Preprocesamiento de Datos

¿Qué aprenderemos hoy?

- Tratar los valores perdidos en un conjunto de datos
- Tratar con diferentes tipos de variables (continuas y discretas)
- Estudiar subgrupos de variables incluidas en conjuntos de datos
- Librería Pandas
- Documentación DataFrame

Tratar con Valores Perdidos (datos)

- Espacio en blanco o información inconsistente
- NaN (no hay número)
- Null (valores desconocidos)

Datos Continuos

- Lectura de datos en formato csv

```
In [ ]:
import pandas as pd

df = pd.read_csv("Imagenes_Clase_07/Clase7_CSV.csv", header=0)
df
```

- documentación

```
In [ ]:
df.head(2)
```

```
In [ ]:
df.dtypes
```

```
In [ ]:
df.isnull().sum()
```

```
In [ ]:
df.dropna(axis=0)
```

```
In [ ]:
df
```

```
In [ ]:
df.dropna(axis=1)
```

```
In [ ]:
df.dropna(thresh=3)
```

```
In [ ]:
df.dropna(subset=["D"])
```

```
In [ ]:
df.values
```

¿Qué hacer con los valores perdido?

¿Simplemente los eliminamos?

```
In [ ]:
import numpy as np
from sklearn.impute import SimpleImputer

imr = SimpleImputer(missing_values=np.nan, strategy="mean")
imr = imr.fit(df.values)
imputed_data = imr.transform(df.values)
imputed_data
```

```
In [ ]:
df[['A','B','C','D']] = imputed_data
df
```

¿Cómo obtenemos la media de cada columna?

```
In [ ]:
df.describe()
```

Procesando Datos Categóricos

- **Ordinales:** Valores categóricos que pueden ordenarse.
- **Nominales:** Valores categóricos que no pueden ordenarse.

```
In [ ]:
df = pd.DataFrame([['green', 'M', 10.1, 'class2',0.0],
                   ['red', 'L', 13.5, 'class1',0.0],
                   ['blue', 'XL', 15.3, 'class2',0.0],
                   ['blue', 'XL', 9.0, 'class2',1.0]])

df.columns = ['color', 'size', 'price', 'class_label', 'metric']
df
```

Los algoritmos que conocemos ¿podríamos ajustarlos directamente a partir del Dataframe anterior?

```
In [ ]:
df.isin([0.0])
```

```
In [ ]:
df.isin([0.0]).sum()
```

```
In [ ]:
df.drop(columns=['metric'])
```

Los algoritmos que conocemos ¿podríamos ajustarlos directamente a partir del Dataframe anterior?

```
In [ ]:
size_mapping = {'XL': 3,'L': 2,'M': 1}

df['size'] = df['size'].map(size_mapping)
df
```

```
In [ ]:
class_mapping = {label: idx for idx, label in enumerate(np.unique(df['class_label']))}
class_mapping
```

```
In [ ]:
df['class_label'] = df['class_label'].map(class_mapping)
df
```

```
In [ ]:
y = df.iloc[:,3].values
y
```

```
In [ ]:
X = df[['color', 'size', 'price']].values
X
```

Los algoritmos que conocemos ¿podríamos ajustarlos directamente a partir de X e y?

```
In [ ]:
X = pd.get_dummies(df[['price', 'size', 'color']]).values
X
```

```
In [ ]:
print('Conjunto de Variables o Características:\n',X)
print('Conjunto de Etiquetas de Clase:\n',y)
```

Trabajando con una Base de Datos real

```
In [ ]:
df_wine = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/wine/wine.data', header=None)

df_wine.columns = ['Class label', 'Alcohol', 'Malic acid', 'Ash',
                   'Alcalinity of ash', 'Magnesium', 'Total phenols',
                   'Flavanoids', 'Nonflavanoid phenols', 'Proanthocyanins',
                   'Color intensity', 'Hue', 'OD280/OD315 of diluted wines',
                   'Proline']

df_wine
```

```
In [ ]:
df_wine.describe()
```

```
In [ ]:
print('Etiquetas de clase', np.unique(df_wine['Class label']))
```

```
In [ ]:
df_wine.isnull().sum()
```

```
In [ ]:
df_wine.describe(include="all")
```

```
In [30]:
df_wine.iloc[:,0] = df_wine.iloc[:,0].astype("category")
```

```
In [ ]:
df_wine.describe(include="all")
```

```
In [33]:
from sklearn.model_selection import train_test_split

X, y = df_wine.iloc[:, 1:].values, df_wine.iloc[:, 0].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0, stratify=y)
```

Reescalamiento de las variables

- **Normalización:**

$$x_{norm}^{(i)} = \frac{x^{(i)} - x_{min}}{x_{max} - x_{min}}$$

```
In [ ]:
from sklearn.preprocessing import MinMaxScaler

mms = MinMaxScaler()

X_train_norm = mms.fit_transform(X_train)
X_test_norm = mms.transform(X_test)

print(X_train_norm.min(axis=0))
print(X_test_norm.min(axis=0))

print(X_train_norm.max(axis=0))
print(X_test_norm.max(axis=0))
```

- **Estandarización:**

$$x_{std}^{(i)} = \frac{x^{(i)} - \mu_x}{\sigma_x}$$

```
In [ ]:
from sklearn.preprocessing import StandardScaler

stdsc = StandardScaler()

X_train_std = stdsc.fit_transform(X_train)
X_test_std = stdsc.transform(X_test)

print(X_train_std.mean(axis=0))
print(X_test_std.mean(axis=0))

print(X_train_std.std(axis=0))
print(X_test_std.std(axis=0))
```

Estudiando las variables para reducir la complejidad del Modelo

- **Regularización L2:** $\|\mathbf{w}\|_2^2 = \sum_{j=1}^{m} w_j^2$

```
In [ ]:
from IPython.display import Image

Image(filename='Imagenes_Clase_07/7_1.jpg', width=540)
```

```
In [ ]:
from sklearn.linear_model import LogisticRegression
import matplotlib.pyplot as plt

fig = plt.figure()
ax = plt.subplot(111)

colors = ['blue', 'green', 'red', 'cyan', 'magenta', 'yellow', 'black',
          'pink', 'lightgreen', 'lightblue', 'gray', 'indigo', 'orange']

weights, params = [], []
for c in np.arange(-4., 6.):
    lr = LogisticRegression(penalty='l2', C=10.**c, random_state=0)
    lr.fit(X_train_std, y_train)
    weights.append(lr.coef_[1])
    params.append(10**c)

weights = np.array(weights)

for column, color in zip(range(weights.shape[1]), colors):
    plt.plot(params,weights[:, column],label=df_wine.columns[column + 1],color=color)
plt.axhline(0, color='black', linestyle='--', linewidth=3)
plt.xlim([10**(-5), 10**5])
plt.ylabel('weight coefficient')
plt.xlabel('C')
plt.xscale('log')
plt.legend(loc='upper left')
ax.legend(loc='upper center',bbox_to_anchor=(1.38,1.03),ncol=1,fancybox=True)
#plt.savefig('images/04_07.png', dpi=300,
#            bbox_inches='tight', pad_inches=0.2)
plt.show()
```

- **Regularización L1:** $\|\mathbf{w}\|_1 = \sum_{j=1}^{m} |w_j|$

```
In [ ]:
Image(filename='Imagenes_Clase_07/7_2.jpg', width=540)
```

```
In [ ]:
fig = plt.figure()
ax = plt.subplot(111)

colors = ['blue', 'green', 'red', 'cyan', 'magenta', 'yellow', 'black',
          'pink', 'lightgreen', 'lightblue', 'gray', 'indigo', 'orange']

weights, params = [], []
for c in np.arange(-4., 6.):
    lr = LogisticRegression(penalty='l1', C=10.**c, random_state=0, solver='saga')
    lr.fit(X_train_std, y_train)
    weights.append(lr.coef_[1])
    params.append(10**c)

weights = np.array(weights)

for column, color in zip(range(weights.shape[1]), colors):
    plt.plot(params,weights[:, column],label=df_wine.columns[column + 1],color=color)
plt.axhline(0, color='black', linestyle='--', linewidth=3)
plt.xlim([10**(-5), 10**5])
plt.ylabel('weight coefficient')
plt.xlabel('C')
plt.xscale('log')
plt.legend(loc='upper left')
ax.legend(loc='upper center',bbox_to_anchor=(1.38,1.03),ncol=1,fancybox=True)
#plt.savefig('images/04_07.png', dpi=300,
#            bbox_inches='tight', pad_inches=0.2)
plt.show()
```

- **Selección de características de manera aleatoria**

```
In [ ]:
df_wine.iloc[:, 1:].sample(n=3, random_state=100, axis=1)
```

Actividad Final

Desde el siguiente enlace descargar la BBDD "Asignaciones y Créditos 2023" para realizar las siguientes actividades:

Enlace: https://datosabiertos.mineduc.cl/asignaciones-de-becas-y-creditos-en-educacion-superior/

Utilice también la base disponible para 2022.

Cargar la BBDD ('Asignacion_2022_WEB.csv')

```
In [ ]:
```

Pregunta 1 ¿Cuántas columnas y registros tiene la BBDD?

```
In [ ]:
```

Pregunta 2 ¿Cuántos tipos de Beneficios diferentes existen? ¿Cuántos beneficios fueron otorgados en cada tipo?

```
In [ ]:
```

Pregunta 3 ¿Cuántos tipos de alumnos existen en la BBDD? ¿Cuántos estudiantes aparecen registrados en cada tipo?

```
In [ ]:
```

Pregunta 4 ¿Cuántos estudiantes tuvieron beneficios el año 2023?

```
In [ ]:
```

Pregunta 5 Realizar un mapeo de lo tipos de beneficios y crear una nueva columna categórica con valores discretos.

```
In [ ]:
```

Repetir preguntas anteriores con BD 2023

In [ ]: