A Brief Introduction to Feature Engineering

Cheng Peng

West Chester University

Contents

1 Introduction

Feature engineering in machine learning is the process of selecting, transforming, and creating input variables (features) that can improve the performance of predictive models. It is a critical step in the machine learning pipeline, as the quality of the features used directly impacts the accuracy and generalization ability of the model. It involves preparing and transforming raw data into features that better represent the underlying problem. This note covers four major types of feature engineering: feature transformation, feature selection, feature extraction, and feature creation, with illustrative examples in R.

The importance of feature engineering is obvious.

- **Improves Model Performance**: Better features lead to better predictive accuracy and robustness.
- **Simplifies Models**: By focusing on the most important features, models can be simpler and less prone to overfitting.
- **Handles Data Issues**: Correctly engineered features can mitigate issues like noise, missing values, and non-linear relationships.

2 Feature Transformation

Feature transformation is a crucial step in the data preprocessing pipeline. It involves modifying features to improve the performance of machine learning models, ensure compatibility with algorithms, or enhance interpretability. Common methods include normalization, scaling, encoding categorical variables, and dimensionality reduction. In this section, we explore these methods with illustrative examples in R.

2.1 Normalization and Standardization

Normalization rescales feature values to a range, typically [0, 1], whereas standardization scales features to have a mean of 0 and a standard deviation of 1. These methods are particularly useful for algorithms sensitive to feature magnitudes, such as k-NN or gradient descent-based models.

Example:

```r
# Sample data
set.seed(123)
data <- data.frame(
  feature1 = runif(10, 1, 100),
  feature2 = rnorm(10, 50, 10)
)

# Normalization
normalize <- function(x) {
  return((x - min(x)) / (max(x) - min(x)))
}
data$feature1_normalized <- normalize(data$feature1)

# Standardization
standardize <- function(x) {
  return((x - mean(x)) / sd(x))
}
data$feature2_standardized <- standardize(data$feature2)

print(data)
```

```
    feature1 feature2 feature1_normalized feature2_standardized
1  29.470174 67.15065           0.2704415            1.75073186
2  79.042208 54.60916           0.8299695            0.36393540
3  41.488715 37.34939           0.4060968           -1.54459366
4  88.418723 43.13147           0.9358038           -0.90522985
5  94.106261 45.54338           1.0000000           -0.63852891
```

```
6   5.510093 62.24082           0.0000000           1.20781880
7  53.282443 53.59814           0.5392146           0.25213972
8  89.349485 54.00771           0.9463095           0.29742931
9  55.592066 51.10683           0.5652837          -0.02334127
10 46.204859 44.44159           0.4593287          -0.76036141
```

2.2 Encoding Categorical Variables

Categorical variables need to be converted into numerical formats for most machine learning algorithms. Common techniques include one-hot encoding and label encoding.

Example:

```r
# Sample data
categories <- data.frame(
  ID = 1:4,
  color = c("red", "blue", "green", "blue")
)

# One-hot encoding using the `model.matrix` function
encoded_data <- model.matrix(~ color - 1, data = categories)
print(encoded_data)
```

```
  colorblue colorgreen colorred
1         0          0        1
2         1          0        0
3         0          1        0
4         1          0        0
attr(,"assign")
[1] 1 1 1
attr(,"contrasts")
attr(,"contrasts")$color
[1] "contr.treatment"
```

```r
# Label encoding
categories$color_label <- as.numeric(factor(categories$color))
print(categories)
```

```
  ID color color_label
1  1   red           3
2  2  blue           1
3  3 green           2
4  4  blue           1
```

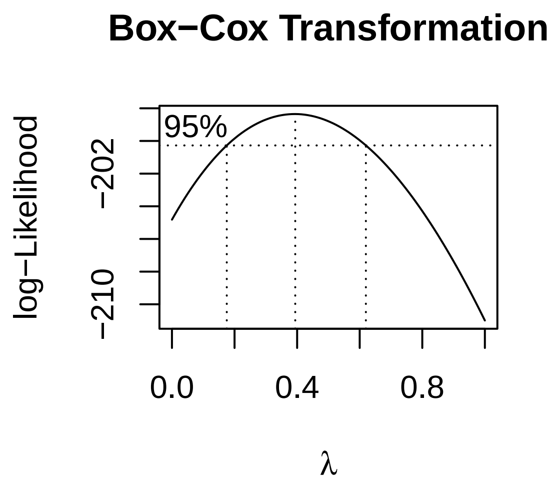
2.3 Box-Cox Transformation

The Box-Cox transformation is a statistical technique used to stabilize variance and make data conform more closely to a normal distribution. It is particularly useful when the assumptions of normality and homoscedasticity (constant variance) are violated. The transformation is defined by the following formula:

$$y(\lambda) = \begin{cases} \frac{y^\lambda - 1}{\lambda}, & \text{if } \lambda \neq 0 \\ \log(y), & \text{if } \lambda = 0 \end{cases}$$

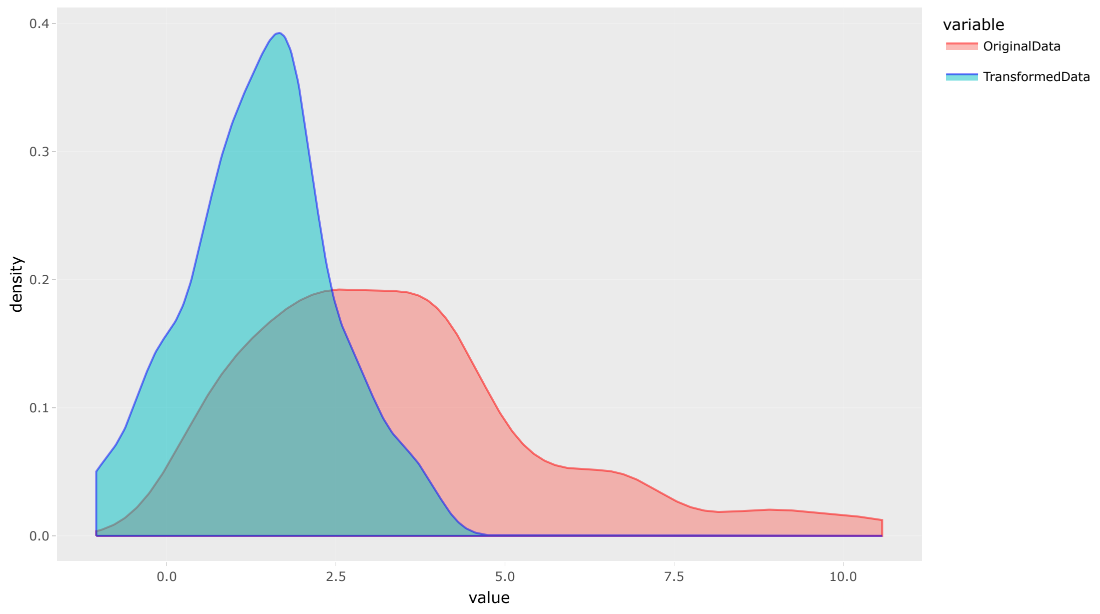
Here, y is the data to be transformed, and λ is a parameter that determines the power of the transformation.

```r
# Load necessary library
library(MASS)
library(ggplot2)
library(reshape2)
library(plotly)
library(tidyverse)
###
# Generate a skewed dataset
set.seed(123)
data <- rgamma(100, shape = 2, scale = 2)

# Perform the Box-Cox transformation
boxcox_result <- boxcox(lm(data ~ 1), lambda = seq(0, 1, by = 0.1))
title("Box-Cox Transformation")
```



```r
# Find the optimal lambda
optimal_lambda <- boxcox_result$x[which.max(boxcox_result$y)]
#cat("Optimal lambda:", optimal_lambda, "\n")
```

```r
# Apply the transformation with the optimal lambda
transformed_data <- if (optimal_lambda == 0) {
  log(data)
} else {
  (data^optimal_lambda - 1) / optimal_lambda
}

pooledData <- data.frame(OriginalData=data, TransformedData=transformed_data)
## Reshaoe the data - stacking the values and assign variable ID.
LongFormatData<- melt(pooledData)
grp.col = c("navy", "darkred")
names(grp.col) = levels(LongFormatData$variable)
##
ggp = ggplot(LongFormatData,aes(x=value, fill=variable, color = variable)) +
      geom_density(alpha=0.5) +
      scale_color_manual(values = c("red", "blue")) +
      theme(legend.title = element_text(face = "bold"),
            legend.position=c(.9,.75))
#
ggplotly(ggp)   # ggplotly's legend is always on topright by default!
```



Log transformation, a special Box-Cox transformation with $\lambda = 0$, is applied to skewed data to reduce the impact of outliers and approximate a normal distribution.

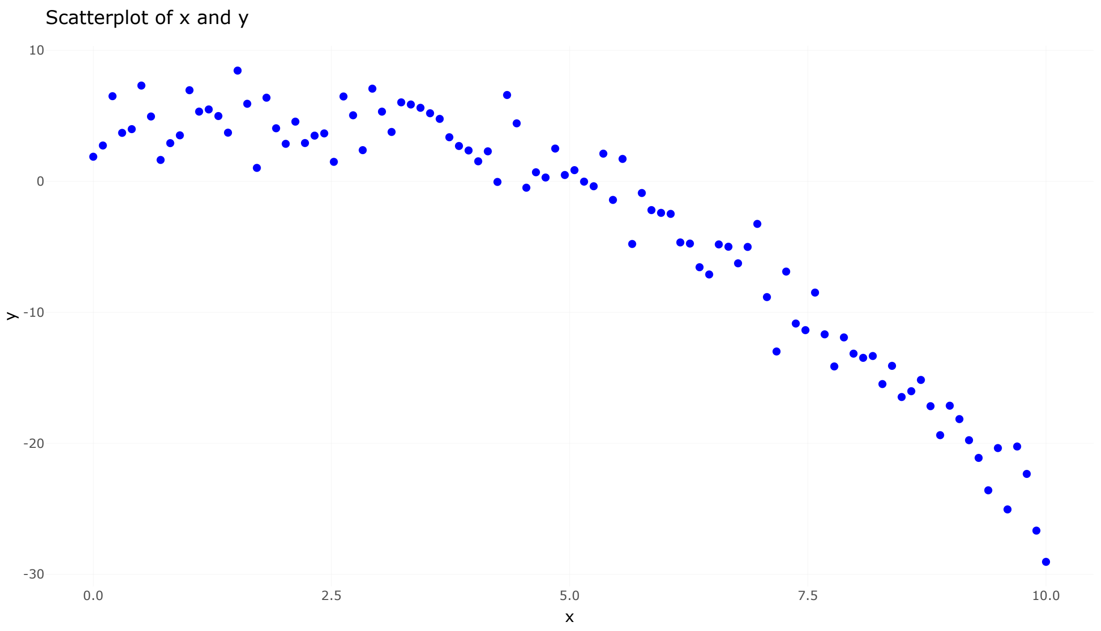
```r
# Sample data
set.seed(123)
skewed_data <- data.frame(
  value = c(1, 2, 3, 10, 100, 1000)
)

# Log transformation
skewed_data$log_value <- log(skewed_data$value)
print(skewed_data)
```

```
  value log_value
1     1 0.0000000
2     2 0.6931472
3     3 1.0986123
4    10 2.3025851
5   100 4.6051702
6  1000 6.9077553
```

2.4 Polynomial Features

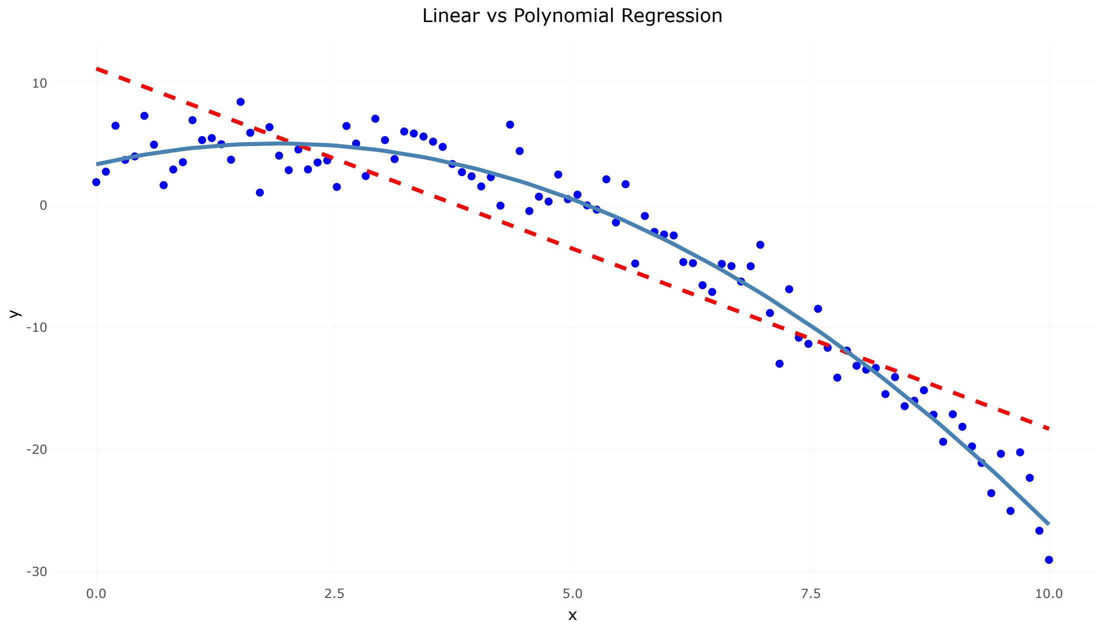
Polynomial transformations allow you to model non-linear relationships by adding polynomial terms (e.g., $x^2, x^3, \cdots$ to the existing dataset. Here's an example in R to demonstrate how to use polynomial features for regression.

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

# Simulate some data
set.seed(123)
x <- seq(0, 10, length.out = 100)
y <- 3 + 2 * x - 0.5 * x^2 + rnorm(100, sd = 2)  # Non-linear relationship

# Create a data frame
data <- data.frame(x, y)

# Plot the data
rawplot = ggplot(data, aes(x, y)) +
  geom_point(color = "blue") +
  ggtitle("Scatterplot of x and y") +
  theme_minimal()
###
ggplotly(rawplot)
```



```r
# Fit a linear regression model
model_linear <- lm(y ~ x, data = data)
```

```r
# Fit a polynomial regression model (degree 2)
model_poly <- lm(y ~ poly(x, 2, raw = TRUE), data = data)

# Summarize the models
#summary(model_linear)
#summary(model_poly)

# Predict and plot the results
data$y_pred_linear <- predict(model_linear, newdata = data)
data$y_pred_poly <- predict(model_poly, newdata = data)

ployplot = ggplot(data, aes(x, y)) +
               geom_point(color = "blue") +
               geom_line(aes(y = y_pred_linear),
                         color = "red",
                         linetype = "dashed",
                         size = 1) +
              geom_line(aes(y = y_pred_poly),
                        color = "steelblue",
                        size = 1) +
              ggtitle("Linear vs Polynomial Regression") +
              theme_minimal() +
              theme(plot.title = element_text(hjust = 0.5))
##
ggplotly(ployplot )
```

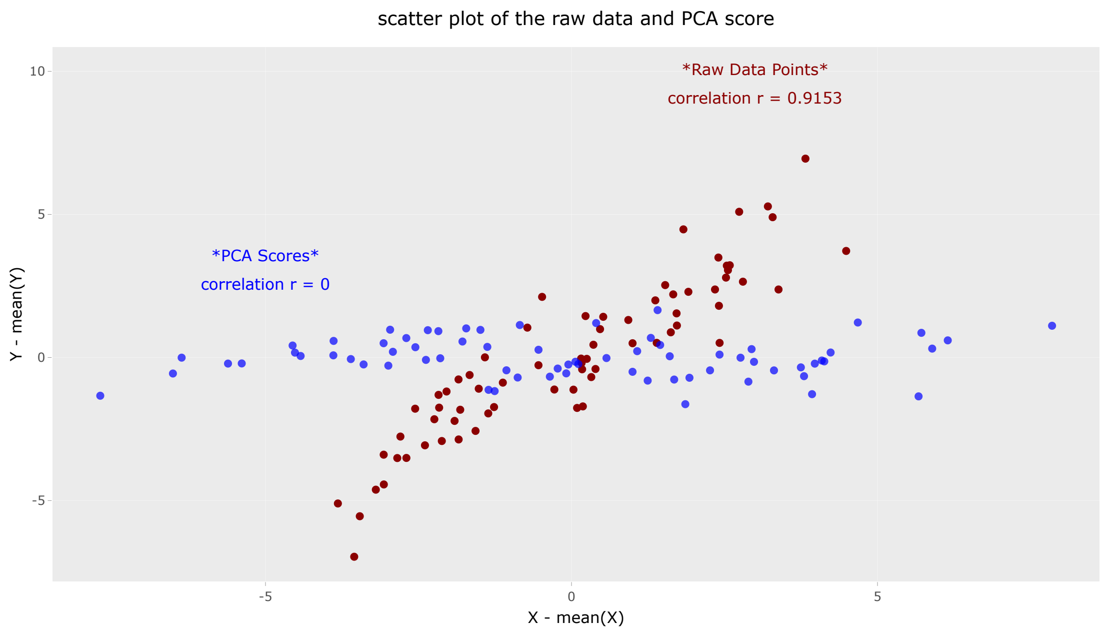

Linear vs Polynomial Regression
10
0
-10
-20
-30
y
0.0
2.5
5.0
7.5
10.0
x

2.5 Principal Component Analysis (PCA)

PCA is a dimensionality reduction technique that transforms a set of highly correlated features into principal components, which are linear combinations of the original features and are uncorrelated. These new principal components are used in the subsequent modeling as **transformed features** as usual. That is, PCA is a solution to multicollinearity issues in regression analysis.

The following illustrative example using a simulated data set.

```r
library(glue)
library(ggtext)
# Sample data
set.seed(123)
## Create a data set to illutrate PCA
X = seq(2,9, length = 70) + rnorm(70,0,0.5)
Y = 1 + 1.2*X + rnorm(70, 0.5,1.2)
pca.data <- data.frame(X=X, Y=Y)
# PCA transformation using R function `prcomp`
# Caution: data is strongly encouraged to be centered in PCA. Otherwise, the
#          PCA transformation must include the intercept term!
#
pca_result <- prcomp(pca.data, center = TRUE, scale. = FALSE)
# Transformed data is the PCA scores: extracting PCs with pca_result$x
pca_transformed <- as.data.frame(pca_result$x)
##
## The next few line of code calculate the above transformed data: pca_result$x
##
# The coefficient of transformation - called factor loading that is calculated
# based on the assumption that the data is centered.
# PC1 = a*x + b*y
# PC2 = c*x + d*y
# The following extract matrix: 1st column: (a,b), 2nd column: (c,d)
load.coef = pca_result$rotation
# The above
PC1 = load.coef[1,1]*(X-mean(X)) + load.coef[2,1]*(Y - mean(Y))
PC2 = load.coef[1,2]*(X-mean(X)) + load.coef[2,2]*(Y - mean(Y))
## conform the extract PCs and the calculated PCs
#cbind(pca_result$x, PC1, PC2,pca.data$x, pca.data$y)
##
## Create a data frame for interactive visualization
pca.viz.dat = as.data.frame(cbind(pca.data, pca_result$x))
##
pca.viz.dat = pca.viz.dat %>%
## create tooltip to show both raw data and PCA scores
mutate(text = paste(
    "X-mean(X): ", round(X-mean(X), 4),
    "\nY -mean(Y): ", round(Y-mean(Y),4),
    "\nPC1: ", round(PC1,4),
    "\nPC2: ", round(PC2,4), sep="")
  )
##
ggp = pca.viz.dat %>%
  ggplot(aes(x=X-mean(X), y=Y-mean(Y), text=text)) +
    geom_point(alpha=0.7,  show.legend = FALSE) +
    geom_point(aes(x=X-mean(X), y=Y-mean(Y)), col = "darkred") +
    geom_point(aes(x = PC1, y = PC2), col = "blue", alpha=0.7) +
    ##
    annotate(geom = "text", x = 3, y = 10,
             label = "*Raw Data Points*",
             col = "darkred",
             fontface =2,
             hjust = "center") +
    annotate(geom = "text", x = 3, y = 9,
             label = paste("correlation r =", round(cor(X,Y),4)),
             col = "darkred",
             fontface =2,
             hjust = "center") +
    annotate(geom = "text", x = -5, y = 3.5,
             label = "*PCA Scores*",
             col = "blue",
             hjust = "center") +
    annotate(geom = "text", x = -5, y = 2.5,
             label = paste("correlation r =", round(cor(PC1,PC2),4)),
             col = "blue",
             fontface =2,
             hjust = "center") +
    ## add title
    ggtitle("scatter plot of the raw data and PCA score") +
    theme(legend.position="none",
          plot.title = element_text(hjust = 0.5))

# turn ggplot interactive with plotly
ggplotly(ggp, tooltip="text")
```



We intentionally plot the original data points and the transformed data points based on PCA scores on the same graph to better view the consequence of the PCA transformation. We can see that the two original features are high correlated and the transformed features (i.e., the two principal components) are not significantly correlated. On the other hand, the total variance of the original two variables is equal to the total variance of the two PCs. The manual calculation of the variance of the original and transformed feature variables.

```
list(var.X = var(X-mean(X)),
     var.Y = var(Y-mean(Y)),
     var.PC1 = var(PC1),
     var.PC2 = var(PC2),
     varXY = var(X-mean(X))+var(Y-mean(Y)),
     varPC12 =var(PC1) + var(PC2))
```

```
$var.X
[1] 4.563593

$var.Y
[1] 7.73993

$var.PC1
[1] 11.81849

$var.PC2
[1] 0.4850331

$varXY
[1] 12.30352

$varPC12
[1] 12.30352
```

Furthermore, we can also calculate the proportion of variance captured in each of the original feature variable and the transformed feature variables (i.e., the two principal components) in the following.

```
PropVarX = var(X-mean(X))/(var(X-mean(X))+var(Y-mean(Y)))
PropVarY = var(Y-mean(Y))/(var(X-mean(X))+var(Y-mean(Y)))
PropVarPC1 = var(PC1)/(var(PC1) + var(PC2))
PropVarPC2 = var(PC2)/(var(PC1) + var(PC2))
list(PropVarX=PropVarX,
     PropVarY=PropVarY,
     PropVarPC1=PropVarPC1,
     PropVarPC2=PropVarPC2,
     PCImportance = summary(pca_result))
```

```
$PropVarX
[1] 0.3709176

$PropVarY
[1] 0.6290824

$PropVarPC1
[1] 0.9605777

$PropVarPC2
[1] 0.03942229

$PCImportance
Importance of components:
```

```
                          PC1     PC2
Standard deviation     3.4378 0.69644
Proportion of Variance 0.9606 0.03942
Cumulative Proportion  0.9606 1.00000
```

That is, the first principle component PC1 captures more than 96% of the total variance and PC2 captures less than 4 of the total variance. This implies that the PC

In practical applications, we should use the transformed PCs and exclude the corresponding numerical features that were used in the PCA.

2.6 Feature Binning and Grouping

Feature binning (also known as discretization) and regrouping are common preprocessing steps in data analysis, especially for converting continuous variables into categorical variables or simplifying categories for better interpretability. Next, we use simple examples in R to illustrate feature binning and grouping.

2.6.1 Feature Binning

Feature binning involves dividing a continuous variable into discrete bins.

Example on Equal-width Binning

```r
# Sample data
data <- data.frame(Age = c(23, 25, 31, 35, 45, 51, 62, 71))

# Equal-width binning into 3 bins
data$Age_Binned <- cut(data$Age,
                       breaks = 3,
                       labels = c("Young", "Middle-aged", "Senior"),
                       include.lowest = TRUE)

print(data)
```

```
  Age  Age_Binned
1  23       Young
2  25       Young
3  31       Young
4  35       Young
5  45 Middle-aged
6  51 Middle-aged
7  62      Senior
8  71      Senior
```

`cut()` divides the range of Age into 3 equal-width intervals. `labels` are provided for the bins. `include.lowest = TRUE` ensures the smallest value is included in the first bin.

Example on Custom Binning

```r
# Custom binning with defined breakpoints
breakpoints <- c(0, 30, 50, 100)
labels <- c("Young", "Middle-aged", "Senior")

data$Age_Binned_Custom <- cut(data$Age,
                              breaks = breakpoints,
                              labels = labels,
                              include.lowest = TRUE)
```

```
print(data)
```

```
  Age  Age_Binned Age_Binned_Custom
1  23       Young             Young
2  25       Young             Young
3  31       Young       Middle-aged
4  35       Young       Middle-aged
5  45 Middle-aged       Middle-aged
6  51 Middle-aged            Senior
7  62      Senior            Senior
8  71      Senior            Senior
```

The `breakpoints` (`c(0, 30, 50, 100)`) define custom intervals. `Labels` (`c("Young", "Middle-aged", "Senior")`) assign meaningful names to each bin.

2.6.2 Feature Grouping

Grouping simplifies the levels of a categorical variable by combining categories **in a meaningful way**. This is a must-have step in processing categorical variables with sparse categories. The following R code snippets illustrate how to use built-in R functions to group (sparse) categories.

Regrouping Categories

```
# Sample data
data <- data.frame(Product_Category = c("A", "B", "C", "D", "E", "F"))

# Regroup categories into broader groups
data$Product_Group <- recode(data$Product_Category,
                             "A" = "Group1",
                             "B" = "Group1",
                             "C" = "Group2",
                             "D" = "Group2",
                             "E" = "Group3",
                             "F" = "Group3")

print(data)
```

```
  Product_Category Product_Group
1                A        Group1
2                B        Group1
3                C        Group2
4                D        Group2
5                E        Group3
6                F        Group3
```

`recode()` (from the dplyr package) maps specific categories to broader groups.

Regrouping Using Base R

```
## Using a named vector for mapping
mapping <- c("A" = "Group1", "B" = "Group1", "C" = "Group2",
             "D" = "Group2", "E" = "Group3", "F" = "Group3")
##
data$Product_Group <- mapping[data$Product_Category]
##
print(data)
```

```
  Product_Category Product_Group
1                A        Group1
2                B        Group1
3                C        Group2
4                D        Group2
5                E        Group3
6                F        Group3
```

A named vector defines the mapping of old to new categories. Categories in `Product_Category` are replaced based on the mapping.

2.7 Summary

Feature transformation is an essential component of preparing data for machine learning. Techniques such as normalization, encoding, and dimensionality reduction address different challenges, ensuring that data is ready for modeling. In R, packages like dplyr, caret, and data.table offer robust tools for implementing these methods efficiently. Selecting the right transformation depends on the data set and the requirements of the machine learning algorithm

3 Feature Selection

Feature selection is a critical step in the development of predictive models, aiming to identify and retain the most relevant features while discarding irrelevant or redundant ones. This process helps reduce dimensionality, improves model interpretability, and can enhance model performance by preventing overfitting. This section explores various feature selection methods and demonstrates their implementation with illustrative examples in R.

Feature selection methods can broadly be categorized into three groups: filter methods, wrapper methods, and embedded methods. Each has its strengths and is suited to different scenarios.

3.1 Filter Methods

Filter methods assess the relevance of features by examining their intrinsic properties, such as statistical measures, without involving a predictive model. These methods are computationally efficient and are often used as a pre-processing step.

Example: Suppose we have a data set with continuous features, and we aim to select features most correlated with the target variable. The `cor()` function in R can calculate correlations:

```r
# Load necessary library
library(datasets)
# Example data set: mtcars
# Target variable: mpg (miles per gallon)
correlations <- abs(cor(mtcars[, -1], mtcars$mpg))
#sorted_correlations <- sort(abs(correlations), decreasing = TRUE)
selected_features <- rownames(correlations)[correlations[,1] > 0.5]
list(Correlation = correlations, Selected.Feature = selected_features)
```

```
$Correlation
          [,1]
cyl  0.8521620
disp 0.8475514
hp   0.7761684
drat 0.6811719
wt   0.8676594
qsec 0.4186840
```

```
vs   0.6640389
am   0.5998324
gear 0.4802848
carb 0.5509251

$Selected.Feature
[1] "cyl"  "disp" "hp"   "drat" "wt"   "vs"   "am"   "carb"
```

This script calculates the absolute correlations of all features with the target and selects features with a correlation greater than 0.5.

Caution: *This feature selection example using correlation coefficients is only for illustrative purpose. In fact, the correlation coefficient is rarely used in feature selection.*

3.2 Wrapper Methods

Wrapper methods use a predictive model to evaluate subsets of features and select the combination that yields the best performance. These methods are computationally intensive but can achieve higher accuracy since they account for feature interactions.

Using the `caret package`, we can apply recursive feature elimination (RFE):

```r
# Load necessary libraries
# library(caret)
# Prepare data set
data(mtcars)
control <- rfeControl(functions = rfFuncs, method = "cv", number = 10)

# Apply RFE
results <- rfe(mtcars[, -1], mtcars$mpg, sizes = c(1:5), rfeControl = control)

# Selected features
print(predictors(results))
```

```
[1] "disp" "hp"   "wt"   "cyl"
```

This script uses a random forest model to evaluate subsets of features and selects the optimal set based on cross-validation performance.

3.3 Embedded Methods

Embedded methods perform feature selection during the model training process. These methods are model-specific and integrate feature selection into the optimization process, making them computationally efficient.

The `glmnet package` can be used for feature selection via LASSO (Least Absolute Shrinkage and Selection Operator):

```r
# Load necessary libraries
# library(glmnet)
# Prepare dataset
x <- as.matrix(mtcars[, -1])
y <- mtcars$mpg

# Apply LASSO
lasso_model <- cv.glmnet(x, y, alpha = 1)
best_lambda <- lasso_model$lambda.min
selected_features <- rownames(coef(lasso_model, s = best_lambda))[coef(lasso_model, s = best_lambda)[,1
print(selected_features)
```

```
[1] "(Intercept)" "cyl"         "hp"          "wt"
```

This script identifies the most important features by penalizing the regression coefficients, shrinking some to zero.

3.4 Conclusion

To conclude, we provide a comparison among the three methods of feature selection in the following table.

Method	Advantages	Disadvantages
Filter	Simple, fast, model-agnostic	Ignores feature interactions
Wrapper	Considers feature interactions	Computationally expensive
Embedded	Integrates feature selection with training	Model-specific, may require tuning

4 Feature Extraction

Feature extraction is a critical step in data analysis and machine learning pipelines. It involves transforming raw data into a format that is more suitable for model building. Effective feature extraction can improve the performance of machine learning models by reducing dimensionality, removing noise, and highlighting relevant patterns. In this essay, we explore various feature extraction methods, including Principal Component Analysis (PCA), clustering-based techniques, Local Outlier Factor (LOF), and others. Each method is discussed with illustrative examples and R code.

4.1 Principal Component Analysis (PCA)

We have introduced PCA as a feature transformation method because the transformed new variables are linear combinations of existing numerical features that are highly correlated.

PCA is a statistical technique used to reduce the dimensionality of a data set while preserving as much variability as possible. It transforms the data into a new coordinate system where the axes (principal components) represent directions of maximum variance.

4.1.1 Exploaring Iris Data

To illustrate the idea of dimension reduction, we use the well-known iris data set which has four numerical features characterizing the size of three different types of iris flowers.

```r
library(datasets)
library(psych)
data(iris)
pairs.panels(iris[,-5],gap=0, bg=c("red","blue","yellow")[iris$Species],
             pch=21)
```

The above scatter plot shows that `Petal.Length` and `Petal.Width`, `Sepal.Length` and `Sepal.Width`, `Sepal.Length` and `Petal.Width` are highly correlated.

4.1.2 Issues of Correlation in Regression

The essence of regression is to assess the relationship between variables, specifically focusing on how one or more independent variables (predictors) influence or relate to a dependent variable (outcome). Regression analysis is a cornerstone of statistical modeling and machine learning because it provides a framework to understand, predict, and quantify these relationships.

What is the implication of high correlation between two feature variables? For example, `Petal.Length` and `Petal.Width` are highly correlated (the Pearson correlation coefficient is 0.96). What are the potential issues

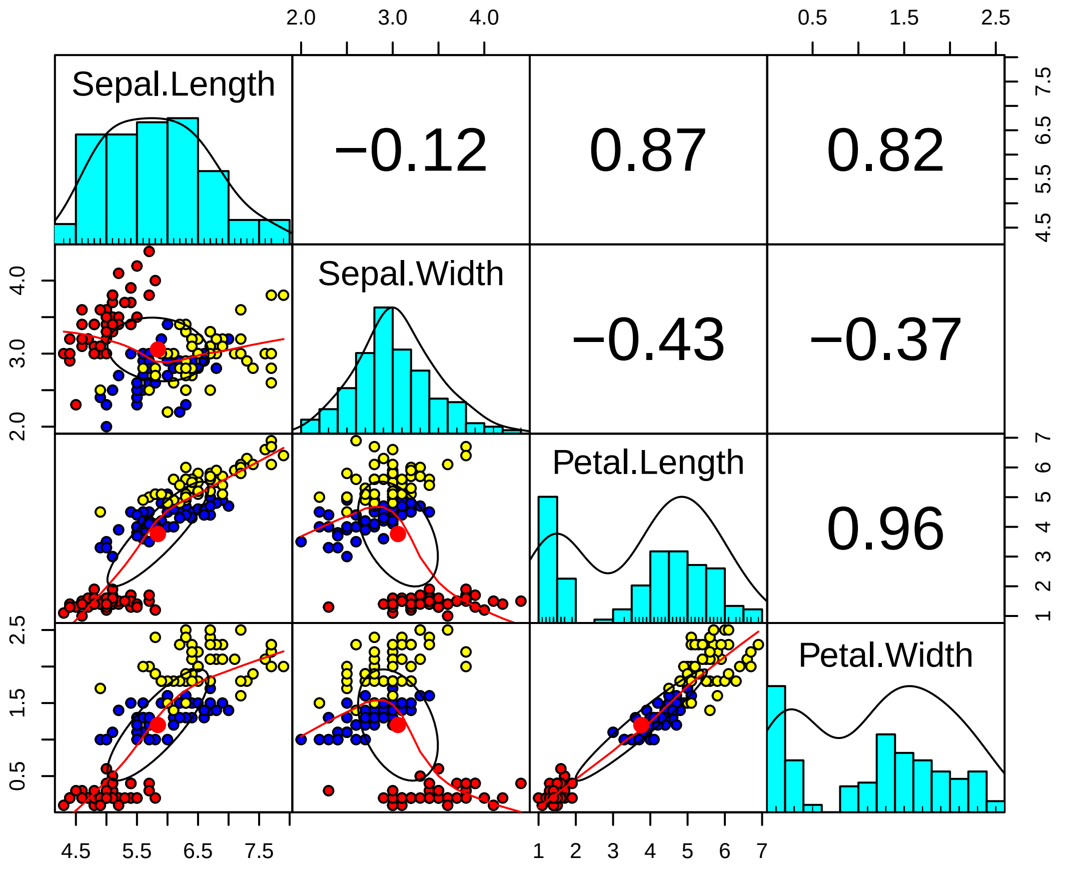


Figure 1: Correlation between numerical feature variables in the iris data set.

in regression modeling? Should we include both variables `Petal.Length` and `Petal.Width` in the model? If there are issues, how to resolve these issues? We try to answer the above questions in the

Potential Issues in Regression Modeling

- *Multicollinearity* arises when two or more predictors are highly correlated. *Consequences* include (1). regression coefficients become unstable and difficult to interpret. (2). standard errors of coefficients inflate, leading to wide confidence intervals. (3). variables may appear statistically insignificant even if they are important.
- *Overfitting*: High redundancy increases model complexity without adding meaningful information, leading to overfitting in small datasets.
- *Decreased Interpretability*: It becomes unclear how each predictor uniquely contributes to the dependent variable.

Should We Include Both Highly Correlated Variables?

- *Interpretation Focus*: Avoid including all highly correlated variables. Choose one or combine them into a single variable.
- *Prediction Focus*: Including all correlated variables might still yield good predictions, but simpler models often generalize better.

How to Resolve Issues

- *Remove Redundant Variables*: Use domain knowledge or statistical diagnostics (e.g., correlation matrix or VIF) to identify and drop redundant variables.
- *Combine Variables*: Use composite metrics, such as their average, sum, or ratio.
- *Principal Component Analysis (PCA)*: Transform correlated variables into uncorrelated principal components and use them in the model.
- *Regularization*: Use Ridge or Lasso regression to handle multicollinearity by penalizing or excluding redundant variables.

Illustrative Example Using PCA in R

Perform PCA on correlated predictors and use the first principal component.

```r
data(iris)
pca <- prcomp(iris[, c("Petal.Length", "Petal.Width", "Sepal.Length", "Sepal.Width")],
              scale = TRUE)
iris0=cbind(iris, pca$x)
iris0[1:6,]
```

```
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species       PC1
1          5.1         3.5          1.4         0.2  setosa -2.257141
2          4.9         3.0          1.4         0.2  setosa -2.074013
3          4.7         3.2          1.3         0.2  setosa -2.356335
4          4.6         3.1          1.5         0.2  setosa -2.291707
```

```
5          5.0         3.6          1.4         0.2  setosa -2.381863
6          5.4         3.9          1.7         0.4  setosa -2.068701
         PC2         PC3          PC4
1 -0.4784238  0.12727962 -0.024087508
2  0.6718827  0.23382552 -0.102662845
3  0.3407664 -0.04405390 -0.028282305
4  0.5953999 -0.09098530  0.065735340
5 -0.6446757 -0.01568565  0.035802870
6 -1.4842053 -0.02687825 -0.006586116
```

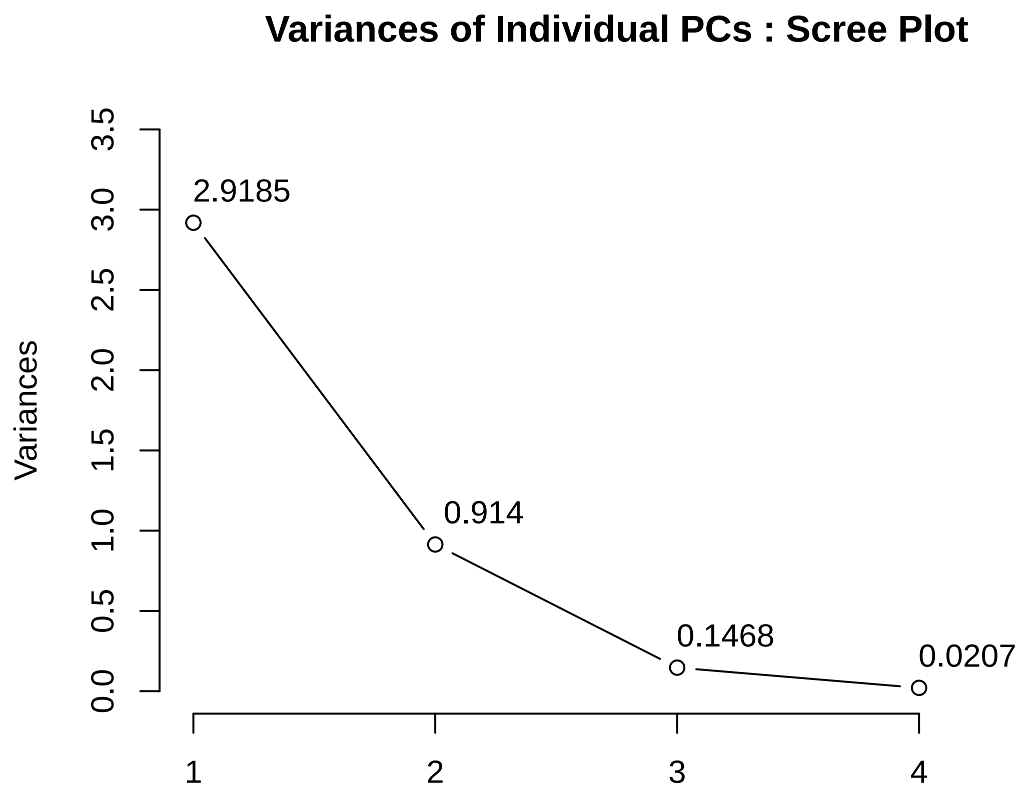
```
summary(pca)
```

```
Importance of components:
                          PC1    PC2     PC3     PC4
Standard deviation     1.7084 0.9560 0.38309 0.14393
Proportion of Variance 0.7296 0.2285 0.03669 0.00518
Cumulative Proportion  0.7296 0.9581 0.99482 1.00000
```

```
pca.var <- (pca$sdev)^2    # or simply apply(pca$x, 2, var)
pca.var
```

```
[1] 2.91849782 0.91403047 0.14675688 0.02071484
```

```
plot(pca, type='l', ylim=c(0,3.5), xlim=c(1,4.5),
     main = "Variances of Individual PCs : Scree Plot")
text((1:4)+0.2, pca.var+.2, as.character(round(pca.var, 4)))
```



```r
apply(pca$x, 2, var)
```

```
       PC1        PC2        PC3        PC4 
2.91849782 0.91403047 0.14675688 0.02071484 
```

The number of principal components (PCs) retained in PCA depends on the specific goals of the analysis and how much of the total variance in the dataset you wish to preserve. Below are common strategies and guidelines for determining the number of PCs to retain.

- **Variance Explained Threshold**: Retain enough PCs to account for a specified cumulative percentage of the total variance (e.g., 70%, 80%, 90%, or 95%).
- **Scree Plot**: Use a scree plot to visualize the eigenvalues and look for an "elbow," which indicates a point where the explained variance starts diminishing significantly
- **Cross-Validation**: Use cross-validation to determine the optimal number of PCs that minimizes prediction error for a downstream model (e.g., regression, classification).
- **Domain Knowledge**: Retain PCs based on practical interpretability or relevance to the problem. This is particularly useful in cases where certain PCs are known to correspond to meaningful patterns or relationships.
- **Parallel Analysis**: Compare the eigenvalues of the observed data to those from random data. Retain PCs where the observed eigenvalue exceeds the random eigenvalue.

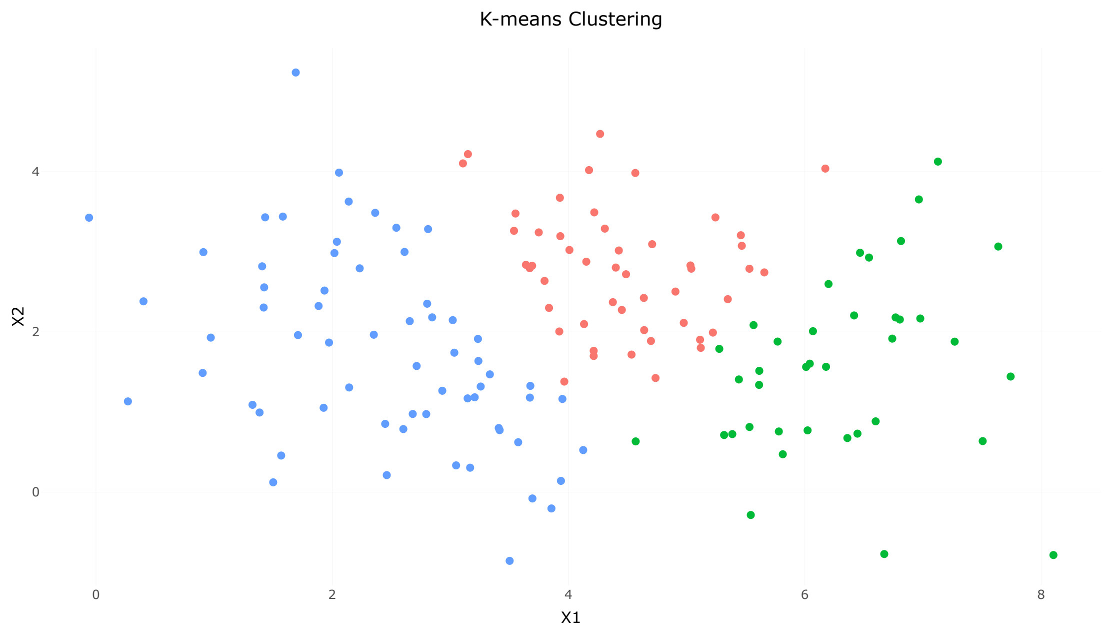
In the above example, the first two PCs capture more than 95% of the total variance in the original centered features.

4.2 Clustering-Based Feature Extraction

Clustering algorithms such as k-means and hierarchical clustering can also serve as feature extraction techniques. By grouping similar data points, cluster membership can be used as a new feature in the data set.

Example: K-means Clustering in R

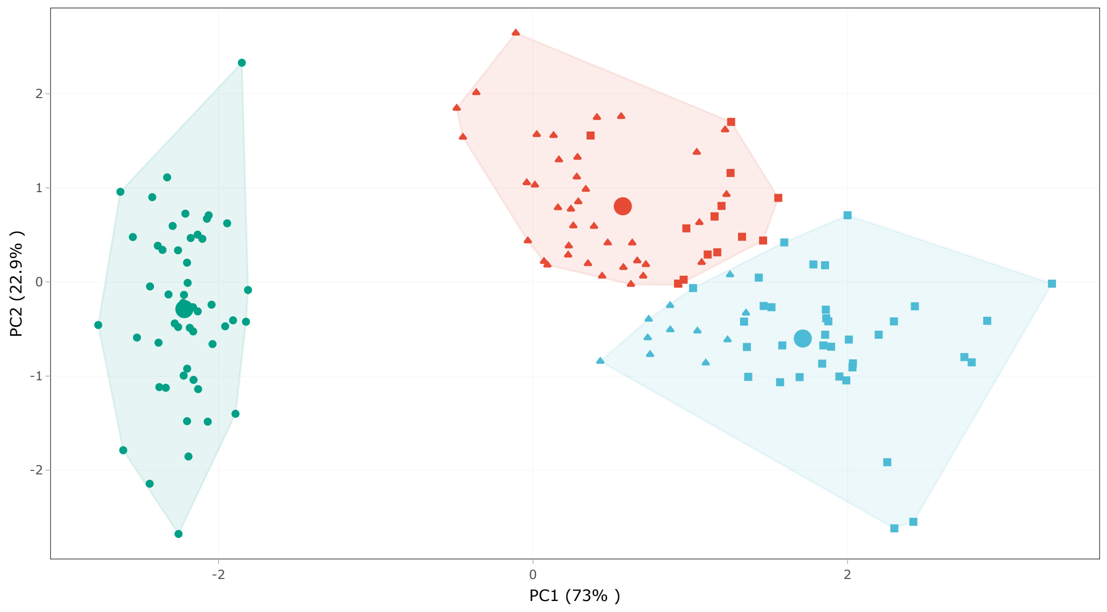
```r
# Load necessary library
#library(cluster)

# Generate sample data
set.seed(456)
data <- data.frame(
  X1 = rnorm(150, mean = 4, sd = 1.8),
  X2 = rnorm(150, mean = 2, sd = 1.2)
)

# Perform K-means clustering
kmeans_result <- kmeans(data, centers = 3)

# Add cluster labels as a new feature
data$Cluster <- kmeans_result$cluster

# Visualize clusters
ggclust = ggplot(data, aes(x = X1, y = X2, color = factor(Cluster))) +
  geom_point() +
  theme_minimal() +
  ggtitle("K-means Clustering") +
  labs(color = "Cluster") +
  theme(legend.position="none",
          plot.title = element_text(hjust = 0.5))
##
ggplotly(ggclust)
```



In this example, cluster labels are extracted as new features, which can be used for further analysis or as input to machine learning models.

Example: clustering based on two or more feature variables

```r
#library(ggpubr)
#library(factoextra)
## PCAs
res.pca <- prcomp(iris[, -5],  scale = TRUE)
## k-means with pre-selected 3 clusters
res.km <- kmeans(scale(iris[, -5]), 3, nstart = 25)
# Coordinates of individuals
ind.coord <- as.data.frame(get_pca_ind(res.pca)$coord)
# Add clusters obtained using the K-means algorithm
ind.coord$cluster <- factor(res.km$cluster)
# Add Species groups from the original data sett
ind.coord$Species <- iris$Species
###
eigenvalue <- round(get_eigenvalue(res.pca), 1)
#rownames(eigenvalue) = c("PC1", "PC2", "PC3", "PC4")
variance.percent <- eigenvalue$variance.percent
###
ggiris_clust <- ggscatter( ind.coord, x = "Dim.1", y = "Dim.2",
                      color = "cluster",
                      palette = "npg",
                      ellipse = TRUE,
                      ellipse.type = "convex",
                      shape = "Species", size = 1.5,
                      legend = "right",
                      ggtheme = theme_bw(),
                      xlab = paste0("PC1 (", variance.percent[1], "% )" ),
                      ylab = paste0("PC2 (", variance.percent[2], "% )" )
                     ) +
                stat_mean(aes(color = cluster), size = 4) +
                theme(legend.position="none",
                      plot.title = element_text(hjust = 0.5))
ggplotly(ggiris_clust)
```

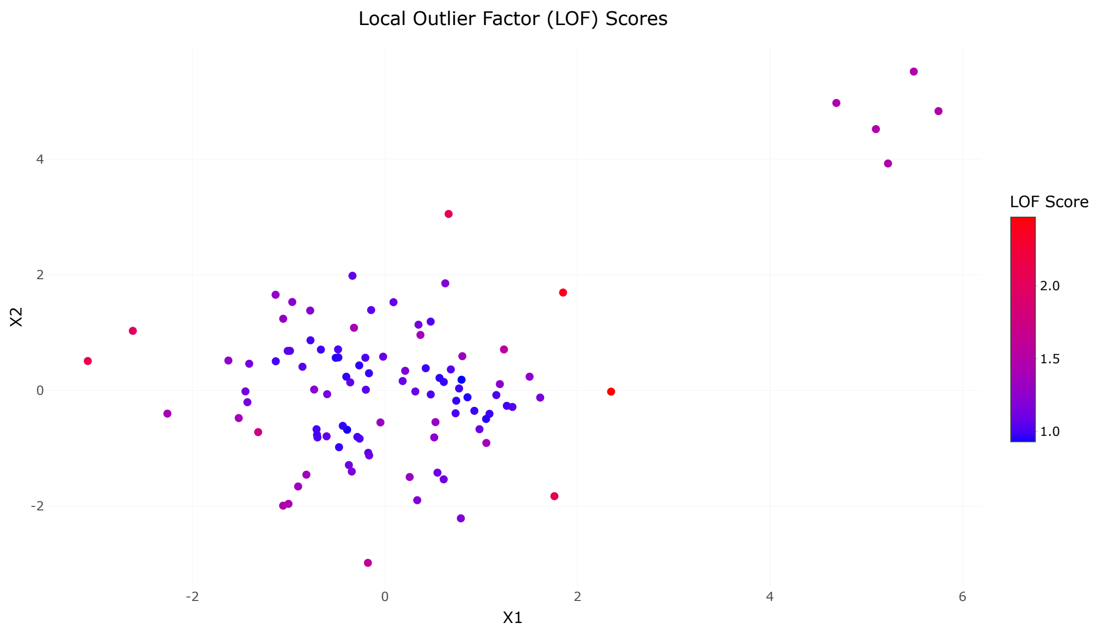


2
1
0
-1
-2
PC2 (22.9%)
-2
0
2
PC1 (73%)

4.3 Local Outlier Factor (LOF, optional)

LOF is a density-based algorithm primarily used for outlier detection. It measures the local density of data points relative to their neighbors. The LOF score can be used as a feature to identify anomalies in the dataset.

```r
# Load necessary library
# Generate sample data
set.seed(789)
data <- data.frame(
  X1 = c(rnorm(95, mean = 0, sd = 1), rnorm(5, mean = 5, sd = 0.5)),
  X2 = c(rnorm(95, mean = 0, sd = 1), rnorm(5, mean = 5, sd = 0.5))
)

# Compute LOF scores
lof_scores <- lof(data, k = 5)

# Add LOF scores as a new feature
data$LOF <- lof_scores

# Visualize LOF scores
gglof = ggplot(data, aes(x = X1, y = X2, color = LOF)) +
  geom_point() +
  scale_color_gradient(low = "blue", high = "red") +
  ggtitle("Local Outlier Factor (LOF) Scores") +
  labs(color = "LOF Score") +
  theme_minimal() +
  theme(plot.title = element_text(hjust = 0.5))
##
ggplotly(gglof)
```



Here, LOF scores help identify outliers, and these scores can be added as a feature to improve model robustness.

4.4 Feature Extraction Using Statistical Measures

Statistical measures like mean, variance, skewness, and kurtosis can be extracted as features, especially for time-series or signal data.

Example: Statistical Features in R

```r
# Load necessary library
#library(e1071)
# Generate sample time-series data
set.seed(123)
data <- data.frame(
  Time = 1:100,
  Value = sin(1:100 / 10) + rnorm(100, mean = 0, sd = 0.2)
)

# Compute statistical features
mean_value <- mean(data$Value)
variance <- var(data$Value)
skewness_value <- skewness(data$Value)
kurtosis_value <- kurtosis(data$Value)

# Add features to the dataset
data$Mean <- mean_value
data$Variance <- variance
data$Skewness <- skewness_value
data$Kurtosis <- kurtosis_value
```

Statistical features can effectively summarize the properties of a dataset and are often used in domains like signal processing.

4.5 Conclusion

Feature extraction is an essential part of the data preprocessing pipeline, enabling better insights and improving model performance. Methods like PCA, clustering, LOF, and statistical measures cater to different types of data and objectives. By leveraging these techniques, analysts and researchers can extract meaningful features and enhance the efficacy of their machine learning workflows. The illustrative R code examples provided here demonstrate the practical application of these methods in real-world scenarios.

5 Feature Creation

Feature creation is a critical step in data preprocessing that involves generating new features from existing data to improve the performance of predictive models. Effective feature creation can capture hidden patterns, improve model interpretability, and address domain-specific challenges. Some feature engineering methods introduced earlier can also be considered as part of this category.

5.1 Aggregation

Aggregation involves summarizing multiple features into a single feature. Common methods include sums, means, medians, and standard deviations.

```r
# Create an aggregate feature as the mean of Sepal.Length and Sepal.Width
iris$Sepal.Avg <- rowMeans(iris[, c("Sepal.Length", "Sepal.Width")])

# View aggregated feature
head(iris)
```

```
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species Sepal.Avg
1          5.1         3.5          1.4         0.2  setosa      4.30
2          4.9         3.0          1.4         0.2  setosa      3.95
3          4.7         3.2          1.3         0.2  setosa      3.95
4          4.6         3.1          1.5         0.2  setosa      3.85
5          5.0         3.6          1.4         0.2  setosa      4.30
6          5.4         3.9          1.7         0.4  setosa      4.65
```

5.2 Interaction Features

Interaction features capture the combined effect of two or more variables. These are especially useful when the relationship between predictors and the target variable depends on their interaction.

```r
# Interaction between Petal.Length and Petal.Width
iris$Petal.Interaction <- iris$Petal.Length * iris$Petal.Width

# View interaction feature
head(iris)
```

```
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species Sepal.Avg
1          5.1         3.5          1.4         0.2  setosa      4.30
2          4.9         3.0          1.4         0.2  setosa      3.95
3          4.7         3.2          1.3         0.2  setosa      3.95
4          4.6         3.1          1.5         0.2  setosa      3.85
5          5.0         3.6          1.4         0.2  setosa      4.30
6          5.4         3.9          1.7         0.4  setosa      4.65
  Petal.Interaction
1              0.28
2              0.28
3              0.26
4              0.30
5              0.28
6              0.68
```

Interaction terms can be critical in regression models where relationships between variables are non-additive.

Caution: *Most statistical software programs and libraries implicitly include interaction terms in the model formula, enabling meaningful interpretation of the associated regression coefficients. Explicitly defining an interaction feature in the model formula, as shown above, can compromise the interpretability of the regression coefficients. However, this is not a concern if the regression model is used solely for prediction purposes.*

5.3 Polynomial Features

Adding polynomial terms of features to capture non-linear relationships.

Example: Polynomial Terms

```r
library(polycor)
data <- data.frame(A = c(1, 2, 3))
data$A_squared <- data$A^2
print(data)
```

```
  A A_squared
1 1         1
2 2         4
3 3         9
```

5.4 Domain-Specific Features

Domain-specific features are derived using domain knowledge and tailored to the specific problem or field being addressed. These features play a critical role in improving model performance, interpretability, and relevance in a given context. The are two representative examples.

Healthcare: Calculating BMI

In healthcare, Body Mass Index (BMI) is a commonly used feature derived from height and weight.

```r
# Sample data
patients <- data.frame(
  height_cm = c(170, 160, 180),
  weight_kg = c(70, 80, 90)
)

# Calculate BMI
patients$BMI <- patients$weight_kg / (patients$height_cm / 100)^2
print(patients)
```

```
  height_cm weight_kg      BMI
1       170        70 24.22145
2       160        80 31.25000
3       180        90 27.77778
```

Finance: Debt-to-Income Ratio

In financial modeling, the debt-to-income ratio (DTI) is often used as a key indicator of financial health.

```r
# Sample data
finance <- data.frame(
  total_debt = c(50000, 25000, 75000),
  annual_income = c(100000, 80000, 150000)
)

# Calculate DTI ratio
finance$DTI <- finance$total_debt / finance$annual_income
print(finance)
```

```
  total_debt annual_income    DTI
1      50000        100000 0.5000
2      25000         80000 0.3125
3      75000        150000 0.5000
```