Biostat 212a Homework 5

Due Mar 16, 2024 @ 11:59PM

Wenqiang Ge UID:106371961

2025-03-11

Table of contents

ISL Exercise 9.7.1 (10pts)

1. This problem involves hyperplanes in two dimensions.

   (a) Sketch the hyperplane $1 + 3X_1 - X_2 = 0$. Indicate the set of points for which $1 + 3X_1 - X_2 > 0$, as well as the set of points for which $1 + 3X_1 - X_2 < 0$.

   (b) On the same plot, sketch the hyperplane $-2 + X_1 + 2X_2 = 0$. Indicate the set of points for which $-2 + X_1 + 2X_2 > 0$, as well as the set of points for which $-2 + X_1 + 2X_2 < 0$.

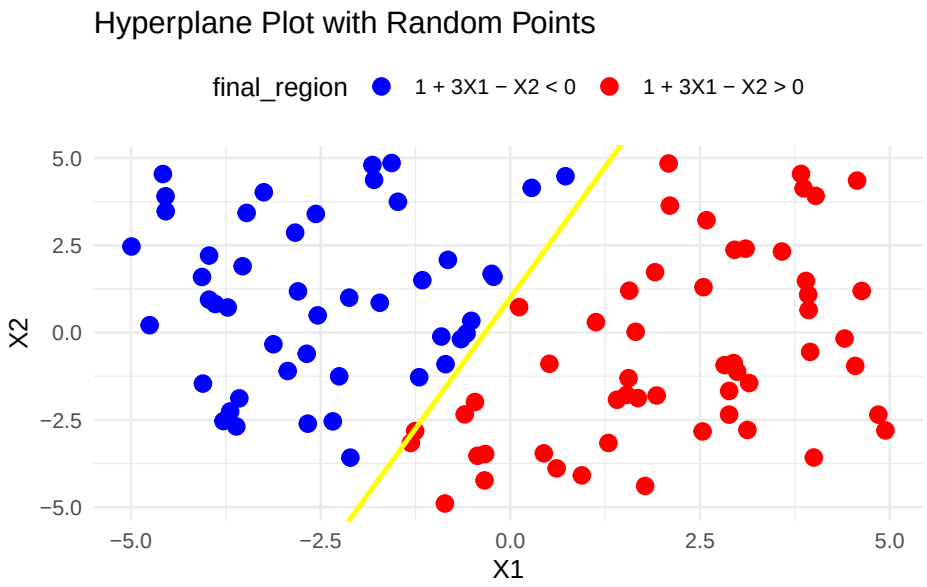
Solution:

(a)

```
# Load necessary libraries
library(ggplot2)
library(dplyr)

# Generate random points
set.seed(123)  # For reproducibility
points <- data.frame(
  X1 = runif(100, min = -5, max = 5),  # Random X1 values
```

```
  X2 = runif(100, min = -5, max = 5)   # Random X2 values
)

# Define hyperplane equation
hyperplane_1 <- function(X1) { 1 + 3*X1 }  # Equation for hyperplane 1

# Calculate the corresponding values of X2
points$hyperplane_1_result <- 1 + 3*points$X1 - points$X2  # Evaluate the hyperplane equa

# Assign category labels for each point based on hyperplane condition
points$region_1 <- ifelse(points$hyperplane_1_result > 0, "1 + 3X1 - X2 > 0", "1 + 3X1 -

# Combine the region labels to distinguish the points based on the regions
points$final_region <- paste(points$region_1)

# Create the plot
ggplot(points) +
  # Plot random points with color based on region
  geom_point(aes(x = X1, y = X2, color = final_region), size = 3) +
  # Plot the hyperplane line
  geom_abline(intercept = 1, slope = 3, color = "yellow", size = 1) +  # Hyperplane: 1 +
  labs(title = "Hyperplane Plot with Random Points", x = "X1", y = "X2") +  # Set plot ti
  theme_minimal() +  # Use minimal theme for cleaner plot
  theme(legend.position = "top") +  # Position the legend at the top
  scale_color_manual(values = c("blue", "red"))  # Set colors for the regions
```

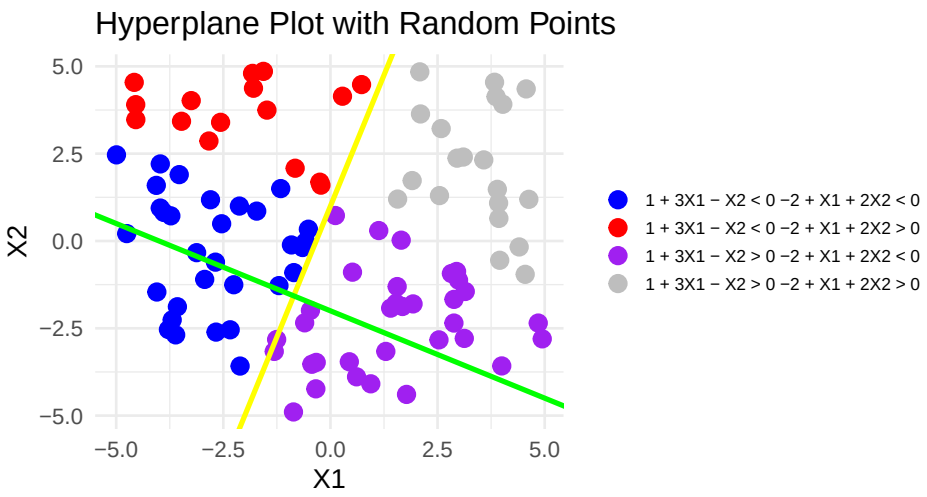
```
Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
i Please use `linewidth` instead.
```



(b)

```r
set.seed(123)  # For reproducibility
points <- data.frame(
  X1 = runif(100, min = -5, max = 5),  # Random X1 values
  X2 = runif(100, min = -5, max = 5)   # Random X2 values
)

# Define hyperplane equations
hyperplane_1 <- function(X1) { 1 + 3*X1 }
hyperplane_2 <- function(X1) { (-2 + X1) / 2 }

# Calculate corresponding X2 values
points$hyperplane_1_result <- 1 + 3*points$X1 - points$X2
points$hyperplane_2_result <- -2 + points$X1 + 2*points$X2

# Assign category labels for each point
points$region_1 <- ifelse(points$hyperplane_1_result > 0, "1 + 3X1 - X2 > 0", "1 + 3X1 -
points$region_2 <- ifelse(points$hyperplane_2_result > 0, "-2 + X1 + 2X2 > 0", "-2 + X1 +

# Combine both labels to distinguish different regions of the points
points$final_region <- paste(points$region_1, points$region_2)

# Plot
ggplot(points) +
```

```r
  geom_point(aes(x = X1, y = X2, color = final_region), size = 3) +  # Random points with
  geom_abline(intercept = 1, slope = 3, color = "yellow", size = 1, show.legend = FALSE)
  geom_abline(intercept = -2, slope = -1/2, color = "green", size = 1, show.legend = FALS
  labs(title = "Hyperplane Plot with Random Points", x = "X1", y = "X2") +
  scale_color_manual(values = c("blue", "red", "purple", "gray")) +  # Set 4 colors for t
  theme_minimal() +
  theme(
    legend.position = "right",
    legend.title = element_blank(),
    legend.text = element_text(size = 7),  # Smaller text for labels
    legend.key.size = unit(0.4, "cm"),  # Adjust legend key size for better spacing
    legend.direction = "vertical",  # Arrange labels vertically
    legend.box = "vertical",  # Stack the labels vertically
    plot.margin = margin(10, 10, 10, 10),  # Adjust margins to give more space
    aspect.ratio = 0.8  # Increase the size of the plot
  )
```



ISL Exercise 9.7.2 (10pts)

2. We have seen that in $p = 2$ dimensions, a linear decision boundary takes the form $\beta_0 + \beta_1 X_1 + \beta_2 X_2 = 0$. We now investigate a non-linear decision boundary.

   (a) Sketch the curve

   $$(1 + X_1)^2 + (2 - X_2)^2 = 4.$$

   (b) On your sketch, indicate the set of points for which

   $$(1 + X_1)^2 + (2 - X_2)^2 > 4,$$

   as well as the set of points for which

   $$(1 + X_1)^2 + (2 - X_2)^2 \leq 4.$$

   (c) Suppose that a classifier assigns an observation to the blue class if

   $$(1 + X_1)^2 + (2 - X_2)^2 > 4,$$

   and to the red class otherwise. To what class is the observation $(0, 0)$ classified? $(-1, 1)$? $(2, 2)$? $(3, 8)$?

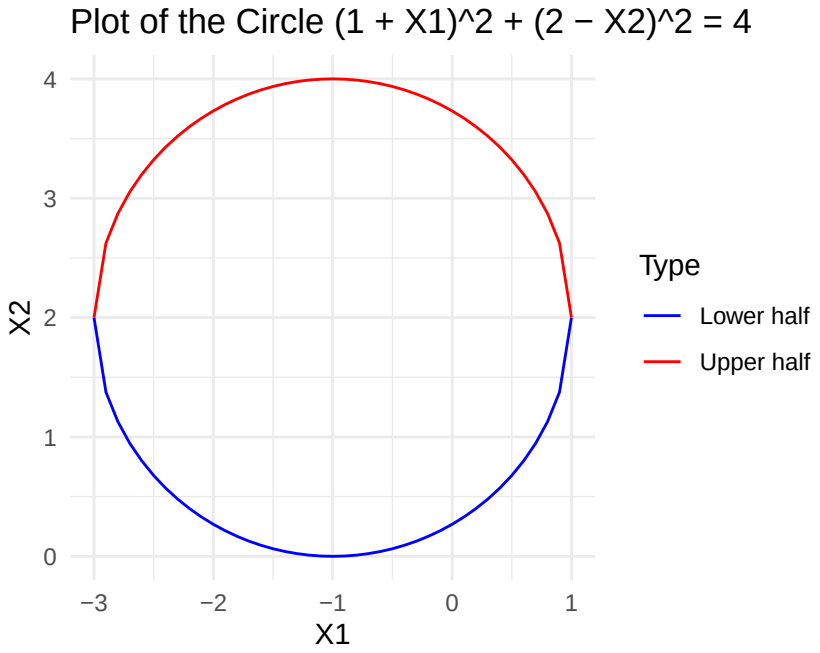
   (d) Argue that while the decision boundary in (c) is not linear in terms of X_1 and X_2, it is linear in terms of X_1, X_1^2, X_2, and X_2^2.

Solution:

(a)

```r
# Define the circle equation
circle_eq <- function(X1) {
  sqrt(4 - (X1 + 1)^2) + 2  # Solve for X2
}

# Create X1 values and calculate corresponding X2 values
X1_values <- seq(-3, 1, by = 0.1)
X2_values_positive <- sapply(X1_values, function(X1) circle_eq(X1))
X2_values_negative <- sapply(X1_values, function(X1) -circle_eq(X1) + 4)

# Create a data frame
circle_data <- data.frame(X1 = rep(X1_values, 2),
                          X2 = c(X2_values_positive, X2_values_negative),
```

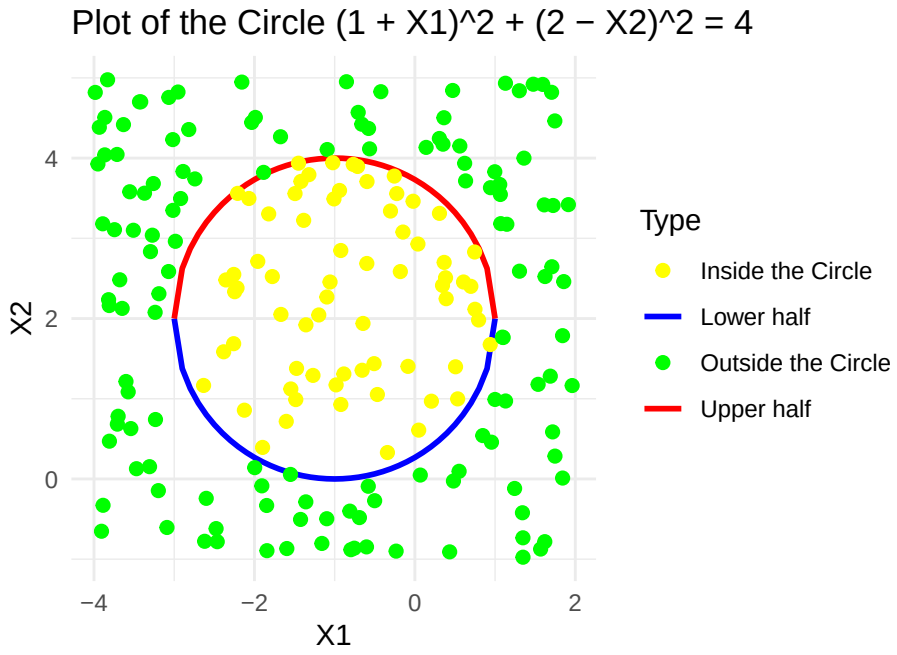
```
                          Type = rep(c("Upper half", "Lower half"), each = length(X1_valu

# Plot the circle
ggplot(circle_data, aes(x = X1, y = X2, color = Type)) +
  geom_line() +
  labs(title = "Plot of the Circle (1 + X1)^2 + (2 - X2)^2 = 4", x = "X1", y = "X2") +
  theme_minimal() +
  scale_color_manual(values = c("blue", "red")) +
  coord_fixed(ratio = 1)
```



(b)

```
# Define random points to classify as inside or outside the circle
set.seed(213)  # For reproducibility
points <- data.frame(
  X1 = runif(200, min = -4, max = 2),  # Random X1 values
  X2 = runif(200, min = -1, max = 5)   # Random X2 values
)

# Calculate the circle equation for each point
points$circle_equation <- (1 + points$X1)^2 + (2 - points$X2)^2
```

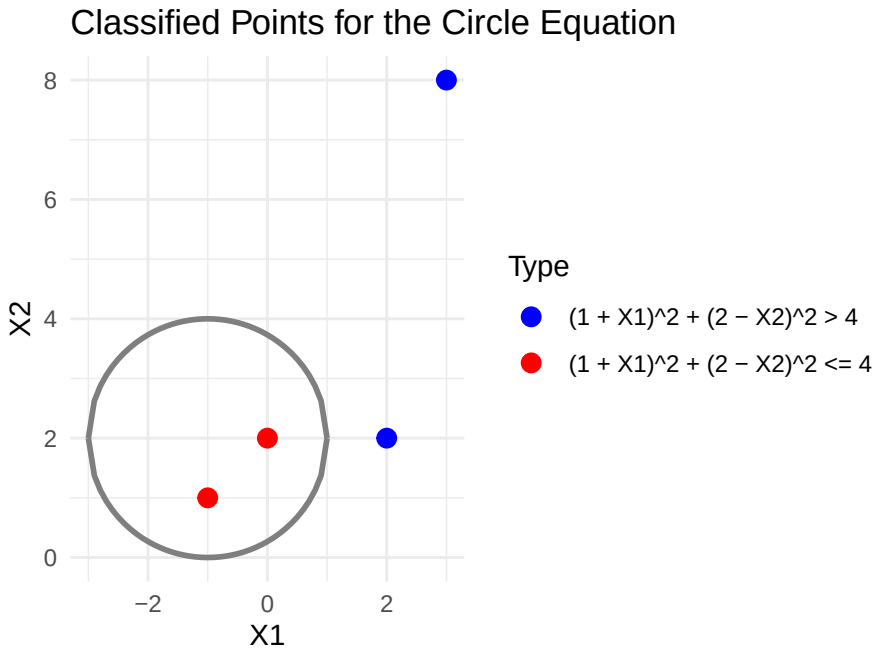
```r
# Assign categories based on whether points are inside or outside the circle
points$region <- ifelse(points$circle_equation <= 4, "Inside the Circle", "Outside the Ci

# Plot the circle and points with different colors for the regions
ggplot() +
  geom_line(data = circle_data, aes(x = X1, y = X2, color = Type), size = 1) +  # Circle
  geom_point(data = points, aes(x = X1, y = X2, color = region), size = 2) +  # Points in
  labs(title = "Plot of the Circle (1 + X1)^2 + (2 - X2)^2 = 4", x = "X1", y = "X2") +
  scale_color_manual(values = c("yellow", "blue", "green", "red")) +  # Added color for a
  theme_minimal() +
  coord_fixed(ratio = 1)
```



(c)

```r
# Define the circle equation
circle_eq <- function(X1) {
  sqrt(4 - (X1 + 1)^2) + 2  # Solve for X2
}

# Create X1 values and calculate corresponding X2 values
X1_values <- seq(-3, 1, by = 0.1)
X2_values_positive <- sapply(X1_values, function(X1) circle_eq(X1))
```

```r
X2_values_negative <- sapply(X1_values, function(X1) -circle_eq(X1) + 4)

# Create a data frame for the circle
circle_data <- data.frame(X1 = rep(X1_values, 2),
                          X2 = c(X2_values_positive, X2_values_negative),
                          Type = rep(c("Upper half", "Lower half"), each = length(X1_valu

# Points for classification
specific_points <- data.frame(
  X1 = c(0, -1, 2, 3),  # X1 values
  X2 = c(2, 1, 2, 8)    # X2 values
)

# Classify the points
specific_points$classification <- ifelse(
  (1 + specific_points$X1)^2 + (2 - specific_points$X2)^2 > 4, "(1 + X1)^2 + (2 - X2)^2 >
)

# Plot the circle and points with different colors for the regions
ggplot() +
  geom_line(data = circle_data, aes(x = X1, y = X2, color = Type), size = 1
            ) +  # Plot circle
  geom_point(data = specific_points, aes(x = X1, y = X2, color = classification), size =
  labs(title = "Classified Points for the Circle Equation", x = "X1", y = "X2") +
  scale_color_manual(values = c("(1 + X1)^2 + (2 - X2)^2 > 4" = "blue", "(1 + X1)^2 + (2
  theme_minimal() +
  coord_fixed(ratio =1)
```



(d) The decision boundary in part (c) is given by the equation:$(1+X_1)^2+(2-X_2)^2=4$. After expand above equation, we get:

$1+{X_1}^2+2X_1+4+{X_2}^2-4X_2=4$

${X_1}^2+{X_2}^2+2X_1-4X_2=0$

In this equation, we have these items $X_1, X_2, {X_1}^2, {X_2}^2$.

This shows that the decision boundary is quadratic when expressed in terms of the original features X_1 and X_2. However, if we introduce new features such as ${X_1}^2$ and ${X_2}^2$ (which are the squared versions of X_1 and X_2), the decision boundary becomes linear in terms of these new features because the equation will now only involve linear terms in X_1, X_2, ${X_1}^2$, and ${X_2}^2$.

Support vector machines (SVMs) on the `Carseats` data set (30pts)

Follow the machine learning workflow to train support vector classifier (same as SVM with linear kernel), SVM with polynomial kernel (tune the degree and regularization parameter C), and SVM with radial kernel (tune the scale parameter γ and regularization parameter C) for classifying `Sales<=8` versus `Sales>8`. Use the same seed as in your HW4 for the initial test/train split and compare the final test AUC and accuracy to those methods you tried in HW4.

Solution:

```
library(GGally)
library(gtsummary)
library(kernlab)
library(tidyverse)
library(tidymodels)
library(ISLR2)
library(caret)
library(doParallel)
library(vip)
library(doParallel)
```

```
# load the data
data("Carseats", package = "ISLR2")
Carseats$Sales<- ifelse(Carseats$Sales > 8, "High", "Low")
Carseats$Sales <- as.factor(Carseats$Sales)
```

Initial split into test and non-test sets

```
#Initial split into test and non-test sets
set.seed(212)
data_split <- initial_split(
  Carseats,
  prop = 0.75,
  strata = Sales
  )
data_split
```

```
<Training/Testing/Total>
<300/100/400>
```

```
Carseats_other <- training(data_split)
dim(Carseats_other)
```

```
[1] 300  11
```

```
Carseats_test <- testing(data_split)
dim(Carseats_test)
```

```
[1] 100  11
```

Recipe

```
svm_recipe <-
  recipe(
    Sales ~ .,
    data = Carseats_other
  ) %>%
  # create traditional dummy variables (necessary for svm)
  step_dummy(all_nominal_predictors()) %>%
  # zero-variance filter
  step_zv(all_numeric_predictors()) %>%
  # center and scale numeric data
  step_normalize(all_numeric_predictors()) # %>%
  # estimate the means and standard deviations
  # prep(training = Heart_other, retain = TRUE)
svm_recipe
```

SVM with linear kernel

Model & Workflow

```
svm_linear_mod <-
  svm_linear(
    mode = "classification",
    cost = tune()
) %>%
  set_engine("kernlab")
svm_linear_mod
```

```
Linear Support Vector Machine Model Specification (classification)

Main Arguments:
  cost = tune()

Computational engine: kernlab
```

```
svm_linear_wf <- workflow() %>%
  add_recipe(svm_recipe) %>%
  add_model(svm_linear_mod)
```

Tuning grid

```
param_grid <- grid_regular(
  cost(range = c(-3, 2)),
  #scale_factor(range = c(-1, 1)),
```

```
    levels = c(5)
    )
  param_grid
```

```
# A tibble: 5 x 1
   cost
  <dbl>
1 0.125
2 0.297
3 0.707
4 1.68
5 4
```

Cross-validation

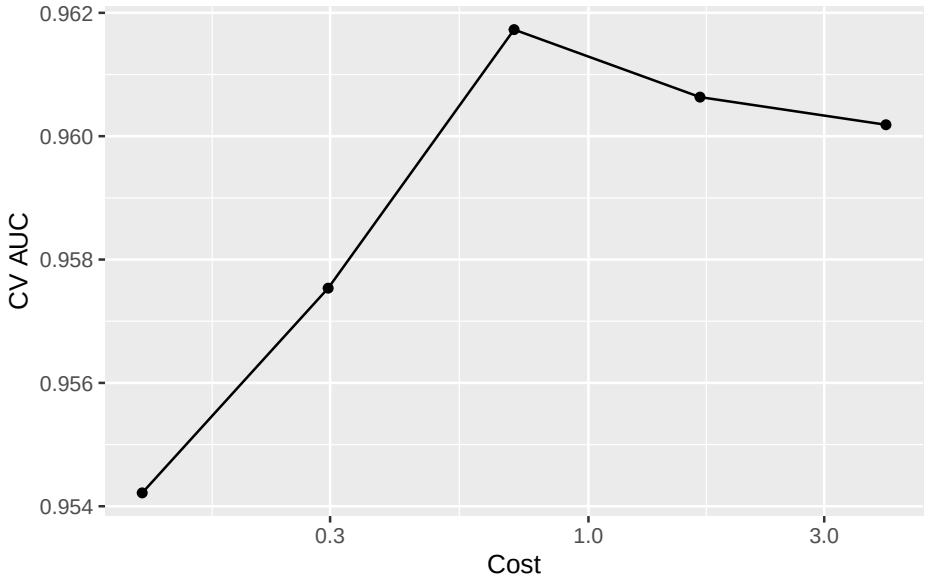
```
  set.seed(212)

  folds <- vfold_cv(Carseats_other, v = 5)
  folds
```

```
#  5-fold cross-validation
# A tibble: 5 x 2
  splits           id
  <list>           <chr>
1 <split [240/60]> Fold1
2 <split [240/60]> Fold2
3 <split [240/60]> Fold3
4 <split [240/60]> Fold4
5 <split [240/60]> Fold5
```

```
  # Fit cross-validation
  svm_linear_fit <-
    svm_linear_wf %>%
    tune_grid(
      resamples = folds,
      grid = param_grid,
      metrics = metric_set(roc_auc, accuracy)
    )
```

```
  #Visualize CV results
  svm_linear_fit %>%
    collect_metrics() %>%
    filter(.metric == "roc_auc") %>%
```

```
  ggplot(aes(x = cost, y = mean)) +
  geom_line() +
  geom_point() +
  labs(x = "Cost",
       y = "CV AUC") +
  scale_x_log10()
```




```
best_linear <- svm_linear_fit %>%
  select_best(metric = "roc_auc")
best_linear
```

```
# A tibble: 1 x 2
   cost .config
  <dbl> <chr>
1 0.707 Preprocessor1_Model3
```

Finalize the model

```
# Final workflow
final_linear_wf <-
  svm_linear_wf %>%
  finalize_workflow(
    best_linear
  )
```

```
# fit the whole training set, then predict test
final_linear_fit <-
  final_linear_wf %>%
  last_fit(data_split)

# test metrics
final_linear_fit %>%
  collect_metrics()
```

```
# A tibble: 3 x 4
  .metric     .estimator .estimate .config
  <chr>       <chr>          <dbl> <chr>
1 accuracy    binary         0.84  Preprocessor1_Model1
2 roc_auc     binary         0.944 Preprocessor1_Model1
3 brier_class binary         0.100 Preprocessor1_Model1
```

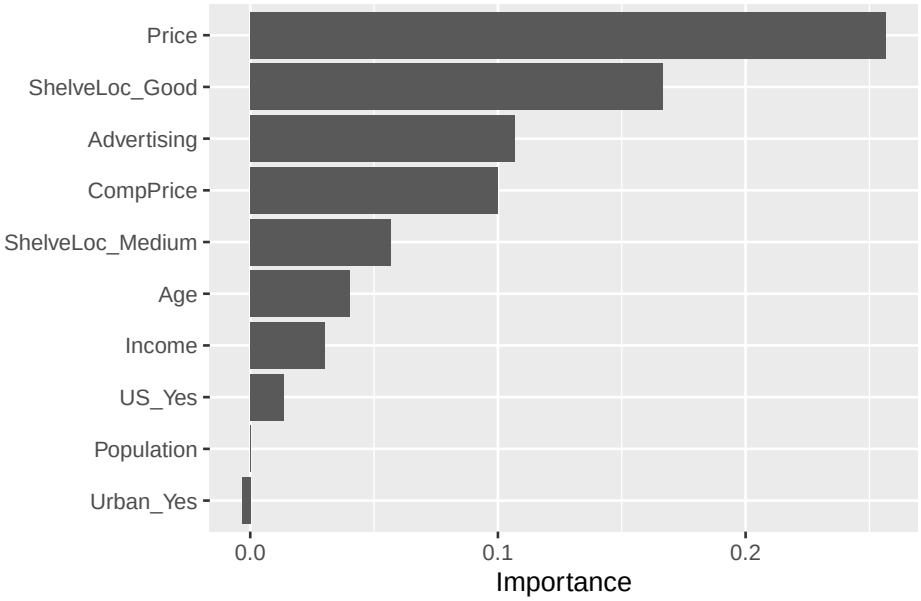
Visualize the final model

```
set.seed(212)
split_obj <- initial_split(data = Carseats, prop = 0.75, strata = Sales)
train <- training(split_obj)
test <- testing(split_obj)

# Create the recipe
recipe(Sales ~ ., data = train) %>%
  # create traditional dummy variables (necessary for svm)
  step_dummy(all_nominal_predictors()) %>%
  # zero-variance filter
  step_zv(all_numeric_predictors()) %>%
  # center and scale numeric data
  step_normalize(all_numeric_predictors()) %>%
  # estimate the means and standard deviations
  prep() -> recipe_obj

# Bake
train <- bake(recipe_obj, new_data=train)
test <- bake(recipe_obj, new_data=test)
```

```
final_linear_fit %>%
  pluck(".workflow", 1) %>%
  pull_workflow_fit() %>%
  # vip(method = "permute", train= Heart)
  vip(method = "permute",
```

```
        target = "Sales", metric = "accuracy",
        pred_wrapper = kernlab::predict, train = train)
```

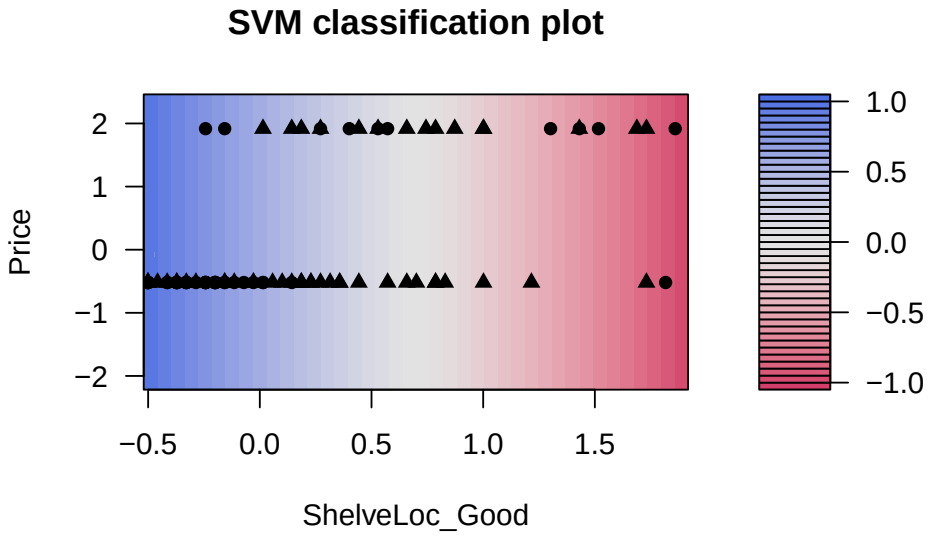
```
Warning: `pull_workflow_fit()` was deprecated in workflows 0.2.3.
i Please use `extract_fit_parsnip()` instead.
```




```
# Use svm_linear() for linear kernel SVM
svm_linear_spec <- svm_linear() %>%
  set_mode("classification") %>%
  set_engine("kernlab")

# Fit the model with linear kernel
svm_linear_fit <- svm_linear_spec %>%
  fit(Sales ~ ., data = train[, c('Price', 'ShelveLoc_Good', 'Sales')])
```

```
Setting default kernel parameters
```

```
# Visualize the decision boundary
svm_linear_fit %>%
  extract_fit_engine() %>%
  plot()
```



SVM with polynomial kernel

Model & Workflow

```
svm_mod <-
  svm_poly(
    mode = "classification",
    cost = tune(),
    degree = tune()
  ) |>
  set_engine("kernlab")
svm_mod
```

```
Polynomial Support Vector Machine Model Specification (classification)

Main Arguments:
  cost = tune()
  degree = tune()

Computational engine: kernlab
```

```
svm_wf <- workflow() %>%
  add_recipe(svm_recipe) %>%
  add_model(svm_mod)
svm_wf
```

```
== Workflow ====================================================================
Preprocessor: Recipe
```

```
Model: svm_poly()

-- Preprocessor ----------------------------------------------------------------
3 Recipe Steps

* step_dummy()
* step_zv()
* step_normalize()

-- Model ----------------------------------------------------------------------
Polynomial Support Vector Machine Model Specification (classification)

Main Arguments:
  cost = tune()
  degree = tune()

Computational engine: kernlab
```

Tuning grid

```
  param_grid <- grid_regular(
    cost(range = c(-3, 3)),
    degree(range = c(1, 5)),
    levels = c(5)
    )
  param_grid
```

```
# A tibble: 25 x 2
    cost degree
   <dbl>  <dbl>
 1 0.125      1
 2 0.354      1
 3 1          1
 4 2.83       1
 5 8          1
 6 0.125      2
 7 0.354      2
 8 1          2
 9 2.83       2
10 8          2
# i 15 more rows
```

Cross-validation

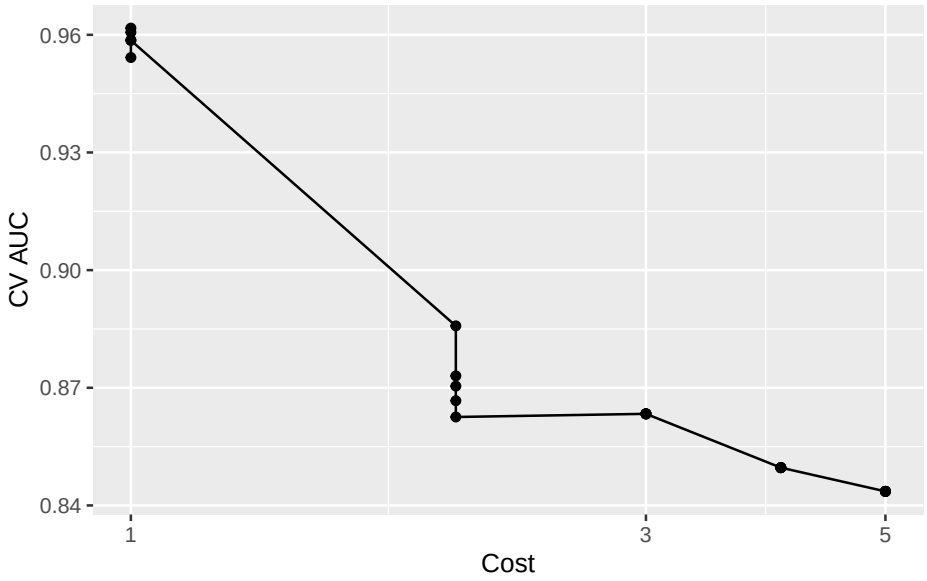
```r
set.seed(212)

folds <- vfold_cv(Carseats_other, v = 5)
folds
```

```
#  5-fold cross-validation
# A tibble: 5 x 2
  splits           id
  <list>           <chr>
1 <split [240/60]> Fold1
2 <split [240/60]> Fold2
3 <split [240/60]> Fold3
4 <split [240/60]> Fold4
5 <split [240/60]> Fold5
```

```r
svm_fit <- svm_wf %>%
  tune_grid(
    resamples = folds,
    grid = param_grid,
    metrics = metric_set(roc_auc, accuracy)
    )
svm_fit
```

```
# Tuning results
# 5-fold cross-validation
# A tibble: 5 x 4
  splits           id    .metrics          .notes
  <list>           <chr> <list>            <list>
1 <split [240/60]> Fold1 <tibble [50 x 6]> <tibble [0 x 3]>
2 <split [240/60]> Fold2 <tibble [50 x 6]> <tibble [0 x 3]>
3 <split [240/60]> Fold3 <tibble [50 x 6]> <tibble [0 x 3]>
4 <split [240/60]> Fold4 <tibble [50 x 6]> <tibble [0 x 3]>
5 <split [240/60]> Fold5 <tibble [50 x 6]> <tibble [0 x 3]>
```

```r
#Visualize CV results
svm_fit %>%
  collect_metrics() %>%
  filter(.metric == "roc_auc") %>%
  ggplot(mapping = aes(x = degree, y = mean)) +
  geom_point() +
  geom_line() +
  labs(x = "Cost", y = "CV AUC") +
  scale_x_log10()
```




```
svm_fit %>%
  show_best(metric = "roc_auc")
```

```
# A tibble: 5 x 8
   cost degree .metric .estimator  mean     n std_err .config
  <dbl>  <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
1 1          1 roc_auc binary     0.962     5  0.0124 Preprocessor1_Model03
2 2.83       1 roc_auc binary     0.961     5  0.0115 Preprocessor1_Model04
3 0.354      1 roc_auc binary     0.959     5  0.0127 Preprocessor1_Model02
4 8          1 roc_auc binary     0.959     5  0.0121 Preprocessor1_Model05
5 0.125      1 roc_auc binary     0.954     5  0.0116 Preprocessor1_Model01
```

```
best_svm <- svm_fit %>%
  select_best(metric = "roc_auc")
best_svm
```

```
# A tibble: 1 x 3
   cost degree .config
  <dbl>  <dbl> <chr>
1     1      1 Preprocessor1_Model03
```

Finalize the model

```r
final_wf <- svm_wf %>%
  finalize_workflow(best_svm)
final_wf
```

```
== Workflow ====================================================================
Preprocessor: Recipe
Model: svm_poly()

-- Preprocessor ----------------------------------------------------------------
3 Recipe Steps

* step_dummy()
* step_zv()
* step_normalize()

-- Model -----------------------------------------------------------------------
Polynomial Support Vector Machine Model Specification (classification)

Main Arguments:
  cost = 1
  degree = 1

Computational engine: kernlab
```

```r
final_fit <-
  final_wf %>%
  last_fit(data_split)
final_fit
```

```
# Resampling results
# Manual resampling
# A tibble: 1 x 6
  splits            id               .metrics .notes   .predictions .workflow
  <list>            <chr>            <list>   <list>   <list>       <list>
1 <split [300/100]> train/test split <tibble> <tibble> <tibble>     <workflow>
```

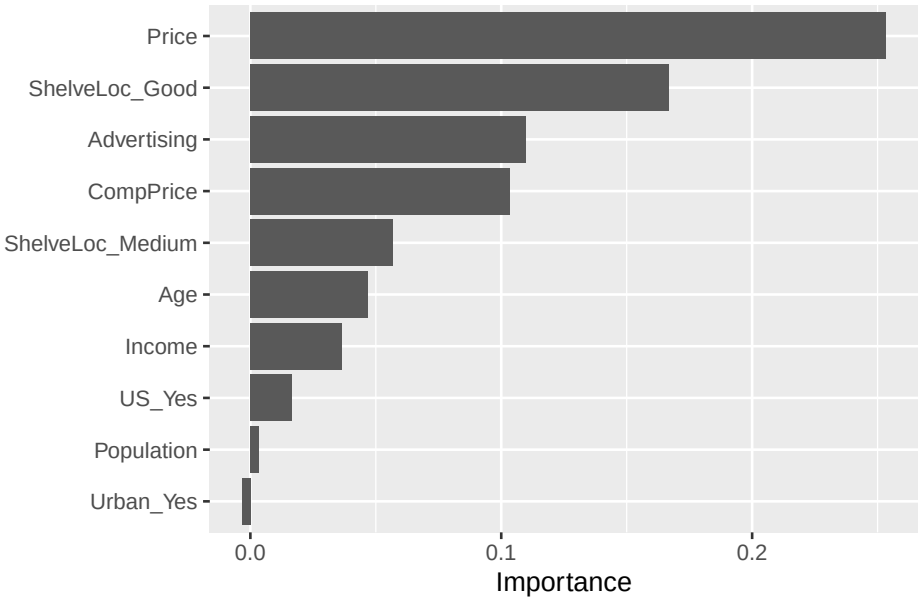
```r
final_fit %>%
  collect_metrics()
```

```
# A tibble: 3 x 4
  .metric     .estimator .estimate .config
  <chr>       <chr>          <dbl> <chr>
1 accuracy    binary        0.84   Preprocessor1_Model1
2 roc_auc     binary        0.944  Preprocessor1_Model1
3 brier_class binary        0.0987 Preprocessor1_Model1
```

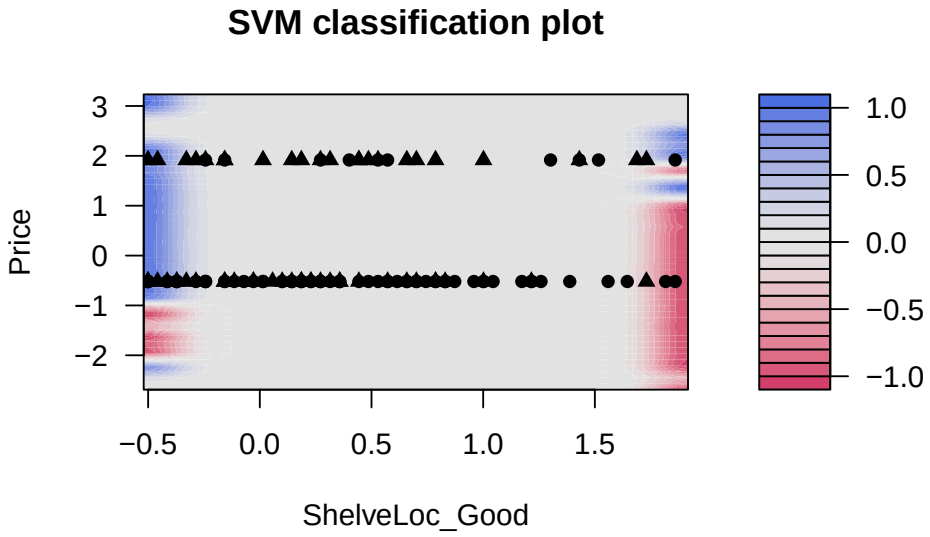
Visualize the final model

```
set.seed(212)
split_obj <- initial_split(data = Carseats, prop = 0.75, strata = Sales)
train <- training(split_obj)
test <- testing(split_obj)


# Create the recipe
recipe(Sales ~ ., data = train) %>%
  # create traditional dummy variables (necessary for svm)
  step_dummy(all_nominal_predictors()) %>%
  # zero-variance filter
  step_zv(all_numeric_predictors()) %>%
  # center and scale numeric data
  step_normalize(all_numeric_predictors()) %>%
  # estimate the means and standard deviations
  prep() -> recipe_obj

# Bake
train <- bake(recipe_obj, new_data=train)
test <- bake(recipe_obj, new_data=test)
```

```
final_fit %>%
  pluck(".workflow", 1) %>%
  pull_workflow_fit() %>%
  vip(method = "permute",
      target = "Sales", metric = "accuracy",
      pred_wrapper = kernlab::predict, train = train)
```




```
# Use svm_poly() for polynomial kernel SVM
svm_rbf_spec <- svm_rbf() %>%
  set_mode("classification") %>%
  set_engine("kernlab")

svm_rbf_fit <- svm_rbf_spec %>%
  fit(Sales ~ ., data = train[, c('Price', 'ShelveLoc_Good', 'Sales')])

svm_rbf_fit %>%
  extract_fit_engine() %>%
  plot()
```



SVM with radial kernel

Model & Workflow

```
svm_mod <-
  svm_rbf(
    mode = "classification",
    cost = tune(),
    rbf_sigma = tune()
  ) %>%
  set_engine("kernlab")
svm_mod
```

```
Radial Basis Function Support Vector Machine Model Specification (classification)

Main Arguments:
  cost = tune()
  rbf_sigma = tune()

Computational engine: kernlab
```

```
svm_wf <- workflow() %>%
  add_recipe(svm_recipe) %>%
  add_model(svm_mod)
svm_wf
```

```
== Workflow ====================================================================
Preprocessor: Recipe
```

```
Model: svm_rbf()

-- Preprocessor ----------------------------------------------------------------
3 Recipe Steps

* step_dummy()
* step_zv()
* step_normalize()

-- Model -----------------------------------------------------------------------
Radial Basis Function Support Vector Machine Model Specification (classification)

Main Arguments:
  cost = tune()
  rbf_sigma = tune()

Computational engine: kernlab
```

Tuning grid

```
  param_grid <- grid_regular(
    cost(range = c(-8, 5)),
    rbf_sigma(range = c(-5, -3)),
    levels = c(14, 5)
    )
  param_grid
```

```
# A tibble: 70 x 2
      cost rbf_sigma
     <dbl>     <dbl>
 1 0.00391   0.00001
 2 0.00781   0.00001
 3 0.0156    0.00001
 4 0.0312    0.00001
 5 0.0625    0.00001
 6 0.125     0.00001
 7 0.25      0.00001
 8 0.5       0.00001
 9 1         0.00001
10 2         0.00001
# i 60 more rows
```

Cross-validation

```r
set.seed(212)

folds <- vfold_cv(Carseats_other, v = 5)
folds
```

```
#  5-fold cross-validation
# A tibble: 5 x 2
  splits           id
  <list>           <chr>
1 <split [240/60]> Fold1
2 <split [240/60]> Fold2
3 <split [240/60]> Fold3
4 <split [240/60]> Fold4
5 <split [240/60]> Fold5
```

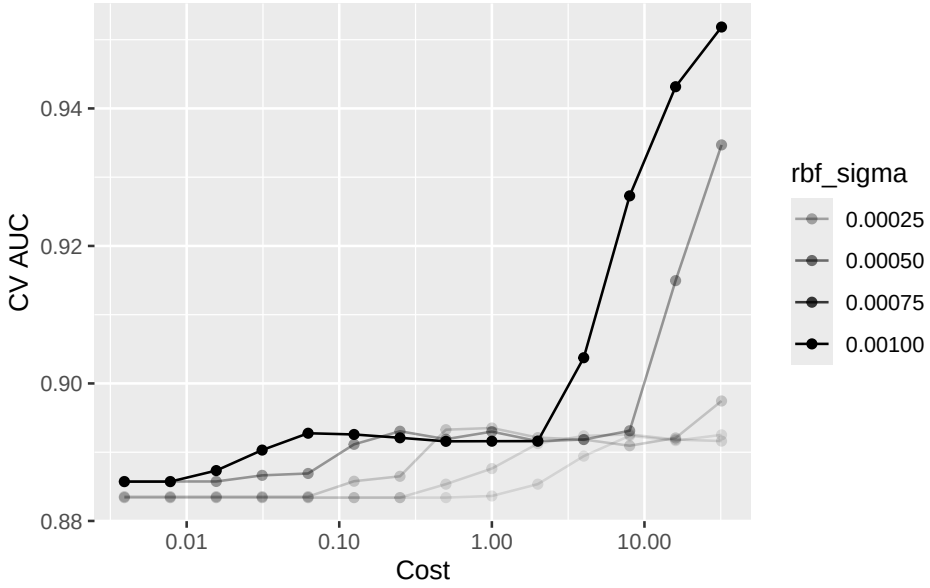
```r
svm_fit <- svm_wf %>%
  tune_grid(
    resamples = folds,
    grid = param_grid,
    metrics = metric_set(roc_auc, accuracy)
    )
svm_fit
```

```
# Tuning results
# 5-fold cross-validation
# A tibble: 5 x 4
  splits           id    .metrics           .notes
  <list>           <chr> <list>             <list>
1 <split [240/60]> Fold1 <tibble [140 x 6]> <tibble [0 x 3]>
2 <split [240/60]> Fold2 <tibble [140 x 6]> <tibble [0 x 3]>
3 <split [240/60]> Fold3 <tibble [140 x 6]> <tibble [0 x 3]>
4 <split [240/60]> Fold4 <tibble [140 x 6]> <tibble [0 x 3]>
5 <split [240/60]> Fold5 <tibble [140 x 6]> <tibble [0 x 3]>
```

```r
svm_fit %>%
  collect_metrics() %>%
  print(width = Inf) %>%
  filter(.metric == "roc_auc") %>%
  ggplot(mapping = aes(x = cost, y = mean, alpha = rbf_sigma)) +
  geom_point() +
  geom_line(aes(group = rbf_sigma)) +
  labs(x = "Cost", y = "CV AUC") +
  scale_x_log10()
```

```
# A tibble: 140 x 8
```

```
      cost rbf_sigma .metric  .estimator  mean     n std_err
     <dbl>     <dbl> <chr>    <chr>      <dbl> <int>   <dbl>
 1 0.00391   0.00001 accuracy binary     0.59      5  0.0319
 2 0.00391   0.00001 roc_auc  binary     0.883     5  0.0218
 3 0.00781   0.00001 accuracy binary     0.59      5  0.0319
 4 0.00781   0.00001 roc_auc  binary     0.883     5  0.0218
 5 0.0156    0.00001 accuracy binary     0.59      5  0.0319
 6 0.0156    0.00001 roc_auc  binary     0.883     5  0.0218
 7 0.0312    0.00001 accuracy binary     0.59      5  0.0319
 8 0.0312    0.00001 roc_auc  binary     0.883     5  0.0218
 9 0.0625    0.00001 accuracy binary     0.59      5  0.0319
10 0.0625    0.00001 roc_auc  binary     0.883     5  0.0218
   .config
   <chr>
 1 Preprocessor1_Model01
 2 Preprocessor1_Model01
 3 Preprocessor1_Model02
 4 Preprocessor1_Model02
 5 Preprocessor1_Model03
 6 Preprocessor1_Model03
 7 Preprocessor1_Model04
 8 Preprocessor1_Model04
 9 Preprocessor1_Model05
10 Preprocessor1_Model05
# i 130 more rows
```



```
  svm_fit %>%
    show_best(metric = "roc_auc")
```

```
# A tibble: 5 x 8
   cost rbf_sigma .metric .estimator  mean     n std_err .config
  <dbl>     <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
1    32  0.001    roc_auc binary     0.952     5  0.0110 Preprocessor1_Model70
2    16  0.001    roc_auc binary     0.943     5  0.0117 Preprocessor1_Model69
3    32  0.000316 roc_auc binary     0.935     5  0.0117 Preprocessor1_Model56
4     8  0.001    roc_auc binary     0.927     5  0.0155 Preprocessor1_Model68
5    16  0.000316 roc_auc binary     0.915     5  0.0162 Preprocessor1_Model55
```

```
  best_svm <- svm_fit %>%
    select_best(metric = "roc_auc")
  best_svm
```

```
# A tibble: 1 x 3
   cost rbf_sigma .config
  <dbl>     <dbl> <chr>
1    32     0.001 Preprocessor1_Model70
```

Finalize the model

```
  # Final workflow
  final_wf <- svm_wf %>%
    finalize_workflow(best_svm)
  final_wf
```

```
== Workflow ====================================================================
Preprocessor: Recipe
Model: svm_rbf()

-- Preprocessor ----------------------------------------------------------------
3 Recipe Steps

* step_dummy()
* step_zv()
* step_normalize()

-- Model -----------------------------------------------------------------------
Radial Basis Function Support Vector Machine Model Specification (classification)

Main Arguments:
  cost = 32
  rbf_sigma = 0.001
```

```
Computational engine: kernlab
```

```r
# Fit the whole training set, then predict the test cases
final_fit <-
  final_wf %>%
  last_fit(data_split)
final_fit
```

```
# Resampling results
# Manual resampling
# A tibble: 1 x 6
  splits            id               .metrics .notes   .predictions .workflow
  <list>            <chr>            <list>   <list>   <list>       <list>
1 <split [300/100]> train/test split <tibble> <tibble> <tibble>     <workflow>
```

```r
# Test metrics
final_fit %>%
  collect_metrics()
```

```
# A tibble: 3 x 4
  .metric     .estimator .estimate .config
  <chr>       <chr>          <dbl> <chr>
1 accuracy    binary        0.89   Preprocessor1_Model1
2 roc_auc     binary        0.959  Preprocessor1_Model1
3 brier_class binary        0.0773 Preprocessor1_Model1
```

```r
# Test metrics
final_fit %>%
  collect_metrics()
```

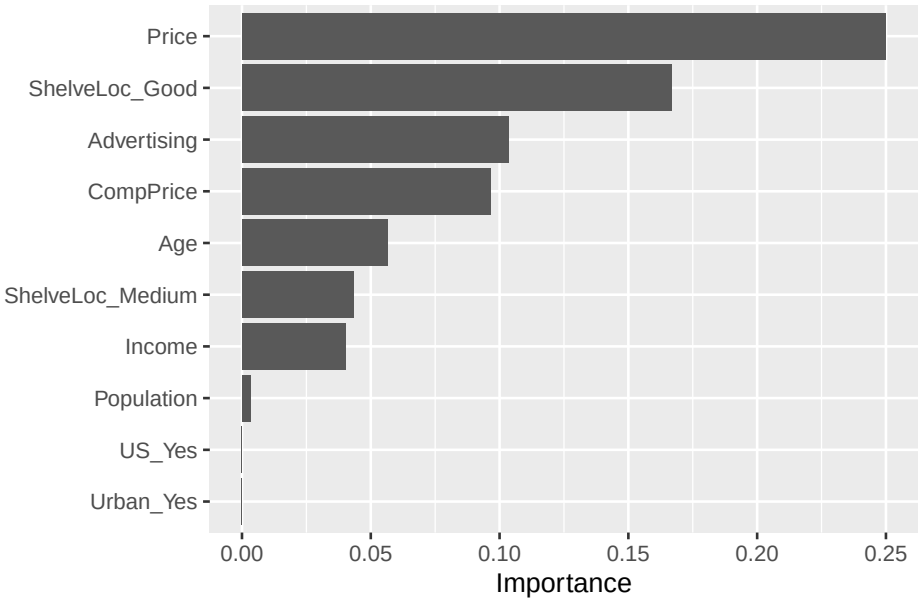
```
# A tibble: 3 x 4
  .metric     .estimator .estimate .config
  <chr>       <chr>          <dbl> <chr>
1 accuracy    binary        0.89   Preprocessor1_Model1
2 roc_auc     binary        0.959  Preprocessor1_Model1
3 brier_class binary        0.0773 Preprocessor1_Model1
```

Visualize the final model

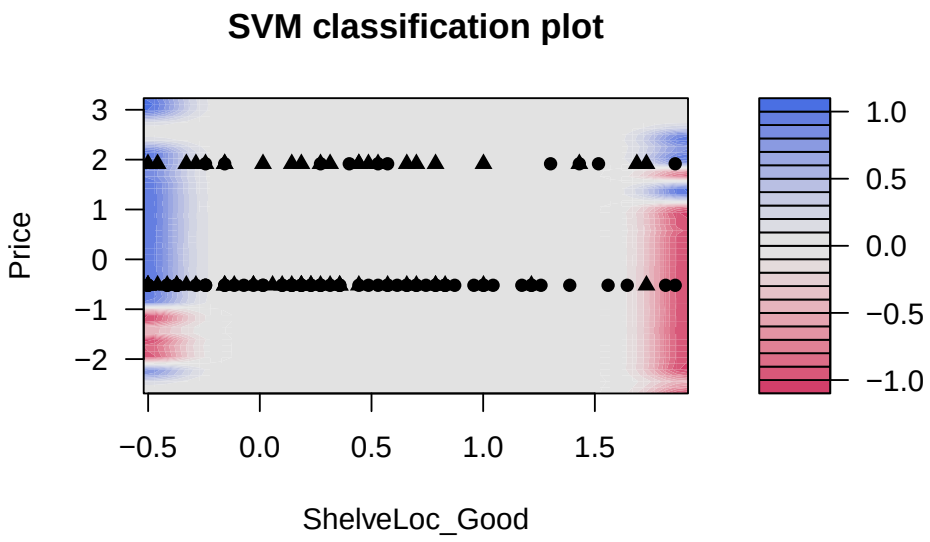
```r
set.seed(212)
split_obj <- initial_split(data = Carseats, prop = 0.75, strata = Sales)
train <- training(split_obj)
test <- testing(split_obj)
```

```
# Create the recipe
recipe(Sales ~ ., data = train) %>%
  # create traditional dummy variables (necessary for svm)
  step_dummy(all_nominal_predictors()) %>%
  # zero-variance filter
  step_zv(all_numeric_predictors()) %>%
  # center and scale numeric data
  step_normalize(all_numeric_predictors()) %>%
  # estimate the means and standard deviations
  prep() -> recipe_obj

# Bake
train <- bake(recipe_obj, new_data=train)
test <- bake(recipe_obj, new_data=test)
```

```
final_fit %>%
  pluck(".workflow", 1) %>%
  pull_workflow_fit() %>%
  vip(method = "permute",
      target = "Sales", metric = "accuracy",
      pred_wrapper = kernlab::predict, train = train)
```



```r
# Use svm_rbf() for RBF kernel SVM
svm_rbf_spec <- svm_rbf() %>%
  set_mode("classification") %>%
  set_engine("kernlab")

# Fit the model with RBF kernel
svm_rbf_fit <- svm_rbf_spec %>%
  fit(Sales ~ ., data = train[, c('Price', 'ShelveLoc_Good', 'Sales')])

# Visualize the decision boundary
svm_rbf_fit %>%
  extract_fit_engine() %>%
  plot()
```



Conlusion:

After comparing the accuracy and roc_auc of each model, the SVM with radial kernel has the highest value, so we choose it as the final model.

Bonus (10pts)

Let

$$f(X) = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p = \beta_0 + \beta^T X.$$

Then $f(X) = 0$ defines a hyperplane in $\mathbb{R}^p$. Show that $f(x)$ is proportional to the signed distance of a point x to the hyperplane $f(X) = 0$.