# Purchase Prediction Using SVM

2023-09-28

# PROJECT OVERVIEW

In this project, I set out to predict the variable “Purchase” using the OJ dataset from the ISLR package. My goal was to explore the effectiveness of different types of Support Vector Machines (SVMs) by evaluating their performance across various kernel functions: linear, radial, and polynomial. This analysis helped identify the optimal cost parameters for each model to achieve the best prediction accuracy.

First I will divide the data into test and training. Now, I will fit a support vector classifier with varying cost parameters over the range [0.01,10] and plot the training and test error across this spectrum of cost parameters and determine the optimal cost.

```
library(ISLR2)
```

```
## Warning: package 'ISLR2' was built under R version 4.3.2
```

```
library(ROCR)
```

```
## Warning: package 'ROCR' was built under R version 4.3.2
```

```
library(e1071)
```

```
## Warning: package 'e1071' was built under R version 4.3.2
```

```
library(tidyverse)
```

```
## Warning: package 'tidyverse' was built under R version 4.3.2
```

```
## Warning: package 'ggplot2' was built under R version 4.3.3
```

```
## Warning: package 'tidyr' was built under R version 4.3.3
```

```
## Warning: package 'readr' was built under R version 4.3.2
```

```
## Warning: package 'dplyr' was built under R version 4.3.3
```

```
## Warning: package 'stringr' was built under R version 4.3.3
```

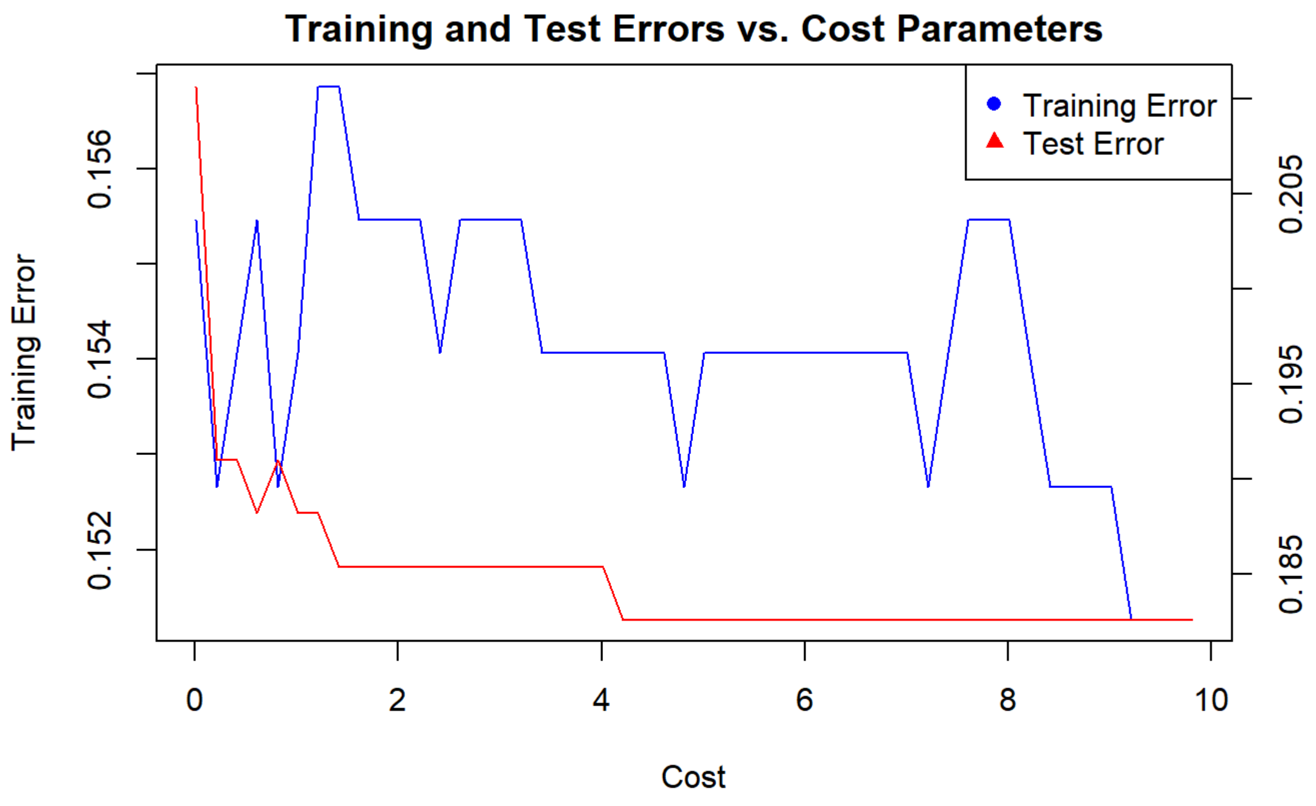
```
## Warning: package 'lubridate' was built under R version 4.3.3
## ── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──
## ✓ dplyr     1.1.4     ✓ readr     2.1.5
## ✓ forcats   1.0.0     ✓ stringr   1.5.1
## ✓ ggplot2   3.5.1     ✓ tibble    3.2.1
## ✓ lubridate 1.9.3     ✓ tidyr     1.3.1
## ✓ purrr     1.0.2
## ── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
## ✕ dplyr::filter() masks stats::filter()
## ✕ dplyr::lag()    masks stats::lag()
## ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
library(caret)
```

```
## Warning: package 'caret' was built under R version 4.3.2
```

```
## Loading required package: lattice
##
## Attaching package: 'caret'
##
## The following object is masked from 'package:purrr':
##
##     lift
```

```r
data("OJ")
set.seed(1234)

test_indis <- sample(1:nrow(OJ), 1/3*nrow(OJ), replace = FALSE)
test <- OJ[test_indis, ]
training <- OJ[-test_indis, ]

costs <- seq(0.01, 10, by = 0.2)

trainerrors <- rep(NA, length(costs))
testerrors <- rep(NA, length(costs))

for (i in 1:length(costs)) {
  fit <- svm(Purchase ~ ., data = training, kernel = "linear", cost = costs[i])

  predtrain <- predict(fit, training)
  predtest <- predict(fit, test)

  trainerrors[i] <- mean(predtrain != training$Purchase)
  testerrors[i] <- mean(predtest != test$Purchase)
}

par(mar = c(8, 5, 2, 2))

plot(costs, trainerrors, type = "l", col = "blue", pch = 16, xlab = "Cost", ylab = "Training Error",
     main = "Training and Test Errors vs. Cost Parameters")

par(new = TRUE)
plot(costs, testerrors, type = "l", col = "red", pch = 17, xlab = "", ylab = "", axes = FALSE)
axis(side = 4, at = pretty(range(testerrors)))

legend("topright", legend = c("Training Error", "Test Error"), col = c("blue", "red"), pch = c(16, 17))
```



```
tune_grid <- expand.grid(cost = costs)
svm_tune <- tune(svm, Purchase ~ ., data = training, ranges = costs, kernel = "linear")


optimal_cost <- svm_tune$best.model
optimal_cost
```

```
##
## Call:
## best.tune(METHOD = svm, train.x = Purchase ~ ., data = training,
##     ranges = costs, kernel = "linear")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  1
##
## Number of Support Vectors:  288
```

```
y_hat <- predict(optimal_cost, newdata = test)
y_true <- test$Purchase
accuracy_linear<- length(which(y_hat == y_true))/length(y_true)
accuracy_linear
```

```
## [1] 0.8117978
```

```
confusion_matrixlinear <- confusionMatrix(y_hat, y_true)
confusion_matrixlinear
```

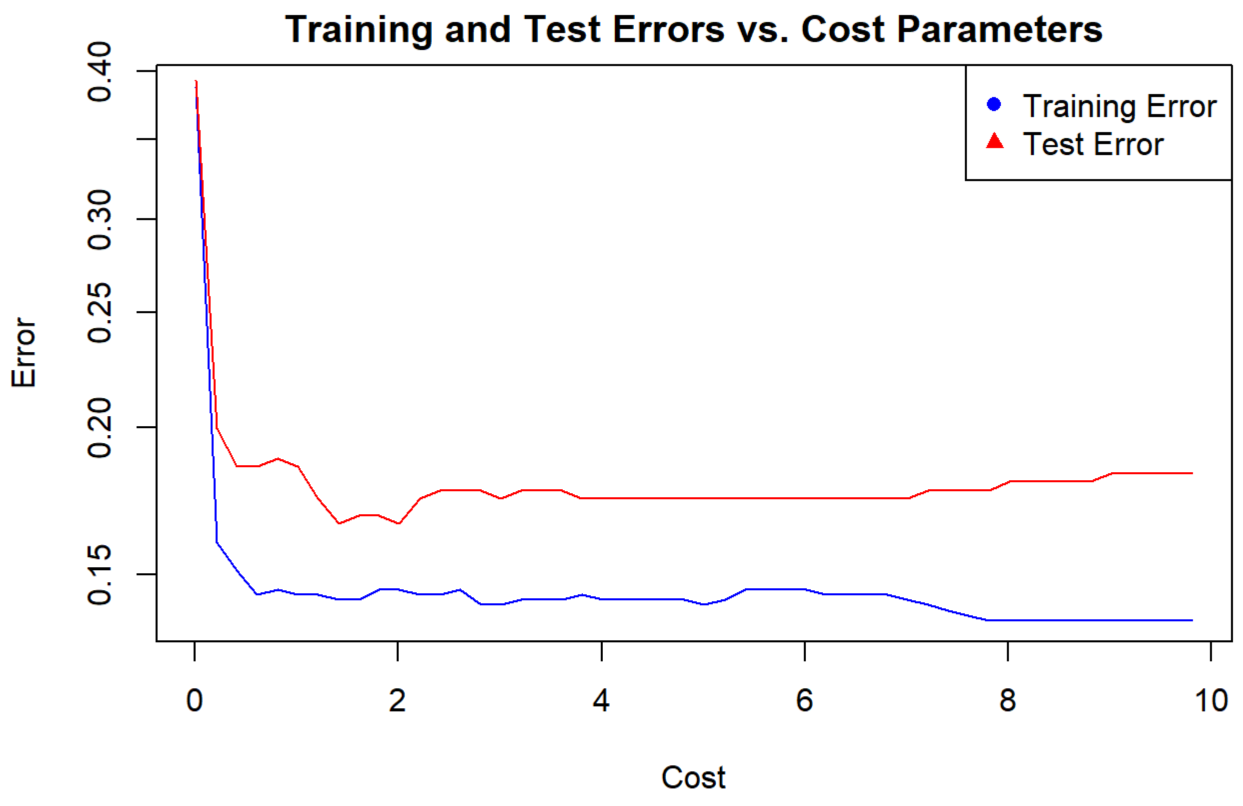
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  CH  MM
##         CH 190  41
##         MM  26  99
##
##                Accuracy : 0.8118
##                  95% CI : (0.7673, 0.8511)
##     No Information Rate : 0.6067
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.598
##
##  Mcnemar's Test P-Value : 0.0872
##
##             Sensitivity : 0.8796
##             Specificity : 0.7071
##          Pos Pred Value : 0.8225
##          Neg Pred Value : 0.7920
##              Prevalence : 0.6067
##          Detection Rate : 0.5337
##    Detection Prevalence : 0.6489
##       Balanced Accuracy : 0.7934
##
##        'Positive' Class : CH
##
```

#We see that the optimal cost is 1

#Now, I will repeat the above process in (A) for a support vector machine with a radial kernel. (Using the default parameter for gamma). Then we will repeat the exercise again for a support vector machine with a polynomial kernel of degree=2. Finally, we will reflect on the performance of the SVM with different kernels, and the support vector classifier, i.e., SVM with a linear kernel.

#radial kernel

```r
trainerrors <- rep(NA, length(costs))
testerrors <- rep(NA, length(costs))

set.seed(12345)
for (i in 1:length(costs)) {
  fit <- svm(Purchase ~ ., data = training, kernel = "radial", cost = costs[i])

  predtrain <- predict(fit, training)
  predtest <- predict(fit, test)

  trainerrors[i] <- mean(predtrain != training$Purchase)
  testerrors[i] <- mean(predtest != test$Purchase)
}

par(mar = c(8, 5, 2, 2))


plot(costs, trainerrors, type = "l", col = "blue", pch = 16, xlab = "Cost", ylab = "Error",
     main = "Training and Test Errors vs. Cost Parameters", log = "y")


points(costs, testerrors, type = "l", col = "red", pch = 17)

legend("topright", legend = c("Training Error", "Test Error"), col = c("blue", "red"), pch = c(16, 17))
```



```r
svm_tune <- tune(svm, Purchase ~ ., data = training, ranges = list(cost = costs), kernel = "radial")


optimal_cost <- svm_tune$best.model
optimal_cost
```

```
##
## Call:
## best.tune(METHOD = svm, train.x = Purchase ~ ., data = training,
##     ranges = list(cost = costs), kernel = "radial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  0.81
##
## Number of Support Vectors:  340
```

```
y_hat <- predict(optimal_cost, newdata = test)
y_true <- test$Purchase
accuracy_radial <- length(which(y_hat == y_true))/length(y_true)
accuracy_radial
```

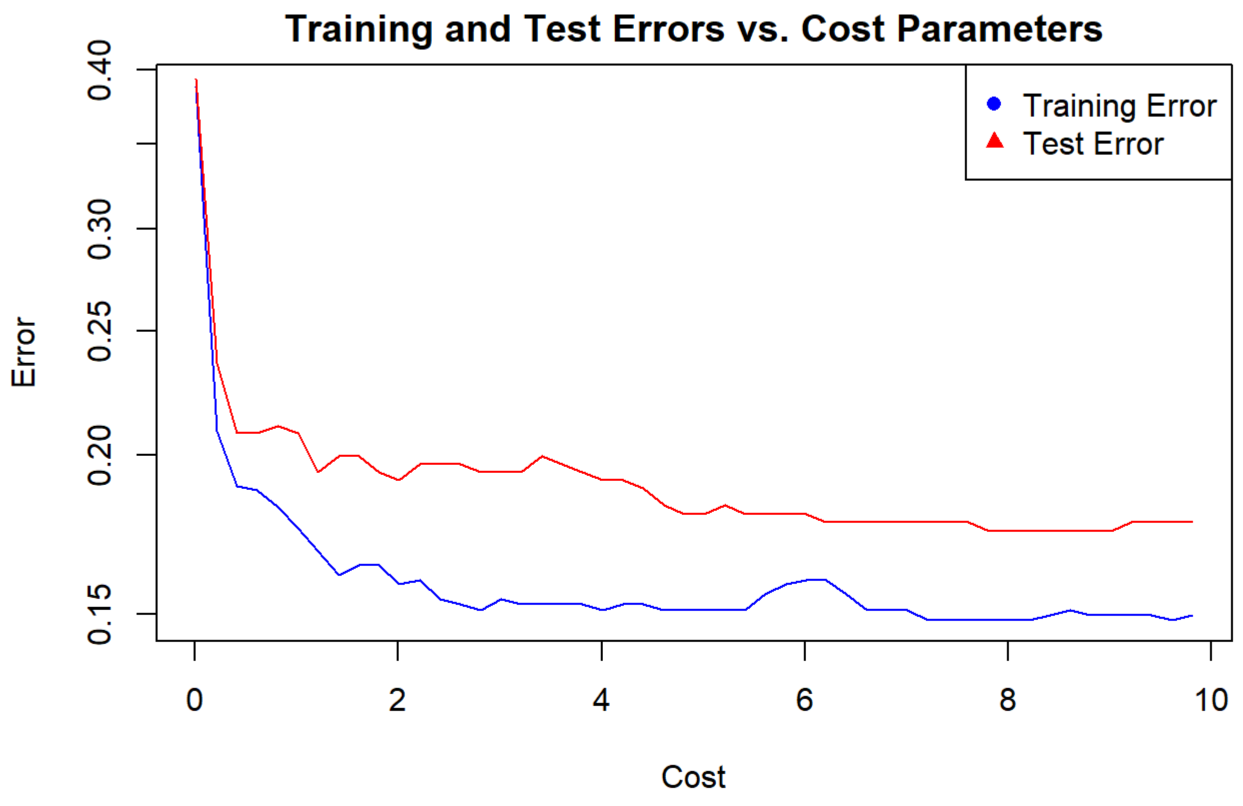
```
## [1] 0.8117978
```

```
confusion_matrixradial <- confusionMatrix(y_hat, y_true)
```

#So, we see that the optimal cost using radial kernel is 0.81

##polynomial kernel

```
trainerrors <- rep(NA, length(costs))
testerrors <- rep(NA, length(costs))
set.seed(12345)

for (i in 1:length(costs)) {
  fit <- svm(Purchase ~ ., data = training, kernel = "polynomial", degree = 2, cost = costs[i])

  predtrain <- predict(fit, training)
  predtest <- predict(fit, test)

  trainerrors[i] <- mean(predtrain != training$Purchase)
  testerrors[i] <- mean(predtest != test$Purchase)
}

par(mar = c(8, 5, 2, 2))


plot(costs, trainerrors, type = "l", col = "blue", pch = 16, xlab = "Cost", ylab = "Error",
     main = "Training and Test Errors vs. Cost Parameters", log = "y")


points(costs, testerrors, type = "l", col = "red", pch = 17)

legend("topright", legend = c("Training Error", "Test Error"), col = c("blue", "red"), pch = c(16, 17))
```



```
svm_tune <- tune(svm, Purchase ~ ., data = training, ranges = list(cost = costs), kernel = "polynomia
l", degree = 2)


optimal_cost <- svm_tune$best.model
optimal_cost
```

```
##
## Call:
## best.tune(METHOD = svm, train.x = Purchase ~ ., data = training,
##     ranges = list(cost = costs), kernel = "polynomial", degree = 2)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  6.21
##      degree:  2
##      coef.0:  0
##
## Number of Support Vectors:  320
```

```
y_hat <- predict(optimal_cost, newdata = test)
y_true <- test$Purchase
accuracy_polynomial <- length(which(y_hat == y_true))/length(y_true)
accuracy_polynomial
```

```
## [1] 0.8230337
```

```
confusion_matrixpolynomial <- confusionMatrix(y_hat, y_true)
```

#Therefore, the optimal cost using the polynomial kernel is 6.21

#reflecting on the performance of SVM with different kernels and the support vector

```
set.seed(123)
accuracy_linear
```

```
## [1] 0.8117978
```

```
accuracy_radial
```

```
## [1] 0.8117978
```

```
accuracy_polynomial
```

```
## [1] 0.8230337
```

```
confusion_matrixlinear
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  CH  MM
##         CH 190  41
##         MM  26  99
##
##                Accuracy : 0.8118
##                  95% CI : (0.7673, 0.8511)
##     No Information Rate : 0.6067
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.598
##
##  Mcnemar's Test P-Value : 0.0872
##
##             Sensitivity : 0.8796
##             Specificity : 0.7071
##          Pos Pred Value : 0.8225
##          Neg Pred Value : 0.7920
##              Prevalence : 0.6067
##          Detection Rate : 0.5337
##    Detection Prevalence : 0.6489
##       Balanced Accuracy : 0.7934
##
##        'Positive' Class : CH
##
```

```
confusion_matrixradial
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  CH  MM
##         CH 195  46
##         MM  21  94
##
##                Accuracy : 0.8118
##                  95% CI : (0.7673, 0.8511)
##     No Information Rate : 0.6067
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5928
##
##  Mcnemar's Test P-Value : 0.003367
##
##             Sensitivity : 0.9028
##             Specificity : 0.6714
##          Pos Pred Value : 0.8091
##          Neg Pred Value : 0.8174
##              Prevalence : 0.6067
##          Detection Rate : 0.5478
##    Detection Prevalence : 0.6770
##       Balanced Accuracy : 0.7871
##
##        'Positive' Class : CH
##
```

```
confusion_matrixpolynomial
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  CH  MM
##         CH 199  46
##         MM  17  94
##
##                Accuracy : 0.823
##                  95% CI : (0.7793, 0.8613)
##     No Information Rate : 0.6067
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6151
##
##  Mcnemar's Test P-Value : 0.0004192
##
##             Sensitivity : 0.9213
##             Specificity : 0.6714
##          Pos Pred Value : 0.8122
##          Neg Pred Value : 0.8468
##              Prevalence : 0.6067
##          Detection Rate : 0.5590
##    Detection Prevalence : 0.6882
##       Balanced Accuracy : 0.7964
##
##        'Positive' Class : CH
##
```

```
#another way to compare the performance is to take out one single test error without involving any rang
e for the cost parameters.

linear_svm <- svm(Purchase ~ ., data = training, kernel = "linear")
pred_linear <- predict(linear_svm, test)
test_error_linear <- mean(pred_linear != test$Purchase)

radial_svm <- svm(Purchase ~ ., data = training, kernel = "radial")
pred_radial <- predict(radial_svm, test)
test_error_radial <- mean(pred_radial != test$Purchase)



poly_svm <- svm(Purchase ~ ., data = training, kernel = "polynomial", degree = 3)
pred_poly <- predict(poly_svm, test)
test_error_poly <- mean(pred_poly != test$Purchase)




test_error_linear
```

```
## [1] 0.1882022
```

```
test_error_radial
```

```
## [1] 0.1853933
```

```
test_error_poly
```

```
## [1] 0.1910112
```

## Data Preparation

### Splitting the Data:

I began by dividing the OJ dataset into a training set and a test set. I selected one-third of the data as the test set, with the remaining two-thirds forming the training set. This approach ensured that the model could be trained and tested on separate data, providing a reliable measure of its performance.

## Support Vector Classifier with a Linear Kernel

### Model Training and Evaluation:

I started by fitting a Support Vector Classifier (SVC) with a linear kernel to the training data. To determine the optimal cost parameter, I tested a range of values from 0.01 to 10. For each cost value, I calculated the training and test errors and plotted these errors against the cost parameter.

## Results:

The plot of training and test errors helped me visualize the trade-off between model complexity and error rates. I also used the tune function to fine-tune the model, which identified a cost of 1 as the optimal value. This cost parameter minimized the test error while maintaining a reasonable training error.

## Support Vector Machine with a Radial Kernel

### Model Training and Evaluation:

Next, I repeated the process using an SVM with a radial kernel. Similar to the linear kernel, I varied the cost parameter over the same range and plotted the training and test errors.

### Results:

For the radial kernel, the optimal cost identified was 0.81. This kernel demonstrated a slight improvement in accuracy compared to the linear kernel, suggesting that the radial kernel may be more effective in capturing the underlying patterns in the data.

### Support Vector Machine with a Polynomial Kernel

### Model Training and Evaluation:

Finally, I trained an SVM with a polynomial kernel (degree 2). The approach was the same as before: I varied the cost parameter, calculated the errors, and plotted them.

## Results:

The polynomial kernel yielded an optimal cost of 6.21. It showed a modest improvement in accuracy over both the linear and radial kernels. However, this kernel had a slightly higher test error, indicating that while it could be more accurate, it might not generalize as well to unseen data.

## Performance Comparison

## Accuracy Comparison:

I compared the accuracy of the models using the different kernels. The linear and radial kernels performed similarly, both achieving an accuracy of approximately 81.18%. The polynomial kernel slightly outperformed the other two with an accuracy of 82.30%.

## Test Error Comparison

The test error rates also revealed interesting insights. The radial kernel had the lowest test error (0.1854), followed closely by the linear kernel (0.1882), with the polynomial kernel showing a slightly higher test error of 0.1910. This suggests that while the polynomial kernel achieved the highest accuracy, the radial kernel was better at generalizing to unseen data, as indicated by its lower test error.

## Overall Performance Analysis

## Linear Kernel:

The linear kernel performed reasonably well, balancing accuracy and test error. It is a good choice when the data is expected to be linearly separable or when simplicity and interpretability are key considerations.

## Radial Kernel:

The radial kernel demonstrated slightly better performance than the linear kernel, particularly in terms of test error. This kernel is beneficial when dealing with more complex data patterns that may not be linearly separable. Its lower test error suggests it generalizes better to new data.

## Polynomial Kernel:

The polynomial kernel, while achieving the highest accuracy, also had a slightly higher test error compared to the radial kernel. This indicates that while it may perform well on the training data, it may not generalize as effectively to new, unseen data. The polynomial kernel could be useful when the relationship between the features and the target variable is inherently non-linear and complex.

## Conclusion

The project provided a comprehensive comparison of different SVM models using linear, radial, and polynomial kernels. Each kernel type exhibited unique strengths:

Linear Kernel: A solid choice for simpler, linearly separable data with moderate accuracy and test error. Radial Kernel: Demonstrated the best generalization performance with the lowest test error, making it suitable for more complex data patterns. Polynomial Kernel: Showed the highest accuracy, but with a slight trade-off in generalization, as indicated by its higher test error. Final Thoughts The analysis highlights the importance of selecting the appropriate SVM kernel based on the specific characteristics of the data and the desired trade-off between accuracy and generalization. Understanding these nuances is crucial for data scientists when developing predictive models that need to perform well on both training and unseen data.