Aprendizaje no supervisado

Lección 3: Preparación y procesamiento de los datos antes de agruparlos

Introducción

Supongamos que el especialista en bases de datos de la empresa de cosméticos donde tú trabajas te envía los datos organizados para que puedas agruparlos. En los datos aparece una variable que indica el color de base facial que compró cada cliente; por ejemplo: café, beige claro o rosado oscuro. ¿Te has preguntado cómo puedes realizar operaciones matemáticas sobre esos valores que, en principio, son una palabra o un texto? Si también te preguntas, ¿por qué debo hacer operaciones matemáticas? Es porque los algoritmos de agrupamiento tienen un fundamento matemático.

En este tema aprenderás como transformar los datos para que puedas aplicar los algoritmos de agrupamiento de datos que ya están implementados en varias plataformas, en particular en *Scikit-learn*. Seguramente, algo ya conoces acerca del preprocesamiento o transformación de los datos, pero es importante que conozcas cuáles de estos procesos son más utilizados antes de agrupar. De esta manera, serás capaz de transformar datos categóricos, incluso datos numéricos, dejando así los datos listos para obtener resultados excelentes.

Como científico de datos, tu labor es más que simplemente recopilar y organizar datos. Tu rol se enfoca en analizar y mostrar información de forma coherente para que los líderes de tu organización puedan:

- Observar propiedades de interés en los datos.
- Determinar relaciones entre ellos.

- Sacar inferencias o conclusiones sobre lo que se ve.
- En fin, apoyar la toma de decisiones.

Transformación de datos numéricos

Imagina que trabajas en una empresa de biotecnología de las plantas, donde uno de sus productos más vendidos es la flor Iris. El director nacional de ventas requiere de un informe de las ventas de los últimos seis meses y te proporciona la información que descargaron de la base de datos.

¡Comencemos!

Presentación de datos

Los datos con los que trabajarás como científico de datos casi siempre aparecen en forma de tabla. En inglés se les conoce como ***raw data***. La mayoría de las plataformas que implementan varios de los algoritmos que utilizarás asumen que los datos siguen esa estructura.

Claro que hay situaciones donde será también tu labor crear esa tabla de datos a partir de datos que provienen de diferentes fuentes. Nosotros asumiremos que ese paso ya fue realizado previamente y contamos con los datos en forma de tabla, como en el siguiente ejemplo lo podrás observar:

Iris	sepallength	sepalwidth	petallength	petalwidth	class
1	5.1	3.5	1.4	0.2	Iris-setosa
2	4.9	3.0	1.4	0.2	Iris-setosa
3	7.0	3.2	4.7	1.4	Iris-versicolor
4	6.4	3.2	4.5	1.5	Iris-versicolor
5	6.3	3.3	6.0	2.5	Iris-virginica
6	5.8	2.7	5.1	1.9	Iris-virginica

- **Aplicación**

Considera la situación planteada al inicio sobre la empresa de biotecnología de las plantas, donde uno de sus productos más vendidos es la flor Iris. Este es el punto de partida para analizar los movimientos de ventas de los últimos meses.

Es por eso que el jefe de producción de la empresa quiere agrupar las flores de acuerdo con sus características para planificar el desarrollo de distintos procesos de interés. Por lo tanto, te envían los datos de las flores en forma de tabla, incluyendo la especie a la que cada una pertenece. Sin embargo, el dato de la especie no es relevante esta vez y debemos eliminarlo de los datos.

¿Cómo lo harías utilizando *Python*?

- ***Scikit-learn***

La plataforma *Scikit-learn* tiene una base de datos similar a la del ejemplo. Vamos a cargarla en un *Google Colab* y eliminar la columna clase (especie). Recuerda importar la librería. El código podría ser así:

Analiza este código. Quizás esperabas cargar los datos desde un archivo con extensión *.csv* utilizando la librería pandas. Es posible que en tu labor diaria lo hagas precisamente usando pandas, pero en este caso vamos a utilizar una base de datos de estudio que te guiará en el aprendizaje de estos conceptos. ¿Puedes cargar por tu cuenta unos datos de ejemplo que estén en un archivo *.csv* y eliminar una columna usando *pandas*? Seguramente sí.

```
# Importando librerias
from sklearn import datasets
import numpy as np

# Cargando iris
iris = datasets.load_iris()

# Cargando los datos sin la clase
X = np.array(iris.data)
```

La función *load_iris*() carga los datos de la base de datos hacia una variable. Estos datos tienen dos campos principales: data y target. Como habrás adivinado, en el campo data están todos los datos de las flores; mientras que en el campo target solo aparecen los valores de la columna clase. Es por ello por lo que en la variable X se almacenan los datos y no hay ninguna otra referencia al campo de la clase. De esta manera eliminamos esa columna. Finalmente, se transforman los datos a una estructura *numpy*, esto también sería necesario para los datos cargados desde pandas. La razón es que los algoritmos en Scikit-learn necesitan como entrada datos con estructura *numpy*.

Transformar datos numéricos

Revisando otra vez los datos de la Tabla 1, notarás que los valores para la longitud del tallo son mayores que los del ancho del pétalo. Como todas las mediciones se hicieron en centímetros, este parece un detalle menor. Sin embargo, ¿te imaginas que la longitud del tallo fuera medida en metros? ¿Ahora que valor sería menor? Para la primera flor, el valor de la longitud del tallo en metros es 0.051, un valor mucho menor que el ancho del pétalo en centímetros.

Iris	sepallength	sepalwidth	petallength	petalwidth	class
1	5.1	3.5	1.4	0.2	Iris-setosa
2	4.9	3.0	1.4	0.2	Iris-setosa
3	7.0	3.2	4.7	1.4	Iris-versicolor
4	6.4	3.2	4.5	1.5	Iris-versicolor
5	6.3	3.3	6.0	2.5	Iris-virginica
6	5.8	2.7	5.1	1.9	Iris-virginica

Tabla 1. Fragmento de la base de datos Iris del Repositorio UCI (Tanveer et al., 2019).

Esta diferencia en las magnitudes puede dañar considerablemente el resultado final del análisis de los datos y, lo que es peor, de las decisiones que se tomen a partir del mismo. En ocasiones, no tenemos opción de conocer todos los detalles de cada variable de los datos, por lo que es común transformar los datos numéricos de manera que todos estén en la misma escala, o sigan la misma distribución, o siguiendo algún otro criterio parecido.

Hay tres tipos principales de preprocesamiento para datos numéricos:

- **Escalado (Alasadi *et al.*, 2017)**

Antes de agrupar los datos de las flores Iris, tu jefe solicita información acerca de cómo están distribuidas las especies. Recuerda que no agruparás teniendo en cuenta la especie, pero esa puede ser una consulta válida. Para hacerlo, podemos implementar el siguiente código usando *Scikit-learn* y *matplotlib*.

```
from sklearn import cluster, datasets, metric
import numpy as np
import matplotlib.pyplot as plt

iris = datasets.load_iris()
X = np.array(iris.data)

# Aquí necesitamos las especies
# para ver cómo de distribuyen
y = np.array(iris.target)

# Ploteamos las últimas dos columnas
plt.scatter(X[:, 2], X[:, 3], c=y)
plt.xlabel('Petal length')
plt.ylabel('Petal width')
plt.show()
```

Resultado

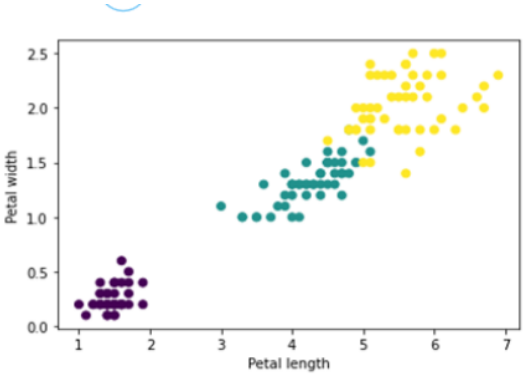

Figura 2. Imagen generada a partir de código en Google Colab.

Se muestra el resultado de cómo se distribuyen las especies, teniendo en cuenta solo las dos últimas columnas o variables (*petallength* y *petalwidth*).

Como lo pudiste apreciar en la imagen, aparecen como puntos en el plano cartesiano los valores de las dos variables seleccionadas. El color de los puntos indica las especies correspondientes. Como se puede notar, hay dos especies parecidas y otra muy diferente, de acuerdo con esas dos variables. También podemos concluir que la especie de color amarillo es

más diversa que las otras y donde las flores tienen tallos más grandes. ¿Qué otras conclusiones puedes sacar de la imagen?

Nota que la variable *petallength* tiene valores que van desde 1 hasta 7, mientras que en *petalwidth* van de 0 a 2.5. Aunque ambas mediciones son en centímetros, esa diferencia en magnitud puede limitar el desempeño de los algoritmos de agrupamiento. ¿Qué podrías hacer? Efectivamente transformar los datos para que estén en la misma escala, por ejemplo, en el intervalo [0, 1]. Pero ¿cómo es esto posible?

Los métodos de escalado transforman los valores de las variables para que se encuentren entre un valor mínimo y máximo determinado, que a menudo es entre cero y uno. Esto permite que el valor absoluto máximo de cada variable se escale al tamaño de la unidad, y el valor mínimo se escale a cero, cuando se habla del intervalo [0, 1]. Uno de los beneficios de esta transformación es que si una variable tiene valores muy aislados (*outliers* en inglés), después de la transformación se espera que esos valores afecten menos a los resultados.

El código para escalar los datos usando *Scikit-learn* es el siguiente:

```
from sklearn import datasets
from sklearn.preprocessing import MinMaxScaler
import numpy as np
import matplotlib.pyplot as plt

iris = datasets.load_iris()
X = np.array(iris.data)
y = np.array(iris.target)

# Ajustamos el método de escalado a los datos para
# transformarlos almacenarlos en otra variable
X2 = MinMaxScaler(feature_range=(10, 20)).fit_transform(X)

# Ploteamos los datos originales y transformados
fig, (ax0, ax1) = plt.subplots(ncols=2, figsize=(10,4))

ax0.scatter(X[:, 2], X[:, 3], c=y)
ax0.set_xlabel('Petal length')
ax0.set_ylabel('Petal width')

ax1.scatter(X2[:, 2], X2[:, 3], c=y)
```

En relación al código, presta especial atención al paquete donde se encuentran los transformadores, *sklearn.preprocessing*. De ahí,

importamos el objeto *MinMaxScaler*. Para transformar los datos, *MinMaxScaler* debe ajustarse primero a los datos que va a transformar; dicho coloquialmente, determinar qué valores de máximo y mínimo van a terminar quedando como cero y uno, además otros detalles. A continuación, puedes transformar los datos. Ambas acciones se pueden ejecutar al mismo tiempo a través de la función *fit_transform()*.

¿Qué pasa con los datos después de ser escalados? Mira la siguiente imagen:

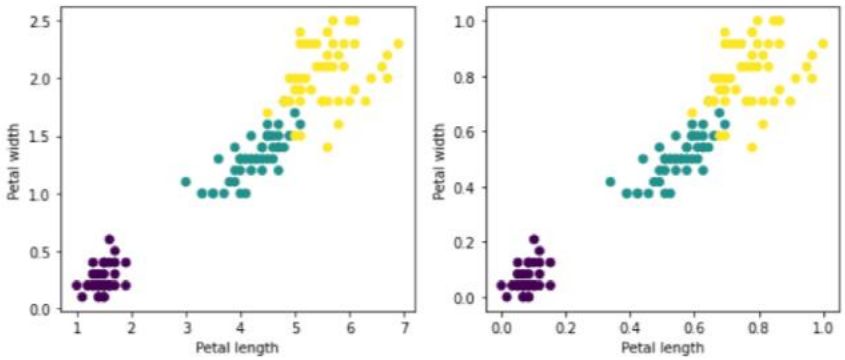

Figura 3. Imagen generada en Google Colab.

¿Notas alguna diferencia entre una imagen y otra? Los colores están iguales, así que no se dañó la distribución de las especies. ¿Y la magnitud de los valores? Resulta que ahora los valores transformados todos están en el intervalo [0, 1] sin que eso cambie las relaciones que ya existían en los datos originales. De esta manera hemos resuelto el problema de las magnitudes.

El objeto *MinMaxScaler* tiene parámetros interesantes, pero quizás el más importantes es *feature_range*, que es una tupla de dos valores (*min, max*). Si quieres cambiar el intervalo a, por ejemplo, de 10 a 20, debes escribir lo siguiente:

```
X2 = MinMaxScaler(feature_range=(10, 20)).fit_transform(X)
```

Prueba por ti mismo otros valores para estos parámetros y analiza los efectos que provocan en los datos.

- **Estandarización (Gal *et al.*, 2019)**

Asumamos ahora que eres el científico de datos de un laboratorio químico especializado que, entre otras responsabilidades, se encarga de analizar químicamente una gran variedad de productos. A petición de una empresa comercializadora de vinos, se requiere analizar químicamente distintas marcas y presentaciones de vinos para determinar sus orígenes y calidad.

Algunas de las variables recolectadas de los vinos son el porciento de alcohol, la cantidad de ácido málico, la intensidad del color o la concentración del aminoácido prolina, entre otros.

Alcohol	Malic acid	Ash	Magnesium	Phenols	Flavanoids	Color	Proline
14.23	1.71	2.43	127	2.8	3.06	5.64	1065
13.2	1.78	2.14	100	2.65	2.76	4.38	1050
12.37	0.94	1.36	88	1.98	0.57	1.95	520
12.33	1.1	2.28	101	2.05	1.09	3.27	680
12.86	1.35	2.32	122	1.51	1.25	4.1	630
12.88	2.99	2.4	104	1.3	1.22	5.4	530

Esta base de datos también aparece en Scikit-learn. Fíjate en la gran diferencia que hay entre los valores de las variables. Para cargar esta base de datos, solo debes escribir:

```
wine = datasets.load_wine()
```

No solo hay diferencias en las magnitudes de las variables, también son muy diferentes sus distribuciones. Sin entrar en muchos detalles acerca de las funciones de distribución, se puede notar que los valores de las variables se distribuyen de manera diferente entre sus histogramas.

Para la mayoría de los algoritmos de agrupamiento, el concepto de centroide es muy importante, ya que forma parte del funcionamiento de varios de ellos. ¿Recuerdas que el centroide es el representante de un

grupo? Este representante casi siempre se calcula como el promedio de los elementos del grupo. Por lo que los promedios de cada variable son también valores significativos.

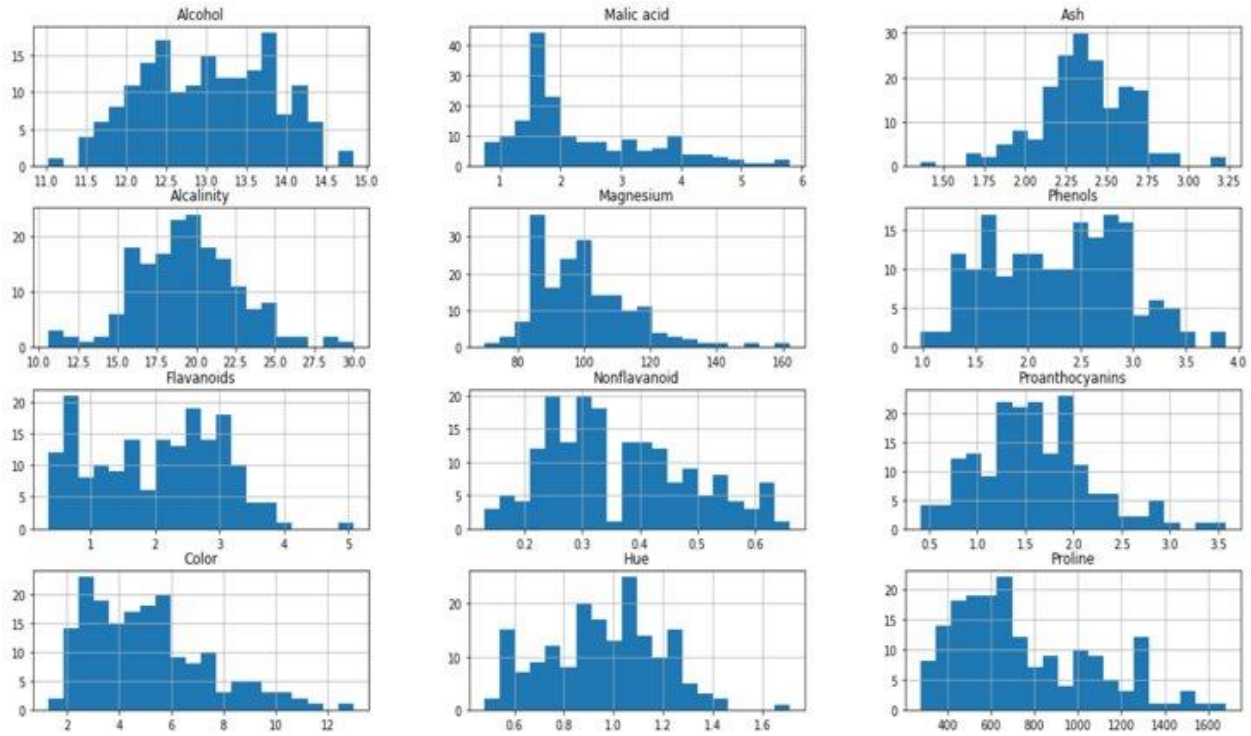

Figura 5. Histogramas de Wine. Imagen generada en Google Colab

Objetivos de los marcos de trabajo

El objetivo principal de los marcos de trabajo es ofrecer una **funcionalidad definida y auto contenida**, se constituyen usando **patrones de diseño**. Su característica principal es su alta **cohesión** y **bajo acoplamiento**.

Conoce los motivos por los que debería utilizarse un framework:

- Acelerar el proceso de desarrollo del software.
- Evitar errores de codificación.
- Reutilizar código mediante su modularización.
- Utilizar buenas prácticas de programación.

- Eliminar vulnerabilidades en las aplicaciones.

Ejemplo de uso de un *framework*
Para comprender el uso de un framework en una situación real, te damos este ejemplo de su utilización en la pandemia por COVID-19.

La crisis por coronavirus resulta un elemento de análisis en donde el sistema de salud y la economía se vieron afectadas. Sin embargo, dicha enfermedad ha contribuido a la generación y manipulación de grandes **volúmenes de información**, por ejemplo:

- Los resultados de pruebas diagnósticas.
- El número de contagios por periodo de tiempo y región.
- La investigación con relación a la sintomatología.
- El comportamiento del virus.
- El número de personas vacunadas.
- Los comentarios en las plataformas digitales para informar de la pandemia.

La creación de aplicaciones con disponibilidad inmediata que hagan uso de estos datos sería muy complicada si la implementación parte de cero. Aquí es donde se hace evidente la importancia de contar con **fragmentos de código preconstruidos** y confiables que puedan ser utilizados para agilizar nuestros desarrollos.
Reflexiona las siguientes preguntas:

- ¿A qué nos referimos con fragmentos de código preconstruidos?
- ¿Cómo podrías utilizarlos en tu día a día?

Para finalizar, si quieres manejar datos, como en este ejemplo de la pandemia, **podrías utilizar un *framework* como Matplotlib** para mostrar los resultados de manera gráfica y hacer que el desarrollo sea agradable de utilizar y fácil de entender para sus usuarios
finales.

Cuando las variables tienen diferentes distribuciones y, sobre todo, diferentes magnitudes, es común que los promedios sean diferentes. Esto afecta a los algoritmos de agrupamiento, ya que puede favorecer unas variables sobre otras. Es aquí donde la estandarización juega un papel importante, ya que permite cambiar el valor de los promedios para que sea el mismo en todas las variables.

Para ello, solo es necesario importar el método, ajustarlo y transformar los datos:

```
from sklearn.preprocessing import StandardScaler

wine = datasets.load_wine()
X = np.array(wine.data)

X2 = StandardScaler().fit_transform(X)
```

Podemos ver que ahora todas las variables tienen su promedio en el valor cero y, además, ¡todas están escaladas! Es por eso por lo que este método es posiblemente el más usado para transformar los datos numéricos antes de agrupar.

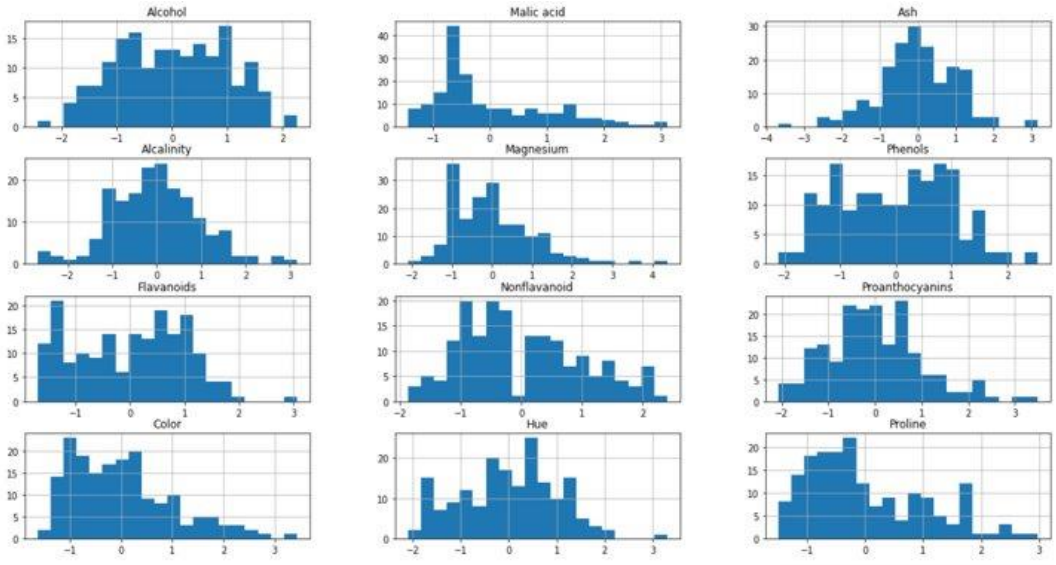

Figura 6. Variables de *Wine* después de estandarizar. Imagen generada en Google Colab

- **Normalización (Singh *et al.*, 2020)**

Ya puedes transformar los datos para que todos los valores estén en el mismo intervalo o tengan el mismo promedio. Eso sin cambiar las distribuciones aún. Seguramente te preguntarás, ¿en qué afectan que tengan diferentes distribuciones? Por ejemplo, la variable *Malic acid* tiene casi todos sus valores alrededor del valor 1.8, lo que significa que casi todos los vinos tienen ese valor en esa variable. Esto no es bueno, ya que, así como está, esa variable no consigue distinguir los vinos que son diferentes.

Este problema es un poco más complejo de resolver y tiene que ver con cuáles comparaciones de datos se ven favorecidas cuando agrupamos con un algoritmo que usa funciones de similaridad, como lo hacen la mayoría. Por lo tanto, el objetivo de normalizar es que, si usamos una función de similaridad tradicional, los resultados de comparar datos con esa función estén en la misma escala de valores.

Cuando normalizas los datos numéricos, estás consiguiendo que los valores de las variables estén escalados de manera que tengan una **norma unitaria**. La forma de hacerlo en Python es similar a las anteriores, solo que este método no necesita ajustarse, sino que normaliza de una manera más directa.

```
from sklearn.preprocessing import normalize

X2 = normalize(np.array(datasets.load_wine().data))
```

El resultado de normalizar Wine es el siguiente:

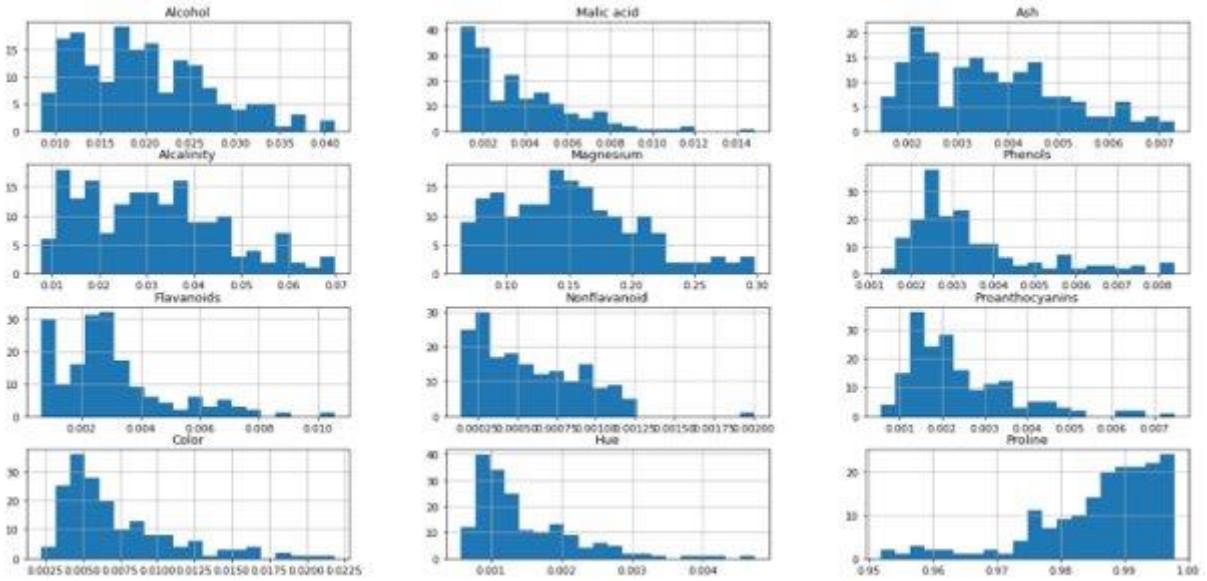

Figura 7. Normalización de *Wine*. Imagen generada en Google Colab.

Analizando estos histogramas podemos ver que ahora sí han cambiado las distribuciones. Revisa, por ejemplo, la variable *Malid acid*, ¿qué cambió? Observa que también otras variables, después de normalizar, están inclinadas hacia la izquierda, mientras que *Proline* está totalmente inclinada hacia la derecha.

Lo esperado después de normalizar es que esencialmente estos cambios en la distribución favorezcan los resultados de las funciones de distancia. Por lo tanto, ya estás listo para enfrentar distintas situaciones con las que debes lidiar cuando los datos numéricos vienen con magnitudes distintas, diferentes promedios o distintas distribuciones. De manera simple puedes usar los métodos que ya están implementados en *Scikit-learn* para ese propósito. El siguiente reto es que sigas el mismo procedimiento para analizar una base de información, pero con datos categóricos, que lo realizarás en el siguiente subtema.

Estos métodos benefician a los algoritmos de aprendizaje en general, pero particularmente a los algoritmos de agrupamiento de datos, debido a que estos últimos usan funciones de distancia para comparar datos similares y estas funciones son sensibles a las magnitudes de los datos.

Transformación de datos categóricos

Ahora estas trabajando para una empresa global de comunicaciones y el administrador de la red requiere monitorear cierta información sobre los usuarios. Conozcamos como podemos ordenar los datos y utilizarlos para calcular lo solicitado.

¡Comencemos!

Variables
Si imaginamos las diferentes conexiones inalámbricas de los usuarios, se verían así:

Figura 8. Concepto de comunicación inalámbrica.

Ahora que tenemos una idea de las diferentes conexiones, conozcamos la información que nos proporciona la empresa.

- **Datos**

Se seleccionaron tres variables: sexo, región y navegador utilizado; de manera que un resumen de los datos puede ser como se muestra a continuación.

ID	Sexo	Región	Navegador
1	masculino	EE. UU.	Internet Explorer
2	femenino	Asia	Chrome
3	masculino	EE. UU.	Safari
4	femenino	Europa	Firefox
5	femenino	EE. UU.	Safari

Como puedes notar, no tenemos datos numéricos en este conjunto de datos.

¡Espera! ¿el ID no es un dato numérico? En este caso sí, y en el ejemplo de las flores Iris también, pero por lo general estos datos que representan IDs, números de nóminas u otros identificadores no son tomados en cuenta.

Entonces, ¿qué puedes hacer con estos datos? ¿Puedes calcular el promedio o analizar cuál es el valor mínimo o máximo? Obviamente no. A este tipo de datos se les conoce como datos categóricos y las operaciones matemáticas sobre ellos son un tanto limitadas.

Datos categóricos

Hay algoritmos que se especializan en este tipo de datos y no necesitan ninguna transformación previa, pero los algoritmos tradicionales de agrupamiento fueron definidos para datos numéricos, por lo que una buena forma de lidiar con datos categóricos es transformarlos a datos numéricos.

Detente a pensar, ¿cómo se pueden transformar esas palabras o textos a números? Aunque hay más variantes, lo tradicional es usar uno de los dos codificadores siguientes:

- **Codificador ordinal (Potdar *et al.*, 2017)**

Este tipo de transformador de dato categórico a numérico es realmente simple y pudiera parecer suficiente. Consiste en expresar con valores numéricos enteros, por lo general consecutivos, cada categoría dentro de cada variable. Como el conjunto de los números enteros es un conjunto ordenado, por eso se le conoce como *ordinal encoder* en inglés.

Realizando esa transformación, los datos podrían quedar:

ID	Sexo	Región	Navegador
1	1	1	2
2	0	0	0

3	1	1	3
4	0	2	1
5	0	1	3

¿Qué puedes comentar acerca de los cambios que sufrieron los datos?

Es posible que pienses que esta transformación es suficiente pues ahora si puedes revisar los intervalos de valores o calcular promedio. Sin embargo, analiza con detenimiento la variable Navegador.

El orden de los números está puesto arbitrariamente, siguiendo el orden del alfabeto, pero esta no es ninguna propiedad matemática válida. Visto así, ¿eso quiere decir que Internet Explorer es mayor que Chrome?, por supuesto que un navegador no es 'mayor' a otro, pero esta codificación así lo interpreta, lo cual no es correcto.

Otra pregunta que seguro se te ocurre es: **¿puedes decir que el promedio entre Firefox y Safari es igual a Internet Explorer ((1 + 3) / 2 = 2)?**... pues tampoco. Por lo que este tipo de transformación puede no ser útil con varios algoritmos que asumen que los datos siguen un orden, como el conjunto de los números reales.

- Transformación

Aun así, esta transformación puede ser utilizada, solo que con precaución. Para realizarla en *Scikit-learn* podemos escribir:

```
from sklearn.preprocessing import OrdinalEncoder

X = [['masculino',      'EE. UU.',      'Internet Explorer'],
     ['femenino',       'Asia', 'Chrome'],
     ['masculino',      'EE. UU.',      'Safari'],
     ['femenino',       'Europa',       'Firefox'],
     ['femenino',       'EE. UU.',      'Safari']]

X2 = OrdinalEncoder().fit_transform(X)

X2
```

```
array([[1., 1., 2.],
       [0., 0., 0.],
       [1., 1., 3.],
       [0., 2., 1.],
       [0., 1., 3.]])
```

¿Cuál sería entonces una transformación válida? Una muy utilizada es la codificación *one-hot*.

- **Codificador *one-hot* (Rodríguez *et al.*, 2018)**

Otra posibilidad para convertir variables categóricas a numéricas es usar codificación one-hot o codificación 'falsa' como también se le conoce. Este tipo de codificación transforma cada variable categórica con *n* categorías o valores, en n nuevas variables binarias, con una de ellas con valor 1 y todas las demás con valor 0.

¿Cómo quedarían los datos del ejemplo después de esta transformación? Observa la siguiente tabla con los resultados.

femenino	Masculino	Asia	EE. UU.	Europa	Chrome	Firefox	Internet Explorer	Safari
0	1	0	1	0	0	0	1	0
1	0	1	0	0	1	0	0	0
0	1	0	1	0	0	0	0	1
1	0	0	0	1	0	1	0	0
1	0	0	1	0	0	0	0	1

Esta transformación modifica considerablemente la tabla inicial de los datos, pero es más adecuada en la mayoría de los casos. Nota que ahora cada variable era antes un valor. Además, después de la transformación, aparece el número uno en las celdas donde los usuarios tenían ese valor. Por ejemplo, si enumeramos las celdas comenzando en cero, la celda (1,2) tiene valor 1, lo que significa que ese usuario es de la Región Asia.

Aunque ahora puedes notar que otra vez hablar de mínimos y máximos, o promedios, tiene poco sentido, lo cierto es que estas operaciones ahora tienen menos influencia en los resultados. Además, nota un detalle curioso, ¡ya los datos están escalados en el intervalo [0,1]!

¿Cómo lo hacemos en *Scikit-learn*?

Détente un momento a analizar este código, hay algunos detalles diferentes

```
from sklearn.preprocessing import OneHotEncoder

enc = OneHotEncoder()
enc.fit(X)
X2 = enc.transform(X).toarray()
```

- Algoritmo

El algoritmo que seguimos para realizar esta codificación fue el siguiente:

Puedes pensar que estos pasos son los mismos para todas las transformaciones, y de hecho lo son, solo que el objeto *OneHotEncoder*, luego de transformar los datos no devuelve una matriz como los demás y por eso necesitamos el paso 5. Para poder hacer este paso necesitamos que el resto de las operaciones se realicen por separado, mientras que para otras transformaciones podíamos unir varios de esos pasos en uno solo.

Información.
Ya conoces cómo transformar datos tanto numéricos como categóricos. Si tienes una tabla de datos donde aparezcan ambos tipos de datos, lo más importante es siempre asegurarte que los datos categóricos fueron convertidos a numéricos. Para ello, recomendamos *OneHotEncoder* casi en

todas las situaciones. Este tipo de codificación genera lo que se conoce en inglés como *sparse matrix*, ya que como puedes ver las filas tienen muchos valores iguales a cero. El estudio de estas matrices no lo veremos en este curso, solo que repetimos que aún así esta es una mejor codificación que *OrdinalEncoder*.

Si te preguntas ¿cómo transformar una tabla con ambos tipos de datos en *Scikit-learn*? Hay varias maneras de hacerlo, puedes usar tus habilidades en pandas para particionar la tabla, seleccionar columnas, concatenarlas o crear tablas nuevas.

Finalmente, analicemos otra transformación muy útil.

- **Llenar datos faltantes**

En ocasiones no es posible tener todos los datos de la tabla debido a múltiples razones. Imagina que trabajas para la empresa Arca Continental, que distribuye Coca Cola y otros productos a las tiendas de conveniencia o recauderías. Muchas de estas tiendas reportan sus ventas, pedidos, ingresos y otros datos escribiéndolos a mano, en papel. Esto puede provocar pérdida de información si se quieren analizar esos datos.

También puedes imaginar que en el laboratorio químico donde se analizan los vinos los técnicos pueden no realizar algunas pruebas, o simplemente escribir mediciones erróneas que luego prefieren dejar en blanco. En pandas esos valores se conocen como *NaN*, que en inglés son las siglas de *not* a *number*. Los algoritmos de *Scikit-learn* no pueden procesar esos valores, por lo que es necesario transformarlos.

Nuestra recomendación es que se realice desde pandas. Veamos un ejemplo. Imagina que para el primer vino de los datos no fue posible medir su porciento de alcohol, por lo que ese dato aparece como faltante.

	index	Alcohol	Malic acid	Ash
0	1	NaN	1.71	2.43
1	1	13.20	1.78	2.14
2	1	13.16	2.36	2.67
3	1	14.37	1.95	2.50
4	1	13.24	2.59	2.87
...	...	...	...	...

Figura 9. Valor *NaN* en *pandas*. Imagen generada en *Google Colab*.

Desde pandas podemos hacer varias cosas:

- Eliminar las filas con NaN => df.dropna()
- Poner el valor 0 en lugar de los NaN => df.fillna(0)
- Sustituir NaN por el promedio de la columna. ¡Esta es la opción que recomendamos!

```
df['Alcohol'] = df['Alcohol'].fillna(df['Alcohol'].mean())
df
```

Desde *Scikit-learn* puedes también reemplazar los datos faltantes. Te invitamos a que revises la documentación y estudies esos métodos. También puedes estudiar otros métodos de transformación de datos que ahí aparecen, como los métodos de Discretización, que son realmente interesantes. No los estudiamos aquí porque consideramos que estas herramientas que hemos analizado son suficientes para que puedas agrupar datos de tu organización realizando las transformaciones previas que se requieran y garanticen excelentes resultados.

Ideas para llevar

Los métodos de preprocesamiento de datos son muy útiles y han sido muy estudiados dentro del aprendizaje automatizado. Además de los métodos estudiados en este tema, hay muchos otros que pueden ayudarte a mejorar los resultados de otros tipos de algoritmos.

En particular, para los algoritmos de agrupamiento, lo importante de preprocesar los datos es que estos algoritmos puedan funcionar, ya que la mayoría están definidos para datos numéricos y asumen ciertas propiedades de esos datos.

Por lo tanto, es importante que, como científico de datos, antes de ir a agrupar datos para resolver determinado problema, sigas de alguna manera los pasos o consejos que se enuncian a continuación:

- Transformar datos antes de agrupar garantiza buenos resultados de agrupamiento.
- Asegurarse que los datos tienen magnitudes similares, o transformarlos cuando es necesario, es fundamental para que los algoritmos no favorezcan unas variables sobre otras.
- Cuando los datos son categóricos muchos algoritmos de agrupamiento no pueden agruparlos, por lo que debes transformarlos. Recomendamos para ello *OneHotEncoder* en la mayoría de las situaciones.
- Los datos faltantes es una situación a la que seguramente tendrás que enfrentarte con frecuencia. Revisa siempre la causa que provoca esa ausencia de información y transforma esos valores para que puedan ser usados por los algoritmos de agrupamiento.

Material de apoyo

Bibliografía

Los contenidos de esta lección están basados en la siguiente bibliografía:

- Alasadi, S. A., & Bhaya, W. S. (2017). Review of data preprocessing techniques in data mining. *Journal of Engineering and Applied Sciences*, *12*(16), 4102-4107.
- Gal, M. S., & Rubinfeld, D. L. (2019). Data standardization. *NYUL Rev.*, *94*, 737.
- Potdar, K., Pardawala, T. S., & Pai, C. D. (2017). A comparative study of categorical variable encoding techniques for neural network classifiers. *International journal of computer applications*, *175*(4), 7-9.
- Rodríguez, P., Bautista, M. A., Gonzalez, J., & Escalera, S. (2018). Beyond one-hot encoding: Lower dimensional target embedding. *Image and Vision Computing*, *75*, 21-31.
- Singh, D., & Singh, B. (2020). Investigating the impact of data normalization on classification performance. *Applied Soft Computing*, *97*, 105524.
- Tanveer, M., Gautam, C., & Suganthan, P. N. (2019). Comprehensive evaluation of twin SVM based classifiers on UCI datasets. *Applied Soft Computing*, *83*, 105617.