# Final Proyect Supervised Methods in Machine Learning.

Julian Abello Orozco -Juan David Ortiz Cortes-Julian Gonzalez Hernandez

2023-11-22

## Introduction

Supervised learning, a pivotal subset of machine learning, operates by training algorithms on labeled datasets. This technique involves teaching models using data where the relationship between input and output is already known. The goal is to enable algorithms to discern patterns, thereby making predictions or classifications on new, unseen data. Its significance spans multiple domains within data science and machine learning engineering. In predictive modeling, it's instrumental for forecasting outcomes based on input features, extensively applied in finance (for stock price predictions), healthcare (diagnosing diseases), and retail (forecasting customer behavior). Moreover, it plays a fundamental role in classification tasks (e.g., differentiating spam from non-spam emails) and regression (such as predicting house prices). Supervised learning models facilitate decision-making processes by automating insights in various realms like image recognition, natural language processing, and recommendation systems. Their forte lies in recognizing intricate patterns within datasets, uncovering relationships not immediately apparent to human analysts, thus offering invaluable insights in diverse industries.

The relevance of supervised learning in data science and machine learning engineering is paramount due to its ability to extract meaningful insights from labeled data. By deciphering these patterns, it enables accurate predictions and classifications, pivotal across various industries. Its applications extend to predictive modeling, decision automation, and pattern recognition, making it an indispensable tool for understanding and leveraging data in numerous sectors. This method's capability to derive significance from labeled datasets renders it vital for driving insights and enabling informed decisions in real-world scenarios.

## Theoretical Framework

**Types of Problems:**

- **Regression**: Involves predicting continuous values. For instance, predicting house prices based on features like area, location, etc.
- **Classification**: Focuses on assigning categories or labels to input data. Examples include spam/non-spam email classification or image recognition to identify objects in images.

**Algorithms:**

- **Linear Regression**: Used for regression tasks, it establishes a linear relationship between input and output variables.
- **Logistic Regression**: For classification, it estimates the probability of an instance belonging to a particular category.
- **Decision Trees, Random Forests, Support Vector Machines (SVM)**: Common algorithms for both regression and classification tasks.

**Training and Testing Models:**

- **Training**: Involves feeding labeled data to the model to learn patterns or relationships between input and output.
- **Testing**: Assessing the model's performance on unseen data to evaluate its predictive accuracy.

**Overfitting and Underfitting:**

- **Overfitting**: When a model learns too much from the training data and fails to generalize well to new data.
- **Underfitting**: Occurs when the model is too simple to capture the underlying patterns in the data.

**Cross-Validation:**

- **K-Fold Cross-Validation**: A technique where the dataset is divided into 'k' subsets. The model is trained on 'k-1' subsets and validated on the remaining subset. This process rotates until all subsets have been used for both training and validation, providing a more robust estimation of the model's performance.

**Performance Metrics:**

- **Regression**: Metrics like Mean Squared Error (MSE), Root Mean Squared Error (RMSE), or R-squared measure the accuracy of continuous value predictions.
- **Classification**: Metrics such as Accuracy, Precision, Recall, F1-score, and ROC-AUC assess the model's performance in assigning correct labels or categories.

## Methodology

The core focus of this study lies in utilizing the DynamicCancerDriverKM package to conduct comprehensive data preprocessing. This package was employed to perform thorough data analysis and refinement, effectively preparing the original dataset for subsequent analysis.

Additionally, subsets of data were created from the original dataset to apply various machine learning models. These subsets were specifically designed and tailored to address the classification of different cancer types and predict relevant clinical outcomes. The diversity of models used facilitated a detailed examination of multiple facets within the data, allowing for a more precise and detailed approach in understanding the complexity of various cancer types and their potential clinical outcomes.

**Initial Data Loading:**

```
datanormal <-(DynamicCancerDriverKM::BRCA_normal)
dataPt <-(DynamicCancerDriverKM::BRCA_PT)
final_data <- bind_rows(datanormal, dataPt)


porcentaje_menor_10 <- final_data %>%
  summarise_all(~ mean(. <400, na.rm = TRUE))


columnas_a_eliminar <- names(porcentaje_menor_10[, porcentaje_menor_10 >= 0.8])


final_data_filtrado <- final_data %>%
  select(-one_of(columnas_a_eliminar))
```

```r
final_data_filtrado2 <- final_data_filtrado

data_pii<-(DynamicCancerDriverKM::PPI)


data_piin <- data_pii %>%
  pivot_longer(cols = c(`Input-node Gene Symbol`, `Output-node Gene Symbol`), names_to = "variable", va
  group_by(gen, variable) %>%
  summarise(frecuencia = n()) %>%
  pivot_wider(names_from = variable, values_from = frecuencia, values_fill = 0)
```

```
## `summarise()` has grouped output by 'gen'. You can override using the `.groups`
## argument.
```

```r
data_piinR <- data_piin %>%
  mutate(total_mode = `Input-node Gene Symbol` + `Output-node Gene Symbol`) %>%
  select(total_mode) %>%
  arrange(desc(total_mode))
```

```
## Adding missing grouping variables: `gen`
```

```r
print(data_piinR)

final_data_filtradox<-colnames(final_data_filtrado)[ 8:ncol(final_data_filtrado)]
aux2 <- AMCBGeneUtils::changeGeneId(final_data_filtradox, from = "Ensembl.ID")

names(final_data_filtrado)[8:12631] <- aux2$HGNC.symbol


genes_en_final_data <- colnames(final_data_filtrado)


data_piinR_filtrado <- data_piinR %>%
  filter(gen %in% genes_en_final_data)
```

Two datasets, namely **datanormal** and **dataPt**, are loaded from the DynamicCancerDriverKM package. Subsequently, these datasets are merged into a single unified dataset called **final_data** using the **bind_rows** function.

**Filtering Columns with Low Values:**

The percentage of values below 400 in the **final_data** dataset is computed. Columns where more than 80% of the values are below 400 are identified. A new refined dataset, **final_data_filtrado**, is created by removing the previously identified columns.

**Protein-Protein Interaction (PPI) Data Processing:**

The **data_pii** dataset is loaded from the DynamicCancerDriverKM package. A pivoting and summarization process is executed to tally the frequency of interactions between genes. Total interaction counts for each gene are computed, sorted, and presented.

**Column Name Modification:**

Column names ranging from the 8th column to the last in **final_data_filtrado** are extracted. The AMCBGeneUtils package is utilized to convert gene identifiers to HGNC gene names.

**Filtering Protein-Protein Interaction Data:**

Genes in **data_piinR** are compared against columns in **final_data_filtrado**, and matching genes are filtered and retained

The preprocessing stages were chosen to prepare the data in a specific and suitable manner for subsequent analysis. Each stage addresses key aspects of the dataset and has a defined purpose:

**Initial Data Loading and Combination:**

Two initial cancer-related datasets are loaded from the DynamicCancerDriverKM package. Merging these datasets into one, final_data, allows working with a more comprehensive database that could contain crucial information from multiple sources.

**Filtering Low-Value Columns:**

Identifies and removes columns with a significant proportion of values below a specific threshold (in this case, below 400). This step helps to reduce noise or irrelevant information in the dataset, focusing on the most significant or relevant variables for analysis.

**Protein-Protein Interaction Data Processing (PPI):**

Addresses protein interaction-related data, crucial in cancer study due to the importance of protein-protein interactions in biological processes. Analyzes the frequency and quantity of interactions between genes, which can reveal valuable information about the biological pathways involved in cancer.

**Column Name Modification:**

Updates column names to reflect more recognizable and understandable identifiers (from gene identifiers to HGNC gene names), facilitating data interpretation and analysis.

**Filtering Protein-Protein Interaction Data:**

Filters and retains only relevant genes that have interactions within the main dataset, further focusing the analysis on the most pertinent genetic information for the specific cancer study.

These preprocessing stages were chosen to clean, prepare, and structure the data in a manner that is more suitable and meaningful for subsequent analysis related to cancer type classification and clinical outcome prediction.

## Implementation

### KNN Model

**Predictor Selection:**

- The first 100 elements of the first column of **data_piinR_filtrado** are selected.
- These elements are converted into a vector and then into characters

```
Predictores <- as.vector(head(data_piinR_filtrado[, 1], 100))
Predictores <- as.character(unlist(Predictores))

colnames(final_data_filtrado)[is.na(colnames(final_data_filtrado))] <- paste0("xs", seq_along(colna
set.seed(23)
```

**Column Names Manipulation:**

- Column names in **final_data_filtrado** that are **NA** are changed to "xs1", "xs2", and so on.

**Training and Test Data Preparation:**

- **final_data_filtrado** is divided into groups based on the **sample_type** variable.
- Random samples with replacement of 123 rows from each group are taken.
- A random sample index of 60% of the rows from **final_data_filtradoe** is chosen to create **train.data**.

- The remaining data is used for **test.data**.

```
final_data_filtradoe <- final_data_filtrado %>%
  group_by(sample_type) %>%
  sample_n(123, replace = TRUE) %>%
  ungroup()
```

**Definition of Target Variables:**

- **sample_type** in both **train.data** and **test.data** is converted into a factor, indicating it's a categorical variable to be used as the target variable in the model.

```
sample.index <- sample(1:nrow(final_data_filtradoe), nrow(final_data_filtradoe) * 0.6, replace = FA

train.data <- final_data_filtradoe[sample.index, c(Predictores, "sample_type"), drop = FALSE]
test.data <- final_data_filtradoe[-sample.index, c(Predictores, "sample_type"), drop = FALSE]

train.data$sample_type <- factor(train.data$sample_type)
test.data$sample_type <- factor(test.data$sample_type)
```

**Training the k-NN Model:**

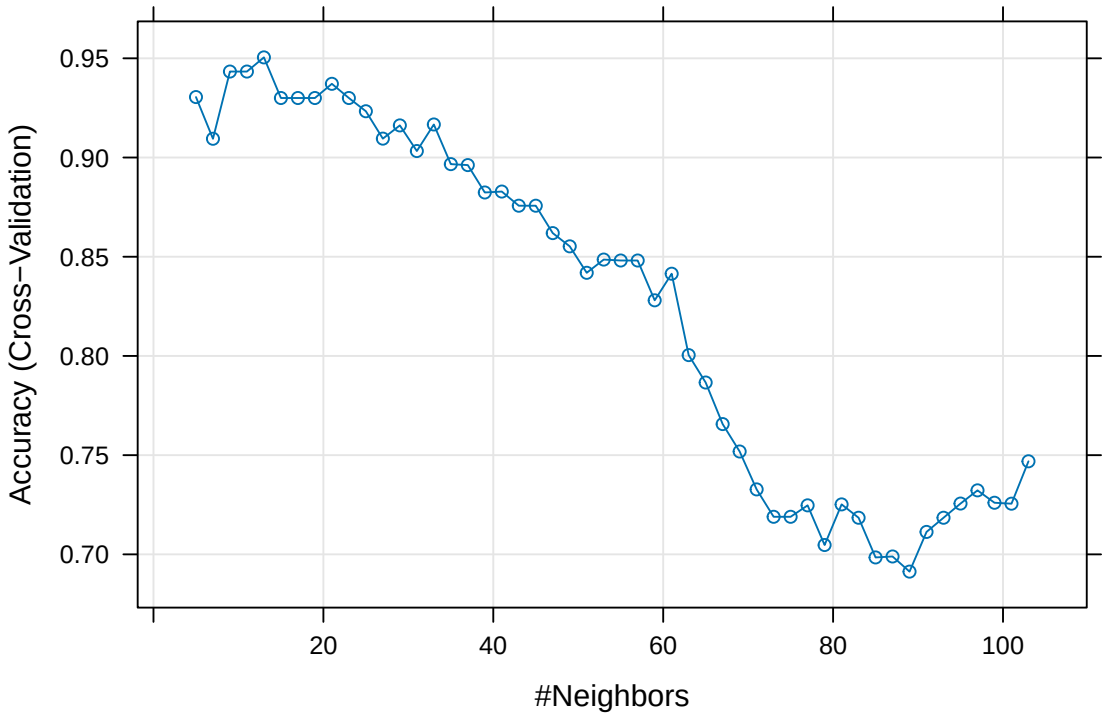
- Control parameters are set for model training using cross-validation.
- The k-NN model is trained using the training data (**train.data**), searching through 50 different nearest neighbor values.
- Data preprocessing is applied (**range** for scaling), and multiple iterations of the model are generated to evaluate its performance.

```
ctrl <- trainControl(method = "cv", p = 0.6)
knnFit <- train(sample_type ~ .,
                data = train.data,
                method = "knn",
                trControl = ctrl,
                preProcess = c("range"),  # c("center", "scale") for z-score
                tuneLength = 50)
```

**Model Visualization:**

- A plot is generated to visualize the k-NN model.

```
plot(knnFit)
```




**Prediction using the k-NN Model:**

- Predictions are made using the trained model (**knnFit**) on the test data (**test.data**).

```
knnPredict <- predict(knnFit, newdata = test.data)
```

**Confusion Matrix Creation:**

- The **confusionMatrix()** function is used to generate a confusion matrix.
- **knnPredict** represents the predicted values obtained from applying the k-NN model on the test dataset.
- **test.data$sample_type** contains the actual values of the target variable from the test dataset.

```
confusionMatrix(data = knnPredict, reference = test.data$sample_type)
```

```
## Confusion Matrix and Statistics
##
##                      Reference
## Prediction            Primary Tumor Solid Tissue Normal
##   Primary Tumor                  34                   0
##   Solid Tissue Normal            16                  49
##
##                Accuracy : 0.8384
##                  95% CI : (0.7509, 0.9047)
##     No Information Rate : 0.5051
##     P-Value [Acc > NIR] : 4.317e-12
##
##                   Kappa : 0.6778
##
##  Mcnemar's Test P-Value : 0.0001768
```

```
##
##             Sensitivity : 0.6800
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 0.7538
##              Prevalence : 0.5051
##          Detection Rate : 0.3434
##    Detection Prevalence : 0.3434
##       Balanced Accuracy : 0.8400
##
##        'Positive' Class : Primary Tumor
##
```

### Linear Regression

### Data Preparation:

- The **sample_type** variable in **final_data_filtradoe** is transformed into binary values (0 and 1) using **mutate** and **ifelse**.

```
final_data_filtradoe <- final_data_filtradoe %>%
  mutate(sample_type = ifelse(sample_type == "Solid Tissue Normal", 1, 0))
```

### Training and Test Data Split:

- The data is split into training (**train.data**) and test (**test.data**) sets using a previously generated index (**sample.index**).

```
train.data <- final_data_filtradoe[sample.index, c(Predictores, "sample_type"), drop = FALSE]
test.data <- final_data_filtradoe[-sample.index, c(Predictores, "sample_type"), drop = FALSE]
```

### Linear Regression Model:

- A linear regression model (**lm**) is fitted with **sample_type** as the response variable and all other variables as predictors in **train.data**.

```
ins_model <- lm(sample_type ~ ., data = train.data)
```

- A summary of the model is printed.

```
summary(ins_model)
```

```
##
## Call:
## lm(formula = sample_type ~ ., data = train.data)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.185117 -0.035942  0.001759  0.037991  0.207019
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  7.342e-01  1.139e-01   6.444 6.24e-08 ***
## TP53         1.436e-04  3.041e-05   4.722 2.22e-05 ***
## CREBBP      -1.951e-05  2.224e-05  -0.877 0.384812
## EP300        2.466e-05  3.022e-05   0.816 0.418698
## YWHAG       -2.062e-05  1.361e-05  -1.515 0.136734
## SMAD3       -3.005e-05  1.932e-05  -1.555 0.126806
## GRB2         2.115e-05  1.138e-05   1.858 0.069578 .
```

```
## SRC          2.869e-05  3.073e-05   0.934 0.355390
## AR          -8.776e-06  7.428e-06  -1.181 0.243516
## ESR1         2.105e-06  2.038e-06   1.033 0.306974
## RB1         -1.042e-05  3.902e-05  -0.267 0.790570
## CSNK2A1     -6.045e-05  3.042e-05  -1.987 0.052879 .
## SMAD2        1.732e-04  4.738e-05   3.656 0.000656 ***
## CDKN1A      -1.487e-05  5.428e-06  -2.740 0.008725 **
## MAPK1        2.350e-05  1.811e-05   1.298 0.200885
## FYN         -3.674e-05  4.720e-05  -0.778 0.440318
## HDAC1        4.342e-05  2.378e-05   1.826 0.074357 .
## PRKCA       -1.489e-04  6.187e-05  -2.406 0.020182 *
## TK1          1.016e-04  3.404e-05   2.985 0.004528 **
## EGFR         3.427e-05  1.017e-05   3.369 0.001534 **
## SMAD4       -1.174e-04  5.680e-05  -2.066 0.044461 *
## JUN          4.161e-06  2.389e-06   1.742 0.088177 .
## CCDC85B      3.574e-05  7.982e-05   0.448 0.656397
## MAPK6       -5.719e-05  4.629e-05  -1.235 0.222971
## GSK3B       -1.038e-04  4.260e-05  -2.437 0.018732 *
## PIK3R1      -2.975e-06  4.168e-06  -0.714 0.479039
## SMAD1       -2.261e-04  5.714e-05  -3.957 0.000261 ***
## SHC1         5.023e-05  2.148e-05   2.339 0.023761 *
## TRAF2       -4.198e-04  1.032e-04  -4.068 0.000184 ***
## YWHAZ        1.530e-06  1.215e-06   1.259 0.214323
## CASP3       -3.021e-05  9.698e-05  -0.312 0.756825
## UBE2I        7.257e-06  2.369e-05   0.306 0.760703
## SP1         -9.046e-05  2.689e-05  -3.364 0.001557 **
## VIM         -1.085e-06  1.959e-06  -0.554 0.582411
## ATXN1        7.646e-05  5.843e-05   1.309 0.197186
## SMN1        -2.857e-04  3.903e-04  -0.732 0.467885
## UBQLN4       3.875e-05  3.530e-05   1.098 0.277967
## MAPK3        1.197e-05  1.831e-05   0.654 0.516604
## PRKACA       9.637e-06  2.680e-05   0.360 0.720795
## TGFBR1      -1.683e-05  1.685e-05  -0.999 0.323093
## CSNK2B      -7.389e-04  1.714e-04  -4.311 8.49e-05 ***
## CALM1       -3.655e-05  1.047e-05  -3.493 0.001068 **
## SETDB1      -3.030e-04  8.951e-05  -3.385 0.001463 **
## YWHAB       -2.942e-05  6.416e-06  -4.585 3.49e-05 ***
## TBP          2.389e-04  1.587e-04   1.506 0.139023
## BRCA1        3.382e-04  7.365e-05   4.591 3.42e-05 ***
## RELA        -1.086e-04  5.439e-05  -1.996 0.051842 .
## CTNNB1       4.497e-06  5.438e-06   0.827 0.412479
## LCK          3.905e-05  1.215e-04   0.321 0.749301
## LYN         -4.315e-05  6.338e-05  -0.681 0.499440
## RXRA        -4.133e-05  1.837e-05  -2.249 0.029312 *
## EEF1A1      -5.969e-07  4.974e-07  -1.200 0.236263
## AKT1        -1.068e-05  2.559e-05  -0.417 0.678296
## SMAD9        6.731e-04  2.237e-04   3.010 0.004235 **
## ANXA7        1.856e-05  2.719e-05   0.682 0.498391
## STAT3       -1.704e-05  9.418e-06  -1.809 0.076991 .
## PTPN11      -4.044e-05  1.437e-05  -2.815 0.007157 **
## NCOA1        1.838e-05  2.673e-05   0.688 0.495033
## PLCG1        6.899e-05  2.501e-05   2.759 0.008298 **
## ACTB         5.728e-07  8.345e-07   0.686 0.495867
## MDFI        -1.139e-05  2.597e-05  -0.439 0.662943
```

```
## EWSR1       -7.556e-05  2.186e-05  -3.456 0.001191 **
## PTK2         1.531e-05  1.363e-05   1.123 0.267150
## RAC1         1.781e-05  1.378e-05   1.293 0.202563
## NFKB1        8.690e-05  2.621e-05   3.316 0.001788 **
## NR3C1       -7.771e-05  1.584e-05  -4.906 1.20e-05 ***
## UNC119       9.832e-06  8.889e-05   0.111 0.912407
## ABL1         7.096e-05  2.893e-05   2.453 0.018017 *
## DLG4        -1.924e-04  1.079e-04  -1.782 0.081316 .
## ATN1         3.461e-06  1.787e-05   0.194 0.847295
## NCOR2        1.061e-05  1.337e-05   0.794 0.431273
## CDK2        -2.399e-05  9.092e-05  -0.264 0.793092
## CHD3         1.162e-05  1.203e-05   0.966 0.339305
## PRKCD        1.961e-05  1.812e-05   1.082 0.284890
## JAK2        -1.604e-05  3.968e-05  -0.404 0.687828
## MAPK14       1.570e-04  5.441e-05   2.885 0.005946 **
## TLE1         8.721e-05  3.557e-05   2.452 0.018074 *
## XRCC6        2.410e-05  1.266e-05   1.904 0.063216 .
## CBL         -3.620e-05  6.176e-05  -0.586 0.560663
## INSR        -1.951e-06  9.954e-06  -0.196 0.845501
## MYC         -2.241e-07  8.416e-06  -0.027 0.978868
## PTN          2.819e-06  3.750e-06   0.752 0.456123
## ZBTB16       3.654e-06  7.860e-06   0.465 0.644213
## HCK         -1.153e-04  5.237e-05  -2.201 0.032811 *
## KAT5         4.999e-04  3.468e-04   1.441 0.156225
## VCL          1.020e-05  6.739e-06   1.514 0.136784
## CAV1         1.211e-05  4.017e-06   3.014 0.004185 **
## RAF1         1.742e-04  1.079e-04   1.615 0.113198
## STAT1       -7.709e-06  3.109e-06  -2.479 0.016887 *
## COPS6       -2.084e-05  3.006e-05  -0.693 0.491598
## KAT2B        3.874e-04  8.213e-05   4.717 2.26e-05 ***
## PTPN6        2.318e-06  5.854e-05   0.040 0.968593
## SKIL        -4.703e-06  1.177e-05  -0.400 0.691365
## SRF          1.284e-05  2.536e-05   0.506 0.614991
## APP         -2.934e-06  1.680e-06  -1.746 0.087441 .
## MAPK8        1.280e-04  4.755e-05   2.692 0.009867 **
## PXN         -5.954e-05  4.186e-05  -1.422 0.161652
## NCOR1       -4.310e-05  1.738e-05  -2.479 0.016886 *
## PDPK1        1.074e-04  5.878e-05   1.828 0.074107 .
## PIN1         1.446e-04  7.419e-05   1.950 0.057344 .
## TRAF6        2.267e-04  1.464e-04   1.549 0.128330
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1107 on 46 degrees of freedom
## Multiple R-squared:  0.9847, Adjusted R-squared:  0.9513
## F-statistic: 29.55 on 100 and 46 DF,  p-value: < 2.2e-16
```

**Training a Linear Regression Model:**

- It uses **`trainControl`** from the **`caret`** package to set up 10-fold cross-validation for model training.
- The **`train`** function fits a linear regression model (**`method = "lm"`**) on the **`train.data`** dataset, using **`sample_type`** as the response variable and all other variables as predictors.
- The model performance and details are printed using **`print(model)`**.

```
train.control <- trainControl(method = "cv", number = 10)
model <- train(sample_type ~ .,
               data = train.data,
               method = "lm",
               trControl = train.control)
print(model)
```

```
## Linear Regression
##
## 147 samples
## 100 predictors
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 132, 133, 133, 132, 132, 132, ...
## Resampling results:
##
##   RMSE       Rsquared   MAE
##   0.9961332  0.3216684  0.6099457
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

### Decision Tree Model

### Building a Decision Tree Model (rpart):

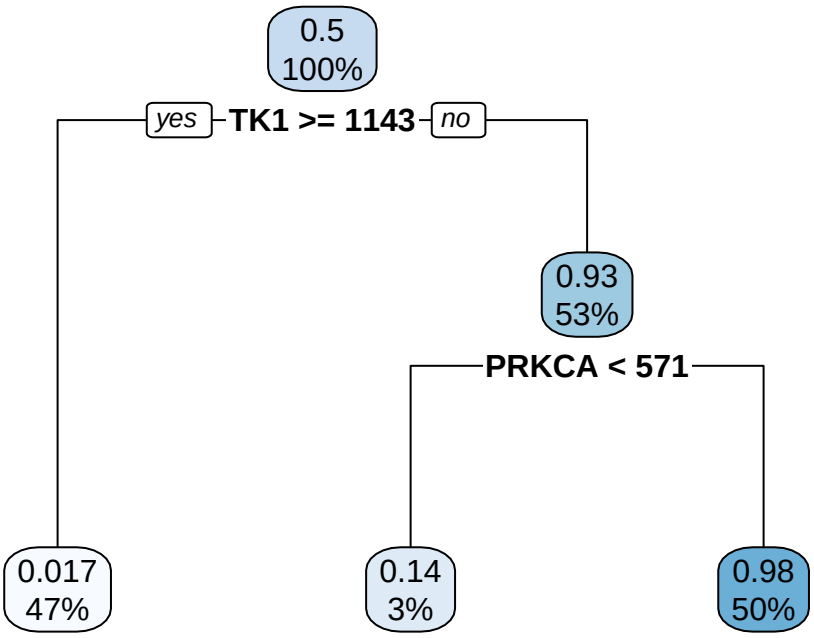
- It fits a decision tree model (**`method = "anova"`**) using the **`rpart`** function.
- The model is trained on the subset of data composed of **`Predictores`** and **`sample_type`** from **`final_data_filtradoe`**.
- The decision tree structure and details are printed using **`print(fit)`**.
- The **`rpart.plot::rpart.plot(fit)`** command creates a graphical visualization of the decision tree.

```
fit <- rpart(sample_type ~ .,
             method = "anova",
             data = final_data_filtradoe[, c(Predictores, "sample_type")],
             control = rpart.control(xval = 10))

print(fit)
```

```
## n= 246
##
## node), split, n, deviance, yval
##       * denotes terminal node
##
## 1) root 246 61.5000000 0.50000000
##   2) TK1>=1142.5 116  1.9655170 0.01724138 *
##   3) TK1< 1142.5 130  8.3769230 0.93076920
##     6) PRKCA< 570.5 7  0.8571429 0.14285710 *
##     7) PRKCA>=570.5 123  2.9268290 0.97560980 *
```

```
rpart.plot::rpart.plot(fit)
```




**First Random Forest:**

- It fits a random forest model (**randomForest**) using **sample_type** as the target variable and all other variables as predictors from the dataset **final_data_filtradoe**.
- A prediction is made on the test data (**test.data**) using the fitted model (**fit.rf**).
- A contingency table is created to compare the actual values of **sample_type** with the generated predictions (**prediction.rf**).

**Second Random Forest:**

- Another random forest model is fitted, similar to the first one, using the same variables from the dataset **final_data_filtradoe**.
- Predictions are made on the test data (**test.data**) using this new model.
- A data frame (**output**) is created containing columns "Actual" and "Predicted," representing actual values and predictions, respectively.
- The Root Mean Squared Error (RMSE) is computed to evaluate the prediction accuracy, and a summary displaying the initial rows of this data frame is printed

```
fit.rf <- randomForest(sample_type ~ .,
                       data = final_data_filtradoe[, c(Predictores, "sample_type")])


prediction.rf <- predict(fit.rf, test.data)
output <- data.frame(Actual = test.data$sample_type, Predicted = prediction.rf)
RMSE = sqrt(sum((output$Actual - output$Predicted)^2) / nrow(output))
```

```r
print(head(output))
```

```
##   Actual   Predicted
## 1      0 0.525066667
## 2      0 0.004000000
## 3      0 0.002000000
## 4      0 0.011066667
## 5      0 0.049966667
## 6      0 0.009733333
```

### Support Vector Machines

**Data Preparation:**

- Converts the response variable (**sample_type**) into a factor, which is the required format for classification models in R.

```r
final_data_filtradoe$sample_type <- as.factor(final_data_filtradoe$sample_type)
```

**Data Splitting:**

- Splits the data into training (**train.data**) and test (**test.data**) sets using a random sampling of 70% of the data for training and the remaining 30% for testing.

```r
set.seed(123)
sample.index <- sample(1:nrow(final_data_filtradoe), nrow(final_data_filtradoe) * 0.7, replace = FA
train.data <- final_data_filtradoe[sample.index, c(Predictores, "sample_type"), drop = FALSE]
test.data <- final_data_filtradoe[-sample.index, c(Predictores, "sample_type"), drop = FALSE]
```

**Hyperparameter Tuning - Linear SVM:**

- Uses the **tune** function from the **e1071** package to search for the best hyperparameters for a linear SVM model.
- Specifies a list of cost values to explore and find the best linear SVM model.

```r
tune.out <- tune(svm,
                 sample_type ~ .,
                 data = train.data,
                 kernel = "linear",
                 ranges = list(cost = c(0.001, 0.01, 0.1, 1, 5, 10, 100)))
```

**Configuration of Linear SVM Model:**

- Configures a linear SVM model (**svm_model**) using the best hyperparameters found during the hyperparameter search.

```r
bestmod <- tune.out$best.model
svm_model <- svm(sample_type ~ ., data = train.data, kernel = "linear", cost = bestmod[["cost"]])
```

**Prediction on Test Set:**

- Makes predictions (**svm_predict**) on the test dataset (**test.data**) using the trained SVM model (**svm_model**).

```r
svm_predict <- predict(svm_model, newdata = test.data)
```

**Model Evaluation:**

- Evaluates the model's performance on the test set by generating a confusion matrix and relevant classification metrics using the **confusionMatrix** function from the **caret** package. This assessment

helps understand how well the model predicts the classes in the test data (**test.data$sample_type**) compared to the predicted classes (**svm_predict**).

```
confusionMatrix(data = svm_predict, reference = test.data$sample_type)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 30  0
##          1  1 43
##
##                Accuracy : 0.9865
##                  95% CI : (0.927, 0.9997)
##     No Information Rate : 0.5811
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9721
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.9677
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 0.9773
##              Prevalence : 0.4189
##          Detection Rate : 0.4054
##    Detection Prevalence : 0.4054
##       Balanced Accuracy : 0.9839
##
##        'Positive' Class : 0
##
```

**Hyperparameter Tuning - Radial SVM:**

- Conducts a search for optimal hyperparameters for a radial SVM model using the **tune** function from the **e1071** package. It explores various cost values (**cost**) specified in the **ranges** parameter.

```
tune.out <- tune(svm,
                 sample_type ~ .,
                 data = train.data,
                 kernel = "radial",
                 ranges = list(cost = c(0.001, 0.01, 0.1, 1, 5, 10, 100)))
```

**Optimal Model Configuration:**

- Retrieves the best model (**svm_model**) obtained from the hyperparameter tuning based on the lowest error metrics, using **bestmod** from the **tune.out** results.

```
bestmod <- tune.out$best.model

svm_model <- svm(sample_type ~ ., data = train.data, kernel = "radial", cost = bestmod[["cost"]])
```

**Prediction and Evaluation with Tuned Model:**

- Applies the optimized SVM model (**svm_model**) on the test set (**test.data**) to generate new predictions (**svm_predict**).

- Evaluates the performance of the tuned SVM model on the test data using the **confusionMatrix** function from the **caret** package, calculating the confusion matrix and related classification metrics.

```
svm_predict <- predict(svm_model, newdata = test.data)
confusionMatrix(data = svm_predict, reference = test.data$sample_type)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 30  1
##          1  1 42
##
##                Accuracy : 0.973
##                  95% CI : (0.9058, 0.9967)
##     No Information Rate : 0.5811
##     P-Value [Acc > NIR] : 5.216e-15
##
##                   Kappa : 0.9445
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.9677
##             Specificity : 0.9767
##          Pos Pred Value : 0.9677
##          Neg Pred Value : 0.9767
##              Prevalence : 0.4189
##          Detection Rate : 0.4054
##    Detection Prevalence : 0.4189
##       Balanced Accuracy : 0.9722
##
##        'Positive' Class : 0
##
```

## Results and Discussion

he provided code encompasses the implementation of various supervised learning models, following a detailed process of data preprocessing and algorithm training. Starting with the loading of datasets from the DynamicCancerDriverKM package, they are merged and prepared for analysis. A crucial step involves removing columns with low values, ensuring the relevance of information and reducing noise in the final dataset.

A deeper dive into Protein-Protein Interaction (PPI) processing involves analyzing interaction frequencies and organizing information to better understand biological patterns relevant to cancer study. Additionally, column name modifications are made to facilitate data interpretation, and relevant genes are filtered to focus the analysis on the most pertinent genetic information.

The implementation of models is broken down into various approaches, from using K-Nearest Neighbors (KNN) to linear regression models, decision trees, random forests, and Support Vector Machines (SVM). Each model is tailored to the prepared data, trained, and its performance evaluated using methods such as cross-validation, prediction, and generating confusion matrices.

However, notable challenges emerged in the process. Handling incomplete or missing data required specific strategies to ensure the integrity of the datasets. Selecting relevant features for each model and optimizing hyperparameters were critical aspects to improve prediction accuracy. Furthermore, evaluating the models' performance using appropriate metrics was essential to understand the effectiveness of each supervised

learning approach.

## Conclusion

The findings from the implemented supervised learning models provide valuable insights for a data scientist or machine learning engineer.

Firstly, the techniques showcased the importance of meticulous data preprocessing. Filtering columns, handling missing data, and processing domain-specific information, like Protein-Protein Interactions (PPI), were pivotal. These steps highlighted the significance of understanding the data domain and tailoring preprocessing techniques accordingly.

The model implementations demonstrated the versatility of supervised learning algorithms—KNN, linear regression, decision trees, random forests, and SVM—in handling diverse data types and tasks. Each model had its strengths and weaknesses, showcasing the need to choose models tailored to specific data characteristics and problem types.

The challenges encountered, such as handling missing data, feature selection, and hyperparameter optimization, underscored the critical thinking and problem-solving skills required in the role of a data scientist or machine learning engineer. These challenges often demand creativity, domain knowledge, and a deep understanding of model behavior to make informed decisions and improve model performance.

Additionally, the emphasis on model evaluation using various performance metrics highlighted the importance of robust evaluation methodologies. Understanding metrics like accuracy, precision, recall, and area under the ROC curve aids in effectively gauging model effectiveness and making informed decisions about model deployment.

Overall, the findings stress the multidimensional skill set needed in these roles—from technical expertise in algorithm implementation and data preprocessing to domain knowledge and a strong grasp of evaluation metrics. The ability to navigate challenges and make data-driven decisions based on model performance insights is crucial for success in these positions.

## References


- Guo, G., Wang, H., Bell, D., Bi, Y., & Greer, K. (2003). KNN model-based approach in classification. In *On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE: OTM Confederated International Conferences, CoopIS, DOA, and ODBASE 2003, Catania, Sicily, Italy, November 3-7, 2003. Proceedings* (pp. 986-996). Springer Berlin Heidelberg.
- Zhang, H., Berg, A. C., Maire, M., & Malik, J. (2006, June). SVM-KNN: Discriminative nearest neighbor classification for visual category recognition. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)* (Vol. 2, pp. 2126-2136). IEEE.
- Hoffmann, J. P. (2021). *Linear regression models: applications in R*. Crc Press.
- Maulud, D., & Abdulazeez, A. M. (2020). A review on linear regression comprehensive in machine learning. *Journal of Applied Science and Technology Trends*, *1*(4), 140-147.
- Rakhmawan, S. A., Omar, M. H., Riaz, M., & Abbas, N. (2023). Hotelling T2 control chart for detecting changes in mortality models based on machine-learning decision tree. *Mathematics*, *11*(3), 566.
- Cervantes, J., Garcia-Lamont, F., Rodríguez-Mazahua, L., & Lopez, A. (2020). A comprehensive survey on support vector machine classification: Applications, challenges and trends. *Neurocomputing*, *408*, 189-215.