Practica- Reduccion de dimensionalidad

Nombre: Paulina Aldape

Reducción de dimensionalidad ¶

Cuando trabajamos con algoritmos de machine learning, ya sea de aprendizaje supervisado como no supervisado, la inclusion de mas caracteristicas a nuestra base de datos puede empeorar el rendimiento de nuestro modelo.

Pocas caracteristicas = underfitting Muchas caracteristicas = overfitting

En el caso de los modelos de aprendizaje no supervisado, causa ruido inecesario.

Por esto se aplica la reduccion de dimensiones de datos, para mejorar la exactitud de nuestro modelo al seleccionar solo el set de caracteristicas optimas o mas importantes de nuestro dataset.

Cuando hablamos de reducción de dimensionalidad, uno de los elementos mas populaes es la extraccion lineal de características, en la que contamos con dos metodos:

- Analisis de componente principal (Principal component analysis o PCA)
- Analisis discriminante linear (Linear discriminant analysis o LDA)

Analisis de componente principal (PCA)

Este metodo nos ayuda a reducir la dimension de nuestros datos y se puede utilizar para lo siguiente:

- Reducir el numero de dimensiones de un dataset.
- Encontrar patrones en un dataset de alta dimension.
- Visualizar mejor datos de alta dimensionalidad.
- Ignorar el ruido
- Mejorar la clasificación

Matematicamente su objetivo es:

- Organizar las dimensiones de los datos por orden de importancia.
- Descarta las dimensiones con menos significancia.
- Se concentra en componentes no correlacionados y de tipo gausianos.

Los pasos bases a seguir para este metodo son:

- Estandarizar los datos para encontrar PCA
- Calcular la matriz de covarianza
- Encontrar los valores de eigenvalues y eigenvectors para la matriz de covarianza.
- Graficar los vectores con los datos escalados

In [ ]:
```
import pandas as pd
import numpy as np
iris_data = pd.read_csv('/content/Iris.csv')
iris_data.columns
```

Out[ ]:
```
Index(['Id', 'SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthC
m',
       'Species'],
      dtype='object')
```

In [ ]:
```
X = iris_data[['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthC
m']]
y = iris_data.Species
```

Estandarizar datos

En la mayoria de los casos, cuando trabajemos con una base de datos a la cual queremos ver sus componentes principales, tenemos que estandarizar los datos.

In [ ]:
```
iris_data.head()
```

Out[ ]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

In [ ]:
```
from sklearn.preprocessing import StandardScaler
X = StandardScaler().fit_transform(X)
```

Forma clasica de PCA

Descomposicion Eigen

Existen valores llamados eigenvectors (vectores de eigen) y eigenvalues (valores de eigen), los cuales representan los valores base de un PCA. Los eigenvectors o componentes principales determinan la direccion hacia un nuevo espacio, mientras que los eigenvalues determinan la magnitud (explican la varianza de los datos dentro de los axes de caracteristicas)

Matriz de covarianza

La manera clasica de representar un PCA es hacer la descomposicion de eigen en una matriz de covarianza. Esta matriz e donde cada elemento representa la covarianza que existe entre dos caracteriticas. Se calcula con la formula siguiente:

$$Cov(X,Y) = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{N-1}$$

Las variables que tienen rangos mayores a otras variables tendran un bias a su favor, causando un desbalance en los datos. Al normalizar los datos con un escalador estandard de Sklearn podemos prevenir este problema.

In [ ]:
```
mean_vec=np.mean(X,axis=0)
cov_mat=(X-mean_vec).T.dot((X-mean_vec))/(X.shape[0]-1)
print("Covariance Matrix \n%s" %cov_mat)
```

```
Covariance Matrix
[[ 1.00671141 -0.11010327  0.87760486  0.82344326]
 [-0.11010327  1.00671141 -0.42333835 -0.358937  ]
 [ 0.87760486 -0.42333835  1.00671141  0.96921855]
 [ 0.82344326 -0.358937    0.96921855  1.00671141]]
```

In [ ]:
```
#usando la libreria de numpy
print("Numpy Covariance matrix \n%s" %np.cov(X.T))
```

```
Numpy Covariance matrix
[[ 1.00671141 -0.11010327  0.87760486  0.82344326]
 [-0.11010327  1.00671141 -0.42333835 -0.358937  ]
 [ 0.87760486 -0.42333835  1.00671141  0.96921855]
 [ 0.82344326 -0.358937    0.96921855  1.00671141]]
```

In [ ]:
```
#obtener la descomposicion de los eigenvalores y eigenvectores de la matriz de
covraianza
cov_mat=np.cov(X.T)

eig_vals, eig_vecs=np.linalg.eig(cov_mat)

print("Eigenvectors \n%s" %eig_vecs)
print("Eigenvelues \n%s" %eig_vals)
```

```
Eigenvectors
[[ 0.52237162 -0.37231836 -0.72101681  0.26199559]
 [-0.26335492 -0.92555649  0.24203288 -0.12413481]
 [ 0.58125401 -0.02109478  0.14089226 -0.80115427]
 [ 0.56561105 -0.06541577  0.6338014   0.52354627]]
Eigenvelues
[2.93035378 0.92740362 0.14834223 0.02074601]
```

In [ ]:
```
# Make a list of (eigenvalue, eigenvector) tuples
eig_pairs = [(np.abs(eig_vals[i]), eig_vecs[:,i]) for i in range(len(eig_val
s))]

# Sort the (eigenvalue, eigenvector) tuples from high to low
eig_pairs.sort(key=lambda x: x[0], reverse=True)

# Visually confirm that the list is correctly sorted by decreasing eigenvalues
print('Eigenvalues in descending order:')
for i in eig_pairs:
    print(i[0])
```

```
Eigenvalues in descending order:
2.930353775589317
0.9274036215173419
0.14834222648163944
0.02074601399559593
```

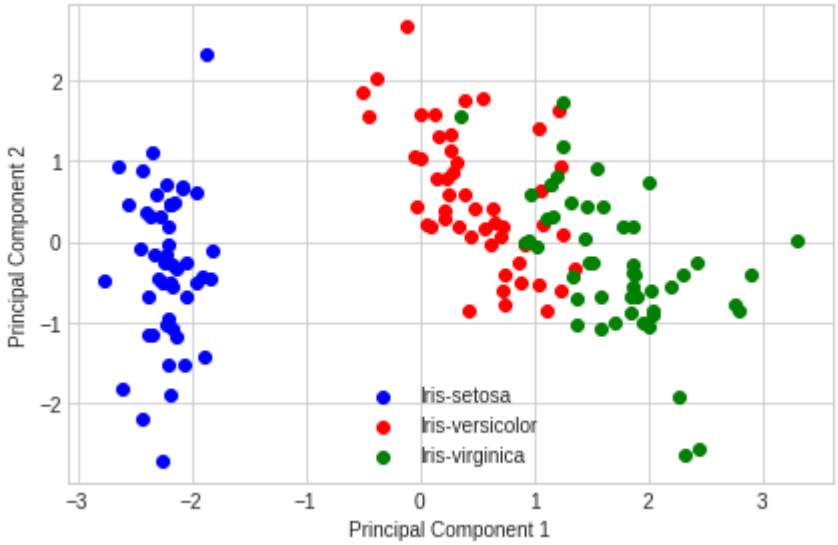
In [ ]:
```
#matriz de proyeccion
matrix_w = np.hstack((eig_pairs[0][1].reshape(4,1),
                      eig_pairs[1][1].reshape(4,1)))

print('Matrix W:\n', matrix_w)
```

```
Matrix W:
 [[ 0.52237162 -0.37231836]
 [-0.26335492 -0.92555649]
 [ 0.58125401 -0.02109478]
 [ 0.56561105 -0.06541577]]
```

In [ ]:
```
#mostrar el nuevo espacio de las variables con la matrix proyectada
Y = X.dot(matrix_w)
```

In [ ]:
```
import matplotlib.pyplot as plt
```

In [ ]:
```
with plt.style.context('seaborn-whitegrid'):
    plt.figure(figsize=(6, 4))
    for lab, col in zip(('Iris-setosa', 'Iris-versicolor', 'Iris-virginica'),
                        ('blue', 'red', 'green')):
        plt.scatter(Y[y==lab, 0],
                    Y[y==lab, 1],
                    label=lab,
                    c=col)
    plt.xlabel('Principal Component 1')
    plt.ylabel('Principal Component 2')
    plt.legend(loc='lower center')
    plt.tight_layout()
    plt.show()
```



Practica:

PCA con SKlearn. Utilizando la libreria de Sklearn, busca el modulo de decomposition de PCA y utiliza los mismos datos que usamos en este ejemplo para generar una grafica similar.

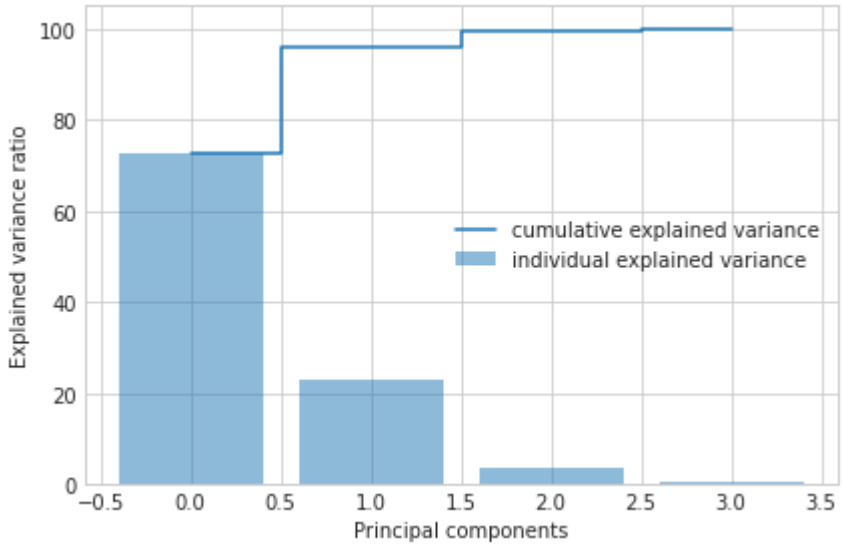
Despues, utiliza los mismos datos para el modulo de discriminant_analisis con Linear discrimination analysis.

Busca la diferencia entre PCA y LDA, escribe tus conclusiones y compara tus resultados.

(x) (https://scikit-learn.org/stable/auto_examples/decomposition/plot_pca_vs_lda.html#sphx-glr-auto-examples-decomposition-plot-pca-vs-lda-py)

In [ ]:
```
#varianza explicada
tot = sum(eig_vals)
var_exp = [(i / tot)*100 for i in sorted(eig_vals, reverse=True)]
cum_var_exp = np.cumsum(var_exp)
```

In [ ]:
```
with plt.style.context('seaborn-whitegrid'):
    plt.figure(figsize=(6, 4))

    plt.bar(range(4), var_exp, alpha=0.5, align='center',
            label='individual explained variance')
    plt.step(range(4), cum_var_exp, where='mid',
             label='cumulative explained variance')
    plt.ylabel('Explained variance ratio')
    plt.xlabel('Principal components')
    plt.legend(loc='best')
    plt.tight_layout()
```

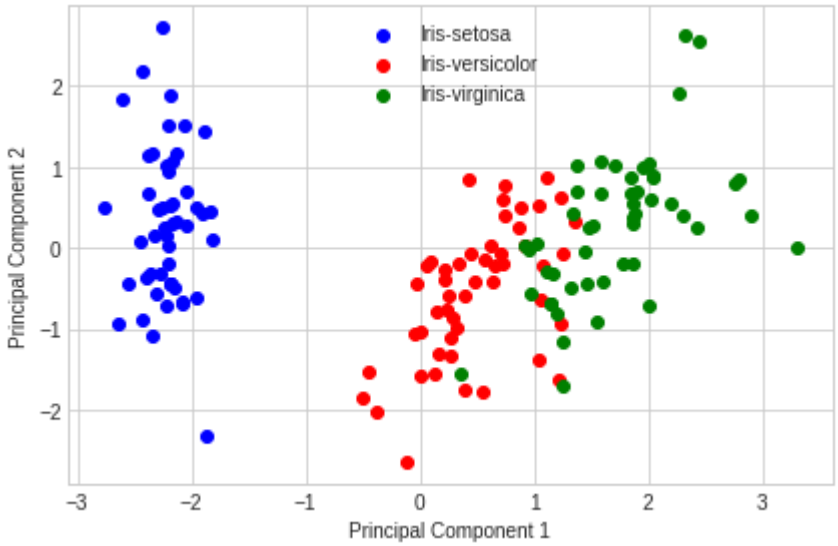


Se observa que a partir del segundo componente ya no se tiene un aumento significativo en la varianza expliada, es por eso que nos mantenemos con los primeros 2 componentes que explican al rededor del 96%.

Aplicación del PCA

In [ ]:
```
from sklearn.decomposition import PCA as sklearnPCA
sklearn_pca = sklearnPCA(n_components=2)
Y_sklearn = sklearn_pca.fit_transform(X)
```

In [ ]:
```
with plt.style.context('seaborn-whitegrid'):
    plt.figure(figsize=(6, 4))
    for lab, col in zip(('Iris-setosa', 'Iris-versicolor', 'Iris-virginica'),
                        ('blue', 'red', 'green')):
        plt.scatter(Y_sklearn[y==lab, 0],
                    Y_sklearn[y==lab, 1],
                    label=lab,
                    c=col)
    plt.xlabel('Principal Component 1')
    plt.ylabel('Principal Component 2')
    plt.legend(loc='upper center')
    plt.tight_layout()
    plt.show()
```



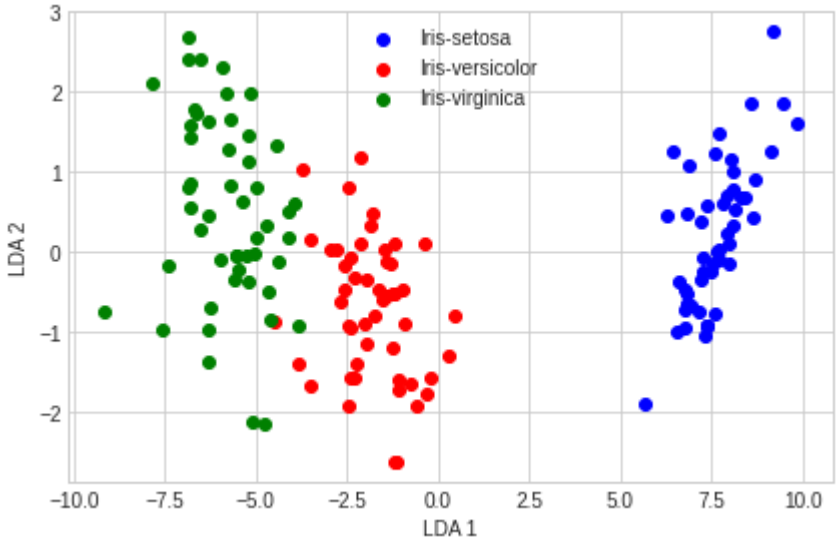
Se puede observar que se llega al mismo resultado planetado con anterioridad, solo que el uso de la paqueteria de sklearn hace que sea más rapido generar el análisis de componentes principales

Aplicación del LDA

In [ ]:
```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
```

In [ ]:
```
lda = LinearDiscriminantAnalysis(n_components=2)
X_r2 = lda.fit(X, y).transform(X)
```

In [55]:
```
with plt.style.context('seaborn-whitegrid'):
    plt.figure(figsize=(6, 4))
    for lab, col in zip(('Iris-setosa', 'Iris-versicolor', 'Iris-virginica'),
                        ('blue', 'red', 'green')):
        plt.scatter(X_r2[y==lab, 0],
                    X_r2[y==lab, 1],
                    label=lab,
                    c=col)
    plt.xlabel('LDA 1')
    plt.ylabel('LDA 2')
    plt.legend(loc='upper center')
    plt.tight_layout()
    plt.show()
```



Conclusiones

Se contraron resultados similares usando las dos metodologias. Sin embargo, al buscar informacion sobre la diferencia entre ambos modelos se recomienda que en el caso de datos distribuidos uniformemente, LDA casi siempre funciona mejor que PCA. Por otra parte, si los datos están muy sesgados (distribuidos de forma irregular), se recomienda utilizar PCA, ya que LDA puede estar sesgado hacia la clase mayoritaria. Finalmente, una de las ventajas del PCA es que se puede aplicar a datos etiquetados y no etiquetados, ya que no depende de las etiquetas de salida. Por otro lado, LDA requiere clases de salida para encontrar discriminantes lineales y, por lo tanto, requiere datos etiquetados. Es por eso que para la base de datos que se utilizó para esta prática se pudo aplicar tambien la metodologia de LDA ya que contabamos con la variable de especie.