5.95e-6 roc_auc binary     0.994    10 0.00231 Prepro~
## 2 22.7          4.45e-9 roc_auc binary     0.993    10 0.00235 Prepro~
## 3  0.00743      1.36e-7 roc_auc binary     0.993    10 0.00238 Prepro~
```

Exercise 03

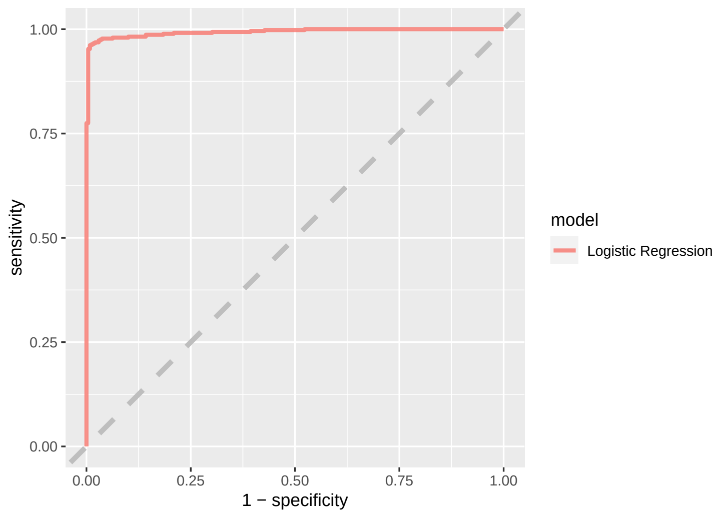
Now that we have already chosen the optimal parameters for the Logistic Regression and SVM with rbf kernel models, we will plot the ROC curves obtained through cross-validation for both models - using the optimal parameters obtained from the previous exercise. In addition to including the best parameters for Logistic Regression, create a plot for the optimized SVM model.

```
## Logistic Regression
```

```r
# Retrieve the optimal parameters found in the previous exercise
pen <- best_log_models %>%
  arrange(desc(mean)) %>%
  slice(1:1) %>% pull(penalty)

mix <- best_log_models %>%
  arrange(desc(mean)) %>%
  slice(1:1) %>% pull(mixture)

# Perform Logistic Regression
lr.model <- logistic_reg( penalty = pen,
mixture = mix   ) %>%
set_engine("glmnet")

# 5-fold cross validation
folds <- vfold_cv(v = 5,bc_df)

fit.res <- fit_resamples( lr.model,
rec,
resamples = folds,
# Save the predicitons
control = control_resamples(save_pred = TRUE) )


# Plot the ROC curve for each fold
predictions <- fit.res %>%
unnest(.predictions)

predictions %>%
group_by(id) %>%
roc_curve(Class,.pred_benign) %>%
ggplot(aes(x = 1-specificity, y = sensitivity, color = id) ) +
geom_path(size = 1.2, alpha = 0.8) +
geom_abline(lty = 2, color = "gray", size = 1.5) +
coord_equal()
```



```r
# Plot the final ROC curve
predictions %>%
mutate( model = "Logistic Regression") %>%
group_by(model) %>%
roc_curve(Class,.pred_benign) %>%
ggplot(aes(x = 1-specificity, y = sensitivity, color = model) ) +
geom_path(size = 1.2, alpha = 0.8) +
geom_abline(lty = 2, color = "gray", size = 1.5) +
coord_equal()
```



```r
## SVM

# Retrieve the optimal parameters found in the previous exercise
cos <- best_svm_models %>%
  arrange(desc(mean)) %>%
  slice(1:1) %>% pull(cost)
print(cos)
```

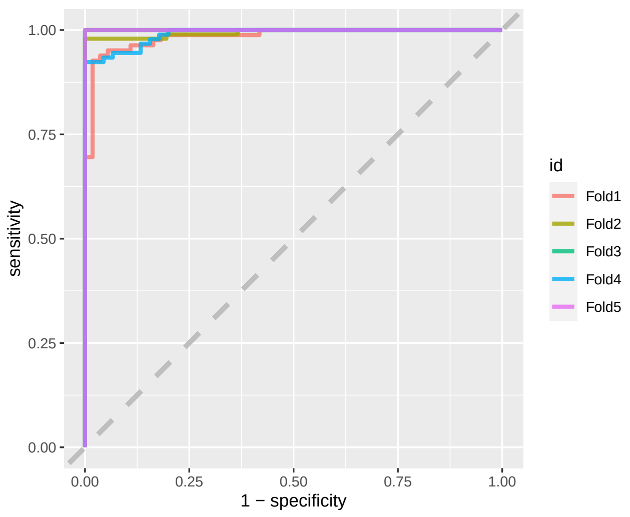
```
## [1] 0.3818131
```

```r
sig <- best_svm_models %>%
  arrange(desc(mean)) %>%
  slice(1:1) %>% pull(rbf_sigma)
print(sig)
```

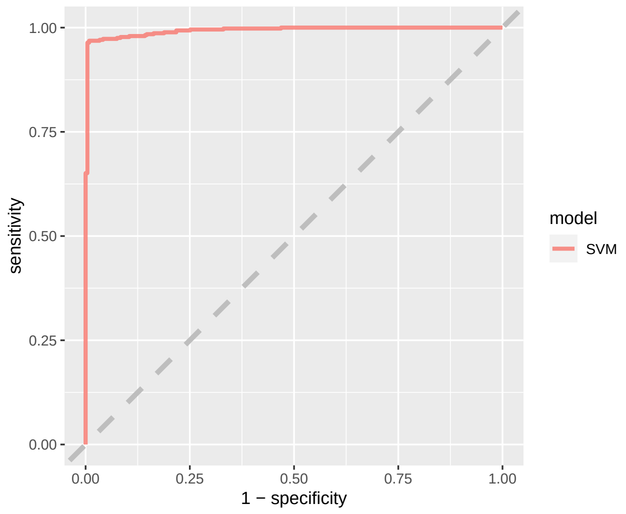
```
## [1] 5.945835e-06
```

```r
# Run SVM
svm.model <- svm_rbf(
  mode = "classification",
  rbf_sigma = sig,
  cost = cos) %>% set_engine("kernlab")

# 5-fold cross validation
```

```r
folds <- vfold_cv(v = 5,bc_df)
fit.res <- fit_resamples( svm.model,
rec,
resamples = folds,
# Save the predictions
control = control_resamples(save_pred = TRUE) )

predictions <- fit.res %>%
unnest(.predictions)

# Plot the ROC curve for each fold
predictions %>%
group_by(id) %>%
roc_curve(Class,.pred_benign) %>%
ggplot(aes(x = 1-specificity, y = sensitivity, color = id) ) +
geom_path(size = 1.2, alpha = 0.8) +
geom_abline(lty = 2, color = "gray", size = 1.5) +
coord_equal()
```



```r
# Plot the final ROC curve
predictions %>%
mutate(model = "SVM") %>%
group_by(model) %>%
roc_curve(Class,.pred_benign) %>%
ggplot(aes(x = 1-specificity, y = sensitivity, color = model) ) +
geom_path(size = 1.2, alpha = 0.8) +
geom_abline(lty = 2, color = "gray", size = 1.5) +
coord_equal()
```



Exercise 04

The Perceptron algorithm is one of the simplest classification algorithms. In this exercise, we will implement and test it on the iris database. We can perform binary classification by distinguishing between ‘setosa’ and ‘other’ classes, which results in a linearly separable problem.

Load the iris database and assign the label ‘other’ for observations where the Species is not ‘setosa’. Implement Perceptron using the perceptron.fit and perceptron.predict functions.

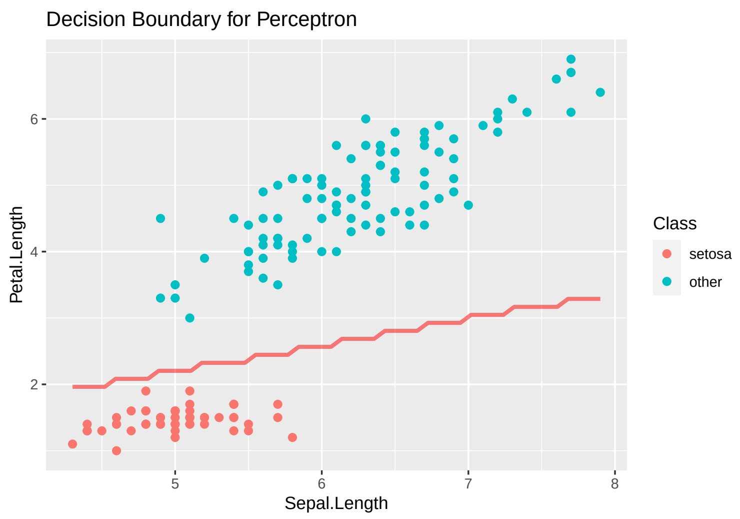
The perceptron.fit function fits the model with the training data, and the perceptron.predict function should return the output vector for x using the following criterion: if $wtx + b > 0$, the output is ‘other’, and if $wtx + b < 0$, the output is ‘setosa’, where w and b are tuned by the perceptron.fit function.

Once you have completed the code implementation, calculate the model accuracy using a k-fold cross-validation process. Please note that the ‘fit_resamples’ function cannot be used in this case, as it is specific to models from the tidymodels ecosystem.

```r
# Load and transform the data
data("iris")
df <- as_tibble(iris) %>%
    mutate(Species = as.character(Species)) %>%
    mutate(Class = if_else(Species == "setosa", Species, "other")) %>%
    select(-Species) %>%
    mutate(Class = factor(Class, levels = c("setosa", "other")))

# Function to fit the Perceptron
perceptron.fit <- function(
    form,
    df,
    eta = 0.01)
{
    # Retrieve the training dataset built from the formula:
    train_df <- model.frame(form, df)
    # Retrieve the column with y values and input the numbers -1 and +1
    classes <- levels(train_df[, 1])
    y <- train_df[, 1] %>% as.integer()
    y <- (y - 1) * 2 - 1
    # Retrieve the column with X values
    X <- as.matrix(train_df[, -1])
    # Normal vector of the hyperplane and the intercept
    w <- vector("numeric", length = ncol(X))
    b <- 0
    # Implement the Perceptron:
    e <- 1
    while (e != 0) {
        e <- 0
        # Shuffling the index set
        index <- sample(1:nrow(X))
        for (i in 1:nrow(X)) {
            j <- index[i]
            x <- as.numeric(X[j, ])
            y_pred <- sign(sum(w * x) + b)
            if (y_pred * y[j] <= 0) {
              w <- w + eta * y[j] * x
              b <- b + eta * y[j]
              e <- e + 1
            }
        }
    }
    return(list(
        "formula" = form,
        "classes" = classes,
        "normal" = w,
        "y_intercept" = b
    ))
}

# Funtion to make predictions
perceptron.predict <- function(
    percep.fit,
    new_data)
{
    form <- percep.fit$formula
    classes <- percep.fit$classes
    w <- percep.fit$normal
    b <- percep.fit$y_intercept

    class_column <- as.character(form[2])
    if (class_column %in% names(new_data)) {
        test_df <- model.frame(form[-2], new_data %>% select(-class_column))
    } else {
        test_df <- model.frame(form[-2], new_data)
    }

    X <- as.matrix(test_df)
    pred <- factor(vector("character", length = nrow(X)), levels = classes)
    for (i in 1:nrow(X))
    {
        x <- as.numeric(X[i, ])
        y_pred <- sign(sum(w * x) + b)
        if (y_pred == -1) {
          pred[i] <- classes[1]
        } else {
          pred[i] <- classes[2]
        }
    }
    return(tibble(.pred = pred))
}

# Cross validation
splits <- vfold_cv(df, k = 10, repeats = 3)
acc_results <- vector("numeric", length = nrow(splits))
for (i in 1:nrow(splits))
{
    s <- splits$splits[[i]]
    train <- analysis(s)
    test <- assessment(s)
    percep.fit <- perceptron.fit(Class ~ ., train)
    test_pred <- perceptron.predict(percep.fit, test) %>%
        bind_cols(test) %>%
        accuracy(Class, .pred)
    acc_results[i] <- test_pred$.estimate
}
cat("Average accuracy score = ", mean(acc_results), "\n")
```

```
## Average accuracy score =  0.9977778
```

```
## Plot

library(modelr)
# Create a grid with data points
plot_grid <- expand_grid( Sepal.Length = seq_range(df$Sepal.Length,50),
Petal.Length = seq_range(df$Petal.Length,50))
df <- df %>% select( Sepal.Length, Petal.Length, Class )

# Add predictions for the classes
percep.fit <- perceptron.fit(Class ~ ., df)
plot_grid_pred <- plot_grid %>%
mutate( pred = perceptron.predict(percep.fit,plot_grid)$.pred)

# create the plot
ggplot(plot_grid_pred, aes(Sepal.Length,Petal.Length))+
geom_contour(aes(z= as.integer(pred)),
alpha = 0.5, show.legend = F,breaks = c(1L,2L),
size=1.2, color ="red") +
geom_point(data = df, aes(z=NULL,colour = Class),size=2) +
labs(title = "Decision Boundary for Perceptron")
```


Decision Boundary for Perceptron
Petal.Length
Sepal.Length
Class
setosa
other
2
4
6
5
6
7
8