Aplicação de PCA e KNN no Dataset Iris utilizando a linguagem R

Alexsandro Da Silva Bezerra

Guarujá, São Paulo

2024

Introdução

Este projeto Este projeto explora a aplicação da Análise de Componentes Principais (PCA) e do algoritmo K-Nearest Neighbors (KNN) no famoso dataset Iris, utilizando a linguagem de programação R. A técnica de PCA é utilizada para reduzir a dimensionalidade do dataset, preservando a maior parte da variabilidade dos dados. O modelo KNN, então, é treinado e testado em dados reduzidos para avaliar sua eficácia em classificar as espécies de flores com base em suas características.

Objetivo

O objetivo deste projeto é aplicar a técnica de PCA para reduzir a dimensionalidade do dataset Iris e, em seguida, utilizar o algoritmo KNN para classificar as espécies de flores, visando maximizar a acurácia e minimizar o risco de overfitting.

O que é PCA ?

A Análise de Componentes Principais (PCA) é uma técnica multivariada utilizada para reduzir a dimensionalidade de um conjunto de dados, mantendo o máximo de informação possível. Seu objetivo é condensar diversas variáveis originais em um número menor de componentes, que são combinações lineares das variáveis, explicando a maior parte da variabilidade presente nos dados. As primeiras componentes principais são as mais relevantes, pois retêm a maior quantidade de variação.

A extração das componentes pode ser feita via matriz de covariância ou de correlação. A matriz de covariância é adequada quando as variáveis têm variâncias semelhantes. No entanto, quando as variáveis possuem escalas muito diferentes, a matriz de correlação é preferida, pois evita que variáveis com maior escala dominem as componentes principais. Para realizar a PCA em R, há funções como prcomp() e princomp() no pacote stats, além de opções em pacotes como FactoMineR, ade4 e amap. Essas funções permitem a padronização dos dados, essencial para garantir a validade dos componentes extraídos via matriz de correlação.

Linguagem R

R é uma linguagem de programação estatística e gráfica que vem se especializando na manipulação, análise e visualização de dados, sendo atualmente considerada uma das melhores ferramentas para essa finalidade.

O que também tornam o R uma excelente opção para a aplicação de machine learnig. Muitos algoritmos de machine learning estão à disposição para utilização no R, muitas vezes sendo executados com uma ou duas linhas de código.

A utilização dos algoritmos de machine learning é apenas uma das etapas do processo. Para efetiva aplicação é necessário que a ferramenta escolhida atenda a todas as etapas, possibilitando assim o conhecimento dos dados, preparação, aplicação do algoritmo e apresentação dos resultados.

Algoritmo KNN

O algoritmo KNN (K-Nearest Neighbors) é um dos mais simples da área de aprendizado de máquina, baseado na similaridade entre os pontos mais próximos para fazer previsões. Ele funciona de forma preguiçosa ("lazy learning"), pois todo o processamento acontece no momento da previsão, em contraste com algoritmos como árvores de decisão ou regressão linear, que realizam cálculos antecipados.

O KNN estima o valor de uma nova observação com base nos "k" pontos mais próximos do dataset. Por padrão, todos os pontos têm o mesmo peso, mas pode-se ajustar o parâmetro "weights" para que os pontos mais próximos tenham mais influência. O algoritmo é sensível à escala das variáveis, sendo necessário padronizá-las para evitar distorções no cálculo das distâncias.

Overfitting e Underfitting

No aprendizado de máquina, o overfitting ocorre quando o modelo se ajusta excessivamente aos dados de treinamento, aprendendo não apenas os padrões relevantes, mas também o "ruído" ou variações aleatórias dos dados, ele fica "viciado" ao conjunto de treinamento. Isso faz com que o modelo tenha um excelente desempenho no conjunto de treinamento porém um terrível desempenho em conjuntos novos de dados, resultando em uma baixa precisão fora do treinamento. Indicadores comuns de overfitting incluem uma baixa taxa de erro no conjunto de treinamento e uma alta taxa de erro no conjunto de teste.

Por outro lado, o underfitting acontece quando o modelo é muito simples ou foi treinado por um tempo insuficiente, não capturando padrões importantes dos dados. Modelos underfitted geralmente têm alta taxa de erro tanto no treinamento quanto no teste, apresentando baixo desempenho em geral.O que o torna o oposto do overfitting que se ajusta demais aos dados de treinamento enquanto o underfitting não se ajusta aos dados do treinamento O objetivo no desenvolvimento de modelos é encontrar o equilíbrio entre overfitting e underfitting, garantindo que o modelo seja tenha um excelente desempenho para generalizar novos dados, sem se ajustar excessivamente ou de forma insuficiente.

Análises

Neste exemplo, utilizamos o dataset `iris`, que é um conjunto de dados clássico na análise de aprendizado de máquina e estatística.

```
data(iris)
head(iris)
```

O resultado do comando `head(iris)` exibe as primeiras seis linhas do dataset:

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa

```
5          5.0           3.6            1.4           0.2       setosa
6          5.4           3.9            1.7           0.4       setosa
```

Se observarmos temos os números em centímetros para cada variável sendo Sépala.Comprimento, Sépala.Largura, Pétala.Comprimento, Pétala.Largura, e a ultima coluna sendo as Espécies de plantas: setosa, virginica, versicolor classificadas com base nessas variaveis.

Selecionando colunas numéricas

Como o PCA só pode ser aplicado em variáveis numéricas, selecionamos apenas as colunas numéricas.

```
colunas_numericas_iris <- iris[, 1:4]
head(colunas_numericas_iris)
```

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

Temos escalas difererentes para cada variavel sendo a maior delas a Sépala.Comprimento e a menor a Pétala.Largura, como vimos anteriormente o PCA é muito sensivel a escala das variaveis sendo assim precisamos padronizalas antes de aplicarmos ele.

Padronizando os dados

```
padronizado_iris <- scale(colunas_numericas_iris)
```

Utilizaremos a função `scale` do R para essa padronização, essa função utiliza a normalização f-score:

$$Z = \frac{x - \mu}{\sigma}$$

- x: O valor original da variável.
- μ: A média da variável.
- σ: O desvio padrão da variável.

```
head(padronizado_iris)
```

```
      Sepal.Length Sepal.Width Petal.Length Petal.Width
[1,]   -0.8976739   1.01560199  -1.335752   -1.311052
[2,]   -1.1392005  -0.13153881  -1.335752   -1.311052
[3,]   -1.3807271   0.32731751  -1.392399   -1.311052
[4,]   -1.5014904   0.09788935  -1.279104   -1.311052
[5,]   -1.0184372   1.24503015  -1.335752   -1.311052
[6,]   -0.5353840   1.93331463  -1.165809   -1.048667
```

Aplicando o PCA

Após a padronização, aplicamos o PCA:

```
pca_resultado <- prcomp(padronizado_iris, center = T, scale. = T )
summary(pca_resultado)
```

```
Importance of components:
                          PC1       PC2       PC3       PC4
Standard deviation         1.7084    0.9560    0.3831    0.1439
Proportion of Variance     0.7296    0.2285    0.0367    0.0052
Cumulative Proportion      0.7296    0.9581    0.9948    1.0000
```

Standard Deviation (Desvio Padrão)

- **Destaques:**
  - PC1: Com 1.7084 de desvio padrão, sendo o componente que captura a maior parte da variação dos dados entre os 4 PCs.
  - PC2: Com 0.9560 de desvio padrão, sendo o segundo componente mais importante entre os 4.

Proportion of Variance (Proporção da Variância)

- **Destaques:**
  - PC1: Temos 0.7296 ou 73% da Proporção da Variância explicada por ele.
  - PC2: Temos 0.2285 ou 22.85% da Proporção da Variância explicada por ele.

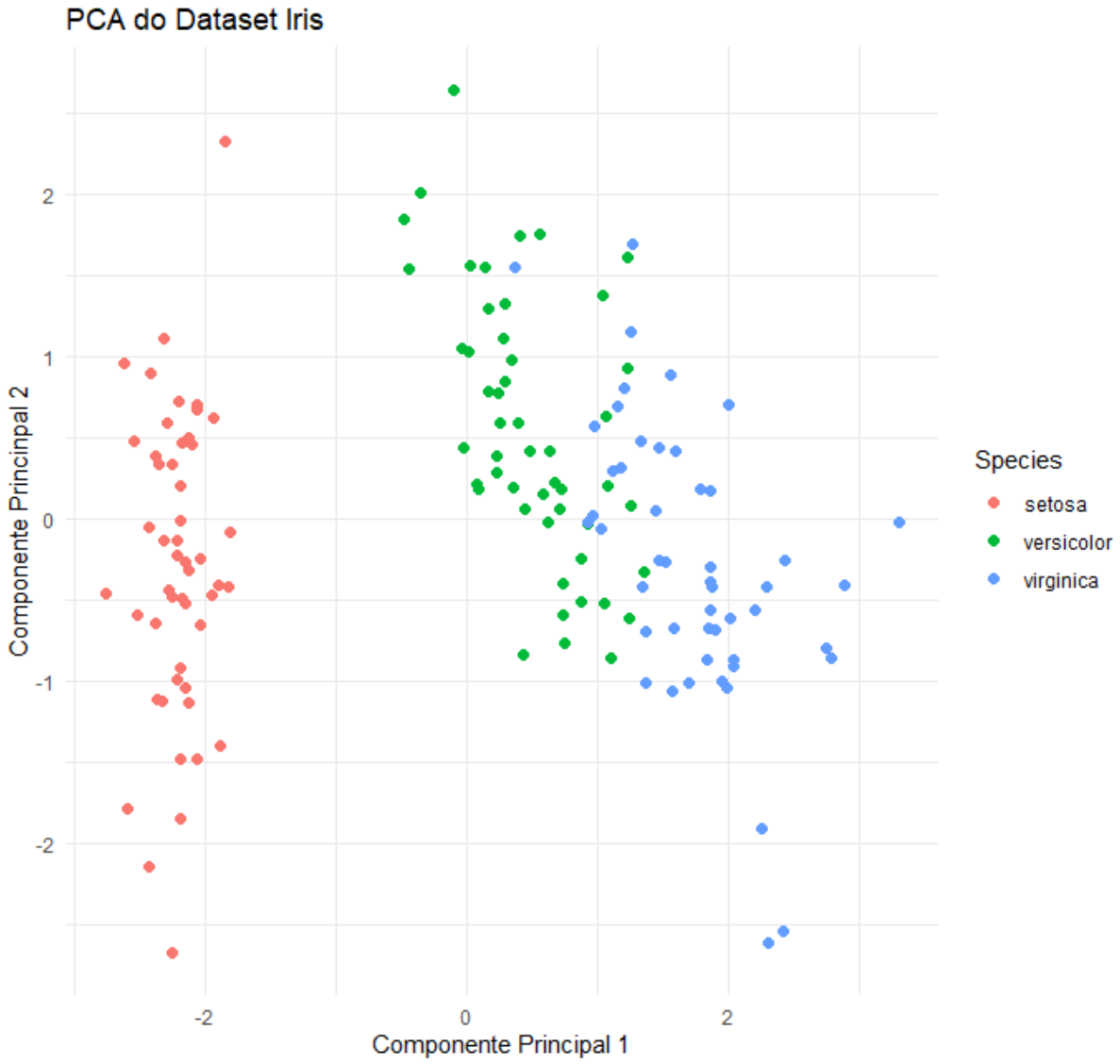
Cumulative Proportion (Proporção Acumulada)

- **Destaques:**
  - PC1: Temos 0.7296 ou 73% da Proporção da Variância explicada após ele.
  - PC2: Temos 0.9581 ou 95.81% da Proporção da Variância explicada após ele, porque estamos acumulando a proporção explicada pelo PC1 + PC2 (0.7296 + 0.2285 = 0.9581). Portanto, apenas esses dois componentes juntos já explicam quase 96% da variância dos dados, ou seja, podemos reduzir a dimensionalidade do dataset *Iris* para duas variáveis sem perder muita informação.

Gráficos

Para visualizar os resultados do PCA, utilizamos o `ggplot2` para criar o gráfico de dispersão entre PC1 e PC2 utilizando as Espécies (Species) em cores para observarmos como elas estão distribuídas neles.:

```
ggplot(pca_data_frame, aes(x = PC1, y = PC2, color = Species)) +
  geom_point(size = 2) +
  labs(title = "PCA do Dataset Iris", x = "PC1", y = "PC2") +
  theme_minimal()
```

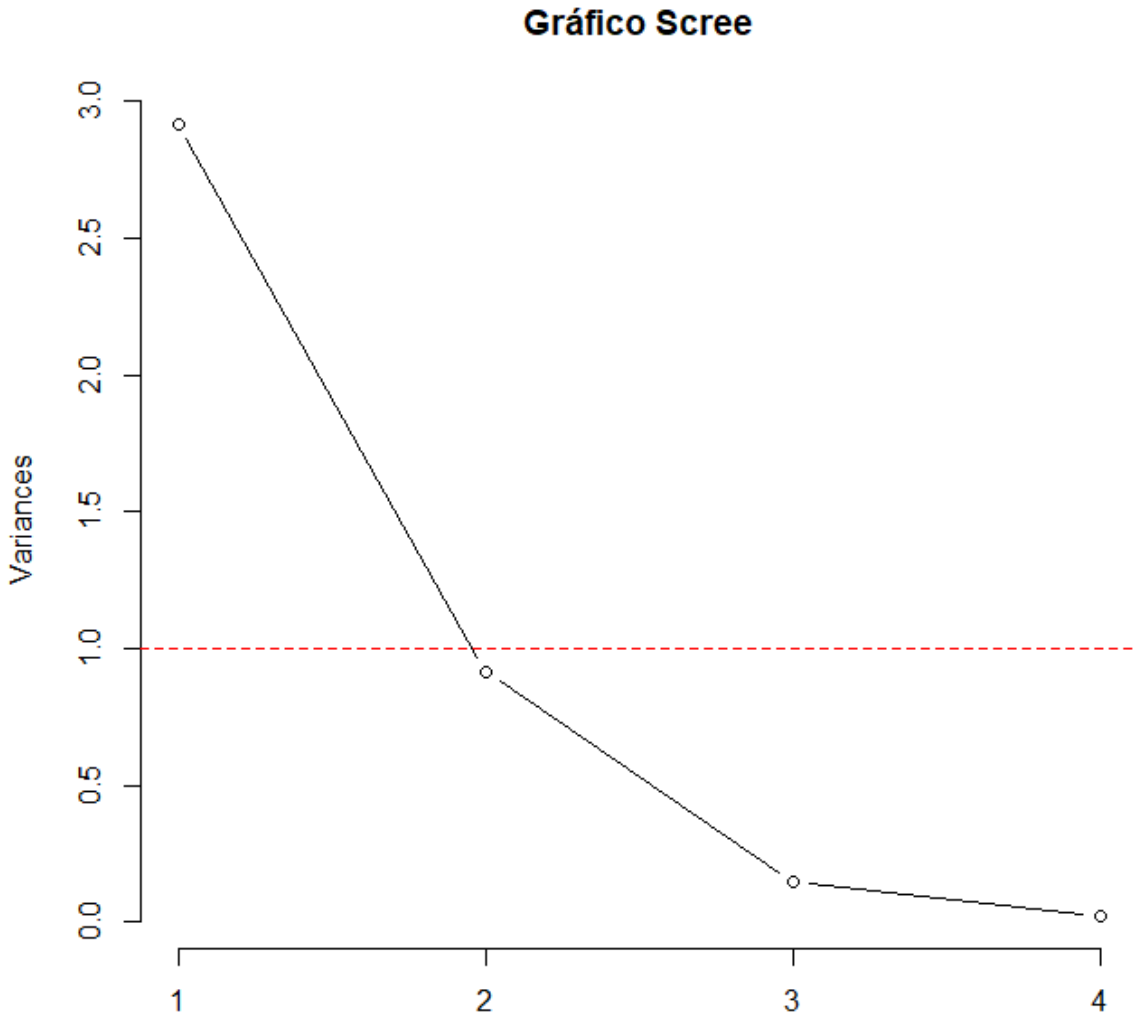


PC1: PC1 como vimos anteriormente ele explica a maior parte da variação dos dados , e no gráfico parece ser o responsável por separar principalmente a espécie Setosa das outras duas. A Setosa tem uma variação muito distinta ao longo de PC1, enquanto Versicolor e Virginica têm uma variação mais distribuída.

PC2: PC2 contribui com uma pequena parte da variação o que já era esperado devido a sua menor explicação na variação dos dados (ao menos comparado ao PC1), ajudando a distinguir Versicolor e Virginica, que estão levemente sobrepostas, mas ainda apresentam algumas pequenas separações ao longo do PC2

Além disso, plotamos o gráfico Scree para ver o critério de Kaiser:

```
plot(pca_resultado, type = "l", main = "Gráfico Scree")
abline(h = 1, col = "red", lty = 2)
```



Com o Critério de Kaiser (*autovalores* > 1), temos o PC1 maior que 1, bem perto de 3 e o PC2 com valor próximo de 1, porém já vemos uma queda quando passamos para o PC2 e seguindo para PC3 e PC4 vemos a continuação dessa queda, o resultado do gráfico reafirma novamente que o PC1 e o PC2, já são o suficiente para representar os dados do dataset Iris.

```
pca_resultado$rotation
```

	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

Com esse resultado conseguimos ver como as variáveis originais contribuem para cada os PC1, PC2, PC3 e PC4.
A Sepal.Length, Petal.Length e Petal.Width contribuem fortemente para o PC1, enquanto no PC2 a Sepal.Width contribui fortemente, porém de maneira negativa, isso pode ocorrer

porque como o PC1 não conseguiu explicar a variabilidade do Sepal.Width, o PC2 está tentando fazer isso, conseguiu porém de forma negativa.

```
iris_pca_reduzido <- pca_data_frame[, 1:2]
```

Criando um data frame apenas com os dois componentes selecionados PC1 e PC2

```
head(iris_pca_reduzido)
```

```
      PC1           PC2
1  -2.257141    -0.4784238
2  -2.074013     0.6718827
3  -2.356335     0.3407664
4  -2.291707     0.5953999
5  -2.381863    -0.6446757
6  -2.068701    -1.4842053
```

Iremos usar esse pca data frame para treinar um algoritmo de Knn

KNN com componentes principais

Por fim, utilizamos os dois primeiros componentes do PCA para treinar um modelo de KNN com 2 vizinhos:

```
set.seed(123)

indices <- sample(1:nrow(iris_pca_reduzido), size = 0.7 * nrow(iris_pca_reduzido))

treino <- iris_pca_reduzido[indices, ]
teste <- iris_pca_reduzido[-indices, ]

treino_rotulos <-iris$Species[indices]
teste_rotulos <- iris$Species[-indices]

resultados <- knn(treino, teste, treino_rotulos, 2)

resultados
```

```
[1] setosa    setosa    setosa    setosa    setosa    setosa    setosa
setosa    setosa    setosa    setosa    setosa    setosa
[13] setosa    setosa    virginica  versicolor versicolor versicolor virginica
versicolor versicolor versicolor versicolor versicolor
[25] versicolor virginica  versicolor virginica versicolor versicolor versicolor
versicolor virginica virginica virginica virginica
[37] virginica virginica virginica virginica virginica virginica virginica
virginica virginica
Levels: setosa versicolor virginica
```

A matriz de confusão resultante:

```
                  teste_rotulos
resultados   setosa  versicolor  virginica
setosa         14          0          0
versicolor      0         14          0
virginica       0          4         13
```

```
taxa_acerto<-(matrix_confusao[1]+matrix_confusao[5]+matrix_confusao[9])/sum(matrix_
```

```
print(paste("Acurácia de acerto:", round(taxa_acerto * 100, 2), "%"))
```

"Acurácia de acerto: 91.11%"
O dataset do pca foi dividido em 70% para treino e 30% para teste, com o KNN configurado para considerar 2 vizinhos (k=2), o modelo foi capaz de classificar as amostras de teste com uma acurácia de 91.11%.
A matriz de confusão mostrou que o modelo classificou corretamente 14 amostras de setosa, 14 de versicolor, mas cometeu 4 erros ao classificar virginica, confundindo algumas amostras com versicolor.

Conclusão

Depois de vários testes aumentando ou diminuindo os k-vizinhos, o máximo de acurácia atingindo pelo modelo foi com número de k-vizinhos em 2, obtendo 91.11% de acurácia.
Em suma a aplicação do PCA no dataset Iris se mostrou muito eficaz onde conseguimos diminuir a dimensionalidade com o mínimo de perda de informação possível, isso ajudou muito uma vez que ao minimizar a complexidade dos dados, evitamos o risco de overfitting onde modelo conseguiu focar nas características mais importantes, resultando em uma melhor generalização. Com isso, obtivemos uma boa acurácia e cumprimos com o objetivo desse projeto.

Referências

- Página oficial do R
- Pacote class do CRAN
- Dataset Iris
- Oliveira, B. (2019). Análise de Componentes Principais
- Didática Tech. A linguagem R
- Araújo, R. (2022). Algoritmo KNN (K-Nearest Neighbors) – Algoritmo de Aprendizado de Máquinas
- IBM. O que é overfitting?
- Link do Repositório no github