Data Mining- Practice 6 Classification- Support Vector machines and Kernel Trick

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 $i = 1, \ldots, n$.

```
perform_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),

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

  # Step 4: Compute eigenvalues and eigenvectors
  eigen_result <- eigen(cov_matrix)
  eigenvalues <- eigen_result$values
  eigenvectors <- eigen_result$vectors

  # Step 5: Compute fraction of total variance
  total_variance <- sum(eigenvalues)
  variance_fraction <- cumsum(eigenvalues) / tota
```

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

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

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

  return(reduced_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 <- perform_pca(features, alpha = 0.95, num_components = 2)

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