Reducción de dimensionalidad

Castro Fernando, Porras Braulio, Tinoco Sergio

Instituto Politécnico Nacional, Escuela Superior de Cómputo

Aprendizaje de Máquina 5BV1

20 de Noviembre de 2022

I. Introducción

Muchos son los métodos que existen para optimizar los distintos algoritmos para el aprendizaje automático, sin duda el más importante es la reducción de dimensionalidad en un conjunto de datos.

En esta práctica se implementarán 3 algoritmos de reducción de dimensionalidad fundamentales; el análisis de componentes principales (PCA), análisis lineal de discriminante (LDA) y análisis de componentes principales con kernel (KPCA), en un mismo conjunto de datos y en los lenguajes de programación Python y R.

II. Objetivo

Comprender la implementación de las técnicas de reducción de dimensionalidad en R y Python.

III. Marco Teórico

Para esta práctica se va a trabajar con un dataset que describe el comportamiento de múltiples alarmas de incendio, las cuales se activan bajo ciertas circunstancias, como lo son: el índice de partículas PM en el aire, la temperatura, la presión, el nivel de CO2 del aire, nivel de hidrógeno, la densidad de partículas, etc.

IV. Desarrollo

Análisis de componentes principales con kernel.

La implementación del KPCA en python y R se ha limitado a utilizar sólo 100 datos de los aproximadamente 60,000 datos totales, esto se ha realizado por cuestiones de rendimiento al probarlo en el equipo local, al usar un equipo con el hardware adecuado se deben usar todos los datos.

Implementación en Python.

Para comenzar se importan las librerías básicas de tratamiento de datos pandas y matplotlib.

```
import matplotlib.pyplot as plt
import pandas as pd
```

posterior a esto cargamos el conjunto de datos con pandas.

```
df = pd.read_csv("dataset.csv")
```

Se declara una cantidad máxima de datos a leer, esto porqué en el equipo en el que esto se probó, daba problemas de rendimiento al cargar el conjunto de datos completo.

```
maxsize = 100
x = df.iloc[0:maxsize,1:15].values
y = df.iloc[0:maxsize,15].values
```

Separamos los datos de prueba y datos de entrenamiento usando la función de sklearn train_test_split

```
from sklearn.model_selection import train_test_split

xtrain,xtest,ytrain,ytest =
train_test_split(x,y,test_size=0.2,random_state=0)
```

Posterior a la carga y separación de los datos, se realiza una normalización de estos para cargarlo de mejor forma.

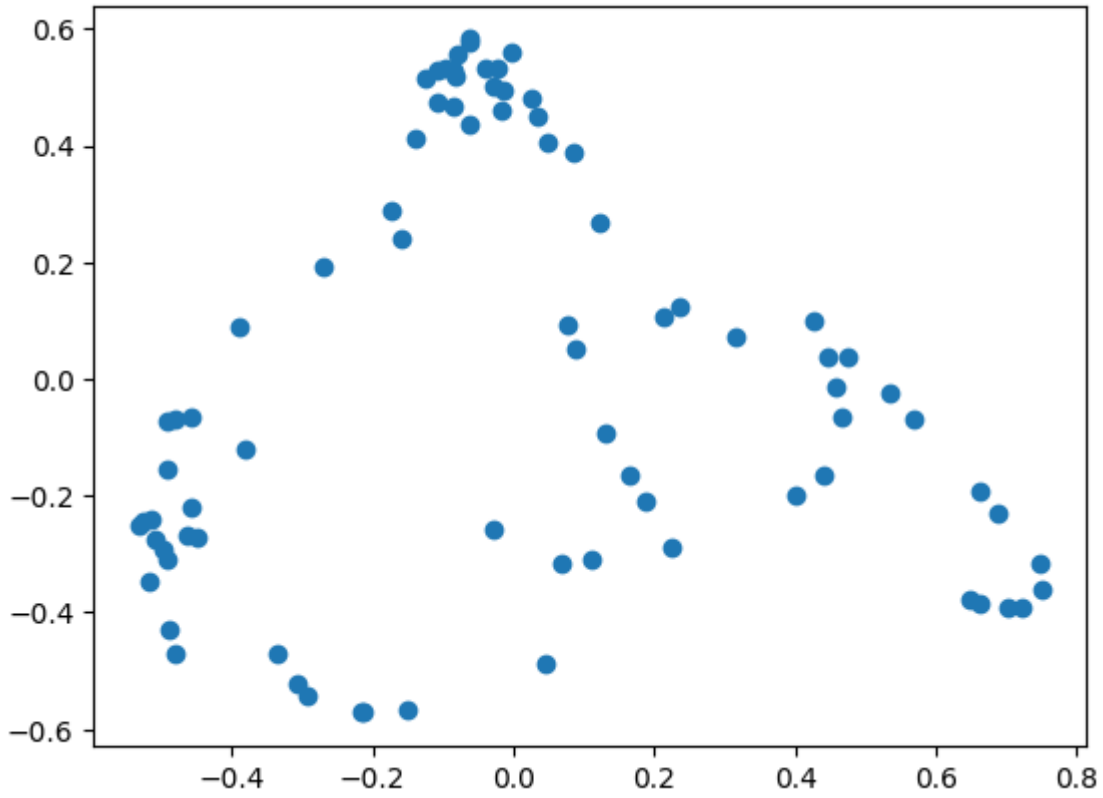
```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
xtrain = scaler.fit_transform(xtrain)
xtest = scaler.fit_transform(xtest)
```

Una vez concluido lo anterior ya podemos utilizar el análisis de componentes principales con kernel con 2 componentes para poder realizar su visualización en 2 dimensiones.

```
from sklearn.decomposition import KernelPCA
components = 2

kpca = KernelPCA(n_components=components,kernel="rbf")
xtrain = kpca.fit_transform(xtrain)
xtest = kpca.fit(xtest)
```

Finalmente, graficamos los datos obtenidos de la muestra que hemos elegido.

```
plt.scatter(xtrain[:,0],xtrain[:,1])
```



Implementación en R

La programación en R se realizó de una manera similar a python por el hecho de que, de igual manera, se tomaron solo 100 elementos de los miles disponibles en el conjunto de datos. Realizamos la lectura y extraemos las columnas de interés.

```
dataset = read.csv("ESCOM/ML/Actividades/RD/dataset.csv")
dataset = dataset[0:100,3:16]
```

Importamos el paquete catTools y configuramos la semilla para obtener siempre los mismos resultados.

```
library(caTools)
set.seed(1000)
```

Realizamos la separación de datos en datos de entrenamiento y datos de prueba utilizando la función sample.split( )

```
split = sample.split(dataset$Fire.Alarm, SplitRatio = 0.8)
trainSet = subset(dataset,split=TRUE)
testSet = subset(dataset,split=FALSE)
```

Al igual que en python realizamos el escalamiento de los datos para facilitar el procesamiento de estos.

```
trainSet[,1:12] = scale(trainSet[,1:12])
testSet[,1:12] = scale(testSet[,1:12])
```

Para continuar instalamos el paquete kernlab e instanciamos el KPCA con 2 componentes y un kernel gaussiano.

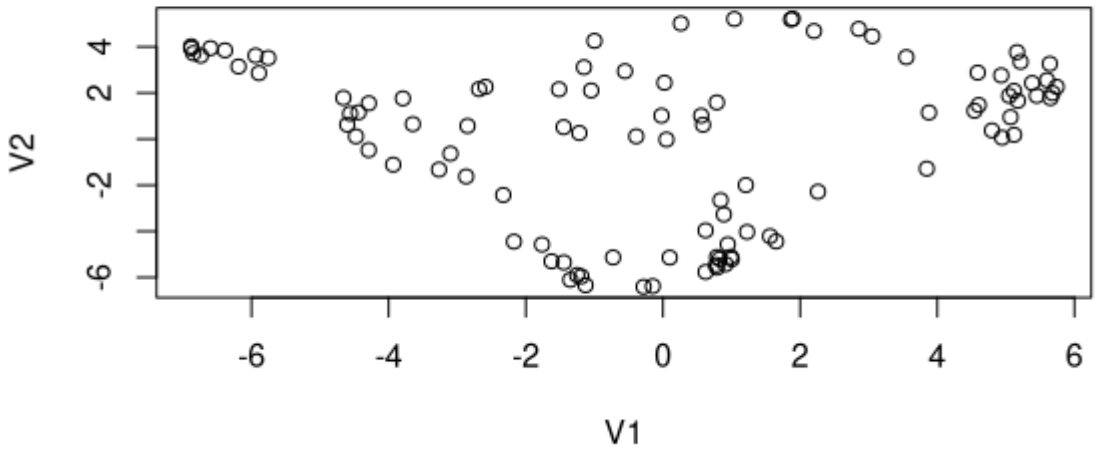
```
library(kernlab)

# Aplicación del KPCA
kpca = kpca(~.,
   data=trainSet[,-13],
   kernel = "rbfdot",
   features = 2
)
```

Para finalizar aplicamos la transformación de los datos llamando la función kpca.

```
trainset_pca = as.data.frame(predict(kpca,trainSet))
testset_pca = as.data.frame(predict(kpca,testSet))
```

Y realizamos la visualización de la muestra de datos.

```
View(trainset_pca)
plot(trainset_pca)
```



Análisis discriminante lineal
Implementación en Python.

Para comenzar se importan las librerías básicas.

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
```

posterior a esto cargamos el conjunto de datos con pandas.

```
dataset = pd.read_csv("dataset.csv")
```

Se declaran las variables dependientes e independientes

```
x = df.iloc[:,2:14].values
y = df.iloc[:,14].values
```

Separamos los datos de prueba y datos de entrenamiento usando la función de sklearn train_test_split

```
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)
```

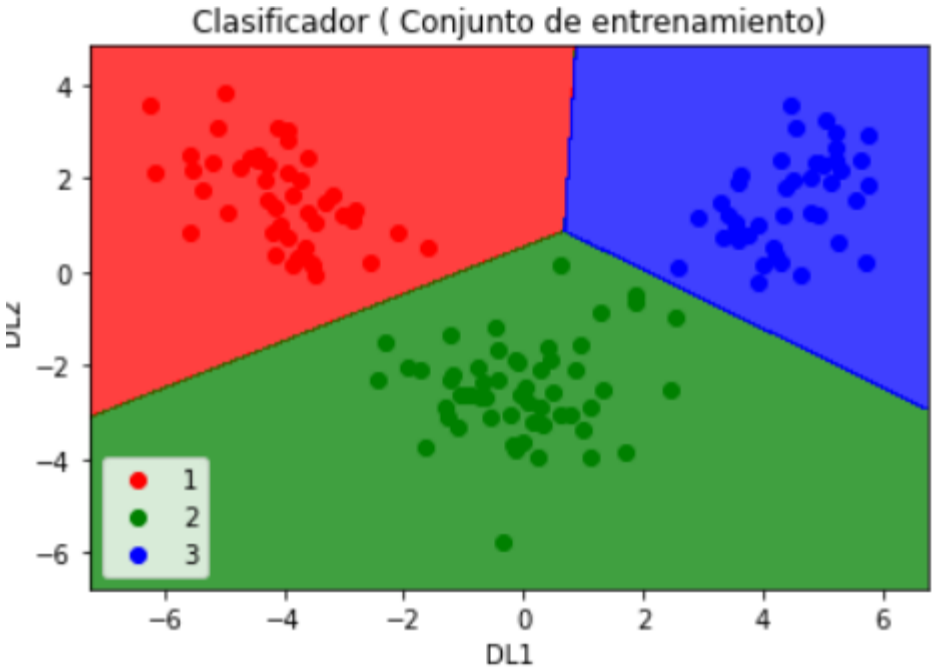
Se realiza una normalización de estos para cargarlo de mejor forma.

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

Una vez concluido lo anterior ya podemos utilizar el análisis discriminante lineal..

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as
LDA
lda = LDA(n_components=2)
X_train = lda.fit_transform(X_train,y_train)
X_test = lda.transform(X_test)
```

Finalmente, graficamos los datos.



Implementación en R

La programación en R se realizó de una manera similar a python. Realizamos la lectura.

```
dataset = read.csv("D:/Data D/5 semestre/Machine Learning/dataset.csv")
```

Importamos el paquete catTools y configuramos la semilla para obtener siempre los mismos resultados.

```
library(caTools)
set.seed(123)
```

Realizamos la separación de datos en datos de entrenamiento y datos de prueba utilizando la función sample.split( ) y se realiza la estandarización de los conjuntos de entrenamiento.

```
split =sample.split(dataset$Fire_Alarm, SplitRatio =0.8)
training_set = subset(dataset, split=TRUE)
testing_set = subset(dataset, split=FALSE)

training_set[,-14]= scale(training_set[,-14])
testing_set[,-14]= scale(testing_set[,-14])
```

A continuación se aplica el LDA .

```
lda = lda(formula = Fire_Alarm ~ ., data = training_set)

training_set = as.data.frame(predict(lda,training_set))
training_set = training_set[,c(3,4,1)]
testing_set = as.data.frame(predict(lda,testing_set))
testing_set = testing_set[,c(3,4,1)]
```

Análisis de Componentes Principales.

Implementación en Python.

Para el análisis de componentes principales se utilizaron los primeros 2000 registros del dataset, de los cuales se tomaron todas las características exceptuando la columna de los índices y el contador de registros.

Se importan las bibliotecas correspondientes

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

Se carga el dataset.

```
dataset=pd.read_csv("dataset.csv")
```

Se seleccionan las columnas para nuestro dataframe.

```
X=dataset.iloc[:,2:14].values
y=dataset.iloc[:,15].values
```

Se añade el paquete para ajustar nuestro modelo de predicción.

```
from sklearn.model_selection import train_test_split
```

Se entrena nuestro modelo con un 80% de nuestros datos y se almacenan los dataframes resultantes.

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

Se incluye el paquete para escalar los datos previamente entrenados.

```
from sklearn.preprocessing import StandardScaler
```

Se crea una instancia de la clase StandardScaler() para ajustar los datos

```
sc_X=StandardScaler()
```

Se ajustan los datos de entrenamiento y prueba por medio del metodo fit_transform

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

Se incluye el paquete de sklearn para utilizar la clase PCA

```
from sklearn.decomposition import PCA
```

Se crea una instancia de la clase PCA pasando como parámetro al constructor el número de componentes que se desean obtener.

```
pca=PCA(n_components=2)
```

Se reajustan los datos de entrenamiento y prueba por medio del método fit_transform y transform respectivamente.

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

Finalmente se obtiene la varianza explicada

```
varianza_explicada=pca.explained_variance_ratio_
```

Resultados

	0	1	2	3	4	5	6	7
0	20	57.36	0	400	12306	18520	939.735	0
1	20.015	56.67	0	400	12345	18651	939.744	0
2	20.029	55.96	0	400	12374	18764	939.738	0
3	20.044	55.28	0	400	12390	18849	939.736	0
4	20.059	54.69	0	400	12403	18921	939.744	0
5	20.073	54.12	0	400	12419	18998	939.725	0
6	20.088	53.61	0	400	12432	19058	939.738	0
7	20.103	53.2	0	400	12439	19114	939.758	0
8	20.117	52.81	0	400	12448	19155	939.758	0
9	20.132	52.46	0	400	12453	19195	939.756	0.9
10	20.146	52.15	0	400	12454	19230	939.757	0.89
11	20.161	51.84	0	400	12467	19264	939.754	0.84
12	20.175	51.62	0	400	12467	19299	939.755	0.81
13	20.19	51.39	0	400	12469	19317	939.758	0.74

Dataframe X

	0	1	2	3	4	5	6	7
0	0.53393	0.521153	0.744485	1.95955	-1.12176	-0.841458	-0.348289	0.773901
1	-0.967095	-0.466599	-0.707497	-0.457549	0.684378	0.88826	0.676415	-1.44296
2	0.393787	0.10984	0.822669	0.18928	-0.811612	-0.867866	-0.938918	0.279857
3	0.695727	0.743323	-1.23431	-0.457549	1.14047	1.30418	0.932591	-0.809574
4	-2.31392	0.707295	1.33645	-0.366766	-0.154836	-1.09563	-1.23305	0.647223
5	0.873344	-1.24719	0.951114	-0.321375	-0.525793	-0.920682	-1.19272	0.140511
6	1.06279	-0.535651	-1.03512	-0.457549	1.4871	1.3471	1.09863	0.457206
7	-1.54786	-0.736804	1.38299	-0.298679	-0.294705	-1.10884	-1.15714	0.583884
8	0.581309	-0.898928	-1.08725	-0.457549	1.17696	1.26127	1.02747	-0.695564
9	-2.31277	-0.970983	1.27129	-0.457549	-0.130511	-1.04942	-1.24016	0.305193
10	0.351552	0.614224	0.701671	1.98224	-1.13392	-0.808448	-0.379125	0.659891
11	-0.0102081	0.0828197	0.65141	-0.457549	0.0884147	-0.775438	-1.13342	1.43263
12	0.239591	0.827387	-1.17474	-0.0376778	-2.6968	-0.24728	1.40225	-1.93701
13	-1.62442	0.407067	1.34017	-0.34407	-0.246055	-1.07913	-1.15477	0.621888

Dataframe X_train

	0	1
0	2.3779	-0.454682
1	-3.10474	-0.490596
2	1.44054	-1.04546
3	-2.82065	0.844676
4	2.30191	-0.23372
5	1.2507	-1.38176
6	-0.880815	2.36964
7	2.25177	-0.621443
8	-2.50708	0.814496
9	1.81202	-0.760324
10	2.20152	-0.55136
11	2.91206	0.877614
12	-3.32541	-2.82533
13	2.21336	-0.391494

Dataframe X_train despues de la reducción.

	0
0	0.4549
1	0.196451

Valores de la varianza explicada

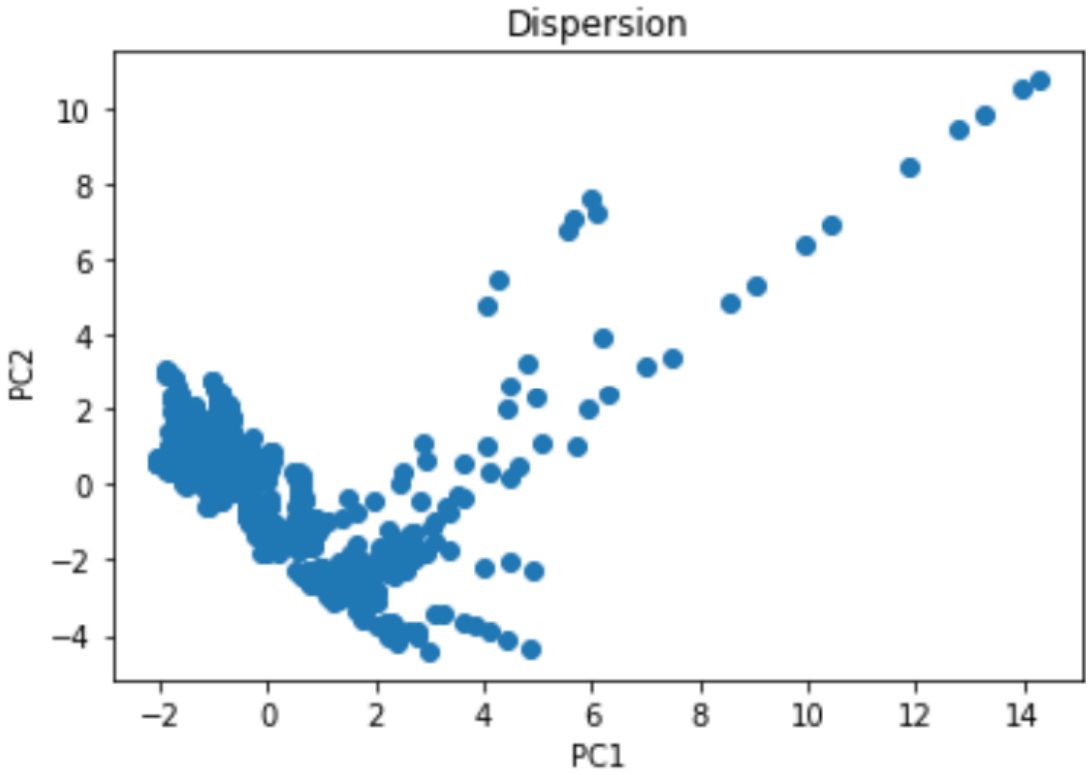


Gráfico de dispersión de las componentes.

Implementación en R.

Se carga el dataset por medio de la ruta absoluta.

```
dataset=read.csv("C:/Users/Sergio/Desktop/Reduccion
Dimension/dataset.csv")
```

Se carga la biblioteca caTools

```
library(caTools)
```

Se define una “semilla” para generar un número de forma pseudoaleatoria.

```
set.seed((123))
```

Se separan los datos del dataframe por medio de la columna de Fire.Alarm y se asigna un 80% de los datos para el entrenamiento.

```
split=sample.split(dataset$Fire.Alarm, SplitRatio=0.8)
```

Se asignan los valores verdaderos al modelo de entrenamiento y los valores falsos al modelo de prueba.

```
training_set=subset(dataset, split==TRUE)
testing_set=subset(dataset, split==FALSE)
```

Se extraen las columnas 3 a la 14 junto con la columna 16 con el fin de omitir la columna del contador. Se asigna el dataframe resultante al modelo de entrenamiento y de prueba.

```
training_set=training_set[,c(3:14,16)]
testing_set=testing_set[,c(3:14,16)]
```

Se escalan los valores de ambos modelos.

```
training_set[,-13]=scale(training_set[,-13])
testing_set[,-13]=scale(testing_set[,-13])
```

Se incluye el paquete caret para generar el modelo de PCA con el cual se van a ajustar los datos y el paquete e1071 para generar las predicciones de los modelos.

```
library(caret)
library(e1071)
```

```
#Variable sobre el cual se reduciran los datos
pca=preProcess(x=training_set[,-13],method="pca",pcaComp=2)
training_set=predict(pca,training_set)
training_set=training_set[,c(2,3,1)]
```

```
testing_set=predict(pca,testing_set)
testing_set=testing_set[,c(2,3,1)]
```

Resultados.

X	UTC	Temperature.C.	Humidity...	TVOC.ppb.	eCO2.ppm.	Raw.H2	Raw.Ethanol	Pressure.hPa.
3	1654733334	20.044	55.28	0	400	12390	18849	939.736
4	1654733335	20.059	54.69	0	400	12403	18921	939.744
7	1654733338	20.103	53.20	0	400	12439	19114	939.758
10	1654733341	20.146	52.15	0	400	12454	19230	939.757
15	1654733346	20.219	50.99	0	400	12475	19362	939.741
19	1654733350	20.277	50.27	0	406	12489	19451	939.752
20	1654733351	20.291	50.15	0	400	12491	19456	939.729
23	1654733354	20.334	49.83	0	415	12495	19489	939.764
30	1654733361	20.434	50.68	16	435	12481	19509	939.772
31	1654733362	20.448	51.23	8	440	12482	19520	939.772
49	1654733380	20.701	56.32	0	408	12510	19587	939.722
52	1654733383	20.743	56.55	11	404	12512	19596	939.693
58	1654733389	20.825	55.38	0	400	12539	19629	939.714
64	1654733395	20.908	52.27	10	411	12534	19635	939.760

Dataframe del training_set original

Temperature.C.	Humidity...	TVOC.ppb.	eCO2.ppm.	Raw.H2	Raw.Ethanol	Pressure.hPa.	PM1.0	PM2.5	NC0.5
20.000	57.36	0	400	12306	18520	939.735	0.00	0.00	0.00
20.015	56.67	0	400	12345	18651	939.744	0.00	0.00	0.00
20.029	55.96	0	400	12374	18764	939.738	0.00	0.00	0.00
20.073	54.12	0	400	12419	18998	939.725	0.00	0.00	0.00
20.088	53.61	0	400	12432	19058	939.738	0.00	0.00	0.00
20.117	52.81	0	400	12448	19155	939.758	0.00	0.00	0.00
20.132	52.46	0	400	12453	19195	939.756	0.90	3.78	0.00
20.161	51.84	0	400	12467	19264	939.754	0.84	3.51	0.00
20.175	51.62	0	400	12467	19299	939.755	0.81	3.38	0.00
20.190	51.39	0	400	12469	19317	939.758	0.74	3.11	0.00
20.204	51.17	0	403	12468	19338	939.742	0.71	2.96	0.00
20.233	50.86	0	400	12480	19382	939.758	0.60	2.52	0.00
20.248	50.66	0	400	12477	19400	939.764	0.53	2.23	0.00
20.262	50.49	0	400	12481	19422	939.761	0.50	2.10	0.00
20.305	50.02	0	400	12487	19470	939.741	0.33	1.39	0.00
20.320	49.96	0	400	12492	19489	939.756	0.31	1.30	0.00

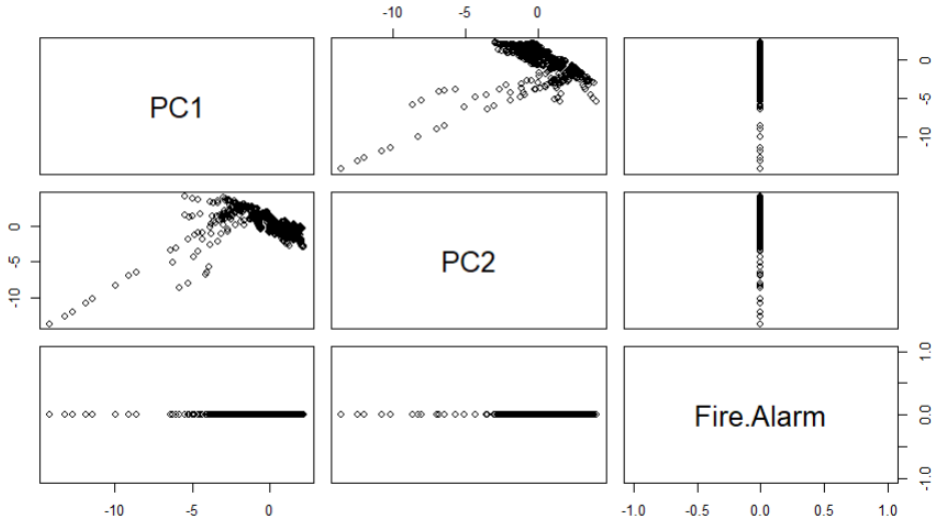
Dataframe del training_set sin considerar las primeras 2 columnas (los índices)

Temperature.C.	Humidity...	TVOC.ppb.	eCO2.ppm.	Raw.H2	Raw.Ethanol	Pressure.hPa.	PM1.0	PM2.5	NC0.5
0.9087321	1.739793276	-1.232484	-0.4583843604	-4.211682	-3.93510811	1.225036	-1.9209495	-1.93497901	-1.912681
0.9098844	1.531975166	-1.232484	-0.4583843604	-3.973730	-3.50293964	1.246389	-1.9209495	-1.93497901	-1.912681
0.9109599	1.318133342	-1.232484	-0.4583843604	-3.796792	-3.13015309	1.232153	-1.9209495	-1.93497901	-1.912681
0.9143400	0.763951715	-1.232484	-0.4583843604	-3.522232	-2.35818804	1.201311	-1.9209495	-1.93497901	-1.912681
0.9154923	0.610347025	-1.232484	-0.4583843604	-3.442915	-2.16024828	1.232153	-1.9209495	-1.93497901	-1.912681
0.9177201	0.369398492	-1.232484	-0.4583843604	-3.345294	-1.84024567	1.279604	-1.9209495	-1.93497901	-1.912681
0.9188724	0.263983508	-1.232484	-0.4583843604	-3.314787	-1.70828584	1.274859	-0.7903782	2.65507914	-1.912681
0.9211002	0.077248395	-1.232484	-0.4583843604	-3.229368	-1.48065511	1.270114	-0.8657497	2.32721784	-1.912681
0.9221757	0.010987548	-1.232484	-0.4583843604	-3.229368	-1.36519026	1.272486	-0.9034354	2.16935870	-1.912681
0.9233280	-0.058285155	-1.232484	-0.4583843604	-3.217166	-1.30580833	1.279604	-0.9913687	1.84149740	-1.912681
0.9244035	-0.124546002	-1.232484	-0.4239509444	-3.223267	-1.23652941	1.241643	-1.0290544	1.65935224	-1.912681
0.9266313	-0.217913559	-1.232484	-0.4583843604	-3.150051	-1.09137359	1.279604	-1.1672353	1.12505976	-1.912681
0.9277836	-0.278150692	-1.232484	-0.4583843604	-3.168355	-1.03199166	1.293839	-1.2551687	0.77291244	-1.912681
0.9288591	-0.329352255	-1.232484	-0.4583843604	-3.143950	-0.95941375	1.286721	-1.2928544	0.61505330	-1.912681
0.9321624	-0.470909519	-1.232484	-0.4583843604	-3.107342	-0.80106195	1.239271	-1.5064067	-0.24710048	-1.912681
0.9333147	-0.488980659	-1.232484	-0.4583843604	-3.076835	-0.73838102	1.274859	-1.5315305	-0.35638758	-1.912681

Dataframe del training_set con los datos escalados.

PC1	PC2	Fire.Alarm
-2.03777968	-5.12993712	0
-2.21236286	-4.84472125	0
-2.34441541	-4.62601731	0
-2.59242915	-4.22857801	0
-2.66988014	-4.11140417	0
-2.78878998	-3.93642941	0
5.48061765	1.58555941	0
4.81124114	1.31252945	0
4.50174342	1.17409227	0
3.87457835	0.80076048	0
3.55673982	0.59548053	0
2.51051726	0.05762028	0
1.86654139	-0.34054812	0
1.54986534	-0.49859662	0
-0.03325842	-1.46304708	0
-0.27778035	-1.54865576	0
-0.87660874	-1.88198714	0
-1.21846560	-2.07825257	0

Dataframe del training_set después de la reducción.



Matriz de correlación

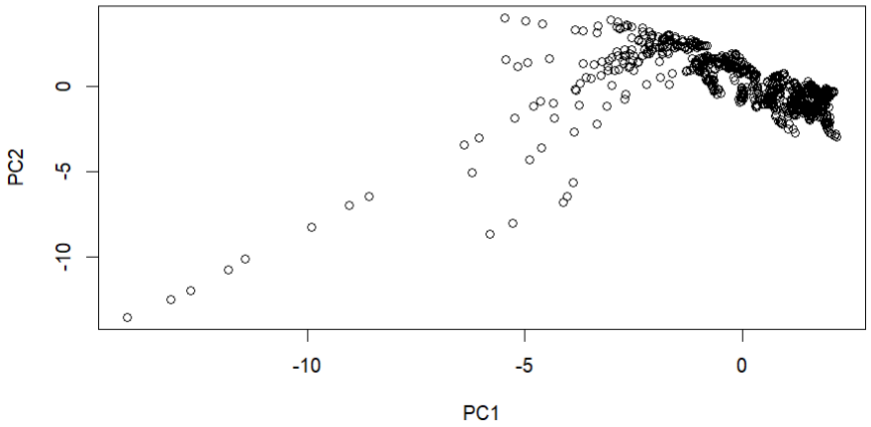


Gráfico de dispersión de las componentes principales.

V. Conclusiones

Las técnicas de reducción de dimensionalidad son una herramienta fundamental para el procesamiento de datos, ya que nos permiten describir de una forma más simple el comportamiento de los mismos por medio de métodos heurísticos que se basan en técnicas matemáticas, como lo es el análisis de componentes principales que se obtiene mediante la matriz de varianzas de las componentes, o el análisis lineal discriminante que se basa en la proyección de espacios vectoriales de tal forma que solo quedan aquellos vectores que aportan una dimensión a nuestro modelo.

VI. Referencias

Smoke Detection Dataset. (2022, 21 agosto). Kaggle. https://www.kaggle.com/datasets/deepcontractor/smoke-detection-dataset

RPubs - Introducción al paquete Caret. (2017, 11 agosto). https://rpubs.com/joser/caret

Comprehensive R Archive Network (CRAN). (2022, 24 octubre). *CRAN - Package e1071*. https://cran.r-project.org/web/packages/e1071/index.html