Reducing dimensionality with principal component analysis with R

Optimize the classification of a data set by applying PCA with R

By Joshua Noble

04 August 2024

Principal component analysis (usually called PCA) is a technique for dimensionality reduction. Dimensionality reduction is the process of decreasing the number of features in a data set by identifying the most critical variables. This is done either by feature selection or feature extraction. Feature selection involves selecting the critical features by filtering out the unimportant ones or outliers. Feature extraction involves creating new and relevant features from the original data set. PCA is widely used in machine learning and data analysis to locate relationships between features and reduce the amount of data required for analysis and modeling.

In this tutorial, we'll implement PCA in R using Jupyter Notebooks on IBM watsonx.ai using the Iris data set . The goal is to classify three species of iris flowers: Iris setosa, Iris versicolor, and Iris virginica. We'll start by exploring and standardizing the data set, and performing PCA. We'll then analyze and visualize the PCA results to showcase how PCA can be a valuable tool for dimensionality reduction in the context of classification. Finally, we'll apply the Naive Bayes classifier to the original data and the efficiency.

In this tutorial, we'll focus on using PCA for feature extraction. The analysis will identify combinations of the original features that account for the highest variance within the data set. The PCA algorithm transforms the original features of the data set into a new set of variables called principal components that capture the most significant variance. These principal components are orthogonal and not correlated with each other. In essence PCA helps transform high-dimensional data into a lower-dimensional form while preserving as much variance as possible.

More about PCA

PCA is an unsupervised learning technique that does not consider class labels or categories when reducing dimensions. However, a key advantage of PCA is that it is not limited to supervised or unsupervised tasks but can be applied to data preprocessing and graphical visualization.

The PCA computation process can be thought of as five key steps:

1. Normalize the range of initial variables
2. Compute the covariance matrix to identify correlations
3. Compute the eigenvectors and eigenvalues of the computed covariance matrix
4. Select the principal components
5. Transform the data into the new coordinate system

There are a few key reasons you might want to use PCA:

- When working with high-dimensional data you may need to reduce number of features to speed up analysis and training.

- When you want to visualize high-dimensional data you can use PCA to project it into a 2D or 3D space and then visualize it with a scatterplot. This can significantly simplify data interpretation and exploration.
- PCA can remove noise, un-necessary complexity, or redundant information from data by focusing on the principal components that capture underlying patterns.

There are two methods to implement PCA in R programming: spectral decomposition and singular value decomposition (SVD). Spectral decomposition analyzes the covariances or correlations among variables, while SVD scrutinizes the covariances or correlations among individuals.

PCA can be implemented using functions built into the R language or through manual computations. In this tutorial we'll focus on the `prcomp` function which uses a technique called singular value decomposition to create the principal components.

Prerequisites

You need an IBM Cloud account to create a watsonx.ai project.

Steps

Step 1. Set up your environment

While you can choose from several tools, this tutorial walks you through how to set up an IBM account to use a Jupyter Notebook. Jupyter Notebooks are widely used within data science to combine code, text, images, and data visualizations to formulate a well-formed analysis.

2. Create a watsonx.ai project.

3. Create a Jupyter Notebook using the R Kernel.

This step will open a notebook environment where we can import a data set and copy the code from this tutorial to tackle a simple classification problem.

Step 2. Import libraries and load the data set

We'll begin by installing and loading the necessary libraries: FactoMineR , corrr , and ggcorrplot . We'll then load the in-built Iris data set.

```
# install.packages("corrr")
# library('corrr')
#
# install.packages("ggcorrplot")
# library(ggcorrplot)

install.packages("FactoMineR")
library("FactoMineR")

install.packages("factoextra")
library("factoextra")

install.packages("naivebayes")
library(naivebayes)

data(iris)
```

Show more

Step 3. Explore and standardize the data set

When looking at a new data set we should always do some exploratory data analysis to check its structure and content first.

```
#Display the structure of the dataset
str(iris)
```

This will output the structure of the data set:

```
'data.frame':	150 obs. of  5 variables:
 $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9
 $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3
 $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4
 $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2
 $ Species     : Factor w/ 3 levels "setosa","versicolor"
```

Show more

It will also output the summary statistics:

```
  Sepal.Length    Sepal.Width     Petal.Length    Petal.W
 Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :
 1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:
 Median :5.800   Median :3.000   Median :4.350   Median :
 Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :
 3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:
 Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :
```

Show more

We can see the structure of the Iris data set, with 150 observations across five variables: sepal length, sepal width, petal length, petal width, and species. The first four variables are numerical measurements, and the "species" variable categorizes the iris flowers into three species: setosa, versicolor, and virginica. The data set is balanced with 50 observations for each species.

The summary statistics provide a comprehensive overview of the central tendency and spread of the data for each numerical variable. For instance, sepal length ranges from 4.3 to 7.9, averaging 5.843. The data set represents all three iris species, making it suitable for machine learning model training and evaluation.

Let's look at one row in this data set:

- Petal.Length: 1.4
- Petal.Width: 0.2
- Species: setosa

These indicate that the iris flower in this observation has a sepal length of 5.1 cm, a sepal width of 3.5 cm, a petal length of 1.4 cm, and a petal width of 0.2 cm. The iris flower species is setosa.

Let's explore the statistical metrics for `Sepal.Length`:

- **Min. (Minimum)**: The shortest sepal length in the data set is 4.3 cm.
- **1st Qu. (First quartile)**: 25% of the iris flowers have a sepal length less than or equal to 5.1 cm.
- **Median**: The data set's middle value of sepal length is 5.8 cm, indicating that 50% of the iris flowers have a sepal length less than or equal to 5.8 cm.
- **Mean**: The average sepal length across all iris flowers in the data set is 5.843 cm.
- **3rd Qu. (Third quartile)**: 75% of the iris flowers have a sepal length less than or equal to 6.4 cm.
- **Max. (Maximum)**: The longest sepal length in the data set is 7.9 cm.

Standardization or normalization is crucial in PCA, as it aligns variables on the same scale, ensuring comparability. In this step, we'll standardize the numeric variables.

```
#Standardize the numeric variables
iris_scaled <- scale(iris[, 1:4])
```

Show more

Step 4. Implement PCA

the maximum variance in the data.

In this step, we will implement PCA on the standardized data set and determine the principal components, eigenvalues, and how well the generated features capture the variance across the data set.

```
#Implement PCA
iris_pca <- prcomp(iris_scaled)
summary(iris_pca)
```

Show more

This outputs:

```
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
```

Show more

The output summarizes the importance of the four principal components (PC1, PC2, PC3, and PC4):

- **Standard deviation**: This measures the degree of variation or dispersion within the set of values. The standard deviation for PC1 is 1.7084, and for PC2, it is 0.9560. These values indicate that PC1 values vary more than PC2 values.
- **Proportion of variance**: This row indicates the total variance in the data set explained by each principal component. Here, PC1 accounts for 72.96% of the total variance, while PC2 accounts for 22.85%, suggesting that PC1 explains a significant portion of the variance in the data set.
- **Cumulative Proportion**: This row represents the cumulative sum of the proportion of variances. It indicates the total variance explained by all the principal components up to that point. For PC1, the cumulative proportion is the same as the

95.81%, which is the sum of the proportion of variances for PC1 and PC2.

The principal components are actually eigenvectors of the covariance matrix and if we want to view the values of those in the original coordinate system we would look at the rotation property of the PCA.

```
iris_pca$rotation
```

Show more

This returns:

```
                    PC1         PC2        PC3        PC4
Sepal.Length  0.5210659 -0.37741762  0.7195664  0.2612863
Sepal.Width  -0.2693474 -0.92329566 -0.2443818 -0.1235096
Petal.Length  0.5804131 -0.02449161 -0.1421264 -0.8014492
Petal.Width   0.5648565 -0.06694199 -0.6342727  0.5235971
```

Show more

These will be a square matrix with the same number of dimensions that the original data contained. If we want to see how the matrix multiplication modifies all the individual records, we would look at the x property of the PCA result:

```
print(head(iris_pca$x))
```

Show more

The x property which returns the first five records after being modified by the eigenvalues:

```
           PC1        PC2         PC3          PC4
[1,] -2.257141 -0.4784238  0.12727962  0.024087508
[2,] -2.074013  0.6718827  0.23382552  0.102662845
[3,] -2.356335  0.3407664 -0.04405390  0.028282305
[4,] -2.291707  0.5953999 -0.09098530 -0.065735340
```

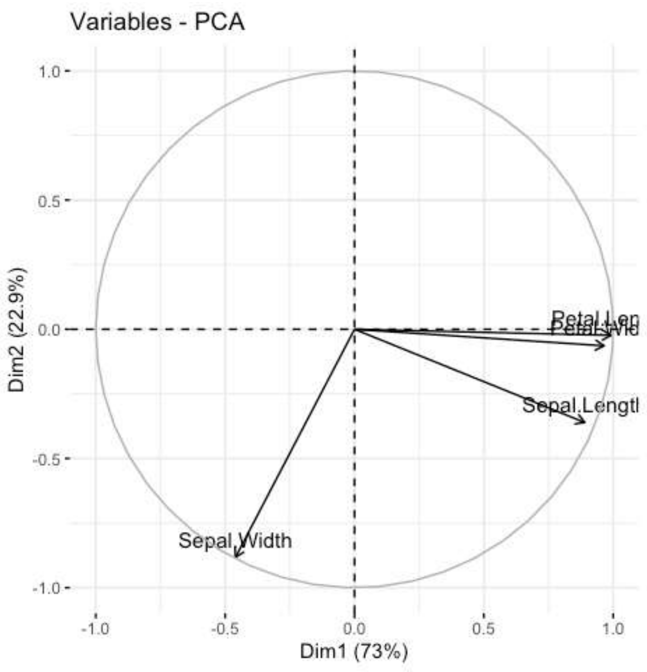
Now we have a generated PCA and we know how well it captures the cumulative variance of the data set. We can now move on to visualizing our calculated principal components and how they change individual records.

Step 5. Visualize PCA results

First, we'll create a graph that depicts how linear combinations of variables contribute to the principal components. Each variable is represented by an arrow pointing in the direction of the principal component, where it exerts the most influence. The length of the arrow indicates the strength of the contribution. The variables that point in the same direction have similar effects on the data.

```
# Plot variables in the PCA space
fviz_pca_var(iris_pca, col.var = "black")
```

Show more



Here's a breakdown of the graph:

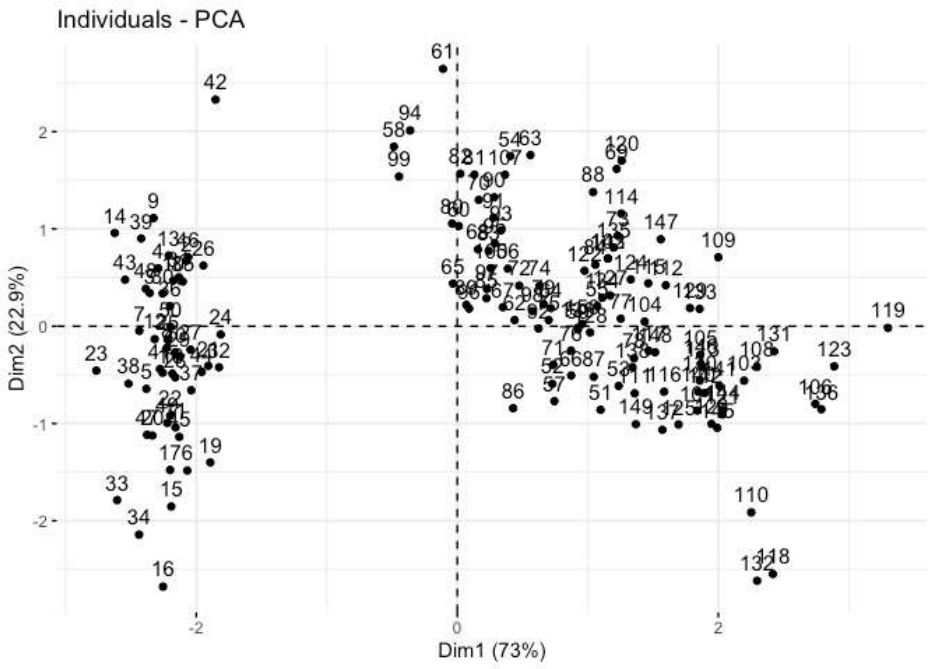
- The axes labeled "Dim 1" and "Dim 2" represent the first two principal components that capture maximum variance in the data. Dim 1 (72.96%) and Dim 2 (22.85%) indicate that these two dimensions together explain about 95.81% of the total variance in the data set.
- Each point on the plot represents a variable from the data set: `Sepal.Width`, `Sepal.Length`, `Petal.Width`, and `Petal.Length`.
- The distance between the points represents the correlation between the variables:
    - Points that are close together are positively correlated.
    - Points that are far apart are not correlated.
    - Points on opposite sides of the plot origin (0,0) are negatively correlated.
- Here, the variable `Sepal.Width` is away from the other three variables. This positioning suggests that `Sepal.Width` is not correlated or is negatively correlated with `Sepal.Length`, `Petal.Width`, and `Petal.Length`. In other words, as the sepal width increases, the other variables remain unaffected or tend to decrease.
- The circle that encompasses all four points is the correlation circle, which indicates the representation quality of the variables on the factor map. Variables close to the circle are well represented on the PCA map.

In conclusion, this graph shows that `Sepal.Width` has a unique behavior compared to the other variables, which could be a vital insight depending on the specific task. For instance, if the goal is to classify the species of the iris flowers, `Sepal.Width` might be a crucial distinguishing feature.

values determine their position along the selected principal components. This visualization will aid in understanding how different observations cluster in the reduced-dimensional space.

```
# Biplot with variables and individuals
fviz_pca_ind(iris_pca)
```

Show more



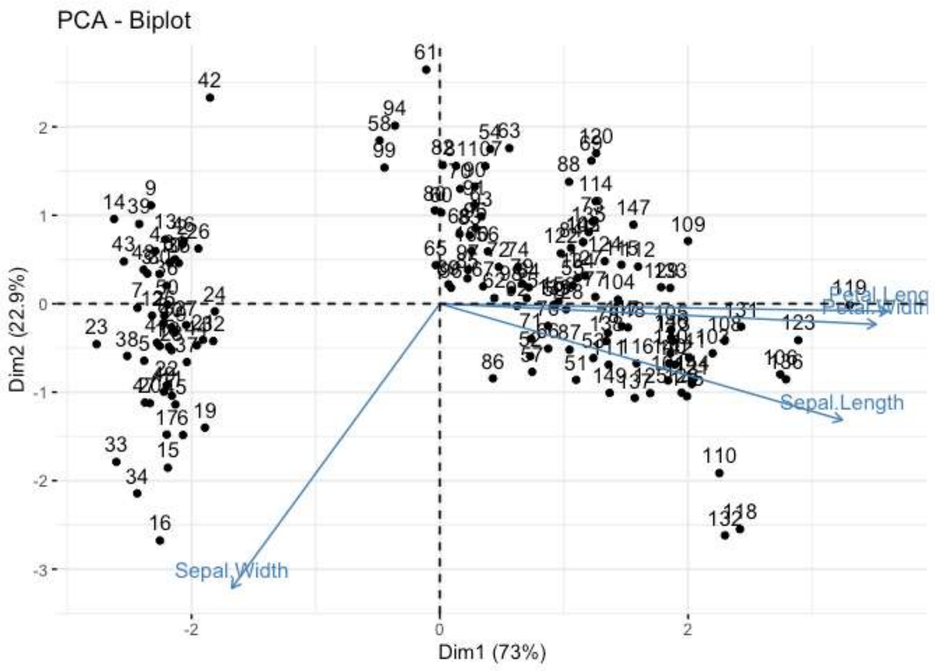
Each point in this PCA plot represents an iris flower from the data set. The axes Dim 1 and Dim 2 represent the first two principal components that capture the maximum variance in the data. By projecting the data onto these principal components, you can visualize the clustering of individual flowers based on their species or other characteristics.

The numerical labels beside each point could represent identification numbers for individual observations. If the species were encoded numerically, these values could also correspond to the different iris flower species. The dashed lines where Dim 1 is 0 and Dim 2 is 0 divide the plot into four quadrants to help visualize the distribution and clustering of the data points.

the relationship between the original features and the principal components and help in interpreting the data set's structure.

```
#Biplot with variables and individuals
fviz_pca_biplot(iris_pca)
```

Show more

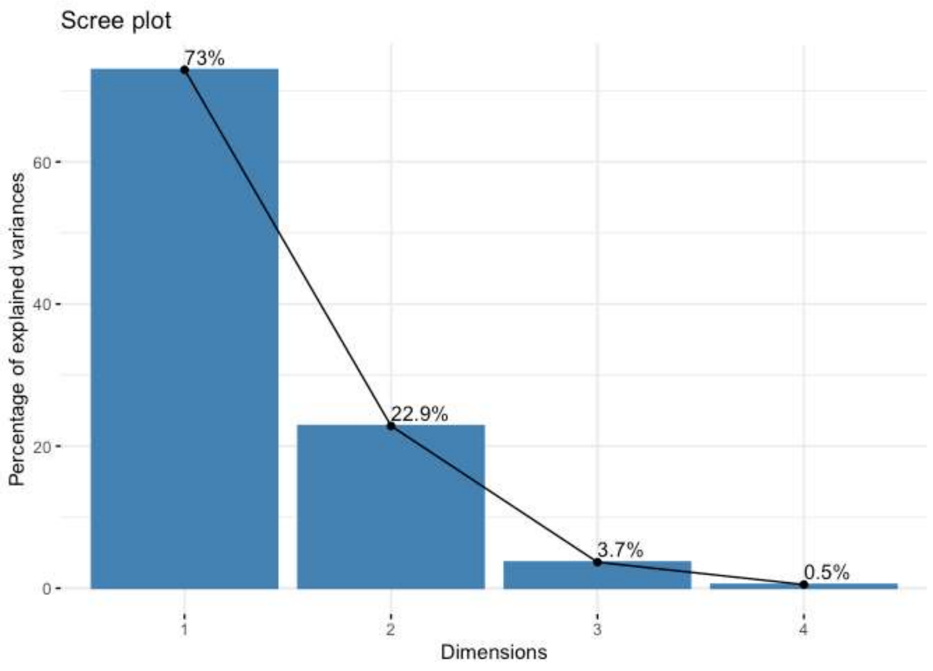


This biplot of the Iris data set shows the relationship between the first two principal components (PC1 and PC2) and the variables in the data set. The blue arrows indicate the direction and magnitude of the variables. The plot shows that Sepal.Length and Sepal.Width are positively associated with PC1 and PC2, respectively. This means that as one variable's value increases, the other variable's value also increases. Conversely, `Petal.Length` and `Petal.Width` are negatively associated with PC1 and PC2, respectively. The plot also shows that the data points are clustered around the origin, indicating the data is relatively homogenous.

Finally, we'll create a scree plot to visualize the eigenvalues associated with each principal component. A scree plot will help us determine the number of components to retain by looking at where the eigenvalues start to level off. You can determine the significant eigenvalues helps preserve most of the variance in the data.

```
#Create a scree plot to visualize eigenvalues
fviz_eig(iris_pca, addlabels = TRUE)
```

Show more



The height of each bar in the scree plot represents the variation explained by each principal component. The first principal component explains the most variation, followed by the second, third, and fourth principal components. This is evident by the decreasing height of the bars. So, here, we might possibly retain either just the PCA or the first two principal components as they explain the most variation.

Step 6. Select a subset of principal components

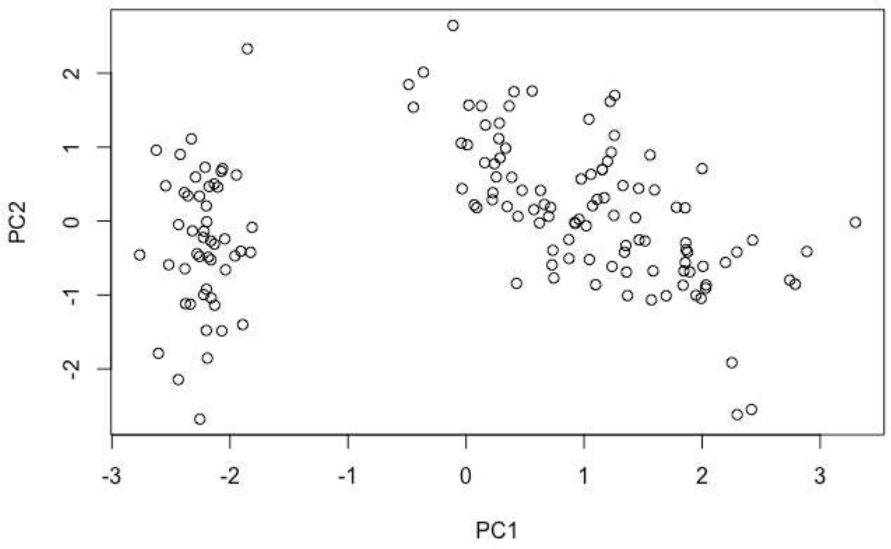
principal components that capture the most significant variance in the data.

Selecting a subset of principal components is essential for several reasons:

- **Dimensionality reduction**: It reduces the number of variables, making the data set more manageable and computationally efficient, especially when dealing with high-dimensional data.
- **Visualization**: It allows easy visualization of the data in lower dimensions. In this case, you can plot the data points in a 2D space defined by the selected principal components, allowing for better data exploration and interpretation.
- **Modeling**: Reduced-dimension data sets can be used in various data analysis tasks, such as clustering, classification, or regression, leading to simpler, more interpretable models.
- **Information retention**: By selecting the top principal components, you aim to retain as much of the original data's variance as possible. So, while the data is simplified, it still contains the essential patterns and structures in the original data set.

```
# Select the first principal components
first_two_components <- iris_pca$x[,1:2]
plot(first_two_components)
```

Show more



We'll select the first and second principal components to give us a 2D representation of the data that's easier to visualize and analyze. This aspect is especially beneficial when dealing with large data sets or when the primary goal is to uncover the primary sources of variation.

You can use the principal components you selected as predictors for various analyses, such as clustering, classification, or visualization, to understand your data better or build predictive models.

Now let's create a Naive Bayes model for classification to explore the difference between fitting a model over the entire (original) data set and just the PCA-transformed data.

Step 7. Fit a Naive Bayes model using the principal components

The Naive Bayes model is a probabilistic classification algorithm that makes predictions based on the probability distribution of the input features. It is widely used for classification tasks, especially when the input features are categorical or continuous.

Selecting a principal component is crucial when applying PCA to a data set, as it makes subsequent modeling and analysis more manageable and helps uncover significant patterns and relationships.

The decision to use just the first principal component may be based the amount of variance it explains, the desired level of dimensionality reduction, or specific analytical objectives. By retaining only the first principal component, we retain the direction of maximum variance in the data, which is often an informative representation. However this approach may not always yield the best results and should be chosen based on the goals of the analysis and the nature of the data.

```
#Select the first principal component for model 2
first_pca_vector <- iris_pca$x[,1]
```

Show more

The `first_pca_vector`, is a vector that contains the values of the first principal component for each record in the data set. Now we can fit the Naive Bayes model using that first principal component to predict the Species.

```
# create a dataframe
pca_iris <- data.frame(Species = iris$Species, PC1 = first_
# now use the new dataframe to build the model
pca_feature_model <- naive_bayes(Species ~ PC1, data = pca_
```

Show more

Step 8. Fit the Naive Bayes model using the whole data set

In this step, we'll fit a Naive Bayes classification model on the original Iris data set.

Show more

This creates `full_iris_model`, which we'll use to compare to the Naive Bayes model trained on the PCA data.

Step 9. Evaluate and compare the models

In this step, we'll make two predictions: `full_iris_pred` and `pca_pred`. The first set of predictions, `full_iris_pred`, will evaluate the performance of the Naive Bayes classification model when applied to the original data. The second set of predictions, `pca_pred`, will show how PCA can reduce dimensionality while still ensuring accurate classification.

```
# Predict using the original data
full_iris_pred <- predict(full_iris_model, newdata = iris[,
# Predict using the PCA-transformed data
pca_pred <- predict(pca_feature_model, newdata = data.frame
```

Show more

Now we'll evaluate both the models using confusion matrices. Using the original data set, we'll create a confusion matrix, `full_cm`, to visualize the classification results. Then we create a second confusion matrix, `pca_cm`, to visualize the classification results using the PCA-transformed data.

```
# Confusion matrix for the original data
full_cm <- table(Actual = iris$Species, Predicted = full_ir
# Display the confusion matrices
print("Confusion Matrix for Original Data:")
print(full_cm)
```

Show more

```
            Predicted
Actual       setosa versicolor virginica
  setosa         50          0         0
  versicolor      0         47         3
  virginica       0          3        47
```

Show more

Actual data versus predicted data: The rows represent the actual species labels, while the columns represent the predicted species labels.

- ***setosa:*** For the setosa species of iris flowers, all 50 instances were correctly classified (true positives). There were no false positives or false negatives for Iris setosa.
- ***versicolor:*** For the versicolor species, 47 instances were correctly classified (true positives). However, the model incorrectly classified three instances as virginica (false negatives). There were no false positives for Iris versicolor.
- ***virginica:*** For the virginica species, 47 instances were correctly classified (true positives). Again, the model incorrectly classified three instances as versicolor (false negatives). There were no false positives.

Now the PCA data, which has only one feature rather than 4:

```
# Confusion matrix for the PCA-transformed data
pca_cm <- table(Actual = iris$Species, Predicted = pca_pred
print("Confusion Matrix for PCA-Transformed Data:")
print(pca_cm)
```

Show more

This outputs the confusion matrix for the PCA-reduced data set:

```
            Predicted
Actual       setosa versicolor virginica
  setosa         50          0         0
```

Show more

In this confusion matrices we can see the following:

- ***setosa:*** Similar to the original data confusion matrix, all 50 instances of Iris setosa were correctly classified (true positives), with no false positives or false negatives.
- ***versicolor:*** For versicolor, 44 instances were correctly classified (true positives). However, the model incorrectly classified six instances as virginica (false negatives). There's an increase in false negatives compared to the original data.
- ***virginica:*** For virginica, 45 instances were correctly classified (true positives). However, the model incorrectly classified five instances as versicolor (false negatives). Again, there's an increase in false negatives compared to the original data.

Step 10. Comparative analysis

Let's compare the results of the confusion matrices.

The model's accuracy is high in both cases, as most data points were correctly classified. However, the PCA-transformed model shows a slight decrease in accuracy, which can be attributed to the higher number of misclassifications for versicolor and virginica.

We can calculate and print the accuracy using the confusion matrices:

```
print(paste(" Full Dataset : ", accuracy <- sum(diag(full_c
print(paste(" PCA Dataset : ", accuracy <- sum(diag(pca_cm)
```

Show more

This returns:

The PCA-transformed data indicates a reduction in dimensionality while maintaining a relatively high classification accuracy. However, this model has a higher number of misclassifications. So while PCA simplifies the feature space it can also cause a loss in classification power as well.

Whether to use PCA or not should weigh the trade-off between dimensionality reduction and maintaining classification performance. Sometimes the decrease in interpretability may be justified by the reduction in feature space complexity, other times it may not.

Summary

In this tutorial, you learned that PCA is a technique for reducing data dimensionality, enhancing data visualization, and speeding up training of machine learning models. We first standardized the data and implemented PCA. We created a scree plot to visualize the eigenvalues. Then, we used only the first principal component as input for the Naive Bayes classification model. That model was compared to a Naive Bayes classification model that used the original Iris data set. To wrap up, we compared the two models to understand the balance between dimensionality reduction and classification accuracy using confusion matrices and accuracy.

Try watsonx for free

Build an AI strategy for your business on one collaborative AI and data platform called IBM watsonx, which brings together new generative AI capabilities, powered by foundation models, and traditional machine learning into a powerful platform spanning the AI lifecycle. With watsonx.ai, you can train, validate, tune, and

Try watsonx.ai, the next-generation studio for AI builders.

Next steps

Explore more articles and tutorials about watsonx on IBM Developer.

Legend

Categories

Machine Learning Artificial intelligence watsonx.ai

Related Topics

Learn classification algorithms using Python and scikit-learn →

Learn regression algorithms using Python and scikit-learn →

Reducing dimensionality with principal component analysis with Python →

Trials

Try watsonx.ai

Interested in generative AI?

Learn generative AI skills →