Basic machine learning tools in R

Dr. Niladri Chakraborty

2025-05-12

What is machine elarning

Machine learning is a branch of artificial intelligence (AI) that focuses on building systems that can learn from and make decisions or predictions based on data. Machine learning algorithms use statistical techniques to learn patterns directly from data without being explicitly programmed.

Importance

Machine learning is a vital technology in today's world due to several reasons:

Data Explosion: In the current digital age, we're generating vast amounts of data every second. Machine learning algorithms can help to process, analyze, and make sense of this data, extracting valuable insights that can inform decision-making.

Automation: Machine learning algorithms can learn from data and make decisions or predictions without being explicitly programmed to do so. This capability is at the heart of many modern automation and AI systems, helping to increase efficiency and productivity in many industries.

**Predictive Capabilities:* Machine learning excels at making predictions based on past data. This is useful in many fields, such as predicting customer behavior in marketing, future stock prices in finance, patient outcomes in healthcare, or potential failures in manufacturing processes.

Personalization: Machine learning algorithms are used to create personalized experiences in many digital services. For example, recommendation systems in online shopping or entertainment platforms use machine learning to suggest products or content based on a user's past behavior.

Improving Decision Making: Machine learning can help organizations make more datadriven decisions. By providing quantitative, data-based assessments, machine learning can help reduce bias and guesswork in decision-making processes. Anomaly Detection: Machine learning is excellent at identifying unusual patterns or anomalies in large datasets. This makes it invaluable in fields like cybersecurity, where it can help to identify potential threats.

Advancements in AI: Machine learning, especially deep learning, has been crucial in the recent advancements in Artificial Intelligence. Tasks like image and speech recognition, natural language processing, and autonomous vehicles have all benefited greatly from machine learning.

Let us begin with Regression analysis.

Regression analysis

Regression analysis is a statistical technique used to model the relationship between a dependent variable and one or more independent variables. In this document, we will explore the basics of regression analysis using R and provide examples using a dataset.

Simple linear regression

Simple linear regression models the relationship between a dependent variable and a single independent variable. It assumes a linear relationship between the variables. Here's an example using the "iris" dataset:

```r
data(iris)
# Fit the simple linear regression model
model <- lm(Sepal.Length ~ Sepal.Width, data = iris)
# Print the model summary
summary(model)
```

```
##
## Call:
## lm(formula = Sepal.Length ~ Sepal.Width, data = iris)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.5561 -0.6333 -0.1120  0.5579  2.2226
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   6.5262     0.4789   13.63   <2e-16 ***
## Sepal.Width  -0.2234     0.1551   -1.44    0.152
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.8251 on 148 degrees of freedom
## Multiple R-squared:  0.01382,    Adjusted R-squared:  0.007159
## F-statistic: 2.074 on 1 and 148 DF,  p-value: 0.1519
```

Multiple linear regression

Multiple linear regression models the relationship between a dependent variable and two or more independent variables. It allows for more complex relationships and provides insights into the combined effects of multiple predictors. Here's an example using the "mtcars" dataset:

```r
# Fit the multiple linear regression model
model <- lm(mpg ~ wt + disp + hp, data = mtcars)
# Print the model summary
summary(model)
```

```
##
## Call:
## lm(formula = mpg ~ wt + disp + hp, data = mtcars)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -3.891 -1.640 -0.172  1.061  5.861
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 37.105505   2.110815  17.579  < 2e-16 ***
## wt          -3.800891   1.066191  -3.565  0.00133 **
## disp        -0.000937   0.010350  -0.091  0.92851
## hp          -0.031157   0.011436  -2.724  0.01097 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.639 on 28 degrees of freedom
## Multiple R-squared:  0.8268, Adjusted R-squared:  0.8083
## F-statistic: 44.57 on 3 and 28 DF,  p-value: 8.65e-11
```

Logistic regression

Logistic regression models the relationship between a binary dependent variable and one or more independent variables. It is used for classification tasks, where the dependent variable represents a categorical outcome. Here's an example using the "mtcars" dataset:

```r
# Fit the logistic regression model
model <- glm(vs ~ mpg + wt + hp, data = mtcars, family = binomial)
# Print the model summary
summary(model)
```

```
##
## Call:
## glm(formula = vs ~ mpg + wt + hp, family = binomial, data = mtcars)
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -10.61945   16.52453  -0.643   0.5205
## mpg           0.50291    0.48656   1.034   0.3013
## wt            3.87749    3.19255   1.215   0.2245
```

```
## hp          -0.09318    0.04318  -2.158   0.0309 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 43.860  on 31  degrees of freedom
## Residual deviance: 14.748  on 28  degrees of freedom
## AIC: 22.748
##
## Number of Fisher Scoring iterations: 8
```

Regression analysis with the ‘mtcars’ data

Now we will provide an overview of regression analysis in R using the built-in mtcars dataset. We’ll be using the lm() function to perform the regression analysis and ggplot2 for visualization.

```
library(ggplot2)
library(broom)
head(mtcars)
```

```
##                    mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

```
summary(mtcars)
```

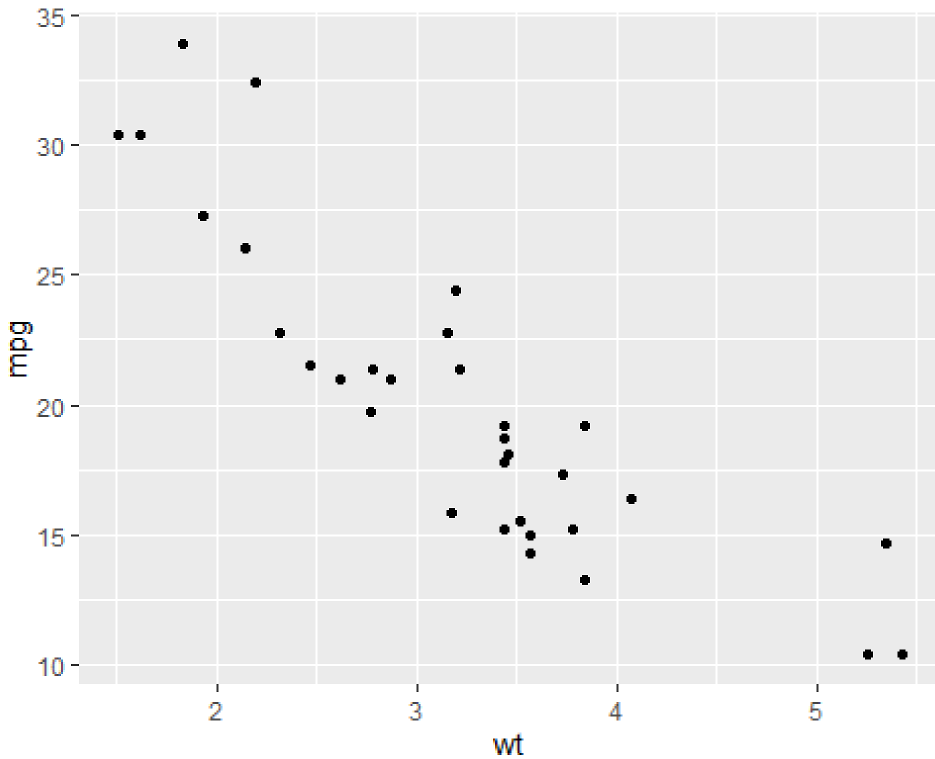
```
##       mpg             cyl             disp             hp
##  Min.   :10.40   Min.   :4.000   Min.   : 71.1   Min.   : 52.0
##  1st Qu.:15.43   1st Qu.:4.000   1st Qu.:120.8   1st Qu.: 96.5
##  Median :19.20   Median :6.000   Median :196.3   Median :123.0
##  Mean   :20.09   Mean   :6.188   Mean   :230.7   Mean   :146.7
##  3rd Qu.:22.80   3rd Qu.:8.000   3rd Qu.:326.0   3rd Qu.:180.0
##  Max.   :33.90   Max.   :8.000   Max.   :472.0   Max.   :335.0
##       drat             wt             qsec             vs
##  Min.   :2.760   Min.   :1.513   Min.   :14.50   Min.   :0.0000
##  1st Qu.:3.080   1st Qu.:2.581   1st Qu.:16.89   1st Qu.:0.0000
##  Median :3.695   Median :3.325   Median :17.71   Median :0.0000
##  Mean   :3.597   Mean   :3.217   Mean   :17.85   Mean   :0.4375
##  3rd Qu.:3.920   3rd Qu.:3.610   3rd Qu.:18.90   3rd Qu.:1.0000
##  Max.   :4.930   Max.   :5.424   Max.   :22.90   Max.   :1.0000
##        am              gear            carb
##  Min.   :0.0000   Min.   :3.000   Min.   :1.000
##  1st Qu.:0.0000   1st Qu.:3.000   1st Qu.:2.000
```

```
##  Median :0.0000   Median :4.000   Median :2.000
##  Mean   :0.4062   Mean   :3.688   Mean   :2.812
##  3rd Qu.:1.0000   3rd Qu.:4.000   3rd Qu.:4.000
##  Max.   :1.0000   Max.   :5.000   Max.   :8.000
```

We'll start with a simple linear regression, using miles per gallon (mpg) as the response variable and weight (wt) as the predictor.

Let us create a scatter plot for the 'mpg' values against 'wt' values.

```
ggplot(mtcars, aes(x=wt, y=mpg)) +
 geom_point()
```



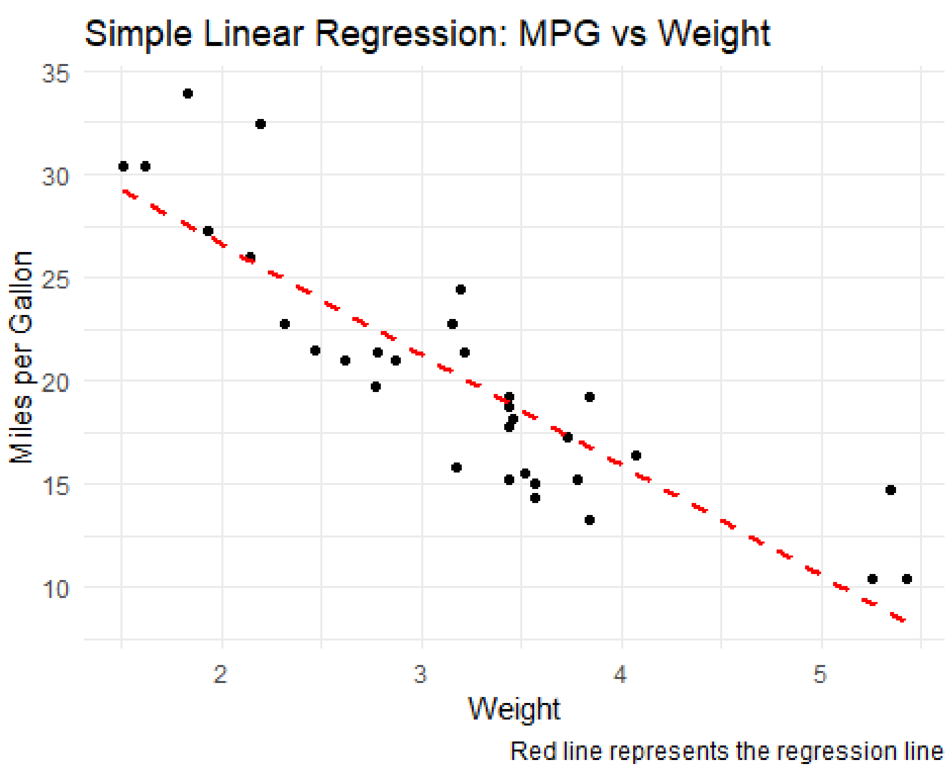
Then we fit a simple linear regression model.

```
simple_model <- lm(mpg ~ wt, data = mtcars)
summary(simple_model)

##
## Call:
## lm(formula = mpg ~ wt, data = mtcars)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.5432 -2.3647 -0.1252  1.4096  6.8727
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
```

```
## (Intercept)  37.2851     1.8776  19.858  < 2e-16 ***
## wt           -5.3445     0.5591  -9.559 1.29e-10 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.046 on 30 degrees of freedom
## Multiple R-squared:  0.7528, Adjusted R-squared:  0.7446
## F-statistic: 91.38 on 1 and 30 DF,  p-value: 1.294e-10
```

Plotting the regression line

```
ggplot(mtcars, aes(x=wt, y=mpg)) +
 geom_point() +
 geom_smooth(method=lm, se=FALSE, color="red", linetype="dashed") +
 theme_minimal() +
 labs(title="Simple Linear Regression: MPG vs Weight",
 x="Weight",
 y="Miles per Gallon",
 caption="Red line represents the regression line")

## `geom_smooth()` using formula = 'y ~ x'
```



Now, let's try a multiple linear regression using mpg as the response variable and wt and hp (horsepower) as predictors.

```
multiple_model <- lm(mpg ~ wt + hp, data = mtcars)
summary(multiple_model)
```

```
##
## Call:
## lm(formula = mpg ~ wt + hp, data = mtcars)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -3.941 -1.600 -0.182  1.050  5.854
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 37.22727    1.59879  23.285  < 2e-16 ***
## wt          -3.87783    0.63273  -6.129 1.12e-06 ***
## hp          -0.03177    0.00903  -3.519  0.00145 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.593 on 29 degrees of freedom
## Multiple R-squared:  0.8268, Adjusted R-squared:  0.8148
## F-statistic: 69.21 on 2 and 29 DF,  p-value: 9.109e-12
```

You can access the fitted values and residuals of a linear model fit with the lm() function as follows:

```
# Print the fitted values
fitted_values <- multiple_model$fitted.values
print(fitted_values)
```

```
##           Mazda RX4       Mazda RX4 Wag          Datsun 710      Hornet 4
Drive
##           23.572329           22.583483           25.275819
21.265020
##   Hornet Sportabout             Valiant          Duster 360           Merc
240D
##           18.327267           20.473816           15.599042
22.887067
##            Merc 230            Merc 280           Merc 280C          Merc
450SE
##           21.993673           19.979460           19.979460
15.725369
##          Merc 450SL         Merc 450SLC  Cadillac Fleetwood Lincoln
Continental
##           17.043831           16.849939           10.355205
9.362733
##   Chrysler Imperial            Fiat 128         Honda Civic       Toyota
Corolla
##            9.192487           26.599028           29.312380
28.046209
##       Toyota Corona    Dodge Challenger         AMC Javelin
Camaro Z28
##           24.586441           18.811364           19.140979
```

```
14.552028
##    Pontiac Firebird           Fiat X1-9       Porsche 914-2        Lotus
Europa
##           16.756745           27.626653           26.037374
27.769769
##      Ford Pantera L        Ferrari Dino       Maserati Bora          Volvo
142E
##           16.546489           20.925413           12.739477
22.983649
```

```
# Print the residuals
residual_values <- multiple_model$residuals
print(residual_values)
```

```
##           Mazda RX4       Mazda RX4 Wag          Datsun 710      Hornet 4
Drive
##         -2.57232940         -1.58348256         -2.47581872
0.13497989
##   Hornet Sportabout             Valiant          Duster 360           Merc
240D
##          0.37273336         -2.37381631         -1.29904236
1.51293266
##            Merc 230            Merc 280           Merc 280C          Merc
450SE
##          0.80632669         -0.77945988         -2.17945988
0.67463146
##          Merc 450SL         Merc 450SLC  Cadillac Fleetwood Lincoln
Continental
##          0.25616901         -1.64993945          0.04479541
1.03726743
##   Chrysler Imperial            Fiat 128         Honda Civic      Toyota
Corolla
##          5.50751301          5.80097202          1.08761978
5.85379085
##       Toyota Corona    Dodge Challenger         AMC Javelin
Camaro Z28
##         -3.08644148         -3.31136386         -3.94097947           -
1.25202805
##    Pontiac Firebird           Fiat X1-9       Porsche 914-2        Lotus
Europa
##          2.44325481         -0.32665313         -0.03737415
2.63023081
##      Ford Pantera L        Ferrari Dino       Maserati Bora          Volvo
142E
##         -0.74648866         -1.22541324          2.26052287           -
1.58364943
```

Let us create a data frame with the fitted values, residuals and actual 'mpg' values.

```
# Create a data frame containing observed mpg, fitted values, and residuals
results <- data.frame(
```

```
  Observed = mtcars$mpg,
  Fitted = fitted_values,
  Residuals = residual_values
)

# View the first few rows of the results
head(results)

##                   Observed   Fitted  Residuals
## Mazda RX4             21.0 23.57233 -2.5723294
## Mazda RX4 Wag         21.0 22.58348 -1.5834826
## Datsun 710            22.8 25.27582 -2.4758187
## Hornet 4 Drive        21.4 21.26502  0.1349799
## Hornet Sportabout     18.7 18.32727  0.3727334
## Valiant               18.1 20.47382 -2.3738163
```

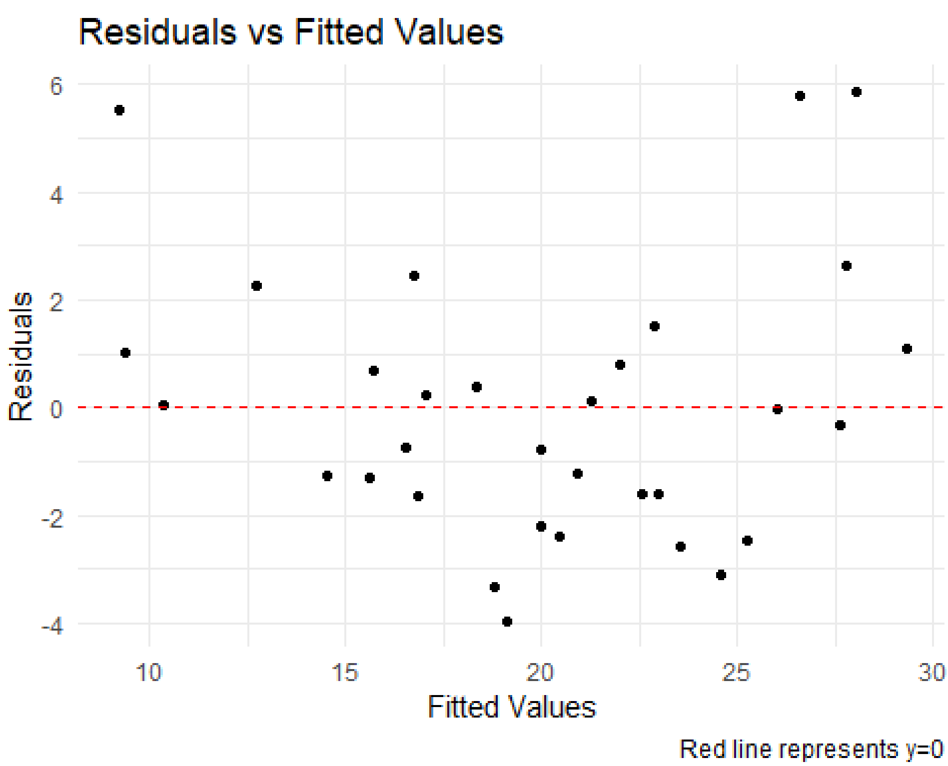
Let's check the residuals vs fitted values to see if the model meets the assumptions of linear regression.

The augment() function is part of the broom package in R, which is used to turn statistical analysis objects from R into tidy data frames. The augment() function, in particular, adds columns to the original data such as fitted values, residuals, and other useful statistics.

```
augment(multiple_model)

## # A tibble: 32 × 10
##    .rownames     mpg    wt    hp .fitted .resid   .hat .sigma .cooksd
.std.resid
##    <chr>       <dbl> <dbl> <dbl>   <dbl>  <dbl>  <dbl>  <dbl>   <dbl>
<dbl>
##  1 Mazda RX4    21    2.62   110    23.6 -2.57  0.0443   2.59 1.59e-2    -
1.01
##  2 Mazda RX4 …  21    2.88   110    22.6 -1.58  0.0405   2.62 5.46e-3    -
0.623
##  3 Datsun 710   22.8  2.32    93    25.3 -2.48  0.0602   2.59 2.07e-2    -
0.985
##  4 Hornet 4 D…  21.4  3.22   110    21.3  0.135 0.0475   2.64 4.72e-5
0.0533
##  5 Hornet Spo…  18.7  3.44   175    18.3  0.373 0.0369   2.64 2.74e-4
0.146
##  6 Valiant      18.1  3.46   105    20.5 -2.37  0.0672   2.60 2.16e-2    -
0.948
##  7 Duster 360   14.3  3.57   245    15.6 -1.30  0.117    2.63 1.26e-2    -
0.533
##  8 Merc 240D    24.4  3.19    62    22.9  1.51  0.116    2.62 1.68e-2
0.620
##  9 Merc 230     22.8  3.15    95    22.0  0.806 0.0600   2.63 2.19e-3
0.321
## 10 Merc 280     19.2  3.44   123    20.0 -0.779 0.0469   2.63 1.55e-3    -
```

```
0.308
## # ℹ 22 more rows
```

```
ggplot(augment(multiple_model), aes(.fitted, .resid)) +
 geom_point() +
 geom_hline(yintercept = 0, linetype = "dashed", color = "red") +
 theme_minimal() +
 labs(title="Residuals vs Fitted Values",
 x="Fitted Values",
 y="Residuals",
 caption="Red line represents y=0")
```



Support Vector Regression (SVR)

Support Vector Regression (SVR) is a powerful machine learning model for regression problems. In this document, we'll demonstrate how to perform SVR in R using the e1071 package and the built-in mtcars dataset.

```
library(e1071)
library(ggplot2)

svr_model <- svm(mpg ~ wt, data = mtcars, kernel = "radial")
summary(svr_model)

##
## Call:
```

```
## svm(formula = mpg ~ wt, data = mtcars, kernel = "radial")
##
##
## Parameters:
##    SVM-Type:  eps-regression
##  SVM-Kernel:  radial
##        cost:  1
##       gamma:  1
##     epsilon:  0.1
##
##
## Number of Support Vectors:  28
```

The output relates to the configuration of a Support Vector Machine (SVM) for an epsilon-regression task. Here's a breakdown of each parameter and what it signifies:

SVM-Type: eps-regression

This specifies the type of SVM being used, which is epsilon-regression (ε-regression). Epsilon-regression is used for regression tasks (predicting continuous values) rather than classification. The goal is to find a function that deviates from the actual observed outputs by a margin that is at most ε for each training example.

SVM-Kernel: radial

This indicates that the kernel used for the SVM is a radial basis function (RBF) kernel. The RBF kernel is a popular choice for SVMs because it can handle non-linear relationships between features. It transforms the input space into a higher-dimensional space where it is easier to find a linear separating hyperplane.

Cost, Gamma, Epsilon are the parameters needed for the model. These parameters collectively define how the SVM will behave, particularly in how it balances the accuracy against the model complexity and how sensitive it is to the training data.

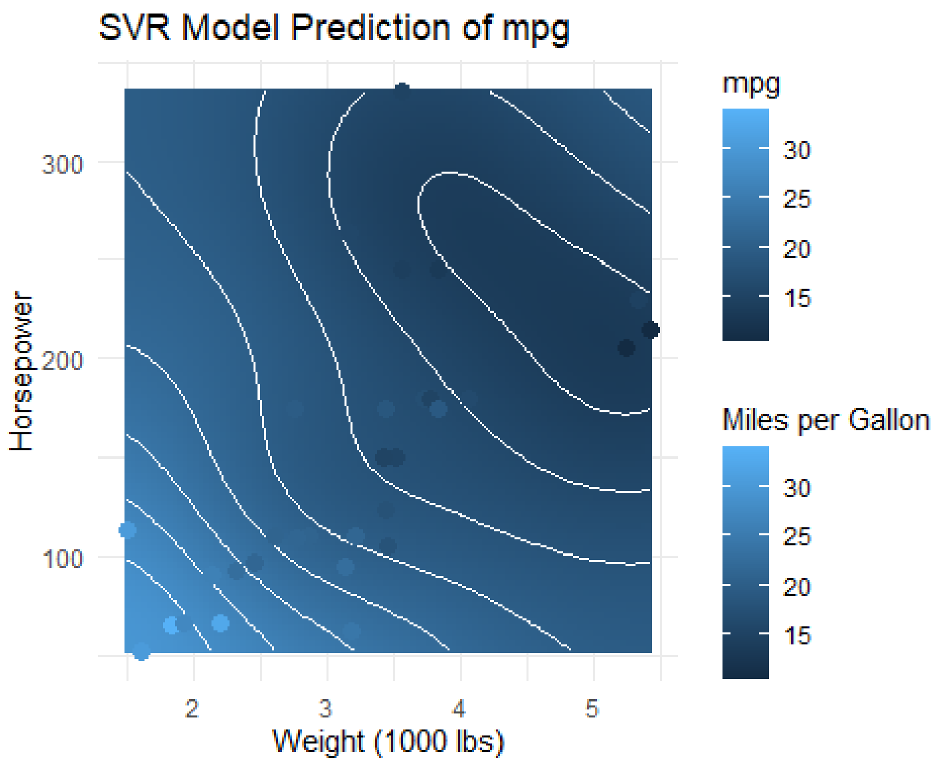
The output "Number of Support Vectors: 28" refers to the total count of support vectors used by the Support Vector Machine (SVM) model. Support vectors are the data points that lie closest to the decision surface (or hyperplane) and are crucial in defining the position and orientation of the hyperplane. These are the points that help the SVM model achieve the best separation between different classes or, in the case of regression, fit the optimal line or curve.

A higher number of support vectors can indicate a model that is highly adapted to the training data, possibly leading to overfitting. Conversely, fewer support vectors might suggest a simpler model, which could potentially underfit the data if not enough complexity is captured.

```
# Load necessary library
library(e1071)
library(ggplot2)
```

```r
# Load the mtcars dataset
data("mtcars")

# Fit the SVR model predicting mpg based on wt and hp
svr_model <- svm(mpg ~ wt + hp, data = mtcars, type = "eps-regression",
kernel = "radial")

# Make predictions over a grid to plot
wt_seq <- seq(min(mtcars$wt), max(mtcars$wt), length.out = 100)
hp_seq <- seq(min(mtcars$hp), max(mtcars$hp), length.out = 100)
grid <- expand.grid(wt = wt_seq, hp = hp_seq)
grid$mpg <- predict(svr_model, newdata = grid)

# Basic plot of the fitted surface
ggplot(grid, aes(x = wt, y = hp, fill = mpg)) +
  geom_tile() +
  geom_contour(aes(z = mpg), color = "white") +
  labs(title = "SVR Model Prediction of mpg",
       x = "Weight (1000 lbs)",
       y = "Horsepower",
       fill = "Miles per Gallon") +
  theme_minimal()+

# Optionally add the actual data points
geom_point(data = mtcars, aes(x = wt, y = hp, color = mpg), size = 3)
```

```
## Warning: The following aesthetics were dropped during statistical
transformation: fill
## ℹ This can happen when ggplot fails to infer the correct grouping
structure in
##   the data.
## ℹ Did you forget to specify a `group` aesthetic or to convert a numerical
##   variable into a factor?
```



1. Axes: The x-axis represents the weight of the cars (wt) in thousands of pounds, and the y-axis represents the horsepower (hp). These are the two features used to predict the miles per gallon (mpg), which is depicted by both the color fill and the contour lines.

2. Color Fill: The varying shades of blue represent different predicted mpg values, with lighter shades indicating higher mpg and darker shades indicating lower mpg. The color legend on the right side of the plot maps the color gradient to the mpg values.

3. Contour Lines: The white lines are contour lines that represent levels of constant mpg. These lines help you to see the predicted mpg at different combinations of wt and hp. Where the lines are closer together, the mpg changes more rapidly with changes in wt and hp.

4. Data Points: The dark dots on the plot are the actual data points from the mtcars dataset. The position of each dot shows the actual wt and hp for each car, and the color of the dot reflects its actual mpg (according to the color scale on the right).

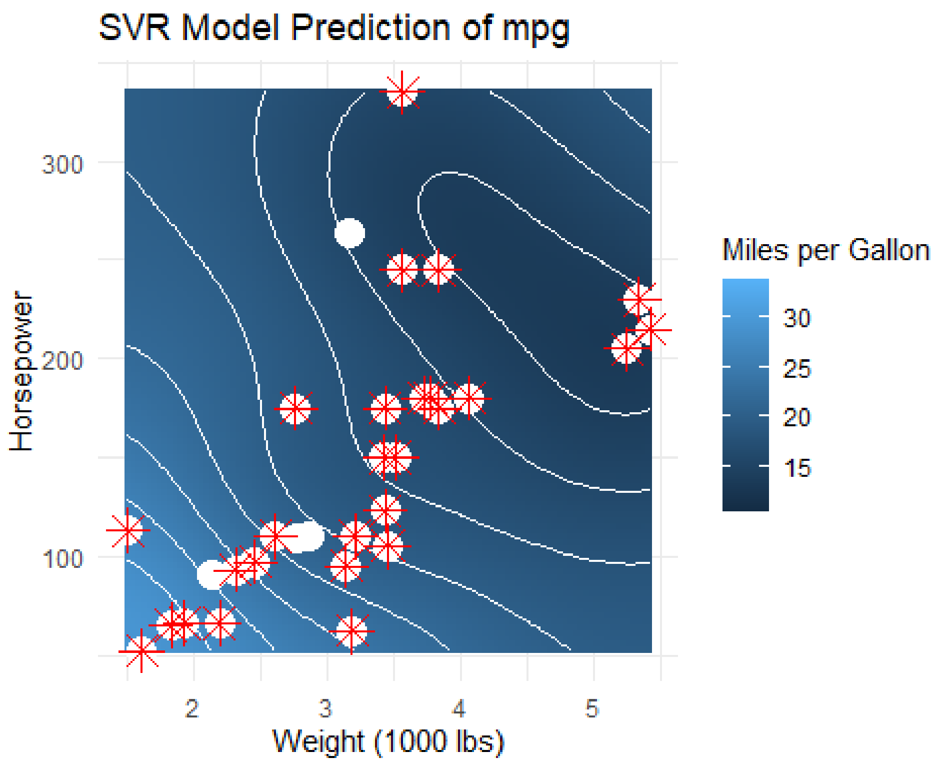
5. Model Interpretation: You can see from the contour lines and color fill that generally, as the weight of the car increases (moving right on the x-axis) and the horsepower increases (moving up on the y-axis), the mpg tends to decrease (the plot gets darker). This means that heavier cars with more horsepower tend to have lower fuel efficiency according to the SVR model's predictions.

Fit Assessment: By looking at how closely the actual data points (dots) align with the contour lines, you can get an initial qualitative sense of how well the SVR model fits the data. If most points are near or on the contour lines that correspond to their color, it suggests a good fit.

A good fit would also be indicated by data points being distributed somewhat evenly across the contour lines, rather than clustering at particular contour lines or regions of the plot. The points are mostly in the lower half of the plot, suggesting that cars with higher weight and horsepower tend to have lower mpg, which fits with our expectations.

```
# Identify the indices of the support vectors
support_vector_indices <- svr_model$index

# Extract the support vectors from the original data
support_vectors <- mtcars[support_vector_indices, ]

# Add the support vectors to the plot with a different shape or color to
distinguish them
ggplot(grid, aes(x=wt,y=hp,fill=mpg))+
  geom_tile()+
  geom_contour(aes(z=mpg),color = "white")+
  geom_point(data = mtcars, aes(x=wt,y=hp),color= "white", size=5)+
  geom_point(data = support_vectors,aes(x=wt,y=hp), color = "red", size = 5,
shape=8)+
    # Red squares for support vectors
  labs(title = "SVR Model Prediction of mpg",
       x = "Weight (1000 lbs)",
       y = "Horsepower",
       fill = "Miles per Gallon") +
  theme_minimal()
```

```
## Warning: The following aesthetics were dropped during statistical
transformation: fill
## ℹ This can happen when ggplot fails to infer the correct grouping
structure in
##   the data.
## ℹ Did you forget to specify a `group` aesthetic or to convert a numerical
##   variable into a factor?
```



Let's evaluate the performance of the model using Mean Squared Error (MSE).

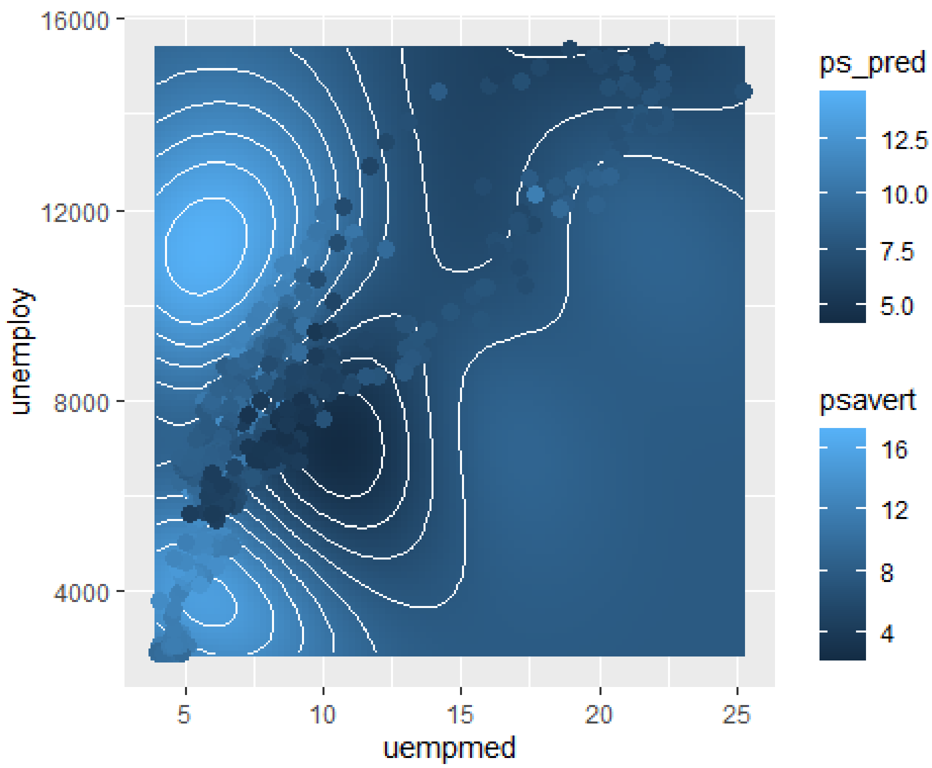
```
predictions <- predict(svr_model, mtcars)
mse <- mean((mtcars$mpg - predictions)^2)
print(paste("MSE: ", mse))

## [1] "MSE:  4.07101515023742"
```

MSE is a measure of how well the model fits the data. It is the average of the squared differences between the predicted and actual values. Lower values indicate a better fit to the data.

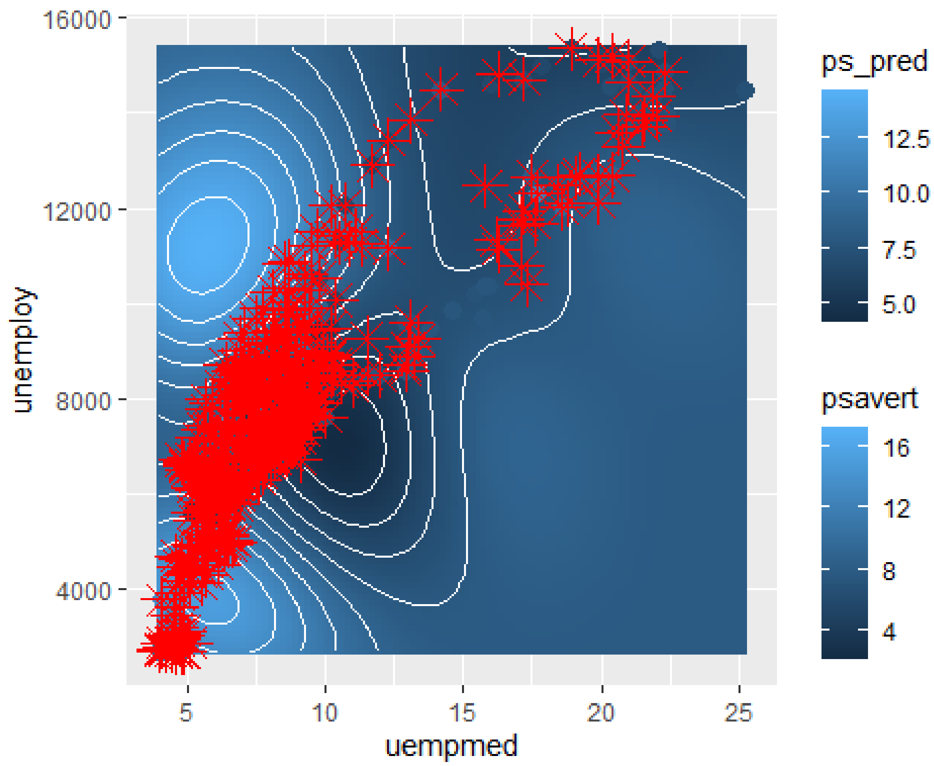
Another example with the 'economics' data

```
library(e1071)
data = economics
# Fit the SVR model predicting mpg based on wt and hp
svr_model <- svm(psavert ~ uempmed + unemploy, data = data, type = "eps-
regression",
                 kernel = "radial")
# Make predictions over a grid to plot
uemp_seq <- seq(min( data$uempmed), max(data$uempmed), length.out = 100)
unemploy_seq <- seq(min(data$unemploy), max(data$unemploy), length.out = 100)
grid <- expand.grid(uempmed = uemp_seq, unemploy = unemploy_seq)
grid$ps_pred <- predict(svr_model, newdata = grid)
# Basic plot of the fitted surface
ggplot(grid, aes(x = uempmed, y = unemploy)) +
```

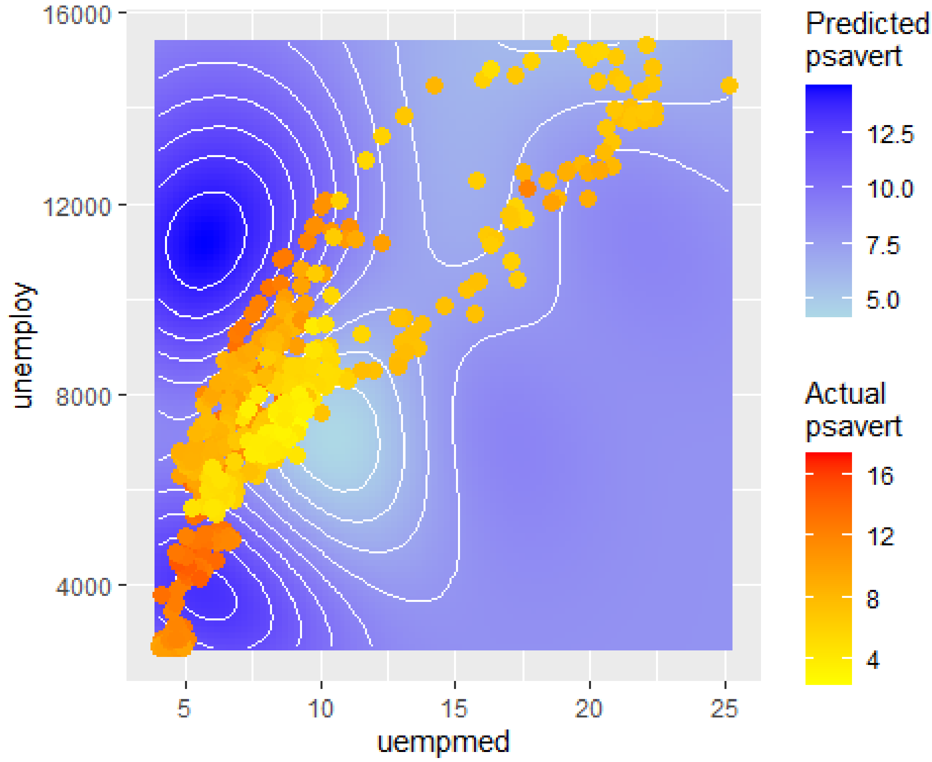
```
  geom_tile(aes(fill = ps_pred)) +
  geom_contour(aes(z = ps_pred), color = "white")+
  geom_point(data = data, aes(x = uempmed, y = unemploy, color = psavert),
size = 3)
```



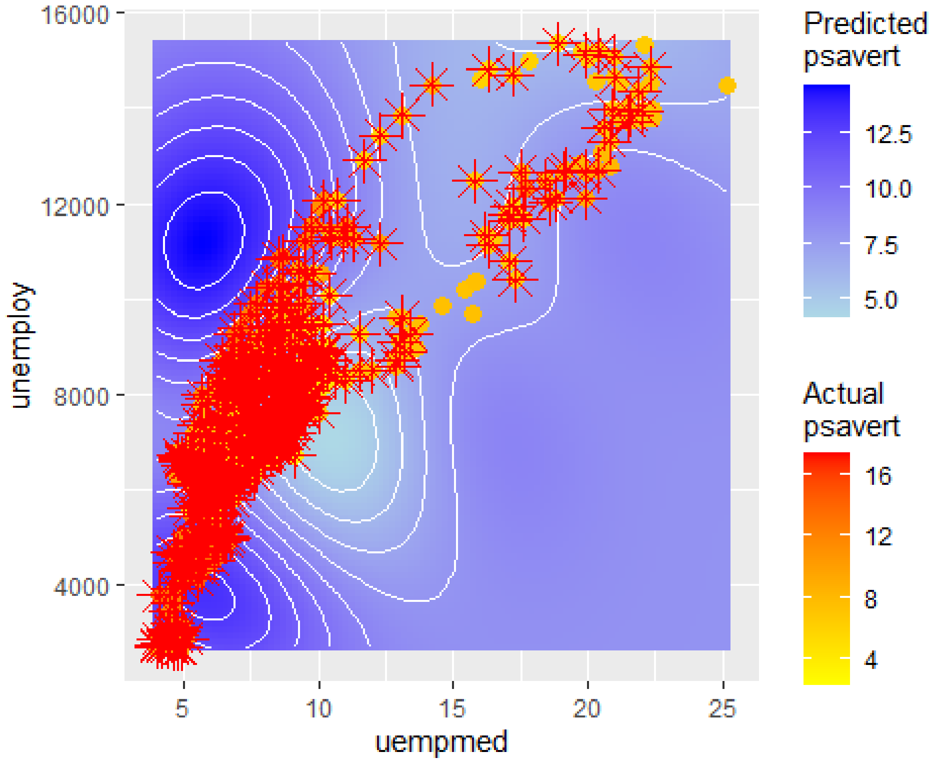

```
# Identify the indices of the support vectors
support_vector_indices <- svr_model$index
# Extract the support vectors from the original data
support_vectors <- data[support_vector_indices, ]

# Basic plot of the fitted surface
ggplot(grid, aes(x = uempmed, y = unemploy)) +
  geom_tile(aes(fill = ps_pred)) +
  geom_contour(aes(z = ps_pred), color = "white")+
  geom_point(data = data, aes(x = uempmed, y = unemploy, color = psavert),
size = 3)+
  geom_point(data = support_vectors,aes(x=uempmed,y=unemploy), color = "red",
size = 5, shape=8)
```




```
ggplot(grid, aes(x = uempmed, y = unemploy)) +
  # Background tile from prediction surface
  geom_tile(aes(fill = ps_pred)) +
  # Contour lines
  geom_contour(aes(z = ps_pred), color = "white") +
  # Overlay original data points, manually colored by actual psavert
  geom_point(data = data, aes(x = uempmed, y = unemploy, color = psavert),
size = 3) +

  # Manual color scale for points
  scale_color_gradient(low = "yellow", high = "red", name =
"Actual\npsavert") +

  # Optional: color scale for tile fill (model predictions)
  scale_fill_gradient(low = "lightblue", high = "blue", name =
"Predicted\npsavert")
```




```
ggplot(grid, aes(x = uempmed, y = unemploy)) +
  # Background tile from prediction surface
  geom_tile(aes(fill = ps_pred)) +
  # Contour lines
  geom_contour(aes(z = ps_pred), color = "white") +
  # Overlay original data points, manually colored by actual psavert
  geom_point(data = data, aes(x = uempmed, y = unemploy, color = psavert),
size = 3) +

  # Manual color scale for points
  scale_color_gradient(low = "yellow", high = "red", name =
"Actual\npsavert") +

  # Optional: color scale for tile fill (model predictions)
  scale_fill_gradient(low = "lightblue", high = "blue", name =
"Predicted\npsavert")+
    geom_point(data = support_vectors,aes(x=uempmed,y=unemploy), color =
"red", size = 5, shape=8)
```



Artificial Neural Network.

Artificial Neural Networks (ANN) are a class of machine learning models that are inspired by the structure of the human brain. In this document, we'll demonstrate how to perform ANN in R using the neuralnet package and the built-in mtcars dataset.

Let's create a neural network model using miles per gallon (mpg) as the response variable and weight (wt) and horsepower (hp) as predictors.

```
library(neuralnet)

nn_model <- neuralnet(mpg ~ wt + hp, data = mtcars, hidden = 2)
print(nn_model)

## $call
## neuralnet(formula = mpg ~ wt + hp, data = mtcars, hidden = 2)
##
## $response
##                      mpg
## Mazda RX4           21.0
## Mazda RX4 Wag       21.0
## Datsun 710          22.8
## Hornet 4 Drive      21.4
## Hornet Sportabout   18.7
## Valiant             18.1
```

```
## Duster 360          14.3
## Merc 240D           24.4
## Merc 230            22.8
## Merc 280            19.2
## Merc 280C           17.8
## Merc 450SE          16.4
## Merc 450SL          17.3
## Merc 450SLC         15.2
## Cadillac Fleetwood  10.4
## Lincoln Continental 10.4
## Chrysler Imperial   14.7
## Fiat 128            32.4
## Honda Civic         30.4
## Toyota Corolla      33.9
## Toyota Corona       21.5
## Dodge Challenger    15.5
## AMC Javelin         15.2
## Camaro Z28          13.3
## Pontiac Firebird    19.2
## Fiat X1-9           27.3
## Porsche 914-2       26.0
## Lotus Europa        30.4
## Ford Pantera L      15.8
## Ferrari Dino        19.7
## Maserati Bora       15.0
## Volvo 142E          21.4
##
## $covariate
##                        wt  hp
## Mazda RX4           2.620 110
## Mazda RX4 Wag       2.875 110
## Datsun 710          2.320  93
## Hornet 4 Drive      3.215 110
## Hornet Sportabout   3.440 175
## Valiant             3.460 105
## Duster 360          3.570 245
## Merc 240D           3.190  62
## Merc 230            3.150  95
## Merc 280            3.440 123
## Merc 280C           3.440 123
## Merc 450SE          4.070 180
## Merc 450SL          3.730 180
## Merc 450SLC         3.780 180
## Cadillac Fleetwood  5.250 205
## Lincoln Continental 5.424 215
## Chrysler Imperial   5.345 230
## Fiat 128            2.200  66
## Honda Civic         1.615  52
## Toyota Corolla      1.835  65
## Toyota Corona       2.465  97
```

```
## Dodge Challenger    3.520 150
## AMC Javelin         3.435 150
## Camaro Z28          3.840 245
## Pontiac Firebird    3.845 175
## Fiat X1-9           1.935  66
## Porsche 914-2       2.140  91
## Lotus Europa        1.513 113
## Ford Pantera L      3.170 264
## Ferrari Dino        2.770 175
## Maserati Bora       3.570 335
## Volvo 142E          2.780 109
##
## $model.list
## $model.list$response
## [1] "mpg"
##
## $model.list$variables
## [1] "wt" "hp"
##
##
## $err.fct
## function (x, y)
## {
##     1/2 * (y - x)^2
## }
## <bytecode: 0x000001f551151a00>
## <environment: 0x000001f551150308>
## attr(,"type")
## [1] "sse"
##
## $act.fct
## function (x)
## {
##     1/(1 + exp(-x))
## }
## <bytecode: 0x000001f55114f0c0>
## <environment: 0x000001f55114e7c8>
## attr(,"type")
## [1] "logistic"
##
## $linear.output
## [1] TRUE
##
## $data
##                      mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4           21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag       21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710          22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive      21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout   18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
```

```
## Valiant             18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
## Duster 360          14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
## Merc 240D           24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## Merc 230            22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
## Merc 280            19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
## Merc 280C           17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
## Merc 450SE          16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
## Merc 450SL          17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
## Merc 450SLC         15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
## Cadillac Fleetwood  10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
## Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
## Chrysler Imperial   14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
## Fiat 128            32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
## Honda Civic         30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
## Toyota Corolla      33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
## Toyota Corona       21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
## Dodge Challenger    15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2
## AMC Javelin         15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2
## Camaro Z28          13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4
## Pontiac Firebird    19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
## Fiat X1-9           27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
## Porsche 914-2       26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
## Lotus Europa        30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
## Ford Pantera L      15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
## Ferrari Dino        19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
## Maserati Bora       15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
## Volvo 142E          21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
##
## $exclude
## NULL
##
## $net.result
## $net.result[[1]]
##                         [,1]
## Mazda RX4           20.09049
## Mazda RX4 Wag       20.09049
## Datsun 710          20.09049
## Hornet 4 Drive      20.09049
## Hornet Sportabout   20.09049
## Valiant             20.09049
## Duster 360          20.09049
## Merc 240D           20.09049
## Merc 230            20.09049
## Merc 280            20.09049
## Merc 280C           20.09049
## Merc 450SE          20.09049
## Merc 450SL          20.09049
## Merc 450SLC         20.09049
## Cadillac Fleetwood  20.09049
## Lincoln Continental 20.09049
```

```
## Chrysler Imperial   20.09049
## Fiat 128            20.09049
## Honda Civic         20.09049
## Toyota Corolla      20.09049
## Toyota Corona       20.09049
## Dodge Challenger    20.09049
## AMC Javelin         20.09049
## Camaro Z28          20.09049
## Pontiac Firebird    20.09049
## Fiat X1-9           20.09049
## Porsche 914-2       20.09049
## Lotus Europa        20.09049
## Ford Pantera L      20.09049
## Ferrari Dino        20.09049
## Maserati Bora       20.09049
## Volvo 142E          20.09049
##
##
## $weights
## $weights[[1]]
## $weights[[1]][[1]]
##           [,1]       [,2]
## [1,] 0.1772530 -0.4781140
## [2,] 1.9826038  0.3011556
## [3,] 0.6673593  1.5121305
##
## $weights[[1]][[2]]
##          [,1]
## [1,] 6.912939
## [2,] 6.578558
## [3,] 6.598996
##
##
##
## $generalized.weights
## $generalized.weights[[1]]
##                     [,1] [,2]
## Mazda RX4              0    0
## Mazda RX4 Wag          0    0
## Datsun 710             0    0
## Hornet 4 Drive         0    0
## Hornet Sportabout      0    0
## Valiant                0    0
## Duster 360             0    0
## Merc 240D              0    0
## Merc 230               0    0
## Merc 280               0    0
## Merc 280C              0    0
## Merc 450SE             0    0
## Merc 450SL             0    0
```

```
## Merc 450SLC            0    0
## Cadillac Fleetwood     0    0
## Lincoln Continental    0    0
## Chrysler Imperial      0    0
## Fiat 128               0    0
## Honda Civic            0    0
## Toyota Corolla         0    0
## Toyota Corona          0    0
## Dodge Challenger       0    0
## AMC Javelin            0    0
## Camaro Z28             0    0
## Pontiac Firebird       0    0
## Fiat X1-9              0    0
## Porsche 914-2          0    0
## Lotus Europa           0    0
## Ford Pantera L         0    0
## Ferrari Dino           0    0
## Maserati Bora          0    0
## Volvo 142E             0    0
##
##
## $startweights
## $startweights[[1]]
## $startweights[[1]][[1]]
##            [,1]       [,2]
## [1,] -1.7911470 -0.4781140
## [2,]  0.0142038  0.3011556
## [3,] -1.3010407  1.5121305
##
## $startweights[[1]][[2]]
##            [,1]
## [1,] -0.2129985
## [2,] -0.5473793
## [3,] -0.5269410
##
##
##
## $result.matrix
##                                [,1]
## error                 563.023594026
## reached.threshold       0.004200819
## steps                  88.000000000
## Intercept.to.1layhid1   0.177252951
## wt.to.1layhid1          1.982603803
## hp.to.1layhid1          0.667359307
## Intercept.to.1layhid2  -0.478113993
## wt.to.1layhid2          0.301155633
## hp.to.1layhid2          1.512130453
## Intercept.to.mpg        6.912939041
## 1layhid1.to.mpg         6.578558185
```

```
## 1layhid2.to.mpg         6.598996498
##
## attr(,"class")
## [1] "nn"

#Plotting the Neural Network
#Let’s plot the neural network structure.


plot(nn_model)

# Making Predictions

#We can use the model to make predictions on the dataset.

predictions <- compute(nn_model, mtcars[,c("wt", "hp")])
head(predictions$net.result)

##                       [,1]
## Mazda RX4         20.09049
## Mazda RX4 Wag     20.09049
## Datsun 710        20.09049
## Hornet 4 Drive    20.09049
## Hornet Sportabout 20.09049
## Valiant           20.09049
```

The neuralnet function in R returns a neural network model that has been trained using the data provided to it. The result is a list that includes many components. Here are some of the key components:

$call: This shows how the neuralnet function was called, including the formula that was used to specify the model, and the data that was used.

$response: This shows the response variable used in the training, in this case, mpg.

$covariate: This is the matrix of covariates (predictor variables) used in the training. In this example, it includes wt and hp.

$model.list: This is a list that contains information about the model, including the response and covariate.

$err.fct: This is the error function that was used in the training. By default, it’s the sum of squared errors (sse).

$act.fct: This is the activation function used in the neurons of the neural network. By default, it’s the logistic function.

$linear.output: This is a logical indicator showing whether the output layer of the neural network uses a linear output function. By default, it’s TRUE.

$data: This is the data frame used to train the neural network.

$weights: This is a list of matrices that represents the final weights of the neural network after training.

$startweights: This is a list of matrices that represents the initial random weights of the neural network before training.

$result.matrix: This is a matrix that contains information about the training process, including the error at each step.

The blue circles and lines in the neural network plot represent bias units and their connections. In neural networks, bias units serve as additional "always-on" neurons that allow the activation function to be shifted left or right, which can be critical for learning and modeling complex patterns.

The plot function visualizes the structure of the neural network, including the input layer (the predictor variables), the hidden layer(s), and the output layer (the response variable). The weights of the connections between the neurons are also shown. The compute function is used to make predictions using the trained neural network model. It takes the neural network model and a data frame of predictor variables, and it returns a list. The net.result element of this list is a matrix of the predicted values.

Which one to choose? SVM or ANN?

Choosing between Support Vector Machines (SVM) and Artificial Neural Networks (ANN) is highly dependent on the specific use case, the nature of your data, and what you are trying to achieve.

Support Vector Machines (SVM): SVMs are effective in high dimensional spaces, and when the number of dimensions is greater than the number of samples.

Artificial Neural Networks (ANN): ANNs are capable of modeling complex, non-linear relationships and can be highly effective on large datasets with many input variables.

However, ANNs can be more computationally intensive than SVMs, and they can require more data preparation.

Here are examples of using both methods in R using the mtcars dataset:

```
# Load necessary libraries
library(ggplot2)
library(lattice)
library(e1071)
library(neuralnet)
library(caret)

# Split the mtcars data into training and testing datasets
```

```r
set.seed(123)
trainIndex <- createDataPartition(mtcars$mpg, p = .8,
 list = FALSE,
 times = 1)
mtcarsTrain <- mtcars[ trainIndex,]
mtcarsTest <- mtcars[-trainIndex,]

# Train a SVM model
svm_model <- svm(mpg ~ ., data = mtcarsTrain)
# Make predictions
svm_predictions <- predict(svm_model, mtcarsTest)

# Train a neural network model
nn_model <- neuralnet(mpg ~ ., data = mtcarsTrain, hidden = 2)
# Make predictions
nn_predictions <- compute(nn_model, mtcarsTest[,-1])

# Evaluate models
svm_MSE <- postResample(svm_predictions, mtcarsTest$mpg)
nn_MSE <- postResample(nn_predictions$net.result, mtcarsTest$mpg)

print(svm_MSE)

##     RMSE Rsquared      MAE
## 3.699094 0.978711 2.582040

print(nn_MSE)

##     RMSE Rsquared      MAE
##  7.82802       NA  6.20000
```

In this example, we train an SVM and an ANN model to predict miles per gallon (mpg) based on all other variables in the mtcars dataset. We then make predictions on a test set and compute the Mean Squared Error (MSE) for both models.

The model with the lower MSE would generally be considered the better model for this specific dataset and problem.