Índice

Integrantes del grupo

José Manuel Vega Gradit, Víctor Montesdeoca Fenoy e Iván Eugenio Tello López.

Descripción del problema

Problema Real

Una fábrica se encuentra con el reto de identificar unos cilindros y clasificarlos en dos grandes grupos: metales y rocas. El primero será utilizado en la construcción de un parque de atracciones, mientras que el segundo se utilizará con el propósito de fabricar elementos decorativos.

Para la empresa es de vital importancia minimizar la aparición de falsos positivos en la detección de metales y que utilizar rocas en lugar de metales para la construcción de las atracciones resultaría catastrófico.

Resolución del Problema

Consideramos como positivo que los cilindros sean metálicos. Debido a que los cilindros metálicos se van a usar para la construcción de una atracción, debemos priorizar que no haya falsos positivos ya que estos pondrían en peligro a la gente que se monte en la atracción.

Por tanto buscamos maximizar la **precisión** en la matriz de confusión.

Librerías empleadas

- Pandas: Librería de código abierto de python para la manipulación, análisis y visualización de datos, las características más atractivas que motivan el uso de esta librería son la flexibilidad, la rapidez y lo poderosa que es.

- Plotly: Librería de código abierto de python para realizar gráficas interactivas y de alta calidad.

- Cufflinks: Librería de python que conecta plotly con pandas para crear gráficas directamente sobre data frames.

- Sklearn: Es un conjunto de rutinas escritas en Python para hacer análisis predictivo, que incluyen clasificadores, algoritmos de clusterización, etc. Está basada en NumPy, SciPy y matplotlib.

Proceso

Análisis Exploratorio de Datos

En primer lugar, visualizamos las 10 primeras filas del conjuntos de datos para hacernos a la idea de cómo son los datos.

```
df.head(10)
```

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	...	V52	V53	V54	V55	V56	V57	V58	V59	V60	Class
0	0.0200	0.0371	0.0428	0.0207	0.0954	0.0986	0.1539	0.1601	0.3109	0.2111	...	0.0027	0.0065	0.0159	0.0072	0.0167	0.0180	0.0084	0.0090	0.0032	R
1	0.0453	0.0523	0.0843	0.0689	0.1183	0.2583	0.2156	0.3481	0.3337	0.2872	...	0.0084	0.0089	0.0048	0.0094	0.0191	0.0140	0.0049	0.0052	0.0044	R
2	0.0262	0.0582	0.1099	0.1083	0.0974	0.2280	0.2431	0.3771	0.5598	0.6194	...	0.0232	0.0166	0.0095	0.0180	0.0244	0.0316	0.0164	0.0095	0.0078	R
3	0.0100	0.0171	0.0623	0.0205	0.0205	0.0368	0.1098	0.1276	0.0598	0.1264	...	0.0121	0.0036	0.0150	0.0085	0.0073	0.0050	0.0044	0.0040	0.0117	R
4	0.0762	0.0666	0.0481	0.0394	0.0590	0.0649	0.1209	0.2467	0.3564	0.4459	...	0.0031	0.0054	0.0105	0.0110	0.0015	0.0072	0.0048	0.0107	0.0094	R
5	0.0286	0.0453	0.0277	0.0174	0.0384	0.0990	0.1201	0.1833	0.2105	0.3039	...	0.0045	0.0014	0.0038	0.0013	0.0089	0.0057	0.0027	0.0051	0.0062	R
6	0.0317	0.0956	0.1321	0.1408	0.1674	0.1710	0.0731	0.1401	0.2083	0.3513	...	0.0201	0.0248	0.0131	0.0070	0.0138	0.0092	0.0143	0.0036	0.0103	R
7	0.0519	0.0548	0.0842	0.0319	0.1158	0.0922	0.1027	0.0613	0.1465	0.2838	...	0.0081	0.0120	0.0045	0.0121	0.0097	0.0085	0.0047	0.0048	0.0053	R
8	0.0223	0.0375	0.0484	0.0475	0.0647	0.0591	0.0753	0.0098	0.0684	0.1487	...	0.0145	0.0128	0.0145	0.0058	0.0049	0.0065	0.0093	0.0059	0.0022	R
9	0.0164	0.0173	0.0347	0.0070	0.0187	0.0671	0.1056	0.0697	0.0962	0.0251	...	0.0090	0.0223	0.0179	0.0084	0.0068	0.0032	0.0035	0.0056	0.0040	R

10 rows × 61 columns

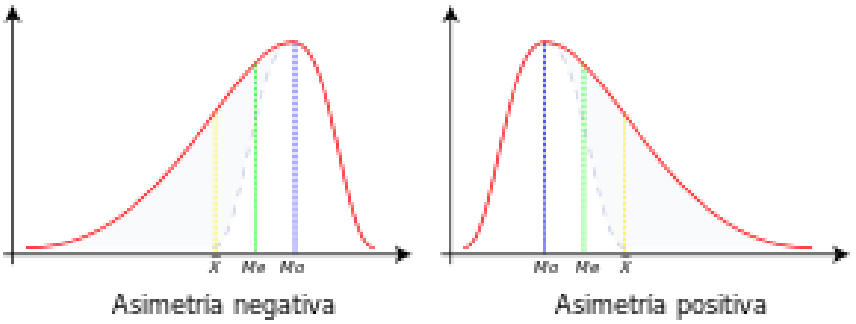
Parece que, salvando la columna de las clases todas las variables son numéricas. Como esto de vital importancia para utilizar los modelos con la librería de Sklearn (no funciona con variables categóricas), nos aseguramos viendo el tipo de datos de cada columna:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 208 entries, 0 to 207
Data columns (total 61 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   V1      208 non-null    float64
 1   V2      208 non-null    float64
 2   V3      208 non-null    float64
 3   V4      208 non-null    float64
 4   V5      208 non-null    float64
 5   V6      208 non-null    float64
 6   V7      208 non-null    float64
 7   V8      208 non-null    float64
 8   V9      208 non-null    float64
 9   V10     208 non-null    float64
 10  V11     208 non-null    float64
 11  V12     208 non-null    float64
 12  V13     208 non-null    float64
 13  V14     208 non-null    float64
 14  V15     208 non-null    float64
 15  V16     208 non-null    float64
 16  V17     208 non-null    float64
 17  V18     208 non-null    float64
 18  V19     208 non-null    float64
 19  V20     208 non-null    float64
 20  V21     208 non-null    float64
 21  V22     208 non-null    float64
 22  V23     208 non-null    float64
 23  V24     208 non-null    float64
 24  V25     208 non-null    float64
 25  V26     208 non-null    float64
 26  V27     208 non-null    float64
 27  V28     208 non-null    float64
 28  V29     208 non-null    float64
 29  V30     208 non-null    float64
 30  V31     208 non-null    float64
 31  V32     208 non-null    float64
 32  V33     208 non-null    float64
 33  V34     208 non-null    float64
 34  V35     208 non-null    float64
 35  V36     208 non-null    float64
 36  V37     208 non-null    float64
 37  V38     208 non-null    float64
 38  V39     208 non-null    float64
 39  V40     208 non-null    float64
 40  V41     208 non-null    float64
 41  V42     208 non-null    float64
 42  V43     208 non-null    float64
 43  V44     208 non-null    float64
 44  V45     208 non-null    float64
 45  V46     208 non-null    float64
 46  V47     208 non-null    float64
 47  V48     208 non-null    float64
 48  V49     208 non-null    float64
 49  V50     208 non-null    float64
 50  V51     208 non-null    float64
 51  V52     208 non-null    float64
 52  V53     208 non-null    float64
 53  V54     208 non-null    float64
 54  V55     208 non-null    float64
 55  V56     208 non-null    float64
 56  V57     208 non-null    float64
 57  V58     208 non-null    float64
 58  V59     208 non-null    float64
 59  V60     208 non-null    float64
 60  Class   208 non-null    object
```

Tras esto, para poder entender mejor los resultados que vamos a obtener y ver si sería necesario normalizar los datos, miramos la distribución de los datos.

Primero, calculamos la métrica de *skewness* o asimetría estadística, que nos dirá la distribución que siguen nuestros datos, para cada columna. Si esta es negativa significará que la distribución es asimétrica positiva o a la derecha. mientras que si es positiva entonces la distribución es asimétrica negativa o a la izquierda. Además, si la asimetría es menor de -1 o mayor de 1, esto significa que los datos siguen una distribución muy asimétrica.



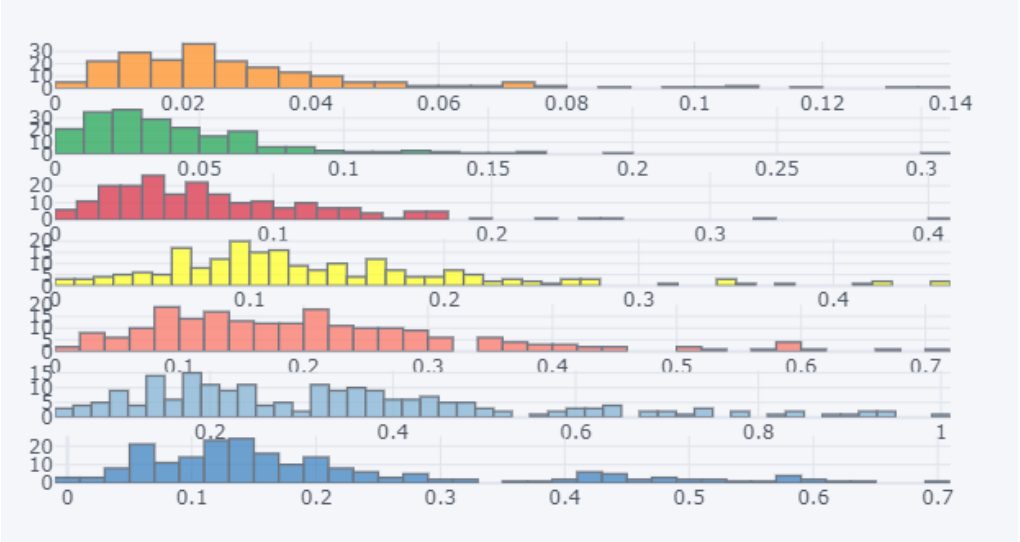
Buscamos las columnas con una asimetría menor que -1 o mayor que 1:

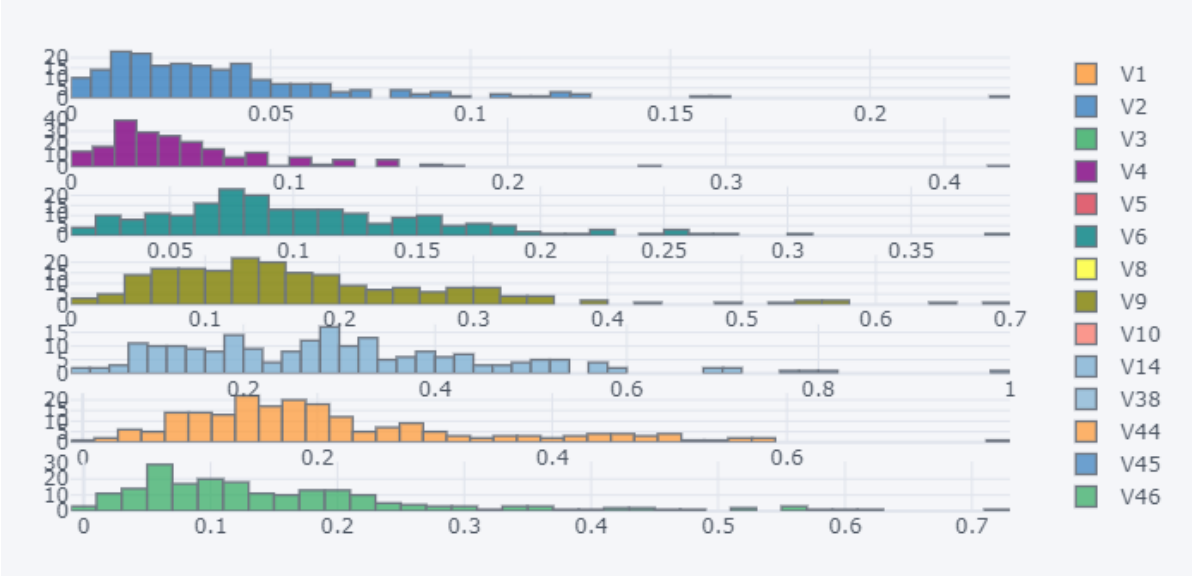
```
skewCols = df.skew()
skewedCols = skewCols[skewCols.ge(1) | skewCols.le(-1)]
skewedCols
```

```
V1     2.131088
V2     2.155644
V3     2.652518
V4     3.401697
V5     2.018141
V6     1.248166
V8     1.481107
V9     1.633870
V10    1.281258
V14    1.022369
V38    1.033366
V44    1.235086
V45    1.366839
V46    1.706674
V47    1.790155
V48    1.277722
V49    1.273385
V50    1.761714
V51    2.716060
V52    2.093976
V53    1.060572
V54    1.093090
V55    1.789946
V56    1.780805
V57    1.653090
V58    2.098330
V59    1.737506
V60    2.775754
```

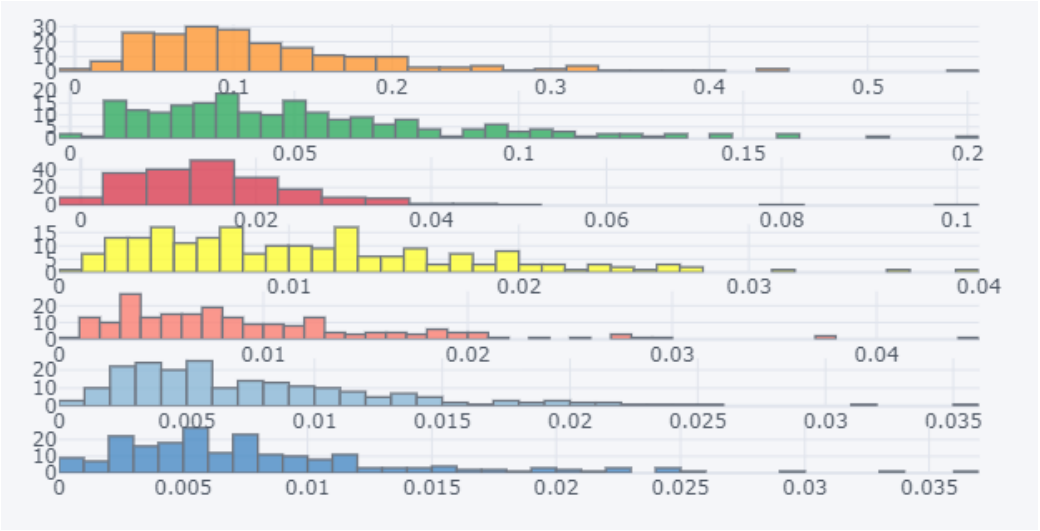
Hay un total de 28 columnas que cumplan esta condición. De manera gráfica, representando mediante histogramas con 50 barras, podmeos observar las distribuciones de cada una de ellas.

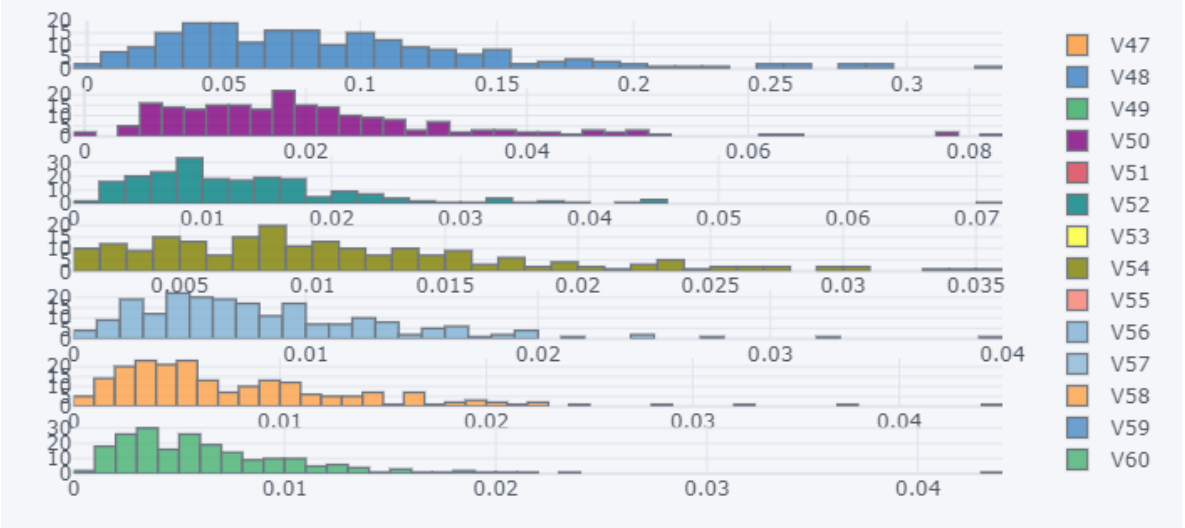
Las primeras 14 columnas:





Las segundas 14:





Estudio de posibles modelos

Los modelos de aprendizaje automático supervisado que vamos a estudiar son DecisionTree, RandomForest, SVM, Naive-Bayes y Naive-Bayes Multinomial.

DecisionTree

Es un algoritmo de clasificación que usa un modelo basado en un árbol de decisiones con sus consecuencias para predecir resultados.

```
print(classification_report(y_train, y_train_pred))
```

	precision	recall	f1-score	support
M	1.00	1.00	1.00	85
R	1.00	1.00	1.00	81
accuracy			1.00	166
macro avg	1.00	1.00	1.00	166
weighted avg	1.00	1.00	1.00	166

```
print(classification_report(y_test, y_test_pred))
```

	precision	recall	f1-score	support
M	0.80	0.62	0.70	26
R	0.55	0.75	0.63	16
accuracy			0.67	42
macro avg	0.67	0.68	0.66	42
weighted avg	0.70	0.67	0.67	42

Podemos ver con con el conjunto de train los resultados obtenidos son óptimos, sin embargo al trasladar el análisis al conjunto de test los resultados son mucho peores. Motivo por el que podemos concluir que ha habido overfitting o *sobreajuste*. Con esto en mente, realizamos Pre-Prunning usando GridSearch para encontrar el arbol más óptimo de entre una serie de árboles con unos parámetros de hojas y anchura ya dados:

```
params = {'max_depth': [2,4,6,8,10,12],
          'min_samples_split': [2,3,4,5],
          'min_samples_leaf': [1,2,3]}
```

```
tree = DecisionTreeClassifier(random_state=0)
gcv = GridSearchCV(estimator=tree,param_grid=params)
gcv.fit(X_train, y_train)
gcv
```

```
GridSearchCV(estimator=DecisionTreeClassifier(random_state=0),
             param_grid={'max_depth': [2, 4, 6, 8, 10, 12],
                         'min_samples_leaf': [1, 2, 3],
                         'min_samples_split': [2, 3, 4, 5]})
```

```
print(classification_report(y_train, y_train_pred))
```

```
              precision    recall  f1-score   support

           M       0.99      0.99      0.99        85
           R       0.99      0.99      0.99        81

    accuracy                           0.99       166
   macro avg       0.99      0.99      0.99       166
weighted avg       0.99      0.99      0.99       166
```

```
print(classification_report(y_test, y_test_pred))
```

```
              precision    recall  f1-score   support

           M       0.75      0.58      0.65        26
           R       0.50      0.69      0.58        16

    accuracy                           0.62        42
   macro avg       0.62      0.63      0.62        42
weighted avg       0.65      0.62      0.62        42
```

Habiendo realizado el Pre-Prunning vemos cómo ya no se produce overfitting como en el caso anterior pero los resultados obtenidos siguen lejos de ser óptimos con una precisión de 0,75.

RandomForest

Es un algoritmo utilizado para clasificación basado en una combinación de árboles predictores tal que cada árbol depende de los valores de un vector aleatorio probado independientemente y con la misma distribución para cada uno de estos.

```
print(classification_report(y_test,y_pred_rf))
```

```
              precision    recall  f1-score   support

           M       0.91      0.77      0.83        26
           R       0.70      0.88      0.78        16

    accuracy                           0.81        42
   macro avg       0.80      0.82      0.81        42
weighted avg       0.83      0.81      0.81        42
```

Para RandomForest, el valor de **precisión** buscado tiene un valor de 0.91

SVM

El objetivo de SVM es encontrar un hiperplano que separe dos clases diferentes de puntos de datos con el margen más amplio entre ambas clases.

```
print(classification_report(y_test,y_pred_svm))
```

```
              precision    recall  f1-score   support

           M       0.96      0.88      0.92        26
           R       0.83      0.94      0.88        16

    accuracy                           0.90        42
   macro avg       0.90      0.91      0.90        42
weighted avg       0.91      0.90      0.91        42
```

La precisión para SVM es de 0.96, siendo esta la mayor hasta el momento.

Naive-Bayes

Es un algoritmo basado en el teorema de bayes para clasificar objetos que asume que las variables del dataset no están relacionadas entre ellas.

```
print(classification_report(y_test,y_pred_nb_gaussian))
```

```
              precision    recall  f1-score   support

           M       0.94      0.62      0.74        26
           R       0.60      0.94      0.73        16

    accuracy                           0.74        42
   macro avg       0.77      0.78      0.74        42
weighted avg       0.81      0.74      0.74        42
```

Para Naive-Bayes tiene un 0.94 de precisión.

Naive-Bayes Multinomial

Se trata de un algoritmo basado en el teorema de Bayes para la clasificación de objetos que asume la independencia entre las variables del dataset, se diferencia de Naive-Bayes en que cuenta con un vector que indica el número de apariciones de cada elemento en particular.

```
print(classification_report(y_test,y_pred_nb_multinomial))
```

```
              precision    recall  f1-score   support

           M       0.88      0.85      0.86        26
           R       0.76      0.81      0.79        16

    accuracy                           0.83        42
   macro avg       0.82      0.83      0.83        42
weighted avg       0.84      0.83      0.83        42
```

La precisión para Naive-Bayes Multinomial es de 0.88 de precisión.

Modelo elegido

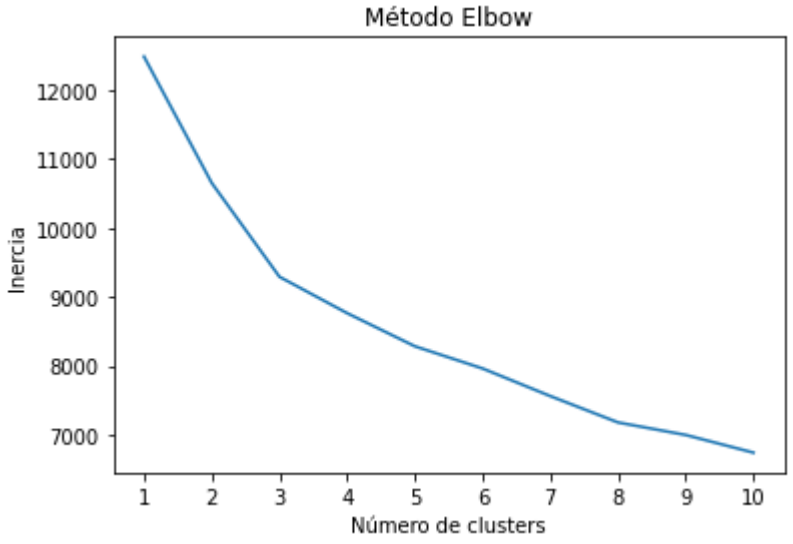
Por lo mostrado en el apartado anterior decidimos que el mejor modelo para este problema es SVM con un 0.96 de precisión.

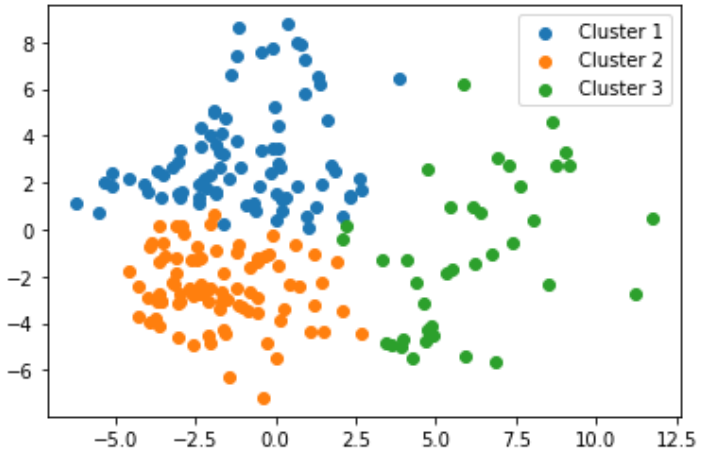
Clustering

Hemos utilizado algoritmos de aprendizaje no supervisado para intentar encontrar características comunes que permitan distinguir entre los metales y las rocas.

Elección del número de clusters

Utilizando el método del codo sobre el dataset, como el punto en el que se observa el cambio más brusco en la inercia nos dirá el número óptimo de Clusters, y en este caso ese punto es 3, el número total de clusters escogidos por el método del codo es 3.



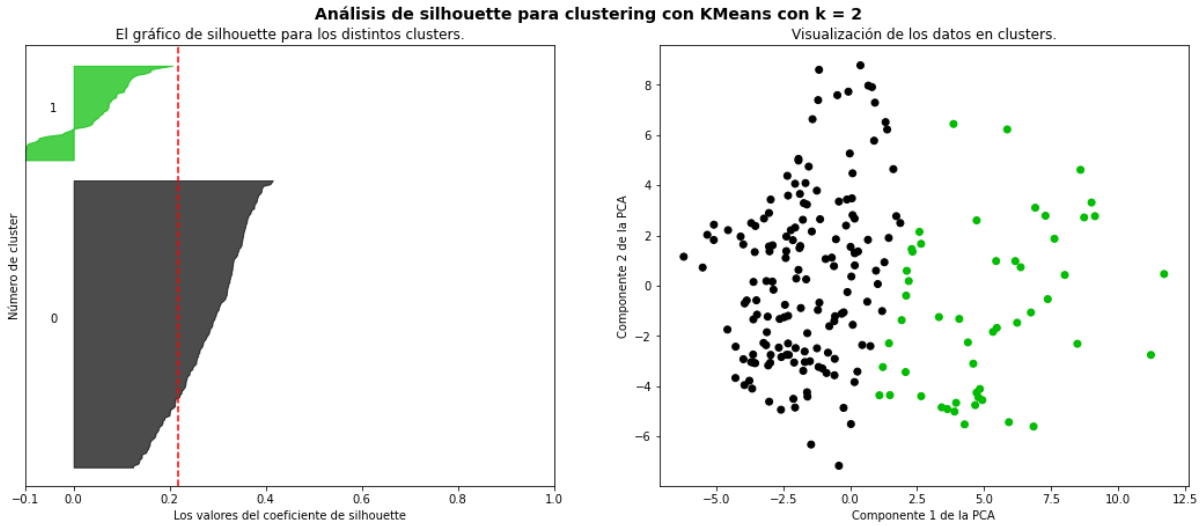


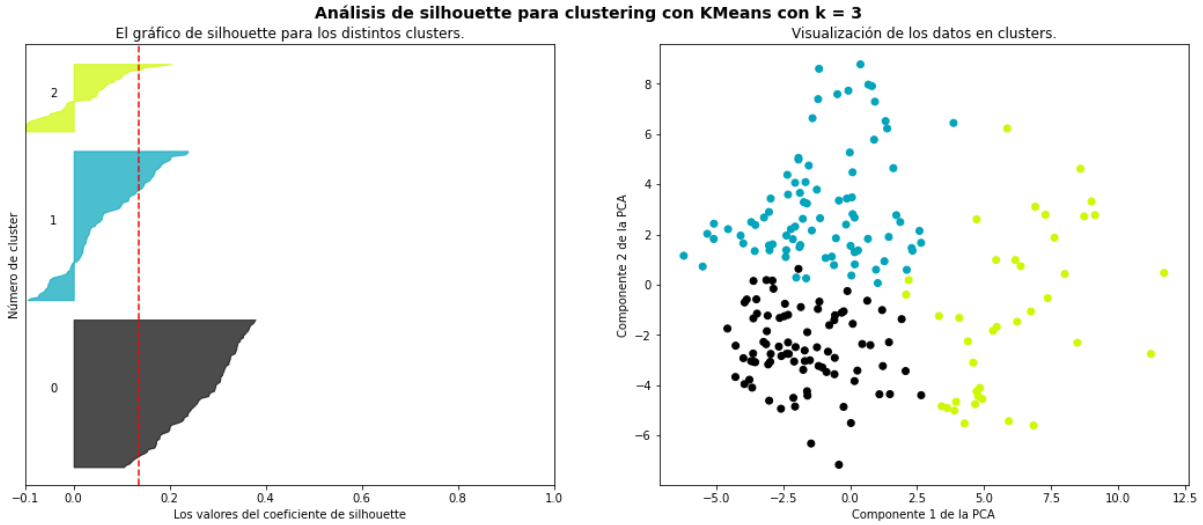
No obstante, no parece muy clara la distinción entre clusters, especialmente en la frontera de los Clusters 1 y 2. Por ello, vamos a buscar el número óptimo de clusters usando otro método.

Para ello vamos a utilizar la métrica de silhoutte la cual consiste en medir cómo de similar es un objeto de un cluster con sus vecinos pertenecientes al mismo y la diferencia con objetos de otros clusters, este ofrece valores comprendidos entre -1 y 1, donde un valor alto indica que un objeto se relaciona de forma adecuada con sus vecinos y se diferencia claramente de elementos de otros clusters. Con esta métrica los resultados obtenidos son:

Para k = 2 La media de silhouette es : 0.2178194633782484
Para k = 3 La media de silhouette es : 0.13471787253074766
Para k = 4 La media de silhouette es : 0.13649098052831896
Para k = 5 La media de silhouette es : 0.12229111527357212
Para k = 6 La media de silhouette es : 0.13686609352797682
Para k = 7 La media de silhouette es : 0.10803864268778789
Para k = 8 La media de silhouette es : 0.11879259314239257
Para k = 9 La media de silhouette es : 0.1165921836401728
Para k = 10 La media de silhouette es : 0.11484793099580967

Como la media más alta de los valores es 0.2178… para k = 2, el número recomendado de clusters es 2.





Viendo la gráfica para ambos números de clusters (k = 2 y k = 3), podemos ver que para k = 2 sí hay una distinción algo más marcada entre los miembros de uno y otro conjunto.

Conclusión de los resultados

Con los resultados obtenidos, podemos asegurar que de cada 100 cilindros que clasifiquemos como un metal, 96 serán siempre metal. Esto garantiza el menor número posible de falsos positivos y la mayor seguridad para la atracción.

Por otro lado, para hacer una mejor interpretación de los clusters, se requeriría de trabajo adicional. Una vez visto que bajo nuestro criterio el número más óptimo de conjuntos es de 2 (lo cual coincide con el número de categorías para nuestro problema de clasificación), Podríamos añadir el cluster como variable en el conjunto de datos para ver si los 2 conjuntos obtenidos se corresponden con la categoría de metal y roca o hacer otros estudios.