Homework 7 Solution

Disclaimer

This homework solution is for the sole benefit of students taking Stat 220 from Prof. Bastola during Fall term 2022. Dissemination of this solution to people who are not registered for this course is not permitted and will be considered grounds for Academic Dishonesty for the all individuals involved in the giving and receiving of the solution.

Assignment prompt

Problem 1: Random classifiers

Suppose you have n observations and you would like to construct a classifier to predict positive outcomes. We know that n_N are actually negative cases and n_P are actually positive cases.

Suppose we are a lazy data scientist and don't actually build a model with predictors to classify our cases. Instead, we use a fair coin to classify negative and positive cases! This "coin classifier" will result in an even split between *expected* positive and negative predictions:

$$\hat{n}_N = \hat{n}_P = \frac{n}{2}$$

Using the same logic, half of the actual negatives will be positives and positives:

$$TN = FP = \frac{n_N}{2}$$

and half of the actual positives will be positives and negatives:

$$FN = TP = \frac{n_P}{2}$$

confusion matrix	truth: negative	truth positive	total
predict negative	$TN = \frac{n_N}{2}$	$FN = \frac{n_P}{2}$	$\hat{n}_N = \frac{n}{2}$
predict positive	$FP = \frac{n_N}{2}$	$TP = \frac{n_P}{2}$	$\hat{n}_P = \frac{n}{2}$
total	n_N	n_P	n

a.

Use the "coin classifier" confusion matrix above to compute accuracy, precision, sensitivity, and specificity for this fair coin prediction method.

Answer: With a fair coin flip predicting positives, the accuracy will be 50%.

$$accuracy = \frac{TN + TP}{n} = \frac{0.5n_N + 0.5n_P}{n} = 0.5\frac{n_N + n_P}{n} = 0.5$$

Precision will be equal to the rate of positives in the sample since 50% of the positives are correctly predicted and 50% of the overall sample is predicted to be positives:

$$precision = \frac{TP}{\hat{n}_P} = \frac{0.5n_P}{0.5n} = \frac{n_P}{n}$$

Sensitivity will be 50% since that is the rate at which we correctly predict positives:

$$Sensitivity = \frac{TP}{n_P} = \frac{0.5 n_P}{n_P} = 0.5$$

Specificity will also be 50% since that is the rate at which we correctly predict negatives:

$$Specificity = \frac{TN}{n_N} = \frac{0.5 n_N}{n_N} = 0.5$$

b.

Consider another classification method that doesn't actually build a model with predictors to classify our cases. Instead, we now predict all cases to be positive! (e.g. a biased coin that lands "positive" with probability 1)

Construct the confusion matrix for this classifier and compute accuracy, precision, sensitivity, and specificity.

Answer: Here all cases are positive, so $TN = 0$ (no negatives predicted) and $FN = 0$ (no negatives predicted). All negative cases are predicted positive, so $FP = n_N$ and all positive cases are predicted positive, so $TP = n_P$.

confusion matrix	truth negative	truth positive	total
predict negative	$TN = 0$	$FN = 0$	$\hat{n}_N = 0$
predict positive	$FP = n_N$	$TP = n_P$	$\hat{n}_P = n$
total	n_N	n_P	n

With a probability p predicting positives, the accuracy will be the rate of positives in the sample:

$$accuracy = \frac{TN + TP}{n} = \frac{0 + n_P}{n} = \frac{n_P}{n}$$

Precision will be equal to the rate of positives in the sample:

$$precision = \frac{TP}{\hat{n}_P} = \frac{n_P}{n}$$

Sensitivity will be 100% since that is the rate at which we correctly predict positives:

$$Sensitivity = \frac{TP}{n_P} = \frac{n_P}{n_P} = 1$$

Specificity will be 0% since that is the rate at which we correctly predict negatives:

$$Specificity = \frac{TN}{n_N} = \frac{0}{n_N} = 0$$

c.

Under what conditions in part (b) will accuracy of this classifier be highest? Under what conditions in part (b) will accuracy of this classifier be lowest?

Answer: Accuracy is the rate of positive cases in the data, $\frac{n_P}{n}$. So when we have a lot of positive cases in the data, predicting all cases to be positive will yield high accuracy. But when we have few positive cases the accuracy will be low.

d.

Describe the trade-off in part (b) in specificity vs sensitivity. What would these values be if instead of classifying every case as positive, you classified all cases as negative?

Answer: We have sensitivity of 1 because we are finding all true positive cases when predicting all positives. But we have specificity of 0 because no true negatives are correctly predicted. If we reverse the classification and predict all negative cases, the opposite will occur with specificy of 1 and sensitivity of 0.

Problem 2: Spam using k-nn

This example looks at a data set of about 4600 emails that are classified as spam or not spam, along with over 50 variables measuring different characteristic of the email. Details about these variables are found on the Spambase example on the machine learning data archive. The dataset linked to below is a slightly cleaned up version of this data. The only extra column in the data is `rgroup` which is a randomly assigned grouping variable (groups 0 through 99) which we will eliminate from the data.

Read the data in using the commands below to create a response `class` variable that contains the factor levels `spam` and `nonspam` with `spam` the first level.

```
> # tsv = tab separated values!
> spam <- read_delim("https://raw.githubusercontent.com/deepbas/statdatasets/main/spam.txt",
+          delim="\t")
>
> # some clean up
> spam <- spam %>%
+   mutate(class = fct_recode(
+     spam,
+     spam = "spam" ,
+     nonspam = "non-spam"), # rename levels because caret doesn't like "non-spam"
+     class = fct_relevel(class, "spam") # make "spam" the first level (our "positive")
+     ) %>%
+   select(-rgroup, -spam) # don't need random group variable and  spam variable
> levels(spam$class)  # verify "spam" is level 1
[1] "spam"    "nonspam"
```

What proportion of cases are spam in this data set? The variable `class` is the true classification of an email.

answer:

```
> spam %>%
+   count(class) %>%
+   mutate(n/sum(n))
# A tibble: 2 x 3
  class       n `n/sum(n)`
  <fct>   <int>      <dbl>
1 spam     1813      0.394
2 nonspam  2788      0.606
```

b.

We want to fit a k-nn classifier for `spam` using the 57 quantitative predictors (columns 1-57) from the spam data to an 80%/20% training and test set split. Create the training and test sets to do this.

Use the following seed and the `sample` command generate training data (this *should* mean your train/test split should be the same as the solution key!):

```
> set.seed(757302859)  # set a seed
```

answer:

```
> set.seed(757302859)  # set a seed
>
> spam_split <- initial_split(spam, prop = 0.80)
> # Create training data
> spam_train <- spam_split %>%
+                     training()
> # Create testing data
> spam_test <- spam_split %>%
+                     testing()
```

c.

Make a recipe for fitting k nearest-neighbor algorithm to the training data by inputting the formula and preprocessing steps.

```
> spam_recipe <- recipe(class ~ ., data = spam_train) %>%
+   step_scale(all_predictors()) %>%
+   step_center(all_predictors())
```

d.

Specify the nearest neighbor model to fit a classification model using 10 neighbors.

```
> spam_knn_spec <- nearest_neighbor(mode = "classification",
+                                   engine = "kknn",
+                                   weight_func = "rectangular",
+                                   neighbors = 10)
```

e.

Define the workflow object feeding in the recipe and model specification. Then, fit the model to the training data. *answer:*

```
> spam_workflow <- workflow() %>%
+   add_recipe(spam_recipe) %>%
+   add_model(spam_knn_spec)
>
> spam_fit <- fit(spam_workflow, data = spam_train)
```

f.

Evaluate the model on the test dataset and predict `class` for the test data set.

```
> test_features <- spam_test  %>% select(-class)
> spam_pred <- predict(spam_fit, test_features, type = "raw")
> spam_results <- spam_test %>%
+   select(class) %>%
+   bind_cols(predicted = spam_pred)
```

g.

Compute the accuracy, sensitivity, specificity and precision for predicting test set responses. Which rate(s) would you like to be high as a user of this spam filter?

Answer:

As someone who rarely looks in my spam folder, I want to make sure that there are very few "false positives", meaning very few non-spam emails that get dumped in my spam folder. So I want high specificity and high precision in the filter.

```
> custom_metrics <- metric_set(accuracy, sens, spec, ppv)
> metrics <- custom_metrics(spam_results,
+                 truth = class,
+                 estimate = predicted)
> metrics <- metrics %>% select(-.estimator)
```

h.

Use the `tidymodels` package to do 10-fold cross validation as follows:

- use the 80% training data split from part b.
- tune your knn spam classifier based on accuracy
- consider neighborhood sizes ranging from size 1 to 31

After running `train` to produce the 10-fold CV results (this could take ~1 minute), use the `results` from your `train` object to get the training set cross-validated estimates of the accuracy, precision, sensitivity and specificity of your final ("best") classifier.

And use the following seed before running your `train` command:

```
> set.seed(30498492)
```

answer

Run CV for knn:

```
> spam_recipe_new <- recipe(class ~ ., data = spam_train) %>%
+   step_scale(all_predictors()) %>%
+   step_center(all_predictors())
```

```
> knn_spec <- nearest_neighbor(
+   weight_func = "rectangular",
+   neighbors = tune()
+ ) %>%
+   set_engine("kknn") %>%
+   set_mode("classification")
```

```
> spam_vfold <- vfold_cv(spam_train, v = 10, strata = class)
```

```
> k_vals <- tibble(neighbors = seq(from = 1, to = 30, by = 1))
```

```
> knn_fit <- workflow() %>%
+   add_recipe(spam_recipe_new) %>%
+   add_model(knn_spec) %>%
+   tune_grid(
+     resamples = spam_vfold,
+     grid = k_vals,
+     metrics = custom_metrics
+     )
```

i.

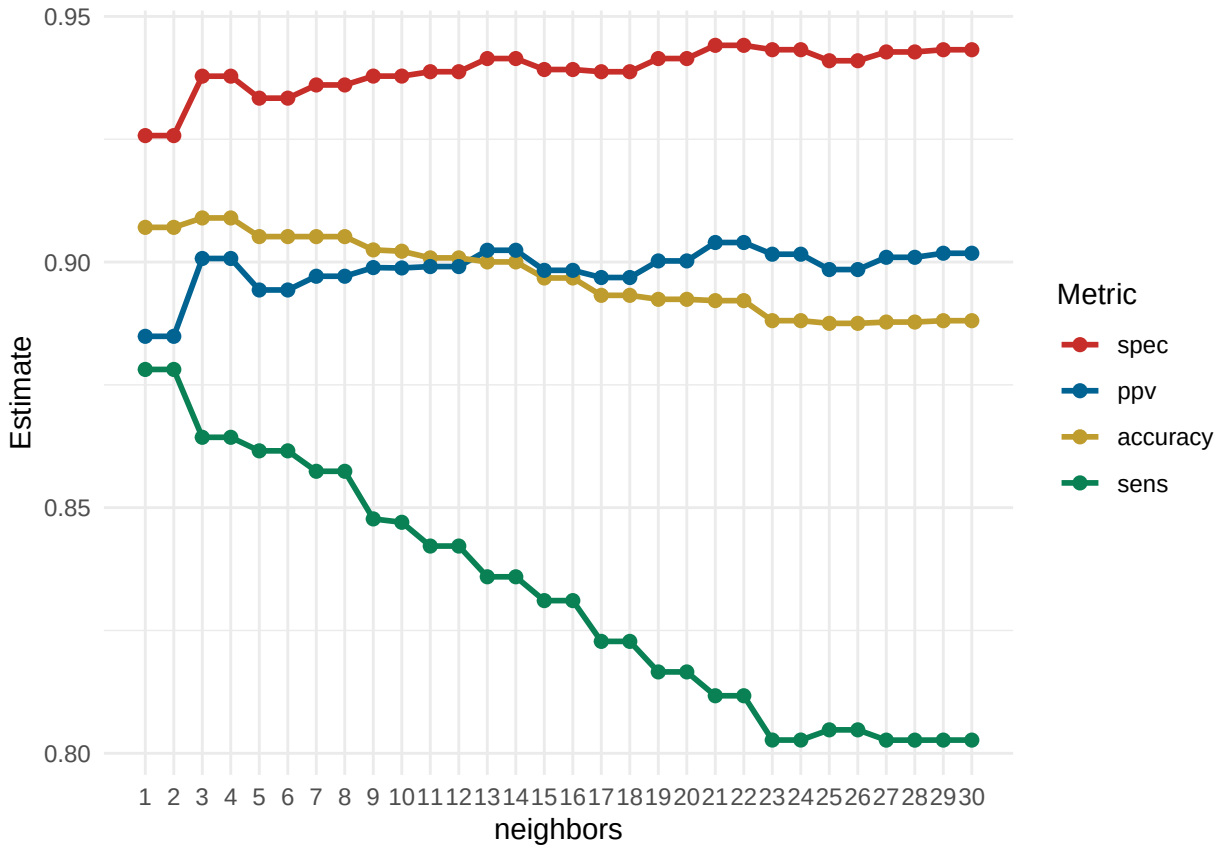
Use the `results` object from your CV fit from part (i) to plot the accuracy, precision, sensitivity and specificity for the values of k that you tuned over. Comment on which neighborhood sizes are "best" in terms

of these four metrics (your answer may depend on the metric).

answer

Low values of k seem best for high precision, accuracy and sensitivity while higher values yield higher specificity. Since "non-spam" is the majority of cases (~2/3 of cases are non-spam), the larger the neighborhood the more likely it is that we see majority "non-spam" which means higher specificity (correctly predicting non-spam cases).

```
> cv_metrics <- collect_metrics(knn_fit)
> final.results <- cv_metrics %>%  mutate(.metric = as.factor(.metric)) %>%
+   select(neighbors, .metric, mean)
>
> final.results %>%
+   ggplot(aes(x = neighbors, y = mean, color = forcats::fct_reorder2(.metric, neighbors, mean))) +
+   geom_line(size = 1) +
+   geom_point(size = 2) +
+   theme_minimal() +
+   scale_color_wsj() +
+   scale_x_continuous(breaks = seq(1,30)) +
+   theme(panel.grid.minor.x = element_blank()) +
+   labs(color='Metric', y = "Estimate")
Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
i Please use `linewidth` instead.
This warning is displayed once every 8 hours.
Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
generated.
```



Problem 3: Logistic Regression

In this problem, you will create a classification model that can predict the presence of heart disease in individuals based on physical attributes and test results. By accurately predicting heart disease, the model can potentially help in reducing the need for more invasive diagnostic procedures.

```
> heart_data <- read_csv("https://raw.githubusercontent.com/deepbas/statdatasets/main/heart.csv")
```

i.

Load and prepare the dataset. Perform necessary preprocessing steps such as handling missing values, normalizing data, and encoding categorical variables. Split the dataset into training and testing sets to ensure the model can be accurately evaluated. The variable `target` is the outcome with 0 for no presence of heart disease and 1 for presence of heart disease.

```
> library(tidymodels)
> set.seed(123322324)
> heart_data <- heart_data %>%
+   janitor::clean_names() %>%
+   mutate(target =  as.factor(target))
>
>
> standardize <- function(x, na.rm = FALSE) {
+   (x - mean(x, na.rm = na.rm)) /
+     sd(x, na.rm = na.rm)
+   }
>
>
> heart_data <- heart_data %>%
+   mutate(across(where(is.character), as.factor)) %>%
+   mutate(across(where(is.numeric), standardize))
>
>
> data_split <- initial_split(heart_data, prop = 0.80)
> train_data <- training(data_split)
> test_data <- testing(data_split)
```

ii.

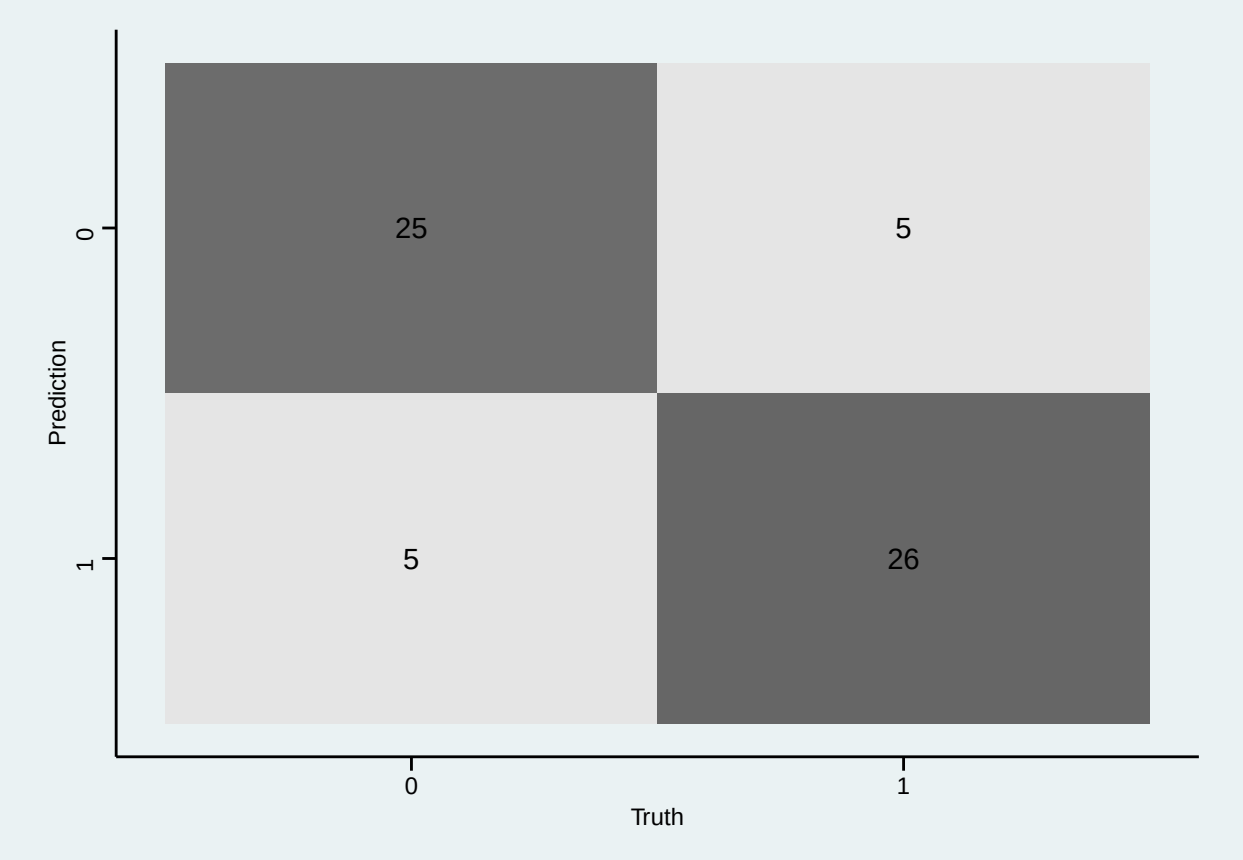
Fit a logistic regression model using the `tidymodels` framework in R. Use the `target` variable 'presence of heart disease' and include relevant predictors based on the dataset.

```
> recipe <- recipe(target ~ ., data = train_data) %>%
+   step_dummy(all_nominal(), -all_outcomes()) %>%
+    step_scale(all_numeric_predictors()) %>%
+   step_center(all_numeric_predictors())
>
> logit_spec <- logistic_reg() %>%
+   set_engine("glm") %>%
+   set_mode("classification")
>
> workflow <- workflow() %>%
+   add_recipe(recipe) %>%
+   add_model(logit_spec) %>%
+   fit(data = train_data)
```

iii.

Plot the confusion matrix. Evaluate your model's performance by calculating metrics such as accuracy, sensitivity, specificity, and the Area Under the Curve (AUC). These metrics will provide insights into how well the model can distinguish between patients with and without heart disease.

```
> predictions <- predict(workflow, test_data)
> results <- bind_cols(test_data %>% select(target), predictions)
>
> results %>%
+   conf_mat(target, .pred_class) %>% # confusion matrix
+   autoplot(type = "heatmap") # with graphics
```



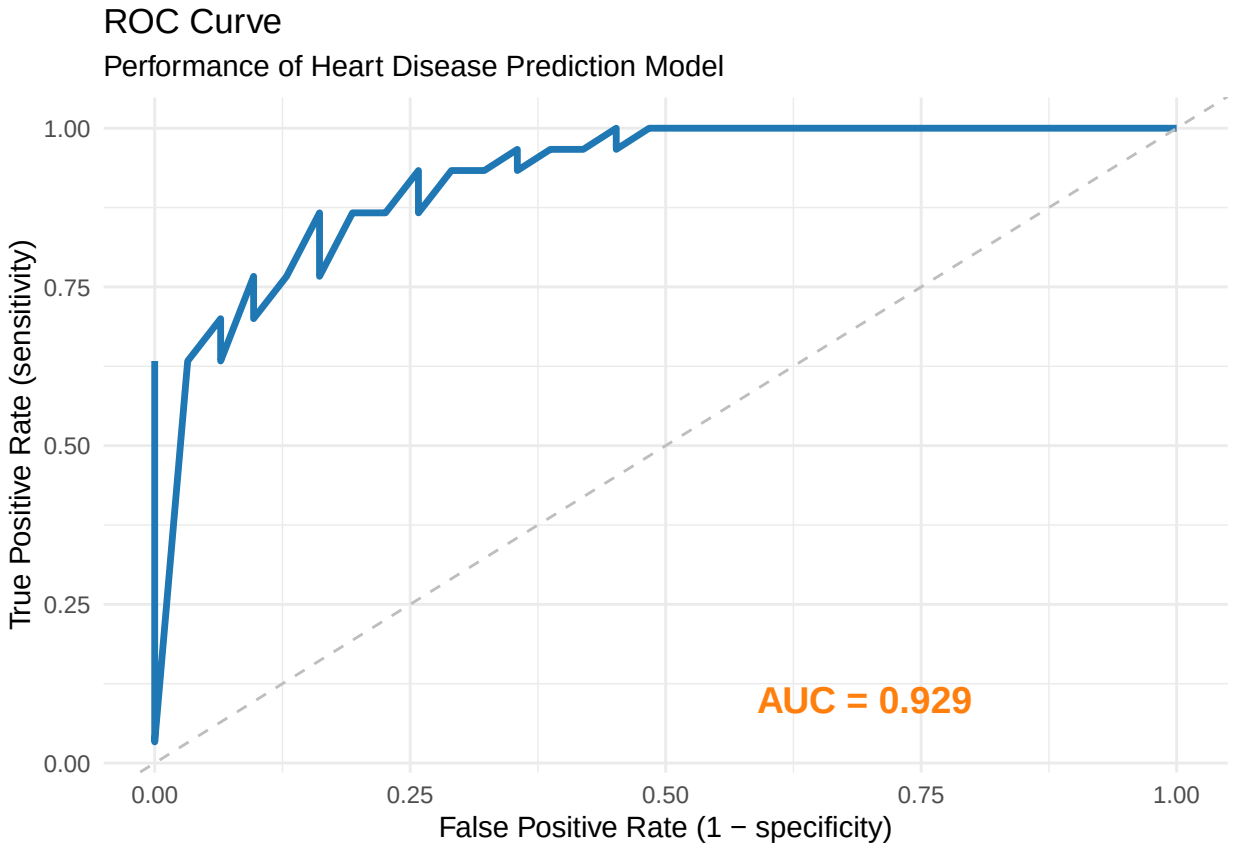

```
>
> library(yardstick)
> accuracy(results, truth = target,
+          estimate = .pred_class)
# A tibble: 1 x 3
  .metric  .estimator .estimate
  <chr>    <chr>          <dbl>
1 accuracy binary         0.836
> sens(results, truth = target,
+          estimate = .pred_class)
# A tibble: 1 x 3
  .metric .estimator .estimate
  <chr>   <chr>          <dbl>
1 sens    binary         0.833
> spec(results, truth = target,
```

```
+          estimate = .pred_class)
# A tibble: 1 x 3
  .metric .estimator .estimate
  <chr>   <chr>          <dbl>
1 spec    binary         0.839
>
> pred_prob <- predict(workflow, test_data, type = "prob")
>
> results2 <- test_data %>% select(target) %>% bind_cols(pred_prob)
>
> auc <- roc_auc(results2, truth = target, .pred_0)
> auc
# A tibble: 1 x 3
  .metric .estimator .estimate
  <chr>   <chr>          <dbl>
1 roc_auc binary         0.929
```

iv.

Explore the impact of varying classification thresholds on the prediction of heart disease. Plot the ROC curve, identifying the optimal threshold by evaluating sensitivity and specificity across a range of values. Discuss the clinical implications of selecting this threshold, particularly in terms of balancing false positives and false negatives.

```
> results2 %>%
+   roc_curve(truth = target, .pred_0) %>%
+   slice(-n())  %>%
+   ggplot(aes(x = 1 - specificity, y = sensitivity)) +
+   geom_line(color = "#1f77b4", size = 1.2) +
+   geom_abline(linetype = "dashed", color = "gray") +
+   annotate("text", x = 0.8, y = 0.1, label = paste("AUC =", round(roc_auc(results2, truth = target, .
+   labs(title = "ROC Curve", subtitle = "Performance of Heart Disease Prediction Model", x = "False Po
+   theme_minimal()
```



The ROC curve indicates a high Area Under the Curve (AUC) of 0.911, suggesting the model is effective in distinguishing between classes (presence and absence of heart disease). The curve approaches closer to the upper left corner, indicating a high true positive rate (sensitivity) with a relatively low false positive rate (specificity) over a wide range of thresholds. The optimal threshold is likely near where the curve first reaches the plateau, approximately between 0.1 and 0.2 on the false positive rate axis, as further increases in sensitivity occur with minimal gains in specificity beyond this point. Selecting a threshold in this region balances detection of heart disease (high sensitivity) while keeping false alarms (false positives) reasonably low, which is crucial in clinical settings to avoid unnecessary treatments.

```
> roc_data <- roc_curve(results2, truth = target, .pred_0)
>
> optimal_point <- roc_data %>%
+   mutate(distance = sqrt((1 - specificity)^2 + (sensitivity - 1)^2)) %>%
+   arrange(distance) %>%
+   slice(1)
> optimal_point
# A tibble: 1 x 4
  .threshold specificity sensitivity distance
       <dbl>       <dbl>       <dbl>    <dbl>
1      0.482       0.839       0.867    0.209
```