

Unsupervised

September 24, 2019

Universidad Nacional de Córdoba - Facultad de Matemática, Astronomía, Física y Computación

Diplomatura en Ciencia de Datos, Aprendizaje Automático y sus Aplicaciones

Práctico 5 - Aprendizaje No Supervisado - 2019

Integrante:

- Tarletta Juan

Trabajaremos con el mismo dataset utilizado en curación de datos (deben utilizar el dataset que se encuentra con los datos modificados o bien aplicar los mismos métodos del trabajo anterior para contar con datos válidos)

El dataset cuenta con X features, siendo las más importantes

- Componente: Indica a que componente pertenece la muestra
- Horas Funcionamiento: Indica la cantidad de horas de funcionamiento del camión (sería como el kilometraje de los camiones)
- Horas del Aceite: Representa la cantidad de horas de utilización del aceite (este dato es importante dado que a medida que, a mayor horas de uso del aceite, el mismo comienza a desgastarse)
- Resultado: (El laboratorio indica si la muestra de aceite está Bien = 1, Regular=2, Mal=3)
- St: Presencia de Hollin en el Aceite
- Al: Presencia de Aluminio en el Aceite
- Fe Presencia de Hierro en el Aceite
- Si Presencia de Silicio en el Aceite
- Na Presencia de Sodio en el Aceite

NOTA: se modifica le dataset y se incluyen el feature * Fecha de Análisis: Indica cuando fué analizada la muestra por el laboratorio

```
[1]: import numpy as np
import pandas as pd
import seaborn as sns
from IPython.display import display, Markdown

import matplotlib.pyplot as plt
%matplotlib inline
import matplotlib
matplotlib.rcParams['figure.figsize'] = 15, 8
```

```
[2]: dataset = pd.read_csv('dataset_intro.csv')
important_cols= ['Equipo', 'Componente', 'Resultado', 'Horas_
↳Funcionamiento', 'Horas del Aceite', 'St', 'Al', 'Fe', 'Si', 'Na' ]

#dataset[important_cols]
```

```
[3]: dataset[['Na']].count()
```

```
[3]: Na      21432
dtype: int64
```

```
[4]: import numpy as np
import matplotlib.pyplot as plt

from ml.visualization import plot_confusion_matrix, plot_learning_curve
from sklearn.datasets import load_wine
from sklearn.linear_model import SGDClassifier
from sklearn.metrics import accuracy_score, classification_report,
↳confusion_matrix
from sklearn.model_selection import GridSearchCV, train_test_split

seed = np.random.seed(1234)

%matplotlib inline
```

```
[5]: #Obtenemos el componente con más apariciones
compo = dataset[['Componente']].mode()
display(compo)
```

```
Componente
0      Motor
```

```
[6]: #Redefinimos el dataset a utilizar, en este caso solo con el componente Motor
dataset = dataset[dataset['Componente'] == 'Motor']
dataset.head(5)
```

```
[6]:      Unnamed: 0  Equipo Componente      Id Resultado  Horas Funcionamiento \
3              3      1355      Motor  76.084008         3.0         21950.0
12             12      1357      Motor  41.099310         1.0         27876.0
29             29      1356      Motor  82.751875         3.0         28295.0
49             49      1355      Motor  79.207921         3.0         22729.0
50             50      1355      Motor  82.456346         2.0         23153.0
```

```
      Horas del Aceite Fecha de Análisis      B  Nit  ...      Zn  Ag  Ti  \
3              409.0      2019-02-20    51.0  9.0  ...  1089.0  0.0  0.0
12             263.0      2017-11-05     NaN  6.0  ...  1246.0  0.0  0.0
29              NaN      2019-05-17    47.0 10.0  ...  1122.0  0.0  0.0
49             380.0      2019-04-03   100.0  6.0  ...  1334.0  0.0  0.0
50             424.0      2019-05-12   105.0  8.0  ...  1331.0  0.0  0.0
```

	V40	V100	TBN	TAN	ISO14	ISO4	ISO6
3	NaN	13.7	9.1	NaN	NaN	NaN	NaN
12	NaN	14.0	NaN	NaN	NaN	NaN	NaN
29	NaN	14.2	10.4	NaN	NaN	NaN	NaN
49	NaN	14.1	9.8	NaN	NaN	NaN	NaN
50	NaN	14.3	9.6	NaN	NaN	NaN	NaN

[5 rows x 45 columns]

```
[7]: dataset.columns
```

```
[7]: Index(['Unnamed: 0', 'Equipo', 'Componente', 'Id', 'Resultado',
        'Horas Funcionamiento', 'Horas del Aceite', 'Fecha de Análisis', 'B',
        'Nit', 'Oxi', 'Sul', 'St', 'V', 'Al', 'Cr', 'Cu', 'Fe', 'Pb', 'Mo',
        'Ni', 'Sn', 'Si', 'K', 'Na', 'W', 'F', 'A', 'ISO', 'PQI', 'Ba', 'Ca',
        'Mg', 'Mn', 'P', 'Zn', 'Ag', 'Ti', 'V40', 'V100', 'TBN', 'TAN', 'ISO14',
        'ISO4', 'ISO6'],
        dtype='object')
```

```
[8]: #utilizamos los datos que estan completos de varios features
dataset_reduce = dataset[['Resultado', 'Horas Funcionamiento', 'Horas del Aceite', 'Sul', 'V', 'Cr', 'Cu', 'Pb', 'Mo', 'Ni', 'K', 'St', 'PQI', 'Al', 'Fe', 'Si', 'Ca', 'Na', 'Mg', 'Zn', 'V100', 'B', 'Oxi', 'P']].dropna()
index_dataset_reduce = dataset_reduce.index.values.astype(int)
index_dataset_reduce.shape
```

```
[8]: (1982,)
```

A continuación evaluaremos k-means sobre el dataset de motor con varios Features.

```
[9]: from sklearn.cluster import KMeans
from sklearn.metrics import accuracy_score

seed
X = dataset_reduce
y = dataset_reduce['Resultado'] #Feature 'Resultado'
#Utilizaremos 2 Clusters
y_pred = KMeans(n_clusters=3, random_state=seed).fit_predict(X)

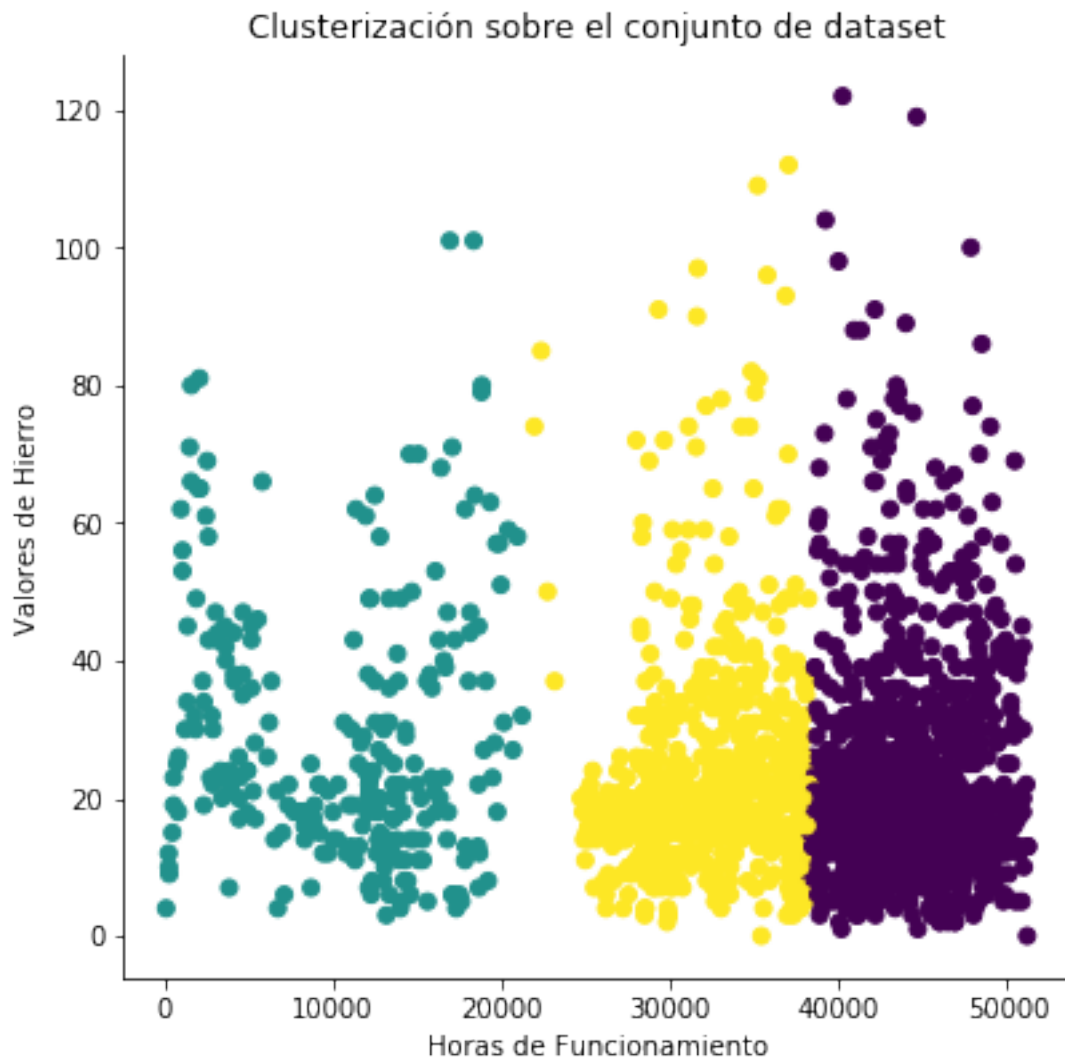
# Accuracy de KMeans
accuracy_kmeans = accuracy_score(y, y_pred)

print('Accuracy KMeans para el dataset: {}'.format(accuracy_kmeans))

plt.figure(figsize=(14, 14))
plt.subplot(221)
plt.scatter(X['Horas Funcionamiento'], X['Fe'], c=y_pred)
plt.title("Clusterización sobre el conjunto de dataset")
```

```
plt.xlabel('Horas de Funcionamiento')
plt.ylabel('Valores de Hierro')
sns.despine()
```

Accuracy KMeans para el dataset: 0.1437941473259334



El grafico es solo de referencia pero no es representativo del dataset, de todas formas observamos que el accuracy no es bueno.

Como 'buscamos' que la agrupacion del dataset se haga en base a la calidad de aceite, vamos a descartar algunos features que pueden estar metiendonos 'ruido'.

```
[10]: #utilizamos los datos que estan completos de varios features
dataset_reduce = dataset[['Resultado', 'Horas Funcionamiento', 'Horas del
    ↳ Aceite', 'St', 'Al', 'Fe', 'Si', 'Ca', 'Na']].dropna()
index_dataset_reduce = dataset_reduce.index.values.astype(int)
```

```
index_dataset_reduce.shape
```

```
[10]: (1983,)
```

```
[11]: from sklearn.cluster import KMeans
      from sklearn.metrics import accuracy_score

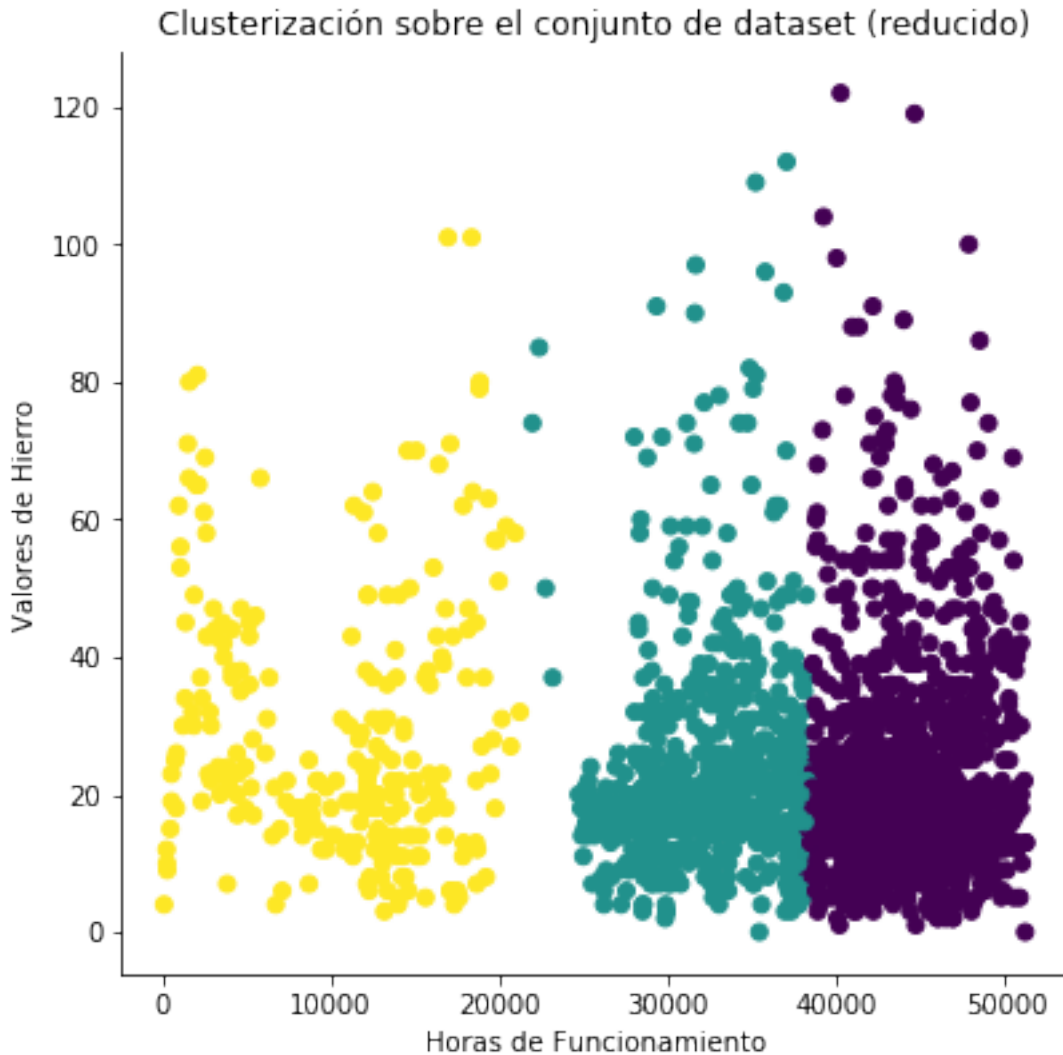
      seed
      X = dataset_reduce
      y = dataset_reduce['Resultado'] #Feature 'Resultado'
      #Utilizaremos n Clusters
      y_pred = KMeans(n_clusters=3, random_state=seed).fit_predict(X)

      # Accuracy de KMeans
      accuracy_kmeans = accuracy_score(y, y_pred)

      print('Accuracy KMeans para el dataset (reducido): {}'.format(accuracy_kmeans))

      plt.figure(figsize=(14, 14))
      plt.subplot(221)
      plt.scatter(X['Horas Funcionamiento'], X['Fe'], c=y_pred)
      plt.title("Clusterización sobre el conjunto de dataset (reducido)")
      plt.xlabel('Horas de Funcionamiento')
      plt.ylabel('Valores de Hierro')
      sns.despine()
```

```
Accuracy KMeans para el dataset (reducido): 0.1956631366616238
```



Observamos que por más que reduzcamos los features, (y que esté contenido el feature 'Resultado' en el dataset), el accuracy sigue siendo malo

A continuación realizaremos scalado/noramlizado de los valores, para evitar que algún feature tenga más peso en la distancia a los centroides

```
[12]: #Separamos el dataset
X = dataset_reduce
Y = dataset_reduce['Resultado'] #Feature 'Resultado'
```

```
[13]: #Escalamos/Noramlizamos los datos
from sklearn import preprocessing

scaler = preprocessing.StandardScaler().fit(X)

# Transformamos los datos de entrenamiento (Media=0 y Desviación_Estandar=1)
X_scaled = scaler.transform(X)
```

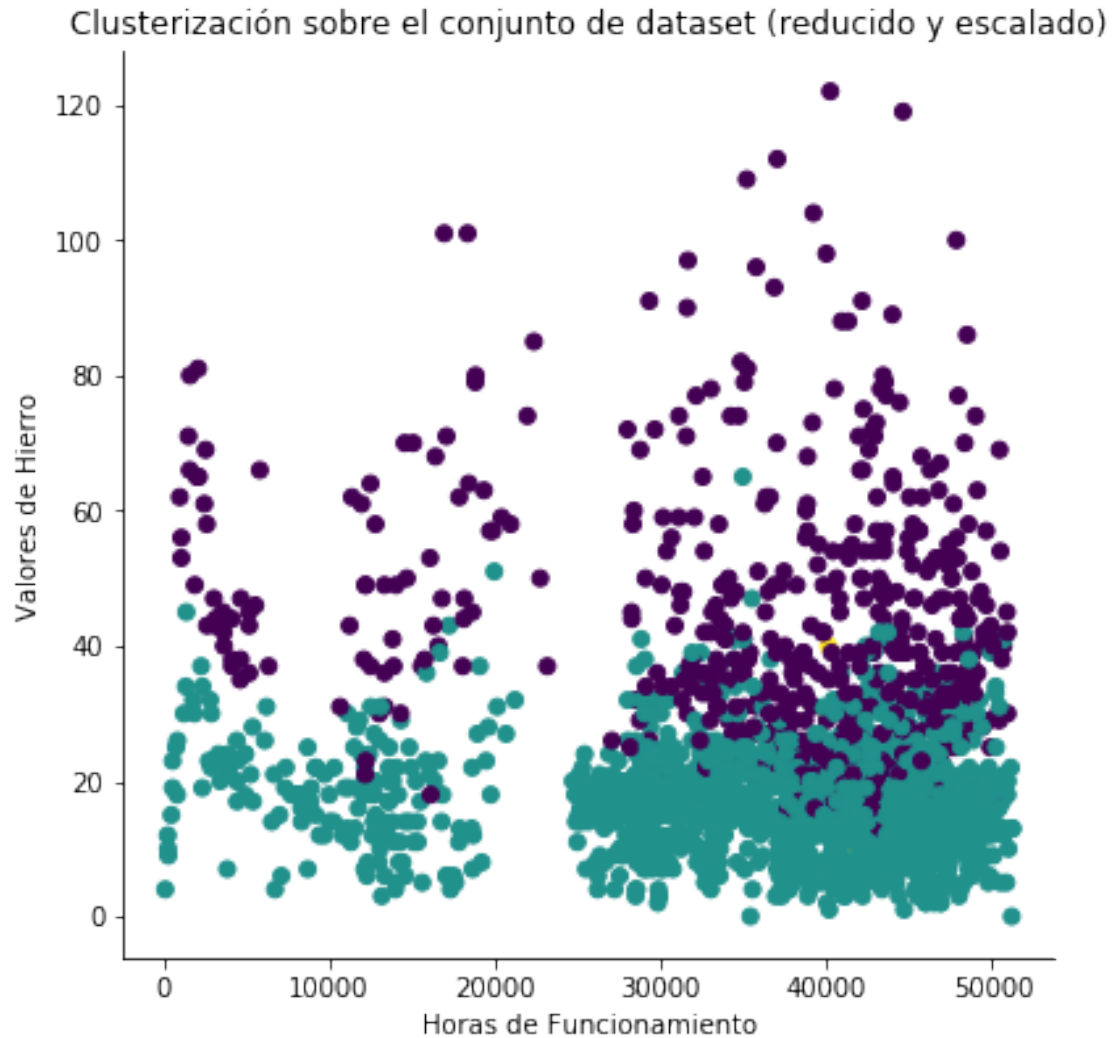
```
[14]: #Clusterizamos con el dataset escalado
y_pred = KMeans(n_clusters=3, random_state=seed).fit_predict(X_scaled)

# Accuracy de KMeans
accuracy_kmeans = accuracy_score(y, y_pred)

print('Accuracy KMeans para el dataset (reducido y escalado): {}'.
      →format(accuracy_kmeans))

plt.figure(figsize=(14, 14))
plt.subplot(221)
plt.scatter(X['Horas Funcionamiento'], X['Fe'], c=y_pred)
plt.title("Clusterización sobre el conjunto de dataset (reducido y escalado)")
plt.xlabel('Horas de Funcionamiento')
plt.ylabel('Valores de Hierro')
sns.despine()
```

Accuracy KMeans para el dataset (reducido y escalado): 0.5370650529500757



Podemos observar una mejora en los resultados

A continuación aplicaremos PCA para obtener features que nos brinden mayor variabilidad en el dataset

```
[15]: X = dataset_reduce.iloc[:,1:] #Features sin 'Resultado'
      y = dataset_reduce['Resultado'] #Feature 'Resultado'
```

```
[16]: from sklearn.decomposition import PCA as sklearnPCA
      sklearn_pca = sklearnPCA(n_components=3)
      X_pca_sklearn = sklearn_pca.fit_transform(X)
      X_pca_sklearn.shape
```

```
[16]: (1983, 3)
```

```
[17]: #Clusterizamos con el dataset PCA
      y_pred = KMeans(n_clusters=3, random_state=seed).fit_predict(X_pca_sklearn)
```



```

print(sklearn_pca.singular_values_, sklearn_pca.explained_variance_ratio_)
# Accuracy de KMeans
accuracy_kmeans = accuracy_score(y, y_pred)

print('Accuracy KMeans para el PCA del dataset (reducido): {}'.
      →format(accuracy_kmeans))

plt.figure(figsize=(14, 14))
plt.subplot(221)
plt.scatter(X['Horas Funcionamiento'], X['Fe'], c=y_pred)
plt.title("Clusterización sobre el PCA del dataset (reducido)")
plt.xlabel('Horas de Funcionamiento')
plt.ylabel('Valores de Hierro')
sns.despine()

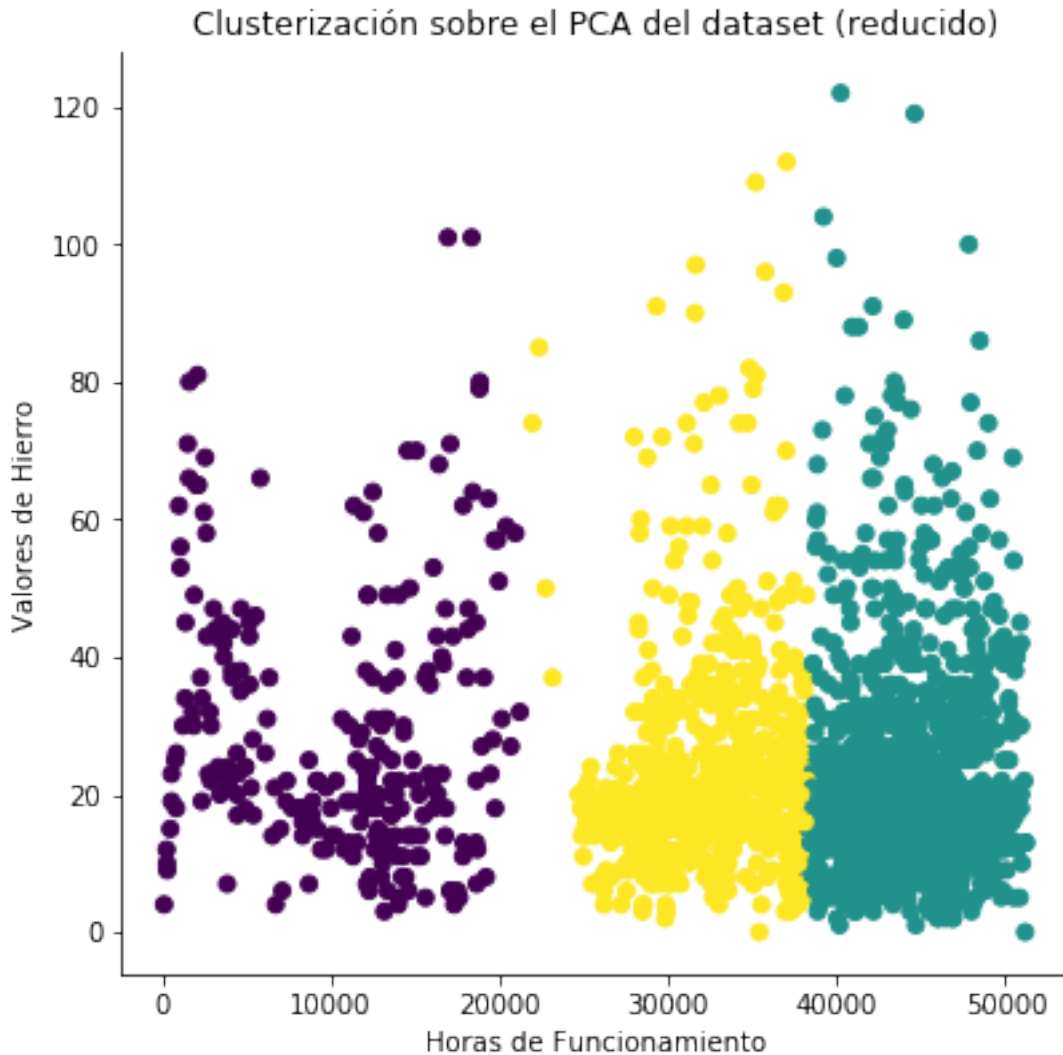
```

```

[516532.3837571  13377.52045999  5919.69385633] [9.99077526e-01 6.70124241e-04
1.31220768e-04]

```

Accuracy KMeans para el PCA del dataset (reducido): 0.4402420574886536



No observamos mejora con PCA, solo que tenemos un solo feature que nos representa el 99% de la variabilidad en los datos

A continuación realizaremos PCA pero con los datos escalados

```
[18]: #Separamos el dataset
X = dataset_reduce.iloc[:,1:]
Y = dataset_reduce['Resultado'] #Feature 'Resultado'
```

```
[19]: #Escalamos/Normalizamos los datos
from sklearn import preprocessing

scaler = preprocessing.StandardScaler().fit(X)

# Transformamos los datos de entrenamiento (Media=0 y Desviación_Estandar=1)
X_scaled = scaler.transform(X)
```

```
[20]: sklearn_pca = sklearnPCA(n_components=5)
X_pca_sklearn = sklearn_pca.fit_transform(X_scaled)
X_pca_sklearn.shape
```

```
[20]: (1983, 5)
```

```
[42]: print(sklearn_pca.singular_values_, sklearn_pca.explained_variance_ratio_)

y_pred = KMeans(n_clusters=3, random_state=seed).fit_predict(X_pca_sklearn)

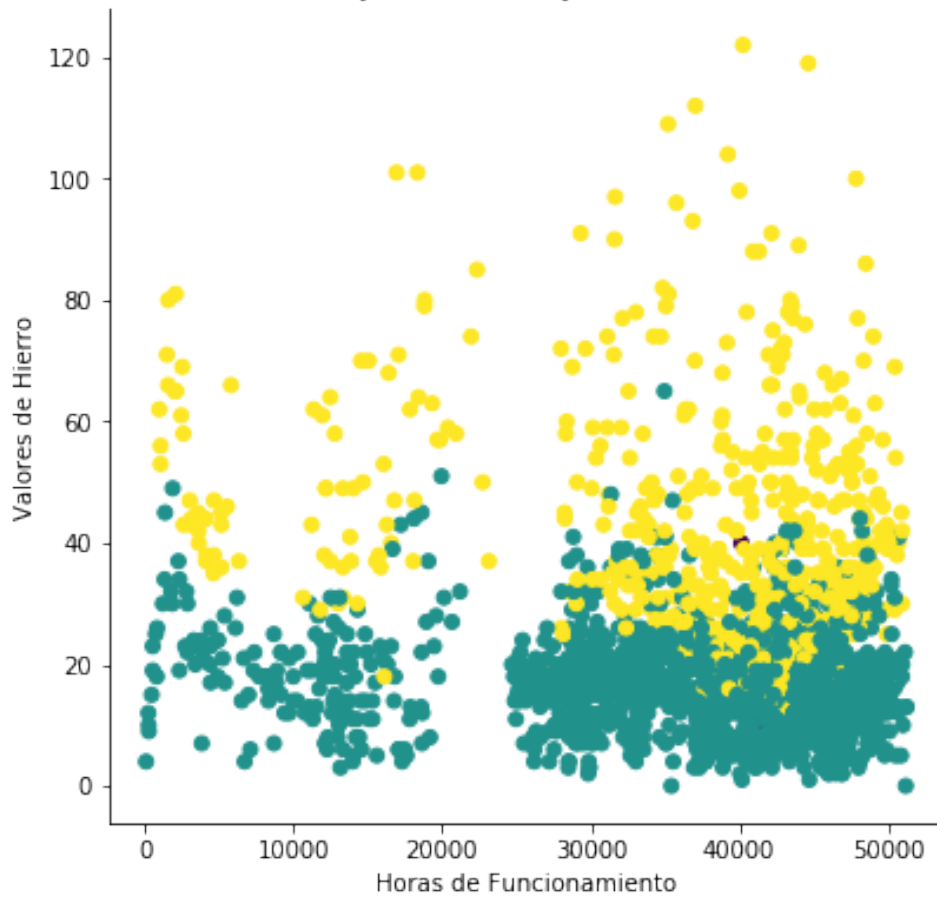
# Accuracy de KMeans
accuracy_kmeans = accuracy_score(y, y_pred)

print('Accuracy KMeans para el PCA del dataset (reducido y escalado): {}'.
      →format(accuracy_kmeans))

plt.figure(figsize=(14, 14))
plt.subplot(221)
plt.scatter(X['Horas Funcionamiento'], X['Fe'], c=y_pred)
plt.title("Clusterización sobre el conjunto scalado y con reduccion de_
      →dimensionalidad")
plt.xlabel('Horas de Funcionamiento')
plt.ylabel('Valores de Hierro')
sns.despine()
```

```
[68.26731915 56.76908018 47.83255803 43.67988954 41.97783209] [0.29377376
0.20314728 0.144223 0.12026808 0.11107781]
Accuracy KMeans para el PCA del dataset (reducido y escalado): 0.642965204236006
```

Clusterización sobre el conjunto scalado y con reduccion de dimensionalidad



Observamos que alcanzamos una mejora mayor en los resultados, aunque no es óptima. El algoritmo identifica mejor 2 de las 3 clases, que en este caso son las mayoritarias.