MATH 3190 Final Project - Quarto

Jun Hanvey, Ian McFarlane, Kellen Nankervis

2024-04-25

Table of contents

Hello World!

#TODO: Find good value for n and fix colors so they are the same as the later plot. - Kellen Nankervis Create example data to test RBF kernel function

```
set.seed(123)
n <- 1000
for (i in 1:10) {
  r <- runif(n, 1, 6 * pi + 1)
  theta <- runif(n, 0, 2 * pi)
}
# Make a sample classification n observations long
class <- sample(c(0, 1), n, replace = TRUE)
# Now fill the classification vector with the correct values
for (j in 1:n) {
  if ((r[j] + theta[j]) %% (2 * pi) < pi) {
    class[j] <- 1
  } else {
    class[j] <- 0
  }
}

# Create a data frame with the data
data <- data.frame(r, theta, class)
# Create a scatter plot of the data in x and y coordinates
data$color <- ifelse(data$class == 1, "red", "blue")
data$x <- data$r * cos(data$theta)
```

```
data$y <- data$r * sin(data$theta)

# Now offset the data a bit so their is some overlap
for (j in 1:n) {
  r <- runif(1, 0, 1)
  theta <- runif(1, 0, 2 * pi)
  # Convert to x and y coordinates
  data$x[j] <- data$x[j] + r * cos(theta)
  data$y[j] <- data$y[j] + r * sin(theta)
  # Convert the new x and y coordinates back to r and theta
  data$r[j] <- sqrt(data$x[j]^2 + data$y[j]^2)
  data$theta[j] <- atan2(data$y[j], data$x[j])
}
```

#TODO: Decide which plots to show and which to delete. Commented out ones would be my current suggestions to delete or at least move to later in the document. Make plots look good when knitted to pdf and/or slides. - Kellen Nankervis

```
# Plot the data in the x and y coordinates
plot(data$x, data$y, col = data$color, pch = 19, xlab = "x", ylab = "y")

# Plot the data in polar coordinates
# plot(data$r, data$theta, col = data$color, pch = 19, xlab = "r", ylab = "theta")

# Plot r*theta vs. r^2 * theta^2
# plot(data$r + data$theta, data$r * data$theta, col = data$color, pch = 19, xlab = "r*the
```

#TODO: Decide on good cost to not overfit. Could be a good spot to also show how we can use cross-validation to find the best cost. Make plots look good when knitted to pdf and/or slides. - Kellen Nankervis

```
# Load the required svm library
library(e1071)
library(caret)

# Convert class to a factor
data$class <- as.factor(data$class)

# Create a data frame with only the class and the x and y coordinates
data2 <- data.frame(class = data$class, x = data$x, y = data$y)
data2$class <- as.factor(data2$class)
```

```
# Use a radial basis function kernel to classify the data with the SVM function
svmfit <- svm(class ~ ., data = data2, kernel = "radial", cost = 1000, gamma = 1)

print(svmfit)

# Plot the data points
plot(data2$x, data2$y, col = data2$class, pch = 19, xlab = "x", ylab = "y")

# Plot the decision boundary
# plot(svmfit, data2$class, grid = 100, dataSymbol = 16)
x1_grid <- seq(min(data2$x), max(data2$x), length.out = 100)
x2_grid <- seq(min(data2$y), max(data2$y), length.out = 100)
grid <- expand.grid(x = x1_grid, y = x2_grid)

predicted_labels <- predict(svmfit, newdata = grid)

plot(data2$x, data2$y, col = data2$class, pch = 19, xlab = "x", ylab = "y")
points(grid$x, grid$y, col = factor(predicted_labels), pch = ".", cex = 3.5)
legend("topright", legend = levels(data2$class), col = c("blue", "red"), pch = 19)

# Create a confusion matrix to evaluate the SVM model
confusionMatrix(predict(svmfit, data2), data2$class)
```

#This was some example code I found to make the decision boundary plot. I'm leaving it for others to see but it will be removed in the final version obviously. - Kellen Nankervis

```
print("")

# Create a toy dataset
set.seed(123)
data <- data.frame(
  x1 = rnorm(50, mean = 2),
  x2 = rnorm(50, mean = 2),
  label = c(rep("Red", 25), rep("Blue", 25)) |> as.factor()
)

# Train an SVM
svm_model <- svm(label ~ ., data = data, kernel = "radial")

# Create a grid of points for prediction
x1_grid <- seq(min(data$x1), max(data$x1), length.out = 100)
```

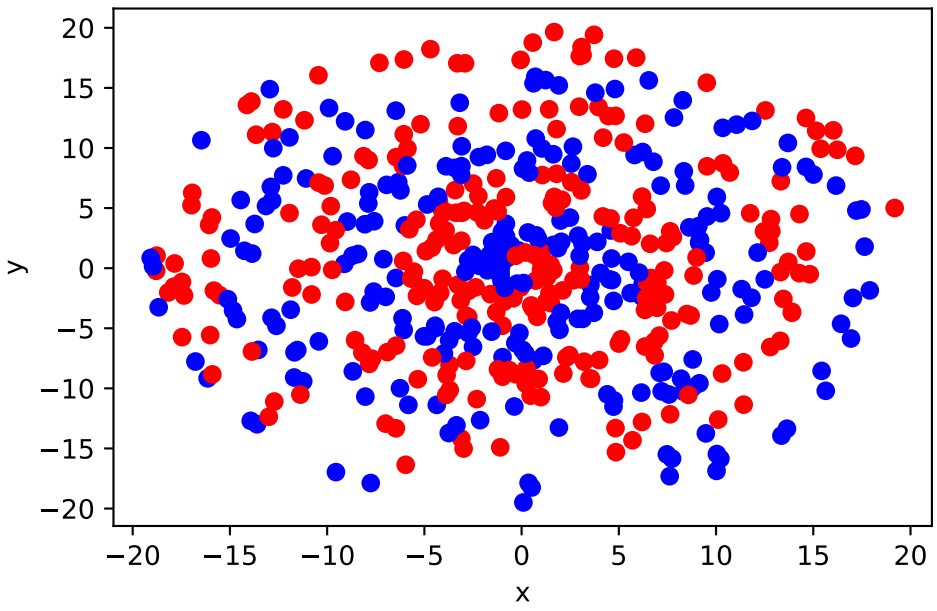
```r
x2_grid <- seq(min(data$x2), max(data$x2), length.out = 100)
grid <- expand.grid(x1 = x1_grid, x2 = x2_grid)

# Predict class labels for the grid
predicted_labels <- predict(svm_model, newdata = grid)

# Plot the decision boundary
plot(data$x1, data$x2, col = factor(data$label), pch = 19, main = "SVM Decision Boundary")
points(grid$x1, grid$x2, col = factor(predicted_labels), pch = ".", cex = 2.5)
legend("topright", legend = levels(data$label), col = c("blue", "red"), pch = 19)
```

1 Create a Python script to generate the data and plot it.

```python
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

np.random.seed(123)
n = 500  # Assuming n is defined

# Collect data in lists
r_values = []
theta_values = []
class_values = []

for i in range(n):
    r = np.random.uniform(1, 6 * np.pi + 1)
    theta = np.random.uniform(0, 2 * np.pi)
    if (r + theta) % (2 * np.pi) < np.pi:
        class_ = 1
    else:
        class_ = 0
    r_values.append(r)
    theta_values.append(theta)
    class_values.append(class_)

# Create DataFrame from lists
data = pd.DataFrame({"r": r_values, "theta": theta_values, "class": class_values})
# Add colors to the data
```

```
data["color"] = np.where(data["class"] == 1, "red", "blue")
data["x"] = data["r"] * np.cos(data["theta"])
data["y"] = data["r"] * np.sin(data["theta"])

# Offset the data a bit so there is some overlap
for i in range(n):
    r = np.random.uniform(0, 0)
    theta = np.random.uniform(0, 2 * np.pi)
    data.at[i, "x"] += r * np.cos(theta)
    data.at[i, "y"] += r * np.sin(theta)
    data.at[i, "r"] = np.sqrt(data.at[i, "x"] ** 2 + data.at[i, "y"] ** 2)
    data.at[i, "theta"] = np.arctan2(data.at[i, "y"], data.at[i, "x"])
```

```
# Plot the data in the x and y coordinates
plt.scatter(data["x"], data["y"], c=data["color"])
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```




```
# Now it's time to use scikit learn to classify the data with the SVM function
from sklearn.svm import SVC
```

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix

# Create a data frame with only the class and the x and y coordinates
data2 = data[["class", "x", "y"]]
data2["class"] = data2["class"].astype("category")

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(data2[["x", "y"]], data2["class"], tes

# Use a radial basis function kernel to classify the data with the SVM function
svm = SVC(kernel="rbf", C=1000, gamma=1)
svm.fit(X_train, y_train)
```

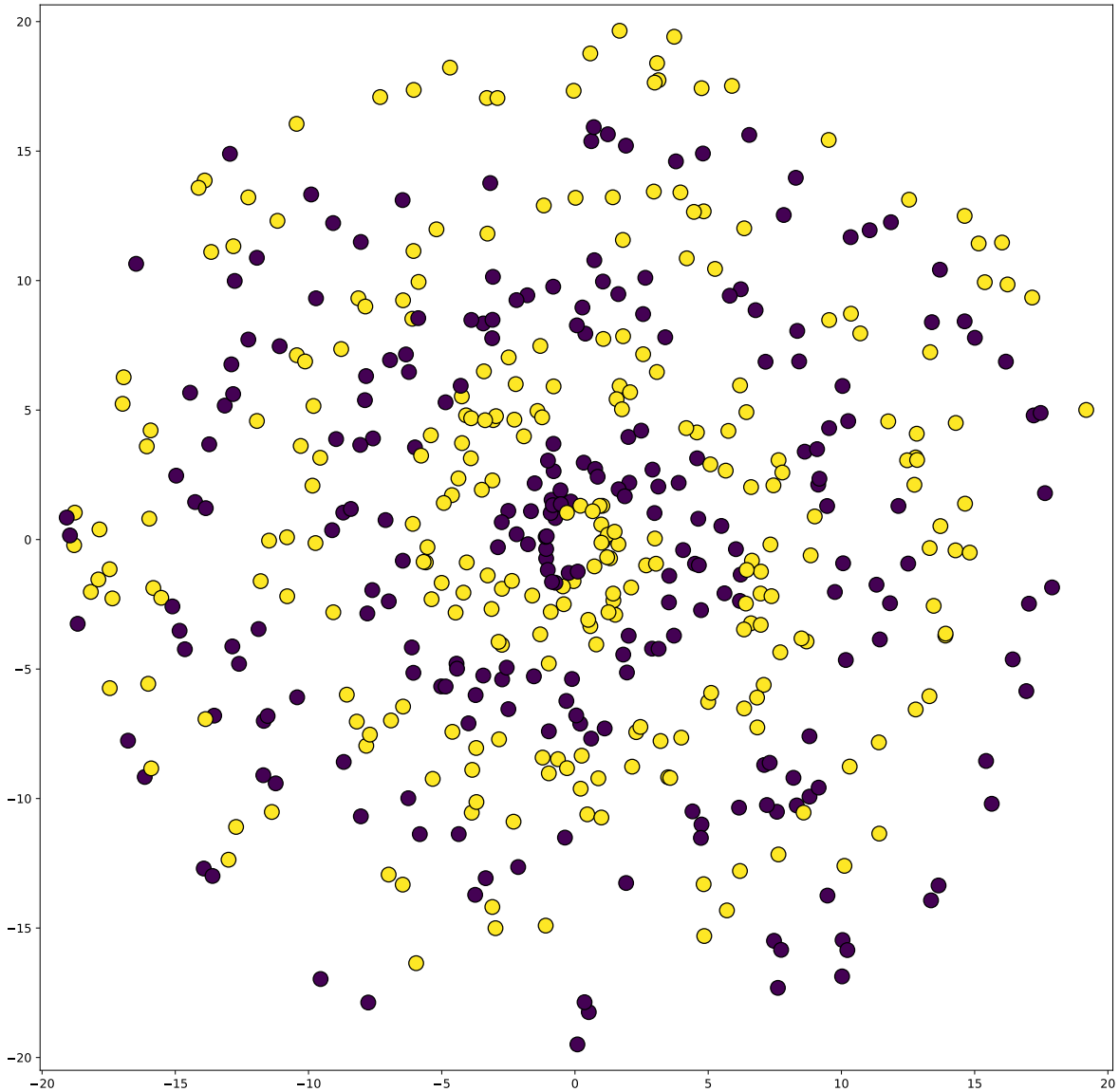
```
C:\Users\grave\AppData\Local\Temp\ipykernel_27552\1462750881.py:8: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guid
  data2["class"] = data2["class"].astype("category")
```

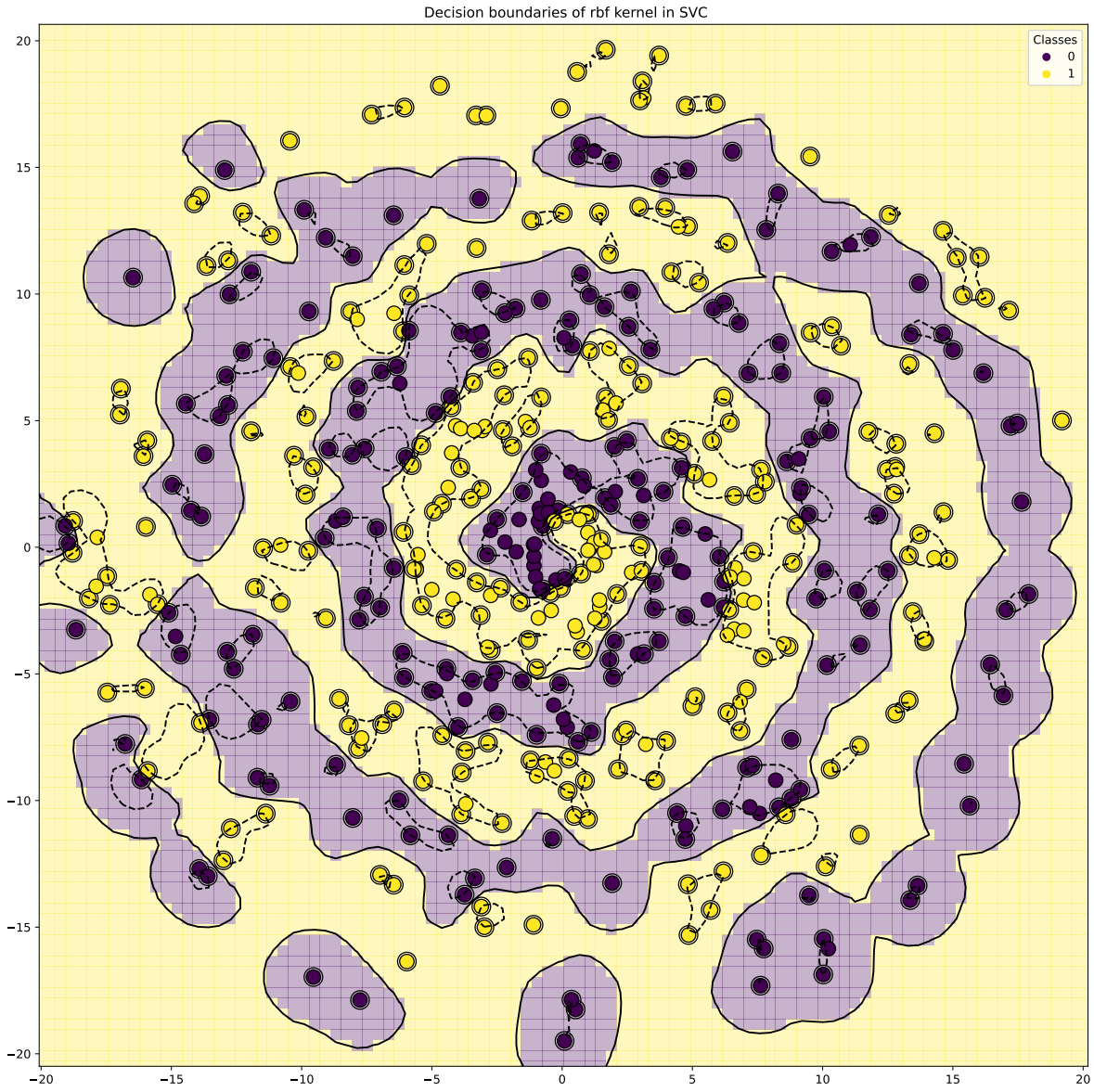
```
SVC(C=1000, gamma=1)
```

```
from sklearn.inspection import DecisionBoundaryDisplay
from sklearn import svm


def plot_training_data_with_decision_boundary(kernel, X, y, gamma=2, C=1, edgeincrease=1,
    # Train the SVC
    clf = svm.SVC(kernel="rbf", gamma=gamma, C=C).fit(X, y)

    # Settings for plotting
    _, ax = plt.subplots(figsize=figsize)
    x_min, x_max, y_min, y_max = min(X[:, 0])-edgeincrease, max(X[:, 0])+edgeincrease, min
    ax.set(xlim=(x_min, x_max), ylim=(y_min, y_max))

    # Plot decision boundary and margins
    common_params = {"estimator": clf, "X": X, "ax": ax}
    DecisionBoundaryDisplay.from_estimator(
        **common_params,
```

```
        response_method="predict",
        plot_method="pcolormesh",
        alpha=0.3,
    )
    DecisionBoundaryDisplay.from_estimator(
        **common_params,
        response_method="decision_function",
        plot_method="contour",
        levels=[-1, 0, 1],
        colors=["k", "k", "k"],
        linestyles=["--", "-", "--"],
    )

    # Plot bigger circles around samples that serve as support vectors
    ax.scatter(
        clf.support_vectors_[:, 0],
        clf.support_vectors_[:, 1],
        s=250,
        facecolors="none",
        edgecolors="k",
    )
    # Plot samples by color and add legend
    ax.scatter(X[:, 0], X[:, 1], c=y, s=150, edgecolors="k")
    ax.legend(*scatter.legend_elements(), loc="upper right", title="Classes")
    ax.set_title(f" Decision boundaries of {kernel} kernel in SVC")

    _ = plt.show()
```

```
# Turn the X and Y data into a numpy array
X = np.array(data2[["x", "y"]])
y = np.array(data2["class"])

# Plotting settings
fig, ax = plt.subplots(figsize=(16, 16))
scatter = ax.scatter(X[:, 0], X[:, 1], c=y, s=150, edgecolors="k")
x_min, x_max, y_min, y_max = min(X[:, 0])-1, max(X[:, 0])+1, min(X[:, 1])-1, max(X[:, 1])+
ax.set(xlim=(x_min, x_max), ylim=(y_min, y_max))

# Plot decision boundary and margins
plot_training_data_with_decision_boundary("rbf", X, y, gamma=1, C=1000, edgeincrease=1, fi
```


20
15
10
5
0
−5
−10
−15
−20
−20
−15
−10
−5
0
5
10
15
20


Decision boundaries of rbf kernel in SVC
Classes
0
1
20
15
10
5
0
−5
−10
−15
−20
−20
−15
−10
−5
0
5
10
15
20