Data Mining- Practice 7 Dimensionality Reduction and PCA

Rajesh Kalakoti[§], Sven Nomm[*]
[*] Taltech, Estonia, 12616
[§] Email: rajesh.kalakoti@outlook.com

***Abstract*—In today's practical session, we will delve into the core concepts of data mining. The session is structured to provide a comprehensive understanding of essential techniques in data preprocessing, dimensionality reduction, and predictive modeling.**

I. Introduction

We begin by exploring the significance of data normalization, a crucial preprocessing step in data mining. Normalization ensures that features are on a consistent scale, facilitating accurate comparisons and analyses. Through hands-on exercises, attendees will learn various normalization techniques in R, enabling them to prepare diverse datasets for effective mining.

A. Missing Entries

Handling missing entries in a dataset is also called Data imputation. Data imputation is a statistical technique used to fill in missing or incomplete data points within a dataset.

1) Mean, Median, or Mode Imputation: Missing values are replaced by the mean (average), median, or mode (most frequently occurring value) of the observed data for the respective variable.

2) Regression Imputation: Missing values are predicted using regression models based on other variables in the dataset. A regression equation is created using variables without missing data, and this equation is used to estimate the missing values.

3) K-Nearest Neighbors (KNN) Imputation: Missing values are imputed based on values from similar cases (neighbors) in the dataset. The KNN algorithm identifies the nearest neighbors for each missing value and imputes the missing value based on their values.

4) Multiple Imputation: Multiple imputation involves creating multiple complete versions of the dataset, each with different imputed values. Statistical analyses are then performed on each dataset, and the results are combined to account for the uncertainty introduced by imputation.

II. Data Normalization

Data normalization is a type of data preprocessing technique that focuses on transforming features to a similar scale.

A. z-Score Normalization (Standardization):

Z-score normalization standardizes the data by transforming it to have zero mean and unit variance.

$$z_i^j = \frac{(x_i^j - \mu_j)}{\sigma_j}$$

- x is the original value, - μ is the mean of the variable, - σ is the standard deviation of the variable.

B. min-max normalization

Min-max scaling transforms the data within range [0, 1].

$$y_i^j = \frac{x_i^j - \min(x^j)}{\max(x^j) - \min(x^j)}$$

1) Code for data Data normalization: Here is the below function are written for two normalization techniques. Min-max normalization is useful when the data has a fixed range. Deep learning based models mostly recommended data normalization because of gradient descent.

```r
z_score <- function(x) {
  if (is.numeric(x)) {
    mean_x <- mean(x)
    sd_x <- sd(x)
    std_values <- (x - mean_x) / sd_x
    return(std_values)
  } else if (is.matrix(x) || is.data.frame(x)) {
    std_matrix <- scale(x)
    return(std_matrix)
  } else {
    stop("Unsupported input type")
  }
}

# Load the iris dataset
data(iris)
z_score_iris <- z_score(iris[, 1:4])
```

```r
print(xtable(
    head(z_score_iris),
    caption='Table showing z-score normalization
```

TABLE I: Table showing z-score normalization top 5 rows.

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
-0.90	1.02	-1.34	-1.31
-1.14	-0.13	-1.34	-1.31
-1.38	0.33	-1.39	-1.31
-1.50	0.10	-1.28	-1.31
-1.02	1.25	-1.34	-1.31
-0.54	1.93	-1.17	-1.05

```
    label='tbl:xtable.floating'),
    align=c(rep('r', 6), 'l'))
```

```
min_max <- function(x) {
  if (is.numeric(x)) {
    min_x <- min(x)
    max_x <- max(x)
    min_max <- (x - min_x)/(max_x - min_x)
    return(min_max)
  } else if(is.matrix(x) || is.data.frame(x))
    {
    normalized_matrix <- as.data.frame(
      lapply(x, min_max_normalize))
    return(normalized_matrix)
  } else {
    stop("Unsupported input type")
  }
}
x <- c(2, 5, 8, 3, 10)
normalized_x <- min_max(x)
print(normalized_x)
```

```
## [1] 0.000 0.375 0.750 0.125 1.000
```

III. Dimensionality Reduction

Let the data **D** consist of n points over d attributes, that is, it is an $n \times d$ matrix.

Principal Component Analysis (PCA) is a technique that seeks a r -dimensional basis that best captures the variance in the data. The direction with the largest projected variance is called the first principal component. The orthogonal direction that captures the second largest projected variance is called the second principal component, and so on.

$$PCA(D, \alpha)$$

1. **Compute the Mean:**

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i$$

   Compute the mean of the dataset.
2. **Center the Data:**

$$\overline{D} = D - 1 \cdot \mu^T$$

   Center the data by subtracting the mean from each data point.
3. **Compute the Covariance Matrix:**

$$\sum = \frac{1}{n}(\overline{D}^T \overline{D})$$

   Compute the covariance matrix.
4. **Compute Eigenvalues ($\sum$):**

$$(\lambda_1, \lambda_2, \ldots, \lambda_d)$$

compute eigen values of co-variance matrix

5. **Compute Eigen vectors ($\sum$):.**

$$U = (u_1, u_2, ...u_d)$$

   compute eigen vectors of co-variance matrix.
6. **Compute Fraction of Total Variance ($f(r)$):**

$$f(r) = \frac{\sum_{i=1}^{r} \lambda_i}{\sum_{i=1}^{d} \lambda_i}, \quad \text{for all } r = 1, 2, \ldots, d$$

   Compute the fraction of total variance for each dimension.
7. **Choose Smallest r such that $f(r) \geq \alpha$:** Choose the dimensionality r such that the fraction of total variance is greater than or equal to the specified threshold (α).
8. **Reduce Eigenvectors (U_r):**

$$U_r = \begin{pmatrix} u_1 & u_2 & \ldots & u_r \end{pmatrix}$$

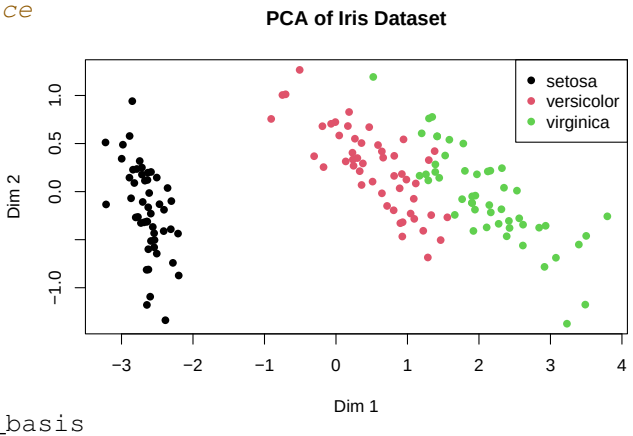
   Select the first r eigenvectors to form the reduced basis.
9. **Transform Data (A):**

$$A = \left\{ a_i \,\middle|\, a_i = U_r^T \overline{x}_i, \text{ for } i = 1, \ldots, n \right\}$$

Obtain the reduced dimensionality data by multiplying the reduced basis (U_r^T) with the original data (x_i) for $i = 1, \ldots, n$.

```
pca <- function(
    D, alpha, num_components) {
  # Step 1: Compute the mean
  mu <- colMeans(D)

  # Step 2: Center the data
  D_centered <- D - matrix(mu, nrow(D),
                           ncol(D),
                           byrow = TRUE)

  # Step 3: Computethe covariance matrix
  cov_matrix <- cov(D_centered)

  # Step 4: Compute eigenvalues
  # and eigenvectors
  eig_result <- eigen(cov_matrix)
  eig_val <- eig_result$values
  eig_vect <- eig_result$vectors
```

```r
  # Step 5: Compute fraction of total variance
  tot_var <- sum(eig_val)
  var_frac <- cumsum(eig_val)/tot_var

  # Step 6: Choose components
  # based on alpha
  num_components <- min(which(var_frac
                              >= alpha))

  # Step 7: Reduce eigenvectors
  red_basis <- eig_vect[, 1:num_components]

  # Step 8: Transform data
  red_data <- as.matrix(D_centered) %*% red_basis

  return(red_data)
}



# Test PCA on iris dataset,
#reducing from 4 features to 2
#features with alpha=0.95
library(datasets)

# Load iris dataset
data(iris)

# Extract features from iris dataset
features <- iris[, 1:4]

# Perform PCA with alpha=0.95
#and 2 components
reduced_data <- pca(
  features, alpha = 0.95,
  num_components = 2)

# Plot PCA results with
# customized axis labels
plot(reduced_data, col = iris$Species,
     pch = 16,
     main = "PCA of Iris Dataset",
     xlab = "Dim 1", ylab = "Dim 2")
legend("topright",
       legend = levels(iris$Species),
       col = 1:3,
       pch = 16)
```



IV. Linear Regression

Given a set of attributes or variables X_1, X_2, ..., X_d, called the predictor, explanatory, or independent variables, and given a real-valued attribute of interest Y, called the response or dependent variable, the aim of regression is to predict the response variable based on the independent variables. That is, the goal is to learn a regression function f, such that

$$Y = f(X_1, X_2, ..., X_d) + \epsilon = f(X) + \epsilon$$

where $X = (X_1, X_2, ..., X_d)^T$ is the multivariate random variable comprising the predictor attributes, and ϵ is a random error term that is assumed to be independent of X. In other words, Y is comprised of two components, one dependent on the observed predictor attributes, and the other, coming from the error term, independent of the predictor attributes. The error term encapsulates inherent uncertainty in Y, as well as, possibly the effect of unobserved, hidden or latent variables.

A. *Linear Regression Model*

In linear regression, the function f is assumed to be linear in its parameters. It can be represented as:

$$f(X) = \beta + \omega_1 X_1 + \omega_2 X_2 + \ldots + \omega_d X_d = \beta + \sum_{i=1}^{d} \omega_i X_i = \beta + \omega^T X$$

Here, the parameter β is the true (unknown) bias term, the parameter ω_i is the true (unknown) regression coefficient or weight for attribute X_i, and $\omega = (\omega_1, \omega_2, \ldots, \omega_d)^T$ is the true d-dimensional weight vector.

1) BIVARIATE REGRESSION: Let us first consider the case where the input data D comprises a single predictor attribute, $X = (x_1, x_2, \ldots, x_n)^T$, along with the response variable, $Y = (y_1, y_2, \ldots, y_n)^T$. Since f is linear, we have:

$$\hat{y}_i = f(x_i) = b + w \cdot x_i$$

Thus, we seek the straight line $f(x)$ with slope w and intercept b that best fits the data. The residual error, which is the difference between the predicted value (also called the fitted value) and the observed value of the response variable, is given as:

$$\epsilon_i = y_i - \hat{y}_i$$

Note that $|\epsilon|$ denotes the vertical distance between the fitted and observed response. The best-fitting line minimizes the sum of squared errors.

$$\min_{b,w} \text{SSE} = \sum_{i=1}^{n} \epsilon_i^2 = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 = \sum_{i=1}^{n} (y_i - b - w \cdot x_i)^2$$

To solve this objective, we differentiate it with respect to b and set the result to 0 to obtain:

$$\frac{\partial \text{SSE}}{\partial b} = -2 \sum_{i=1}^{n} (y_i - b - w \cdot x_i) = 0$$

$$w = \frac{(\sum_{i=1}^{n} x_i \cdot y_i) - n \cdot \mu_X \cdot \mu_Y}{\sum_{i=1}^{n} x_i^2 - n \cdot \mu_X^2}$$

The regression coefficient w can also be written as

$$w = \frac{\sum_{i=1}^{n} (x_i - \mu_X)(y_i - \mu_Y)}{\sum_{i=1}^{n} (x_i - \mu_X)^2} = \frac{\sigma_{XY}}{\sigma_X^2} = \frac{\text{cov}(X, Y)}{\text{var}(X)}$$

where σ_X^2 is the variance of X, and σ_{XY} is the covariance between X and Y. Noting that the correlation between X and Y is given as $\rho_{XY} = \frac{\sigma_{XY}}{\sigma_X \cdot \sigma_Y}$, we can also express w as:

$$w = \rho_{XY} \cdot \frac{\sigma_Y}{\sigma_X}$$

2) Code For LInear regression: Here I have created sample for regression exmaple, you can check the sample dataset csv file with regression_data in data folder.

```
library(here)
```

```
## here() starts at /home/rajeshkalakoti/Doc
```

```
data_file = here("data",
                 "regression_data.csv");
data = read.csv(data_file,header=TRUE)
dim(data)
```

```
## [1] 1000    6
```

```
# random simple using sample()
row.number <- sample(1:nrow(data),
                     0.8*nrow(data))
# 800 observation to train the data
training = data[row.number,]
 # 200 observation to test the data
testing = data[-row.number,]
# dimension of training data,
#800 rows with 6 variables
dim(training)
```

```
## [1] 800   6
```

```
dim(testing)
```

```
## [1] 200   6
```

```
# select all X's column for training
trainingX <- subset(training,
                    select = -c(Y))
# select only Y column for training
trainingY <- subset(training,
                    select = c(Y))
 # select all X's column for testing
testingX <- subset(testing,
                   select = -c(Y))
# select only Y column for testing
testingY <- subset(testing,
                   select = c(Y))
```

```
linearRegression <- function(x,y)
  { # create a function of x and y data
  # vector of 1's with the
  #same amount af rows.
  intercept <- rep(1, nrow(x))
  # Add intercept column to x
  x <- cbind(intercept, x)
  # create x matrix of feature variables
  matrix_X <- as.matrix(x)
  # create y vector of
  #  the response variable
  vector_Y <- as.matrix(y)
  betas  <- solve(
    t(matrix_X) %*% matrix_X) %*% t(
      matrix_X) %*% vector_Y
  betas <- round(betas, 2)
  return(betas)
}

PredictY <- function(x, betas) {
  betas_matrix <- t(as.matrix(betas))
  intercept <- rep(1, nrow(x))
  x <- cbind(intercept, x)
  matrix_X <- t(as.matrix(x))
  Ŷ <- betas_matrix %*% matrix_X
```

```
  return(Ŷ)
}
```

Error function

```
errors <- function(Y, Ŷ){
  Y <- as.matrix(Y)
  Ŷ <- t(as.matrix(Ŷ))
  # compute the sum squared errors
  RSS = sum((Y- Ŷ)^2)
  # RSS gives a measure of
  # error of prediction,
  #the lower it is the more our model
  #is accurate
  # compute the
  #total sum of squares
  TSS = sum((Y - mean(Ŷ))^2)
  # R2 represents
  #the proportion of variance
  R2 <- 1 - (RSS/TSS)
  # Root mean square error
  # we will use it to evaluate our model
  RMSE <- sqrt(mean((Ŷ - Y)^2))
  # return list of R2 and RMSE
  return(list(R2 = R2,
              RMSE = RMSE,
              RSS = RSS, TSS = TSS))
}
```

```
betas <- linearRegression(
  trainingX,trainingY)
# dimension of betas
#is 6 values of Y intercept
dim(betas)
```

```
## [1] 6 1
```

```
print(betas)
```

```
##                Y
## intercept  2.11
## X1         0.00
## X2         0.00
## X3        -1.30
## X4         0.40
## X5         2.09
```

```
# compute Ŷ with PredictY function using
Ŷ <- PredictY(testingX,
              betas)
dim(Ŷ)
```

```
## [1]   1 200
```

TABLE II: Error values

names	values
R2	0.98
RMSE	0.92
RSS	167.68
TSS	6757.78

```
error <- errors(testingY, Ŷ)
error_df <- data.frame(
  names = names(error),
  values = unlist(error))
```

```
print(xtable(
    error_df,
    caption='Error values',
    label='tbl:xtable.floating'),
    align=c(rep('r', 4), 'l'))
```

we will use lm function which can be used to create a simple linear regression model, in dataset We have Y column that will depend on 5 columns of X's (X1-x5)

```
Rversion <- lm(
  formula = Y ~ X1 + X2 + X3 + X4 + X5,
  data =data)
summary(Rversion)
```

```
##
## Call:
## lm(formula = Y ~ X1 + X2 + X3 + X4 + X5, data
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.4238 -0.2019 -0.0309  0.1434  7.0502
##
## Coefficients:
##             Estimate Std. Error  t value Pr(>
## (Intercept)  2.127945   0.024946   85.301   <2
## X1           0.010831   0.066027    0.164    0
## X2          -0.003681   0.013002   -0.283    0
## X3          -1.305743   0.009444 -138.257   <2
## X4           0.397638   0.017141   23.198   <2
## X5           2.084183   0.007731  269.601   <2
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.
##
## Residual standard error: 0.6138 on 994 degrees
## Multiple R-squared:  0.9899, Adjusted R-square
## F-statistic: 1.949e+04 on 5 and 994 DF,  p-val
```

V. Overall, both R2 values from the regressionlinear function from scatch and using also built in function with R

VI. are close to 1 which means the two method are very close.