بسم الله الرحمن الرحيم

University of Khartoum

Faculty of mathematical sciences and information

Research titled: Principal companent analysis

Present by:

Madiha Osman Adam Mohammed

Marafy Hago Aldow Alli

Asawer Haider Aldow Ahmed

Supervisor:

Dr. Randa Abdelhalim

2024-2025

إهداء:

الى أعز الناس إلى قلبي امي، إلى عائلتي الكريمة التي كانت دائما سندي في كل اقول لكم ان هذا الإنجاز هو ثمرة دعمكم اللامحدود ،خطوه، إلى كل فرد منهم وتضحياتكم الغالية.

في الأوقات الى أصدقائي الذين كانوا خير رفيق في مسيرتي، ووقفوا بجانبي الصعبة كما في اللحظات المشرقة.

الى كل من علمني وساهم في تشكيل افكاري، وخاصة من قدم لي النصائح القيمة التي كانت محورية في إتمام هذا المشروع، اخص بالشكر دكتورتي / رنده عبدالحليم.

هذا العمل هو نتاج مثابرتي وجهودي، ولاكن بفضل محبتكم ودعمكم أصبحت اليوم أقوى وأكثر عزيمه.

لكم بكل فخر وامتنان، واتمنى ان يكون بداية لمزيد من النجاحات. اهديه

Taple of contents

Biplot Intrepretation

4.3 Intrepretation of Results

Dimensionality Reduction Effectiveness

Variables Relationships

Cluster Identifictation

Chapter 5: Application of PCA

5. 1 Case Study in PCA Variants in Mobile Malwer Detection

Dataset overview (CICMalDroid-2020)

PCA Variants : Sparse PCA Randomized PCA, Incremental PCA, Kernal PCA

Observations Malwer Classification

5.2 Independe to Comparative Study of PCA, ICA, and LDA on the FERET

Algorithms Overview(PCA, ICA, LDA)

Mathodology and Dataset Description

Results and performance Comperison

Chapter 1

Introduction

When we want to deal with a dataset that contains a large number of variables and features without losing important information, or when the variables are highly correlated (multicollinearity) we resort to one of the dimensionality reduction techniques in statistics, with Principal Component Analysis (PCA) being one of the most popular methods.

Principal Component Analysis (PCA) is usually used for data visualisation, noise reduction (removing unimportant information), and identifying patterns in various applications such as image processing, computer biology, finance, and others.

In this research, we will provide an overview of Principal Component Analysis (PCA) and explain the cases in which it is used (simplifying and analyzing data, identifying important patterns, improving the efficiency of machine learning algorithms, reducing noise), and how to apply it.

1.1 Curse of Dimensionality:

Dealing with datasets having a large number of features or variables compared to the number of observations or samples can arise the problem of dimensionality in statistics . It's a common issue in machine learning and data analysis, especially in high-dimensional data like images, text, or genomic data.

Let's break down the problem and its implications:-

- Sparse Data: As the number of dimensions increases, the data points become more spread out in the feature space. This leads to a sparsity of data points in high-dimensional space, meaning each point is relatively far from its neighbors.

- Computational Challenges: Algorithms for analyzing and modeling high-dimensional data require more processing power, memory, and time. They can become computationally expensive and inefficient, especially with limited computational resources.

- Impact on Sample Size:

- Need for More Samples: To achieve statistically significant results and reliable model performance, more samples are generally required as the number of dimensions increases. This is because the curse of dimensionality makes it harder to distinguish true patterns from random noise with limited data.
- Sampling Bias: In high-dimensional data, it's harder to obtain representative samples. Sampling bias can occur if the data collection process systematically excludes or overrepresents certain groups of observations.

• Consequences of Dimensionality Issues:

- Overfitting: Models can become overly complex and learn the noise in the data, leading to poor generalization performance on unseen data.
- Reduced Accuracy: Predictions made by models trained on high-dimensional data may be less accurate due to the increased complexity and lack of sufficient data to learn the true relationships between features.

1.2 Dimensionality reduction techniques

Dimensionality reduction techniques are methods used to reduce the number of input variables or features in a dataset while preserving as much relevant information as possible. We use it because high-dimensional data can be difficult to visualize and analyse.

Dimensionality reduction techniques can be b categorized into two main types: linear and non-linear methods.

1. Linear Dimensionality Reduction:
    - Principal Component Analysis (PCA): PCA is a popular linear dimensionality reduction technique that transforms the original features into a new set of uncorrelated variables called principal components. These components are ordered by the amount of variance they explain in the data, allowing for the reduction of dimensions while retaining as much variance as possible.
    - Linear Discriminant Analysis (LDA): LDA is a supervised linear dimensionality reduction technique that aims to find the feature subspace that maximizes class separability.

2. Non-linear Dimensionality Reduction:
    - t-Distributed Stochastic Neighbor Embedding (t-SNE): t-SNE is a non-linear dimensionality reduction technique commonly used for visualization. It

aims to map high-dimensional data to a low-dimensional space while preserving local structure and clusters in the data.

- Isomap: Isomap is a non-linear dimensionality reduction method that focuses on preserving the geodesic distances between all pairs of data points, capturing the intrinsic geometry of the data.

1.3 History of principal Component analysis (PCA)

We can divide the history of PCA into two stages: the fundamental stage and the post-modern and computer stage.

In a fundamental stage While Beltrami and Jordan independently developed the singular value decomposition (SVD) in the 1870s, which forms the basis of PCA, the earliest formal descriptions of PCA are credited to Pearson (1901) and Hotelling (1933). Pearson focused on finding lines and planes that best fit a set of points in multidimensional space. This geometric approach also led to Principal Components (PCs). Hotelling's work, considered the more important of the two, presented a standard algebraic derivation of PCA. His motivation was to find a smaller set of "fundamental" variables that could explain the original variables. Hotelling coined the term "principal components" and "method of principal components" to distinguish it from factor analysis, which uses similar concepts but has a different purpose. Hotelling's derivation used correlation matrices, expressed original variables in terms of components, and didn't rely on matrix notation. The calculations for PCA were "cumbersome" in the pre-computer era, even for a few variables, highlighting the challenges of early statistical analysis.

After initial development in the early 20th century. The "power method" for calculating PCs and Girshick's work on the statistical properties of sample PCs (1936-1939) laid important groundwork. However, PCA saw limited real-world applications for about 25 years due to the computational complexity of hand calculations, which made it feasible only for datasets with a very limited number of variables. The widespread adoption of computers in the 1960s revolutionized PCA, enabling its application to much larger datasets and unlocking its full potential. The text highlights four key papers that furthered the understanding and application of PCA: Anderson (1963), Rao (1964), Gower (1966), and Jeffers (1967). The book by
Preisendorfer and Mobley (1988) is also considered a significant contribution, offering numerous novel ideas that have yet to be fully explored. PCA is now widely used in diverse fields, as evidenced by thousands of research publications related to it and its applications in areas like agriculture, biology, economics, and more.

Chapter 2

Theoretical Framework

In this chapter, we will provide a comprehensive explanation of the mathematical foundations underlying Principal Component Analysis (PCA), including an in-depth discussion of the covariance matrix, eigenvalues, and eigenvectors. We will also explore the crucial steps of data preprocessing, specifically focusing on normalization and centering, which are essential for ensuring the effectiveness of PCA. In addition, we will provide a definition of what are principal components and how PCA constructs the principal components. Finally, we will present a step-by-step breakdown of the PCA algorithm, detailing each stage of the process to enhance understanding and facilitate practical application.

2.1 Mathematical Framework

Introduction to Eigenvalues and Eigenvectors:

Consider a square matrix n × n. If X is the non-trivial column vector solution of the matrix equation $AX = \lambda X$, where λ is a scalar, then X is the eigenvector of matrix A, and the corresponding value of λ is the eigenvalue of matrix A.

Suppose the matrix equation is written as $A X – \lambda X = 0$. Let I be the n × n identity matrix.
If I X is substituted by X in the equation above, we obtain

$A X – \lambda I X = 0$.

The equation is rewritten as $(A – \lambda I) X = 0$.

The equation above consists of non-trivial solutions if and only if the determinant value of the matrix is 0. The characteristic equation of A is $Det (A – \lambda I) = 0$. ‘A’ being an n × n matrix, if $(A – \lambda I)$ is expanded, $(A – \lambda I)$ will be the characteristic polynomial of A because its degree is n.

Properties of Eigenvalues:

Let A be a matrix with eigenvalues λ1, λ2,…., λn.
The following are the properties of eigenvalues.

(1) The trace of A, defined as the sum of its diagonal elements, is also the sum of all eigenvalues,

$$te(A) = \sum_{i=1}^{n} a_{ii} = \sum_{i=1}^{n} \lambda_i = \lambda_1 + \lambda_2 + \quad . + \lambda$$

(2) The determinant of A is the product

.

of^n all its eigenvalues,

$$det(A) \prod_{i=1}^{n} \lambda_i = \lambda_1 \lambda_2 \quad . \lambda_n$$

.

(3) The eigenvalues of the kth power of A, that is, the eigenvalues of Ak, for any positive integer k, are

λ_{k1} , ., λ_{kn}

(4) The matrix A is invertible if and only if every eigenvalue is nonzero.

(5) If A is invertible, then the eigenvalues of A^{-1} are $\lambda 1_1, . , \lambda 1_N$

(6) If A is equal to its conjugate transpose, or equivalently if A is Hermitian, then every eigenvalue is real. The same is true for any real symmetric matrix.

(7) If A is not only Hermitian but also positive-definite, positive-semidefinite, negative-definite, or negative-semidefinite, then every eigenvalue is positive, non-negative, negative, or non-positive, respectively.

(8) If A is unitary, every eigenvalue has absolute value $|\lambda i| = 1$.

(9) If A is a n×n matrix and {λ1, λ2,…., λk} are its eigenvalues, then the eigenvalues of the matrix I + A (where I is the identity matrix) are {λ1+ 1, λ2+1,…., λk+1

The eigenvector of a matrix is also known as a latent vector, proper vector or characteristic vector. They are defined in the reference of a square matrix. Matrix is an important branch that is studied under linear algebra. A matrix is a rectangular array of numbers or other elements of the same kind. It generally represents a system of linear equations.

Eigenvector Method:

The method of determining the eigenvector of a matrix is given below:

If A be an n × n matrix and λ be the eigenvalues associated with it. Then, eigenvector v can be defined by the following relation:

$Av = \lambda v$

If I is the identity matrix of the same order as A, then

$(A – \lambda I)v = 0$

The eigenvector associated with matrix A can be determined using the above method.

Here, v is known as the eigenvector belonging to each eigenvalue and is written as:

$$v = \begin{bmatrix} v_1 \\ v_2 \\ . \\ . \\ . \\ v_n \end{bmatrix}$$

Eigenvector Equation

The equation corresponding to each eigenvalue of a matrix is given by:

AX = λX

It is formally known as the eigenvector equation.

In place of λ, put each eigenvalue one by one and get the eigenvector equation which enables us to solve for the eigenvector belonging to each eigenvalue.

For example: Suppose that there are two eigenvalues λ1 = 0 and λ2 = 1 of any 2 × 2 matrix. Then,

AX = λ1X AX

= 0…..(1)

and

AX = λ2X

A = 1

(A – I) X = O ….. (2)

Equations (1) and (2) are eigenvector equations for the given matrix.

Where, I = Identity matrix of the same order as A

O = zero matrix of the same order as A

$$\begin{bmatrix} x \\ y \end{bmatrix}$$

X = Eigenvector which is equal to (as A is of order 2) How

to Find an Eigenvector?

In order to find the eigenvectors of a matrix, one needs to follow the steps, given below:

Step 1: Determine the eigenvalues of given matrix A using the equation det (A – λI) = 0, where I is the equivalent order identity matrix as A. Denote each eigenvalue of λ1, λ2, λ3,…

Step 2: Substitute the value of λ1 in equation AX = λ1X or (A – λ1I)X = 0.

Step 3: Calculate the value of eigenvector X, which is associated with eigenvalue λ1.

Step 4: Repeat steps 3 and 4 for other eigenvalues λ2, λ3, … as well.

Now let us explain more with examples :

Example 1: Find the eigenvalues for the following matrix.

$$A = \begin{bmatrix} 4 & 6 \\ 1 & 5 \end{bmatrix}$$

Solution:

Given,

$$A = \begin{bmatrix} 4 & 6 \\ 1 & 5 \end{bmatrix}$$

$$A - \lambda I = \begin{bmatrix} 4-\lambda & 6 \\ 1 & 5-\lambda \end{bmatrix}$$

$$|A - \lambda I| = 0$$

$$\Rightarrow \begin{vmatrix} 4-\lambda & 6 \\ 1 & 5-\lambda \end{vmatrix} = 0$$

$(4 - \lambda)(5 - \lambda) - 6 = 0$

$\Rightarrow 20 - 5\lambda - 4\lambda + \lambda^2 - 6 = 0$

$\Rightarrow \lambda^2 - 9\lambda + 14 = 0$

$\Rightarrow (\lambda - 7)(\lambda - 2) = 0$

$\Rightarrow \lambda = 7$ or $\lambda = 2$

Example 2: Find the eigenvectors for the following matrix.

$$A = \begin{bmatrix} 1 & 4 \\ -4 & -7 \end{bmatrix}$$

Solution :

$$A = \begin{bmatrix} 1 & 4 \\ -4 & -7 \end{bmatrix}$$

$$A - \lambda I = \begin{bmatrix} 1-\lambda & 4 \\ -4 & -7-\lambda \end{bmatrix}$$

$$|A - \lambda I| = \begin{vmatrix} 1-\lambda & 4 \\ -4 & -7-\lambda \end{vmatrix}$$

$$(1–\lambda)(-7–\lambda)–4(-4) = 0$$

$$(\lambda + 3)^2 = 0$$

λ = -3, -3

Using eigenvector equation,

A X = − 3 X

A + 3 I = O

$$\left(\begin{bmatrix} 1 & 4 \\ -4 & -7 \end{bmatrix} + \begin{bmatrix} 3 & 0 \\ 0 & 3 \end{bmatrix} \right) \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Which gives,

4x + 4y = 0

or

x + y = 0

Let us set x = k, then y = -k

Therefore, the required eigenvector is:

$$X = \begin{bmatrix} x \\ y \end{bmatrix} = k \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

Covariance Matrix:

Covariance matrix is a type of matrix that is used to represent the covariance values between pairs of elements given in a random vector. The covariance matrix can also be

referred to as the variance covariance matrix. This is because the variance of each element is represented along the main diagonal of the matrix.

A covariance matrix is always a square matrix. Furthermore, it is positive semi-definite, and symmetric. This matrix is very useful in stochastic modeling and principle component analysis. In this part , we will learn about the variance covariance matrix, its formula, examples, and various important properties associated with it.

To determine the covariance matrix, the formulas for variance and covariance are required. Depending upon the type of data available, the variance and covariance can be found for both sample data and population data. These formulas are given below.

Population Variance: $\text{var(x)} = \sum_{1}^{n}(x_i - \mu)/n$

Population Covariance: $\text{cov(x, y)} = \sum_{1}^{n}(x_i - \mu_x)(y_i - \mu_y)/n$

Sample Variance: $\text{var(x)} = \sum_{1}^{n}(x_i - \bar{x})/n - 1$

Sample Covariance: $\text{cov(x, y)} = \sum_{1}^{n}(x_i - \bar{x})(y_i - \bar{y})/n - 1$

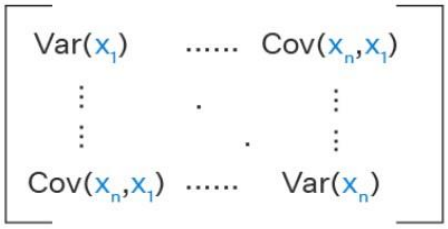
μ = mean of population data.

$\bar{x}$ = mean of sample data. n = number of

observations in the dataset.

x_i = observations in dataset x.

Using these formulas, the general form of a variance covariance matrix is given as follows:



Example:

Suppose there are two data sets X = {3, 2} and Y = {7, 4}. The sample variance of dataset X = 0.5, and Y = 4.5. The covariance between X and Y is 1.5. The covariance matrix is expressed as follows:

$$\begin{bmatrix} 0.5 & 1.5 \\ 1.5 & 4.5 \end{bmatrix}$$

Properties of Covariance Matrix:

- A covariance matrix is always a square matrix. This means that the number of rows of the matrix will be equal to the number of columns.

- The matrix is symmetric. Suppose M is the covariance matrix then M^T = M.

- It is positive semi-definite. Let u be a column^T vector, U^T is the transpose of that vector

  and M be the covariance matrix then U Mu ≥ 0.

- All eigenvalues of the variance covariance matrix are real and non-negative.

2.2 Data preprocessing

Centering and normalizing data before running Principal Component Analysis (PCA) is crucial for several reasons:

1. Centering (Mean Subtraction)

Removes Bias: Centering the data by subtracting the mean ensures that the PCA focuses on the variance around the mean rather than the absolute values. This is important because PCA aims to identify directions (principal components) that maximize variance.
Focus on Variance: If the data is not centered, the first principal component may be influenced by the mean rather than the underlying structure of the data. Centering allows PCA to find components that are more representative of the data's distribution.

2. Normalizing (Scaling)

Equal Weighting: Normalization (often done by dividing by the standard deviation) ensures that all features contribute equally to the analysis. If features are on different scales (e.g., one feature ranges from 0 to 1 and another from 0 to 1000), the PCA will be biased towards the feature with the larger scale.
Improved Interpretability: Normalizing the data can help in interpreting the principal components, as each feature contributes equally to the variance in the dataset, making it easier to understand the significance of each component.

3. Mathematical Properties

Covariance Matrix: PCA relies on the covariance matrix of the data. If the data is not centered, the covariance matrix will include the mean values, which can distort the analysis. Centering ensures that the covariance matrix reflects only the variability of the data.
Eigenvalues and Eigenvectors: PCA involves calculating the eigenvalues and eigenvectors of the covariance matrix. Centered and normalized data lead to a more accurate representation of the data's structure, ensuring that the principal components derived from the eigenvectors are meaningful.

2.3 Principal Components

Principal components are new variables that are constructed as linear combinations or mixtures of the initial variables. These combinations are done in such a way that the new variables (i.e., principal components) are uncorrelated and most of the information within the initial variables is squeezed or compressed into the first components. So, the idea is 10-dimensional data gives you 10 principal components, but PCA tries to put maximum possible information in the first component, then maximum remaining information in the second and so on..
Organizing information in principal components this way will allow you to reduce dimensionality without losing much information, and this by discarding the components with low information and considering the remaining components as your new variables.

An important thing to realize here is that the principal components are less interpretable and don't have any real meaning since they are constructed as linear combinations of the initial variables.

Geometrically speaking, principal components represent the directions of the data that explain a maximal amount of variance, that is to say, the lines that capture most information of the data. The relationship between variance and information here, is that, the larger the variance carried by a line, the larger the dispersion of the data points along it, and the larger the dispersion along a line, the more information it has. To put all this simply, just think of principal components as new axes that provide the best angle to see and evaluate the data, so that the differences between the observations are better visible.

How PCA Constructs the Principal Components:

As there are as many principal components as there are variables in the data, principal components are constructed in such a manner that the first principal component accounts for the largest possible variance in the data set. For example, let's assume that the scatter plot of our data set is as shown below, can we guess the first principal component ? Yes, it's approximately the line that matches the purple marks because it goes through the origin and it's the line in which the projection of the points (red dots) is the most spread out. Or mathematically speaking, it's the line that maximizes the variance (the average of the squared distances from the projected points (red dots) to the origin).

The second principal component is calculated in the same way, with the condition that it is uncorrelated with (i.e., perpendicular to) the first principal component and that it accounts for the next highest variance.

This continues until a total of p principal components have been calculated, equal to the original number of variables.

2.4 PCA Algorithm

After we have covered the mathematical aspect, In this section, you will get to know about the steps involved in the Principal Component Analysis technique.

Step 1: Standardization

The aim of this step is to standardize the range of the continuous initial variables so that each one of them contributes equally to the analysis.

More specifically, the reason why it is critical to perform standardization prior to PCA, is that the latter is quite sensitive regarding the variances of the initial variables. That is, if there are large differences between the ranges of initial variables, those variables with larger ranges will dominate over those with small ranges (for example, a variable that ranges between 0 and 100 will dominate over a variable that ranges between 0 and 1), which will lead to biased results. So, transforming the data to comparable scales can prevent this problem.

Step 2: Covariance Matrix Computation

The aim of this step is to understand how the variables of the input data set are varying from the mean with respect to each other, or in other words, to see if there is any relationship between them. Because sometimes, variables are highly correlated in such a way that they contain redundant information. So, in order to identify these correlations, we compute the covariance matrix.

What do the covariances that we have as entries of the matrix tell us about the correlations between the variables?

It’s actually the sign of the covariance that matters:

-If positive then: the two variables increase or decrease together (correlated)
-If negative then: one increases when the other decreases (Inversely correlated)

Step 3: Compute the eigenvectors and eigenvalues of the covariance matrix to identify the principal components

Eigenvectors and eigenvalues are the linear algebra concepts that we need to compute from the covariance matrix in order to determine the principal components of the data.

What you first need to know about eigenvectors and eigenvalues is that they always come in pairs, so that every eigenvector has an eigenvalue. Also, their number is equal to the number of dimensions of the data. For example, for a 3-dimensional data set, there are 3 variables, therefore there are 3 eigenvectors with 3 corresponding eigenvalues.

It is eigenvectors and eigenvalues who are behind all the magic of principal components because the eigenvectors of the Covariance matrix are actually the directions of the axes where there is the most variance (most information) and that we call Principal Components. And eigenvalues are simply the coefficients attached to eigenvectors, which give the amount of variance carried in each Principal Component.

By ranking your eigenvectors in order of their eigenvalues, highest to lowest, you get the principal components in order of significance.

Step 4: Create a Feature Vector

As we saw in the previous step, computing the eigenvectors and ordering them by their eigenvalues in descending order, allow us to find the principal components in order of significance. In this step, what we do is, to choose whether to keep all these components or discard those of lesser significance (of low eigenvalues), and form with the remaining ones a matrix of vectors that we call Feature vector.

So, the feature vector is simply a matrix that has as columns the eigenvectors of the components that we decide to keep. This makes it the first step towards dimensionality reduction, because if we choose to keep only p eigenvectors (components) out of n, the final data set will have only p dimensions.

Step 5: Recast the Data Along the Principal Components Axes

In the previous steps, apart from standardization, you do not make any changes on the data, you just select the principal components and form the feature vector, but the input data set remains always in terms of the original axes (i.e, in terms of the initial variables).

In this step, which is the last one, the aim is to use the feature vector formed using the eigenvectors of the covariance matrix, to reorient the data from the original axes to the ones represented by the principal components (hence the name Principal Components Analysis). This can be done by multiplying the transpose of the original data set by the transpose of the feature vector.

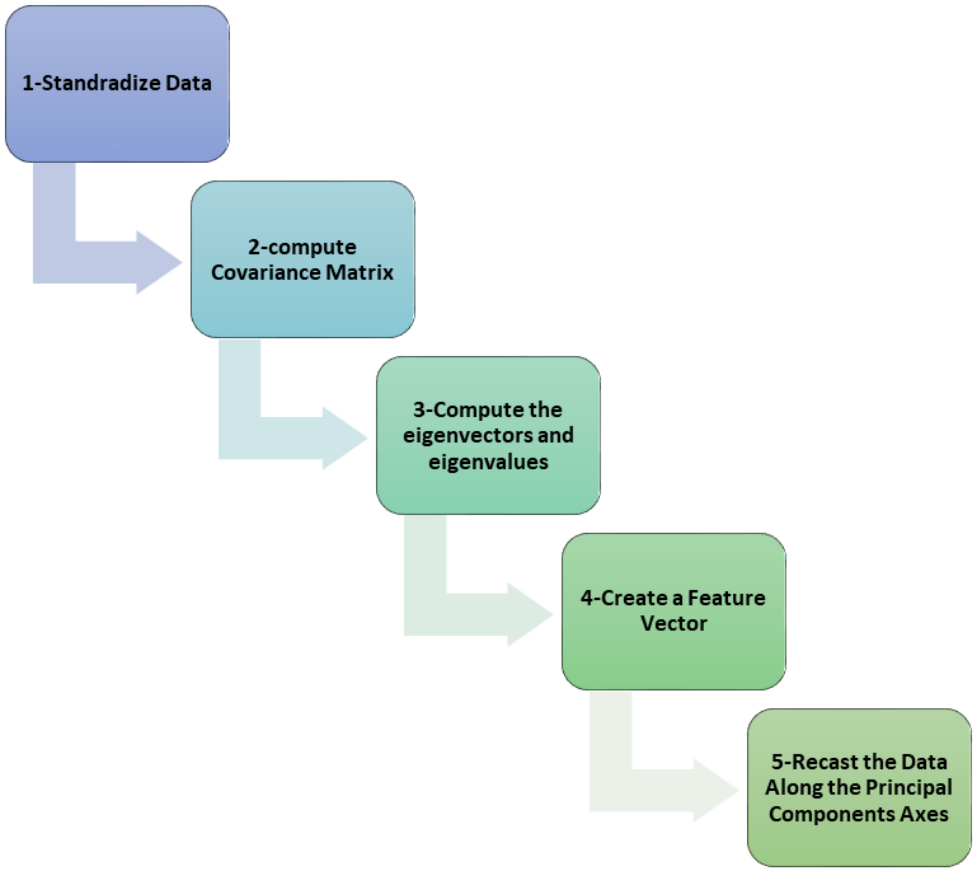


Figure (1) PCA algorithm

Chapter 3

Implementation of PCA

Introduction

In this chapter , we will use the Python programming language due to its status as one of the most popular programming languages in the world, making it an ideal choice for both beginners and professionals alike. In the field of data analysis, Python plays a vital role thanks to its diverse and powerful libraries such as Pandas, NumPy, Matplotlib, and Seaborn.

Python enables analysts to easily collect, process, and clean data, as well as perform complex statistical analyses and create effective visualizations. It also provides tools for big data analysis and machine learning, making it a comprehensive and powerful tool for data-driven decision-making.

Libraries and modules:-

When discussing Principal Component Analysis (PCA) and the use of libraries like **NumPy**, **Matplotlib**, **Pandas**, and modules such as **sklearn.decomposition, Confusion_matrix** , it's important to understand their roles in implementing and analyzing this statistical technique. Here's how each library and module contributes:

1. NumPy (imported as np):

NumPy is a powerful library for handling arrays and numerical computations. In PCA, NumPy is used to process numerical data, such as finding the eigenvalues and eigenvectors of the covariance matrix.
NumPy provides efficient functions for performing the necessary computations for PCA, such as calculating means, standard deviations, and transforming data into the appropriate form.

2. Pandas (imported as pd):

Pandas is used to manage and explore data before applying PCA. Raw data often needs cleaning and organization, and this is where Pandas comes in with its data structures like DataFrames, which make it easier to handle tabular data.
Pandas helps in loading the data, performing basic statistical operations, and preparing the data for analysis by transforming it into formats that NumPy and PCA can easily handle.

3. Matplotlib (imported as plt):

Matplotlib is used for creating visualizations, which are an essential part of PCA analysis. After reducing dimensions using PCA, results are visually displayed to understand the distribution of data in the new space (e.g., plotting the data in 2D or 3D charts).

Matplotlib can be used to plot charts showing the new data components and how the data is distributed across these components.

4. sklearn.decomposition

It is a module in the scikit-learn library provides algorithms for dimensionality reduction, which is the process of reducing the number of features in a dataset while preserving as much of the variance as possible

5-Confusion_matrix

It is in the sklearn.metrics library ,uesed to evaluate the classification performance. It compares the predicted labels with the true labels, helping you identify false positives, false negatives, and the overall accuracy of your model after dimensionality reduction with PCA.

To implement PCA using Python, follow these steps:

Step 1: Importing the libraries

 # importing required libraries
import numpy as np import
matplotlib.pyplot as plt import
pandas as pd

Step 2: Importing the dataset
Import the dataset and distribute the dataset into X and y components for data analysis.

 # importing or loading the dataset dataset
= pd.read_csv('wine.csv')
 # distributing the dataset into two components X and Y
X = dataset.iloc[:, 0:13].values y = dataset.iloc[:,
13].values

Step 3: Splitting the dataset into the Training set and Test set

```
# Splitting the X and Y into the # Training set and
Testing set from sklearn.model_selection import
train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

Step 4: Feature Scaling

Doing the pre-processing part on a training and testing set such as fitting the Standard scale.

```
# performing preprocessing part from
sklearn.preprocessing import StandardScaler sc =
StandardScaler()

X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

Step 5: Applying PCA function

Applying the PCA function into the training and testing set for analysis.

```
# Applying PCA function on training # and
testing set of X component from
sklearn.decomposition import PCA

pca = PCA(n_components=2)

X_train = pca.fit_transform(X_train)
X_test = pca.transform(X_test)

explained_variance = pca.explained_variance_ratio_
```

Step 6: Fitting Logistic Regression To the training set

```
# Fitting Logistic Regression To the training set from
sklearn.linear_model import LogisticRegression

classifier = LogisticRegression(random_state=0)
classifier.fit(X_train, y_train)
```

Step 7: Predicting the test set result

```
# Predicting the test set result using #
predict function under LogisticRegression
y_pred = classifier.predict(X_test)
```

Step 8: Making the confusion matrix

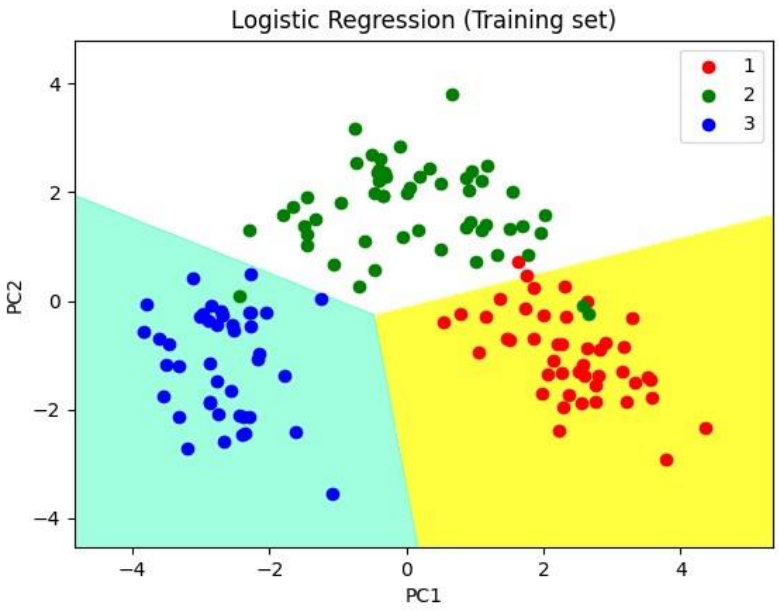
```
# making confusion matrix between #
test set of Y and predicted value.
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, y_pred)
```

Step 9: Predicting the training set result

```
# Predicting the training set #
result through scatter plot
from matplotlib.colors import ListedColormap

X_set, y_set = X_train, y_train
X1, X2 = np.meshgrid(np.arange(start=X_set[:, 0].min() - 1,
stop=X_set[:, 0].max() + 1, step=0.01),
np.arange(start=X_set[:, 1].min() - 1,
stop=X_set[:, 1].max() + 1, step=0.01))

plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(),
X2.ravel()]).T).reshape(X1.shape), alpha=0.75,          cmap=ListedColormap(('yellow',
'white', 'aquamarine')))

plt.xlim(X1.min(), X1.max()) plt.ylim(X2.min(),
X2.max())

for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                color=ListedColormap(('red', 'green', 'blue'))(i), label=j)

plt.title('Logistic Regression (Training set)')
plt.xlabel('PC1')  # for Xlabel
plt.ylabel('PC2')  # for Ylabel plt.legend()
# to show legend
```

 # show scatter plot plt.show()

Output :-

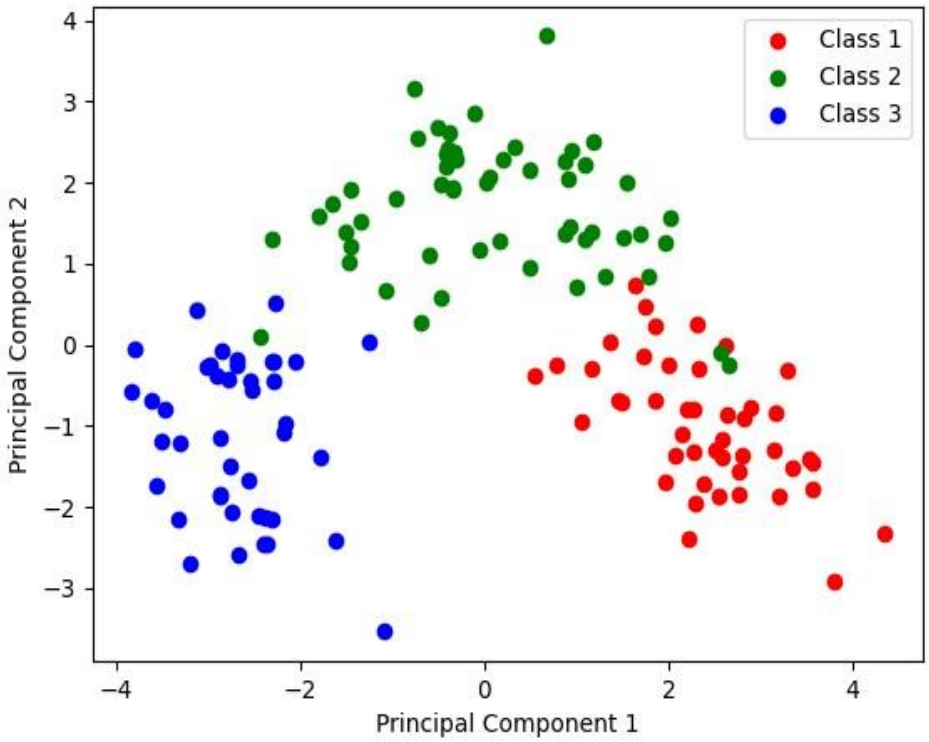


Step 10: Visualizing the Test set results

```
# Visualising the Test set results through scatter plot
X_set, y_set = X_test, y_test

X1, X2 = np.meshgrid(np.arange(start=X_set[:, 0].min() - 1,
stop=X_set[:, 0].max() + 1, step=0.01),
np.arange(start=X_set[:, 1].min() - 1,
stop=X_set[:, 1].max() + 1, step=0.01))

plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(),
                                                  X2.ravel()]).T).reshape(X1.shape), alpha=0.75,
cmap=ListedColormap(('yellow', 'white', 'aquamarine')))

plt.xlim(X1.min(), X1.max()) plt.ylim(X2.min(),
X2.max())
```

```
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                color=ListedColormap(('red', 'green', 'blue'))(i), label=j)

# title for scatter plot plt.title('Logistic
Regression (Test set)')
plt.xlabel('PC1')  # for Xlabel
plt.ylabel('PC2')  # for Ylabel
plt.legend() # show scatter plot
plt.show()
```

Output:-



The output of this code will be a scatter plot of the first two principal components and their explained variance ratio. By selecting the appropriate number of principal components, we can reduce the dimensionality of the dataset and improve our understanding of the data.

Chapter 4

Results and Discussion

Introduction

This chapter presents and analyzes the findings by outlining the evaluation metrics used to assess the effectiveness of Principal Component Analysis (PCA), such as the explained variance ratio, to provide an accurate understanding of the model's performance. Subsequently, the analysis of results is presented through visual representations of the data using tools such as scree plots and biplots, offering a clear depiction and interpretation of the findings. Finally, the discussion section delves deeper into interpreting these results, drawing comparisons with previous studies or theoretical expectations, thereby situating the findings within their scientific context and enhancing their overall comprehension.

4.1 Evaluation Metrics

To evaluate the effectiveness of Principal Component Analysis (PCA), several metrics are commonly utilized to ensure the method's reliability and accuracy. In this section, we will explore the most significant evaluation metrics.

4.1.1 Explained variance ratio

Explained variance ratio is a measure of the proportion of the total variance in the original dataset that is explained by each principal component. The explained variance ratio of a principal component is equal to the ratio of its eigenvalue to the sum of the eigenvalues of all the principal components.

In Sklearn PCA, the explained variance ratio of each principal component can be accessed through the explained_variance_ratio_ attribute. For example, if pca is a Sklearn PCA object, pca.explained_variance_ratio_[i] gives the explained variance ratio of the i-th principal component.

The total explained variance ratio of a set of principal components is simply the sum of the explained variance ratios of those components. In Sklearn PCA, the total explained variance ratio of the first k principal components can be accessed through the explained_variance_ratio_ attribute as follows: pca**.**explained_variance_ratio_**[:**k**].**sum**()**

Example:

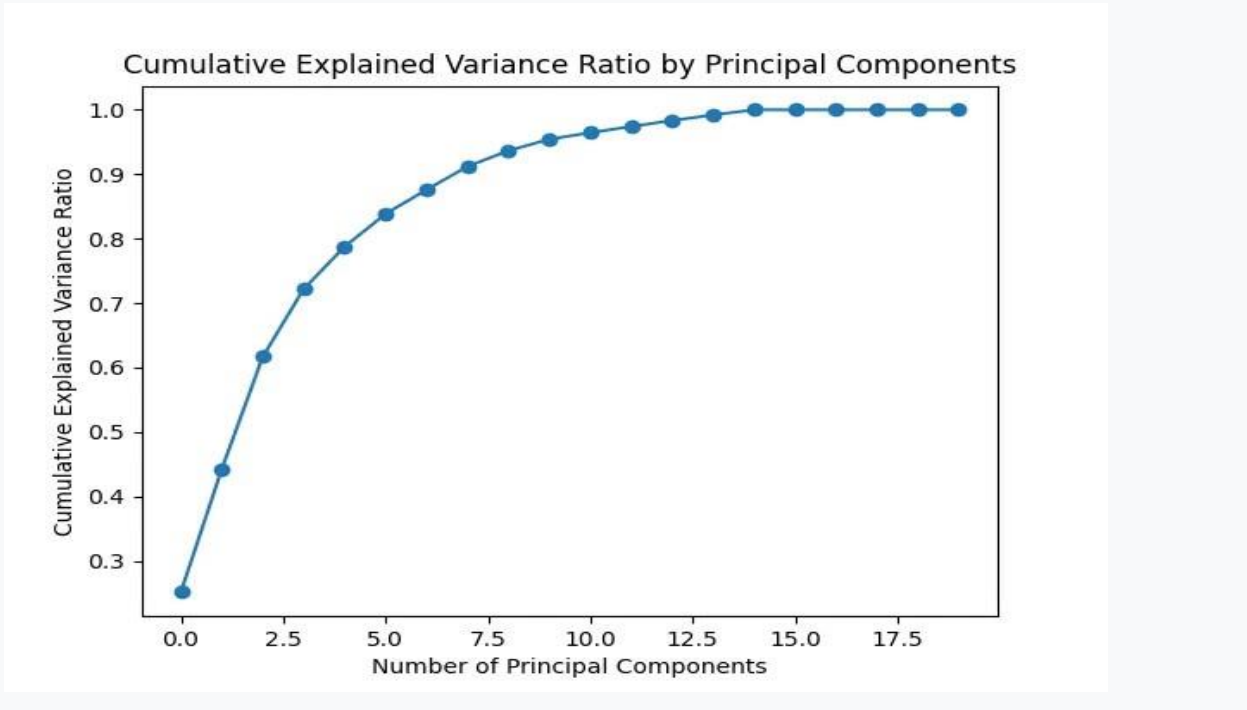
```
# Explained variance ratio explained_variance_ratio = 
pca.explained_variance_ratio_ total_explained_variance_ratio = 
explained_variance_ratio.sum()
```

```
# Print results print(f"\nExplained Variance
Ratio:\n{explained_variance_ratio}") print(f"Total Explained Variance Ratio:
{total_explained_variance_ratio:.4f}")
```

Output:

```
Explained Variance Ratio:
[2.51381826e-01 1.91027743e-01 1.74173740e-01 1.05213977e-01
 6.57562016e-02 5.09055218e-02 3.74919920e-02 3.60336202e-02  2.45297254e-
02 1.77444536e-02 1.02026854e-02 9.58555726e-03
 9.27050381e-03 8.67264583e-03 8.00980652e-03 2.42430646e-32
 2.01048672e-32 1.09528969e-32 4.45708476e-33 2.20001505e-33]
Total Explained Variance Ratio: 1.0000
```

```
# Import necessary libraries
import numpy as np import
matplotlib.pyplot as plt

# Plot explained variance ratio cumulative_variance_ratio =
np.cumsum(explained_variance_ratio)
plt.plot(cumulative_variance_ratio, marker='o') plt.xlabel('Number of
Principal Components') plt.ylabel('Cumulative Explained Variance
Ratio') plt.title('Cumulative Explained Variance Ratio by Principal
Components') plt.show()
```



The plot above illustrates the cumulative explained variance ratio by the number of principal components. Each point on the curve represents the cumulative proportion of total variance explained as we incrementally add principal components. This visualization can help in determining the optimal number of principal components to retain, balancing the goal of dimensionality reduction with preserving a sufficient amount of information from the original dataset.

Interpreting Explained Variance and Explained Variance Ratio :

- Explained variance and explained variance ratio are both measures of how much of the total variance in the original dataset is explained by each principal component. However, they differ in their interpretation.

- Explained variance measures the absolute amount of variance that is explained by each principal component. This is useful when you want to know how many principal components you need to retain in order to capture a certain percentage of the total variance in the original dataset.

- Explained variance ratio measures the relative amount of variance that is explained by each principal component. This is useful when you want to know how much information each principal component contributes to the overall structure of the data.

For example, suppose you have a dataset with 100 features, and you perform Sklearn PCA with 10 principal components. The total explained variance of the first 10 principal components is 80%, which means that these 10 principal components capture 80% of the total variance in the original dataset.

Suppose further that the explained variance ratio of the first principal component is 50%. This means that the first principal component explains 50% of the total variance in the original dataset, and is therefore the most important principal component. The second principal component might have an explained variance ratio of 20%, which means that it explains 20% of the total variance in the original dataset, and is therefore less important than the first principal component

4.1.2 Sampling adequacy (KMO Statistic)

The Kaiser-Meyer-Olkin (KMO) statistic is a measure used to evaluate sampling adequacy and determine whether the data is suitable for factor analysis or Principal Component Analysis (PCA). It does so by examining the relationships between variables, focusing on correlations and partial correlations. Here is a detailed explanation:

Purpose of KMO:

- The KMO statistic assesses the degree of shared variance among variables compared to their unique variance.
- A high KMO value indicates that variables share sufficient common variance, making factor analysis or PCA appropriate. Conversely, a low KMO value suggests that the dataset is not suitable for these techniques, as there is limited shared variance.

Individual and Overall KMO:

- Each variable in the dataset has its own individual KMO statistic, which reflects its adequacy for inclusion in the model.
- The **overall KMO statistic** is the aggregate measure, calculated as the sum of individual KMO values, representing the suitability of the entire dataset for analysis.

KMO Value Range and Interpretation:

- KMO values range from **0 to 1.0**:
  - **KMO ≥ 0.90:** Excellent sampling adequacy.
  - **0.80 ≤ KMO < 0.90:** Good sampling adequacy.

        - **0.70 ≤ KMO < 0.80:** Average sampling adequacy.
        - **0.60 ≤ KMO < 0.70:** Marginal sampling adequacy (acceptable threshold for factor analysis).
        - **KMO < 0.50:** Unacceptable, indicating that factor analysis or PCA may not be appropriate.
    - If the KMO overall statistic is below 0.60, it is necessary to improve it by identifying and removing variables with the lowest individual KMO values.

Improving the KMO Statistic:

- To improve the overall KMO, variables with low individual KMO statistics can be dropped. These variables often exhibit high multicollinearity, which interferes with the identification of distinct factors.
- This iterative process continues until the overall KMO value exceeds the threshold of 0.60.

Calculation of the KMO Statistic:

- The **numerator** in the KMO formula is the sum of squared correlations for all variables in the dataset (excluding self-correlations, which are always 1.0).
- The **denominator** is the sum of squared correlations plus the sum of squared partial correlations between each pair of variables (ii and jj), controlling for all other variables.
- The logic behind this calculation is that small partial correlations indicate that the variables share sufficient common variance, making it possible to extract distinct and meaningful factors during analysis.

Conceptual Basis:

- The KMO statistic relies on the idea that partial correlations between variables should not be large if the data is suitable for factor analysis. Large partial correlations suggest that the variables do not group well into factors, undermining the utility of factor analysis or PCA.

4.1.3 Residual Correlation Analysis in Principal Component Analysis (PCA)

Residual correlation analysis is a tool used to evaluate the quality of a model resulting from the application of Principal Component Analysis (PCA). This analysis focuses on measuring the residual relationships between the original variables after

dimensionality reduction, helping to determine the extent of information loss or unexplained variance in the model.

Steps for Residual Correlation Analysis:

1. **Calculate the Original Correlations:**
    - First, compute the correlation matrix among all variables in the original dataset before applying PCA.
    - This matrix reflects the complete relationships between the variables.
2. **Reconstruct Data Using Principal Components:**
    - After selecting a specific number of principal components based on the explained variance ratio, reconstruct the original dataset using these components.
    - Compute the correlation matrix for the reconstructed data.
3. **Calculate the Residual Correlation Matrix:**
    - Residual Correlation Matrix = (Original Correlation Matrix) - (Correlation Matrix from Reconstructed Data).
    - This matrix reflects the amount of relationship that remains unexplained by the selected principal components.

Interpreting Residual Correlation Analysis:

1. **Values Close to Zero:**
    - If the values in the residual correlation matrix are small (close to zero), it indicates that the selected principal components explain most of the relationships between variables.
    - This is considered a sign of a high-quality model.
2. **Large Values:**
    - If the residual correlation values are high, it suggests that significant information has not been explained by the principal components.
    - This may indicate the need to select more principal components or reconsider the suitability of PCA for the dataset.
3. **Residual Patterns:**

- Analyzing the residual patterns in the matrix can reveal variables or groups of variables that may require further processing (e.g., noise reduction or transformation).

Significance of Residual Correlation Analysis in PCA:

Evaluating Information Loss:

- Helps measure the amount of information ignored during dimensionality reduction.
- Ensures that the model retains most of the useful variance in the data.

Selecting the Number of Principal Components:

- Aids in determining the optimal number of principal components to retain, achieving a balance between simplification and information loss.

Ensuring the Suitability of PCA:

- If the residual correlation matrix shows that the original relationships between variables are not adequately explained, it may indicate that PCA is not the most suitable method for the data.

Practical Example:

- Suppose the original dataset contains 10 variables, and PCA reduces them to 3 principal components.
- After reconstructing the data using the three components, the residual correlation matrix is calculated.
- If the residual values in the matrix are small (e.g., less than 0.1), this indicates that the analysis was successful.
- If the residual values are large (e.g., between 0.2 and 0.5), this indicates significant information loss, which may affect the model's accuracy.

4.2 Results Analysis in PCA

This section provides an analysis of the results obtained from the Principal

Component Analysis (PCA), highlighting key findings and interpreting visualizations that offer insights into the structure of the data. The main tools for presenting results include scree plots and biplots, which are essential for understanding the variance explained by the components and the relationships among variables and observations.

Key Results Presentation

4.2.1 Scree Plot:

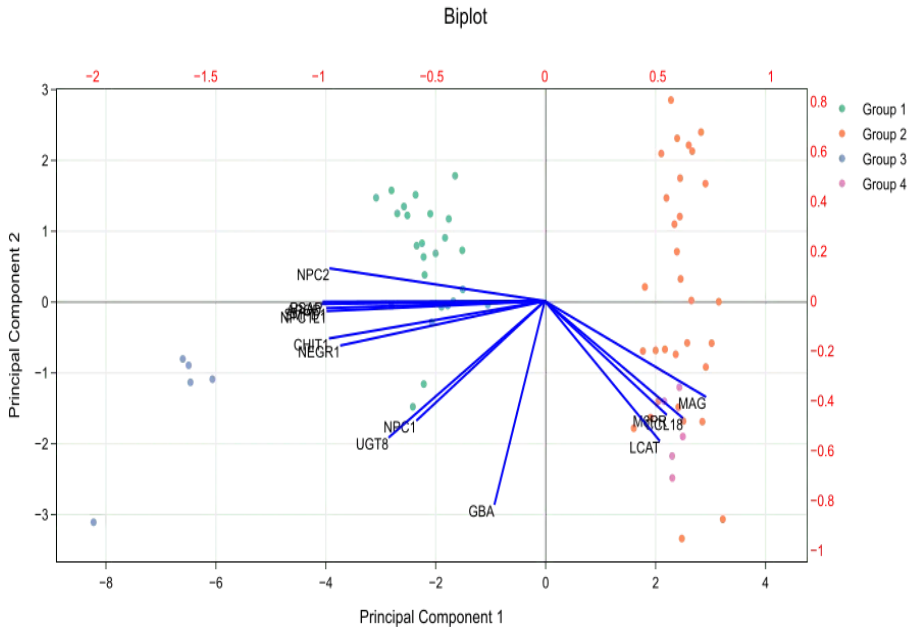
A common method for determining the number of PCs to be retained is a graphical representation known as a scree plot. A Scree Plot is a simple line segment plot that shows the eigenvalues for each individual PC. It shows the eigenvalues on the y-axis and the number of factors on the x-axis. It always displays a downward curve. Most scree plots look broadly similar in shape, starting high on the left, falling rather quickly, and then flattening out at some point. This is because the first component usually explains much of the variability, the next few components explain a moderate amount, and the latter components only explain a small fraction of the overall variability. The scree plot criterion looks for the “elbow” in the curve and selects all components just before the line flattens out. (In the PCA literature, the plot is called a ‘Scree’ Plot because it often looks like a ‘scree’ slope, where rocks have fallen down and accumulated on the side of a mountain.)

When the eigenvalues drop dramatically in size, an additional factor would add relatively little to the information already extracted

4.2.2 Biplot:

A biplot combines information about the observations and variables in a single visualization.

- Observations are represented as points.
- Variables are represented as arrows, with their length indicating their contribution to the principal components.
- **Interpretation:**
- The direction of the arrows indicates the correlation between variables.
- Points closer together represent observations with similar profiles, while those farther apart are dissimilar.



Variables with arrows pointing in similar directions are positively correlated, whereas those pointing in opposite directions are negatively correlated.

4.3 Interpretation of Results

4.3.1 Dimensionality Reduction:

- Based on the scree plot, it was determined that the first three components account for approximately 88% of the variance, making them sufficient for analysis.
- This significant reduction in dimensionality ensures that most of the information is retained while simplifying the dataset.

4.3.2 Variable Relationships:

- The biplot shows that certain variables are highly correlated (e.g., variables X1 and X2 with PC1).
- Variables with shorter arrows contribute less to the explained variance and may be less significant in the context of the dataset.

4.3.3 Cluster Identification:

- Patterns in the biplot may reveal clusters of observations, indicating groups with similar characteristics.
- For example, observations in quadrant I may represent a distinct subgroup influenced by variables strongly associated with PC1 and PC2.

Conclusion

The results of PCA demonstrate a clear reduction in dimensionality with minimal loss of information. The scree plot and biplot offer valuable insights into the structure of the data, enabling better understanding and interpretation of relationships among variables and observations. This analysis aids in identifying the most significant components and their contributions, forming a solid basis for subsequent analysis or decision-making.

Chapter 5

Applications of PCA

Introduction

In this chapter, we will explain a case study as an example of applying the Principal Component Analysis (PCA) technique.The experiment involves using dimensionality reduction techniques like normal PCA and its variants to detect malware in Android applications using the CICMalDroid_2020 dataset

Additionally, we will briefly discuss a comparative study of PCA, Independent Component Analysis (ICA), and Linear Discriminant Analysis (LDA) on the FERET dataset. This comparison was conducted under uniform working conditions using the standard FERET dataset.

5.1 Case study on variants of PCA in mobile malware detection

To explore the dimensionality reduction using normal PCA and its variants for mobile malware detection in the CICMalDroid_2020 dataset [11] are experimented. The dataset is taken from the University of New Brunswick (UNB), Canadian Institute of Cyber Security (CIC). The dataset consists of 11,598 records with 471 feature attributes. The dataset commences with 17,341 Android samples gathered from VirusTotal, the Contagio security blog, AMD, MalDozer, and other sources. Samples

were obtained between December 2017 and December 2018. Detecting Android apps with malicious data points is crucial for cyber security specialists. There are five key categories in the dataset: Adware, Banking malware, SMS malware, Riskware, and Benign are the different forms of malicious software. The experiment is carried out in a Python Jupyter notebook environment using sklearn library [12, 13, 14, 15, 16].

5.1.1 Results of PCA in machine learning for mobile malware detection

The following are the outcomes of normal PCA and variants of PCA where the 471 feature dimensions are reduced into two PCs are visualized below. Figure 2 shows the importing data into the Python Jupyter notebook environment

```
df = pd.read_csv("C:/Users/Acer/Desktop/feature_vectors_syscallsbinders_frequency_5_Cat.csv")
#df = pd.read_csv(r"C:\Users\hp\Desktop\Malware dataset_CCI.csv", encoding = 'utf-8')
df.shape
```

```
(11598, 471)
```

Figure 2.

Data import—CICMalDroid_2020 dataset.

Figure 3 shows the data pre-processing to check whether the data contains any null values or not. Data pre-processing is an important step in the machine learning process,

and it helps to purify the irrelevant and undefined raw data into the relevant defined form.

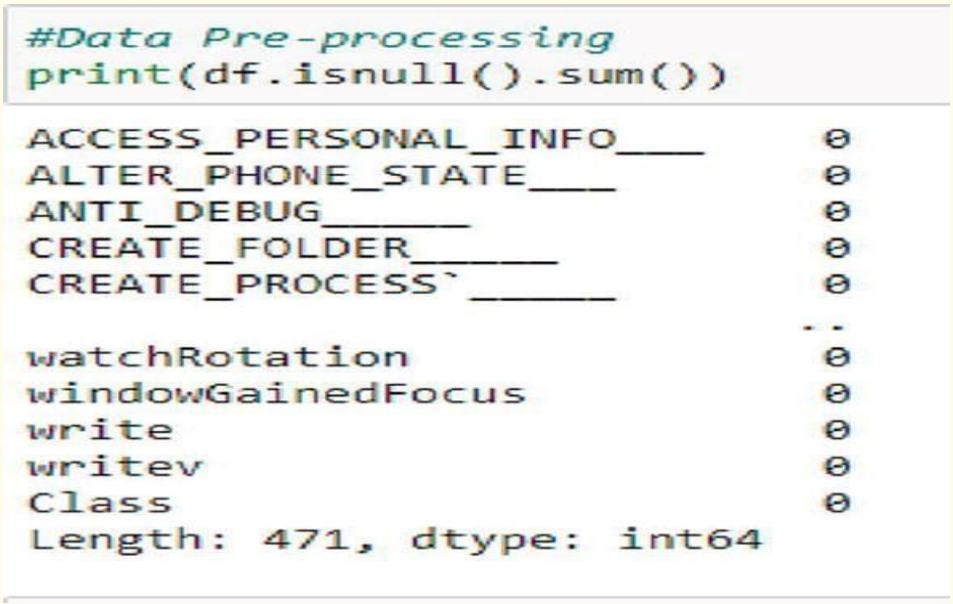


Figure 3.

Check if the data contains any null values or not.Figure 3 depicts that the dataset does not contain any null values, and it is fit for further processing (i.e.) from the results value

‘0’indicates there are no null values in the data Figure 4 shows the removal of duplicate data values to ensure the originality of the dataset. Duplicate data leads to

misinterpretation of the results.

```
#to check for any duplicate data values
df2 = df1.drop_duplicates()
df2.shape
```

```
(11526, 471)
```

Figure 4.

Drop duplicate values.Figure 4 depicts that out of 11,598 records, 72 were duplicated,

and the duplicated records were dropped. After dropping the 72 duplicate records, now the dataset consists of 11,526 instances with 471 features.

Figure 5 shows the data splitting for training and testing so that the machine learning model can detect and cluster the mobile malware data points in the dataset. Splitting the

data for training and testing is a significant phase in the machine learning process. So, the data will be adequately trained and provide the best results in testing, which helps to

derive a high efficacy rate.

```
#Splitting data for training and testing
X = df2.iloc[:,:-1].values #features
Y = df2['Class'].values #target variable
```

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.3,random_state = 0)
print("Training samples: {}; Test samples: {}".format(len(X_train), len(X_test)))
```

```
Training samples: 8068; Test samples: 3458
```

Figure 5.

Train and test split.Figure 5 explains that the dataset is divided into 70% for training and

30% for testing (i.e.) out of 11,526 records, 8068 are used for training and 3458

samples are used for testing the model. Now, the dataset is suitable to perform the PCA with the machine learning technique.

Figure 6 shows the feature scaling; before applying PCA, one must scale the data so that the data can be properly scaled within a particular range appropriately to support

data modelling. Without incorporating feature scaling, during the model development the data takes more time to fit into the prescribed model form.

```
#Feature Scaling
from sklearn.preprocessing import MinMaxScaler
ms = MinMaxScaler()
X = ms.fit_transform(X)
```

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
X = sc.fit_transform(X)
```

Figure 6.

Feature scaling.Figure 6, depicts the method for feature scaling using MinMaxScaler and standard scalar to bring the scattered data points within a typical specified range.

Hence, the data is further applicable for PCA.

Figure 7 shows the normal PCA method used in the dataset to reduce the 471 featured dimensions into two PCs. It also shows that from the explained variance PC1 has more

information than PC2. Normal PCA is the default form of PCA, more suitable for all kinds of data to reduce the dimensions effectively without any information loss.

```
# Applying PCA function on training
# and testing set of X component
from sklearn.decomposition import PCA

pca = PCA(n_components = 2)

X_train = pca.fit_transform(X_train)
X_test = pca.transform(X_test)

explained_variance = pca.explained_variance_ratio_
print(explained_variance)
```

```
[0.70241931 0.29758069]
```

Figure 7.

Normal PCA.Figure 8 shows the normal PCA results of two PCs as PC1 and PC2.

	Principal component 1	Principal component 2
8113	-3.371104	0.361204
8114	-2.592353	-0.461123
8115	3.150280	1.218977
8116	8.846860	5.503562
8117	-2.846843	-1.267918

Figure 8.

Normal PCA with PC1 and PC2.Figure 8 represents that the total 471 features are reduced into two PCs PC1 and PC2, without removing any of the data features. This

helps to train the data and develop the machine learning model effectively with less memory consumption.

42

Figure 9 shows the method of sparse PCA in mobile malware data. Similarly, the sparse PCA is widely suited for sparse data so that the 471 data features are reduced into two set of PCs PC1 and PC2.

```
#Sparse PCA
from sklearn.decomposition import SparsePCA

spca = SparsePCA(n_components=2, alpha=0.0001)
X_train = spca.fit_transform(X_train)
X_test = spca.transform(X_test)

#explained_variance = spca.explained_variance_ratio_
#print(explained_variance)
```

```
principal_data_Df = pd.DataFrame(data = X_train
             , columns = ['Principal component 1', 'Principal component 2'])
principal_data_Df
```

	Principal component 1	Principal component 2
0	-24.650157	13.666278
1	-33.683180	17.554447
2	-0.555096	-4.540046
3	2.199703	0.124262
4	3.263998	-0.028142
...	...	...
8063	6.213232	9.590552
8064	2.451784	0.380245
8065	-4.936274	-9.065537
8066	-2.058104	-1.706935
8067	1.720193	0.536436

8068 rows × 2 columns

Figure 9.

Sparse PCA.Figure 10 shows the method of randomized PCA in mobile malware data. Randomized PCA is suitable for big data processing so that the features are randomly selected to derive the two set of PCs PC1 and PC2.

```
#Randomized PCA
from sklearn.decomposition import PCA

rpca = PCA(n_components=2, svd_solver='randomized')
X_train = rpca.fit_transform(X_train)
X_test = rpca.transform(X_test)

explained_variance = rpca.explained_variance_ratio_
print(explained_variance)
```

```
[0.70241931 0.29758069]
```

```
principal_data_Df = pd.DataFrame(data = X_train
             , columns = ['Principal component 1', 'Principal component 2'])
principal_data_Df
```

	Principal component 1	Principal component 2
0	-2.783878	-0.664049
1	-2.187618	-0.819019
2	-0.833045	0.270127
3	23.384158	-12.316224
4	-3.639008	-2.031355
...	...	...
8113	-3.207486	0.341588
8114	-2.466535	-0.438440
8115	2.997384	1.160743
8116	8.417485	5.237434
8117	-2.708675	-1.205984

8118 rows × 2 columns

Figure 10.

Randomized PCA.Figure 11 shows the method of incremental PCA in mobile malware data. Incremental PCA is similar to randomized PCA, but it gradually increases the

batch size to reduce the total number of features into two set of PCs PC1 and PC2.


```
#Incremental PCA
from sklearn.decomposition import IncrementalPCA

ipca = IncrementalPCA(n_components=2, batch_size=200)
X_train = ipca.fit_transform(X_train)
X_test = ipca.transform(X_test)

explained_variance = ipca.explained_variance_ratio_
print(explained_variance)
```

```
[0.70241931 0.29758069]
```

```
principal_data_Df = pd.DataFrame(data = X_train
             , columns = ['Principal component 1', 'Principal component 2'])
principal_data_Df
```

	Principal component 1	Principal component 2
0	-2.783878	-0.664049
1	-2.187618	-0.819019
2	-0.833045	0.270127
3	23.384158	-12.316224
4	-3.639008	-2.031355
...	...	...
8113	-3.207486	0.341588
8114	-2.466535	-0.438440
8115	2.997384	1.160743
8116	8.417485	5.237434
8117	-2.708675	-1.205984

8118 rows × 2 columns

Figure 11.

Incremental PCA.

Figure 12 shows the method of kernel PCA in mobile malware data. Kernel PCA is

widely applicable for non-linear data modelling. It also helps to reduce the dimensions of the data based on the kernel function like gamma etc. to derive the two or more sets of

PCs PC1and PC2.

```
#Kernel PCA
kpca = KernelPCA(kernel='rbf', gamma=1)
X_kpca = kpca.fit_transform(X_train)

plt.title("Kernel PCA")
plt.scatter(X_kpca[:,0], X_kpca[:,1], c=y_train)
plt.show()
```

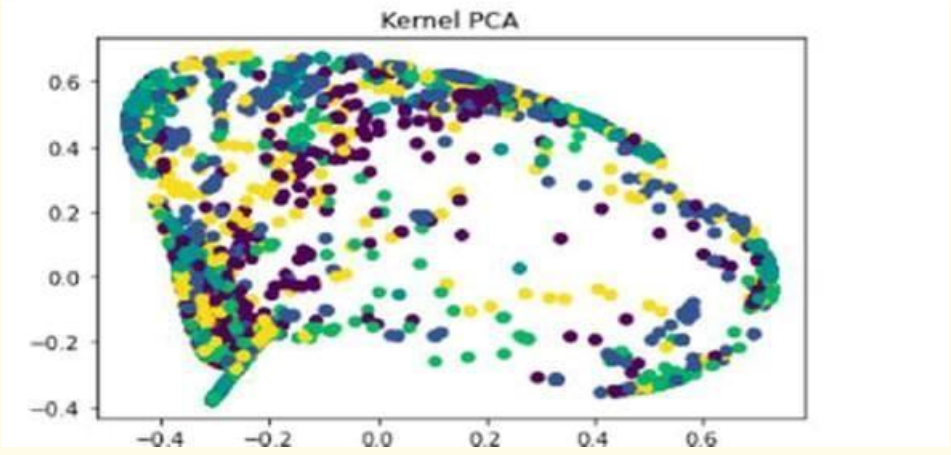


Figure 12.

Kernel PCA.In this case study, the 471 data features of CICMalDroid_2020 dataset is

transformed into two sets of PCs PC1 and PC2 for all variants of PCA, depending upon the suitability of the data values.

5.1.2 Observations

Based on the results obtained from the variants of PCA for mobile malware detection

depicts that for the CICMalDroid_2020 dataset is discussed. It is a numerical labelled data that highly supports normal standard PCA technique. It helps to reduce the

dimensions of the PCA from 471 features into two sets of PCs (PC1 and PC2). It supports processing the data model quickly and forms the clusters effectively. Figure 13

shows the group of clusters that discovered the five different malware involved in the

CICMalDroid_2020 dataset based on PC1 containing huge information about the

dataset of normal PCA.6.2 Observations

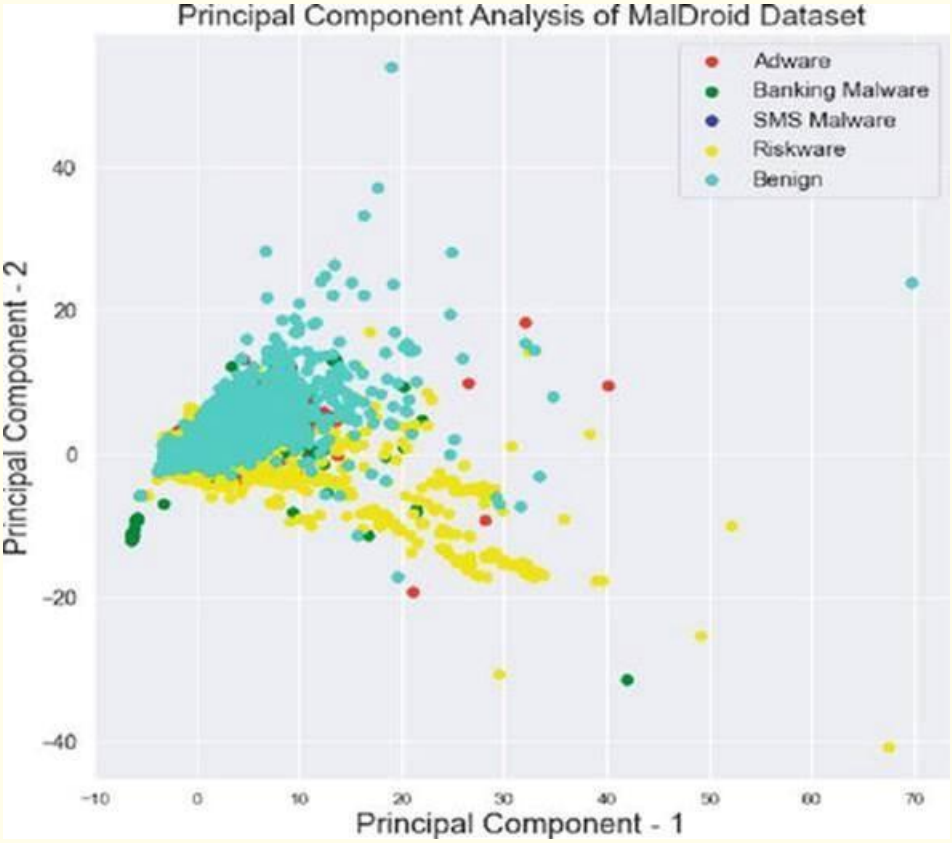


Figure 13.

Class of malwares in the CICMalDroid_2020 dataset.Table 1 shows the size of malware

samples available under the category of Adware, Banking malware, SMS malware, Riskware, and Benign.

Class	Type of malware	Size
1	Adware	1253
2	Banking malware	2100
3	SMS malware	3904
4	Riskware	2546
5	Benign	1795

Table 1.

CICMalDroid_2020 dataset—sample size

Hence, the other variants of PCA, such as sparse PCA are applicable for sparse data, randomized PCA and incremental PCA are suitable for big data processing and kernel

PCA is widely supported by non-linear data modelling. So, depending upon the type of data and their accessibility, the appropriate type of PCA is incorporated into machine

learning algorithms specifically for unsupervised learning for dimensionality reduction to develop a suitable **predictive model. It also helps to identify the hidden patterns of**

the data effectively

5.2 Independent Comparative Study of PCA, ICA, and LDA on the FERET Dataset

Introduction:

Face recognition is one of the most prominent applications of image analysis and understanding, with numerous algorithms proposed yielding varying results in previous studies. This study aims to provide an independent comparison of three of the most widely used face representation methods: Principal Component Analysis (PCA), Independent Component Analysis (ICA), and Linear Discriminant Analysis (LDA). The comparison was conducted under uniform working conditions using the standard FERET dataset.

Studied Algorithms:

1. **Principal Component Analysis (PCA):** Extracts the most distinguishing features from images by identifying projection vectors that retain the highest variance in the data.
2. **Independent Component Analysis (ICA):** Captures both second-order and higher-order statistics, using statistically independent basis vectors.
3. **Linear Discriminant Analysis (LDA):** Focuses on maximizing between-class variance while minimizing within-class variance.

Methodology:

- The study used the FERET dataset with its standard test sets (fb, fc, dup1, and dup2).
- The same preprocessing steps were applied to all images, including resizing, rotation, and histogram equalization.
- The performance of the algorithms was evaluated using four distance metrics: L1 (Manhattan), L2 (Euclidean), Cosine (COS), and Mahalanobis.
- PCA was trained using 675 images from 225 individuals, retaining the top 40% of eigenvalues (270 dimensions). The same projection was used for ICA and LDA, but LDA was limited to 224 dimensions due to theoretical constraints.

Results and Analysis:

- No single algorithm outperformed others in all scenarios; performance depended on the specific task.
- **Expression Variation Task (fb):** ICA2 + COS performed best at rank 1, but LDA + COS surpassed it at higher ranks.
- **Illumination Variation Task (fc):** PCA + L1 was the best performer from rank 17 onward, while ICA1 performed poorly in this task.
- **Temporal Changes Task (dup1 and dup2):** ICA2 + COS was the best overall, and LDA performed worse than expected.
- **Distance Metrics Comparison:** L1 performed best with PCA, COS was optimal for ICA2, and L2 was not the best choice in most cases.

Statistical Difference Analysis: McNemar's test was used to evaluate the statistical significance of performance differences between algorithms. The results showed that some differences were not statistically significant, highlighting the importance of caution when comparing classification results based solely on rank 1.

Comparison with Previous Studies:

- The results were consistent with Phillips et al. (2000) regarding the ranking of test sets, where fb was the easiest and dup2 the hardest.
- The study confirmed Bartlett et al. (2002)'s findings that ICA2 outperforms PCA for temporally separated images but did not support its superiority in expression variation tasks.
- Unlike some previous studies, LDA did not consistently outperform PCA, indicating its sensitivity to training data selection.

Conclusion: This study demonstrated that optimal performance depends on the nature of the task. ICA2 + COS was best for temporal changes, while PCA + L1 was effective for illumination variations. Choosing the appropriate distance metric for each algorithm is crucial. The findings emphasize the need for statistical testing when comparing face recognition algorithms to ensure accurate evaluation.

The References:

1/ Principal Component Analysis

Second Edition

2/Advance in Principal companent Analysis

//3/https://www.geeksforgeeks.org/principal-component-analysis-with-python

4 /Independend Comparative Study of PCA, ICA, and LCA on the FERET Dataset

Kresimir Delac, Mislav Grgiv, Sonja Grgic

University of Zagreb, FER, Unska 3/XII, Zagreb, croatia

Received 28 December 2004, Accepted 27 Febrary 2006