

Machine Learning en la mejora de la precisión de CRISPR-Cas9

```
In [1]: import pandas as pd
```

Cargamos la base de datos y visualizamos las 5 primeras filas.

```
In [2]: df = pd.read_csv("Datos.csv")
df.head()
```

```
Out[2]:
```

	23-nt sequence	30-nt sequence	modFreq
0	GGCTGCTTTACCCGCTGTGGGGG	TGGAGGCTGCTTTACCCGCTGTGGGGGCGC	0.874864
1	TCCGGGTTGGCCTTCCACTGGGG	CGTCTCCGGGTTGGCCTTCCACTGGGGCAG	0.871393
2	CAGCATCCTTCGGAAAGCTCTGG	CCCTCAGCATCCTTCGGAAAGCTCTGGACA	0.860592
3	CGGTAGAAGCAGGTAGTCTGGGG	GGCGCGGTAGAAGCAGGTAGTCTGGGGATA	0.802276
4	CATCCCGCTGCCCCAGTGAAGG	GTGTCATCCCGCTGCCCCAGTGAAGGCCA	0.796638

Eliminamos la columna "30-nt sequence", ya que no la vamos a utilizar. Almacenamos en la misma variable. Visualizamos las 5 primeras filas.

```
In [3]: df = df.drop("30-nt sequence", axis = 1)
df.head()
```

```
Out[3]:
```

	23-nt sequence	modFreq
0	GGCTGCTTTACCCGCTGTGGGGG	0.874864
1	TCCGGGTTGGCCTTCCACTGGGG	0.871393
2	CAGCATCCTTCGGAAAGCTCTGG	0.860592
3	CGGTAGAAGCAGGTAGTCTGGGG	0.802276
4	CATCCCGCTGCCCCAGTGAAGG	0.796638

Convertimos la columna "23-nt sequence" en una lista.

```
In [4]: lista = list(df["23-nt sequence"])
lista[0:5]
```

```
Out[4]: ['GGCTGCTTTACCCGCTGTGGGGG',
'TCCGGGTTGGCCTTCCACTGGGG',
'CAGCATCCTTCGGAAAGCTCTGG',
'CGGTAGAAGCAGGTAGTCTGGGG',
'CATCCCGCTGCCCCAGTGAAGG']
```

Empezamos a construir listas con las features que afectan a la eficiencia del gRNA. En este ejemplo, se van a utilizar las features de secuencia: "G en la posición 20", "C en la posición 21", "número total de A en un ADN diana", "C en la posición 18" y "C en la posición 16"; y la feature estructural "Contenido de GC". Empezamos con la feature "G en la posición 20". Construimos una lista con el nucleótido de la posición 20 de cada ADN diana.

```
In [5]: num_20 = []
        for i in range(len(lista)):
            num_20.append(lista[i][19])

        num_20[0:5]
```

```
Out[5]: ['G', 'G', 'C', 'G', 'A']
```

Construimos una lista en la cual el nucleótido G en la posición 20 codifica el número 1 y los nucleótidos A, C o T codifican el número 0.

```
In [6]: num_20_bool = []
        for i in num_20:
            if i == "G":
                num_20_bool.append(1)
            else:
                num_20_bool.append(0)

        num_20_bool[0:5]
```

```
Out[6]: [1, 1, 0, 1, 0]
```

Es favorable para la eficiencia del gRNA que el PAM sea "CGG" en vez de "TGG", que es un PAM ineficiente. Por lo que, construimos una lista con el nucleótido en la posición 21, el primer nucleótido del PAM.

```
In [7]: num_21 = []
        for i in range(len(lista)):
            num_21.append(lista[i][20])

        num_21[0:5]
```

```
Out[7]: ['G', 'G', 'T', 'G', 'A']
```

Construimos una lista en la cual el nucleótido C en la posición 21 codifica el número 1 y los nucleótidos A, G y T codifican el número 0.

```
In [8]: num_21_bool = []
        for i in num_21:
            if i == "C":
                num_21_bool.append(1)
            else:
                num_21_bool.append(0)

        num_21_bool[0:5]
```

```
Out[8]: [0, 0, 0, 0, 0]
```

Construimos una lista con el número de nucleótidos A en cada ADN diana.

```
In [9]: num_a = []
        for i in lista:
            num_a.append(i.count("A"))

        num_a[0:5]
```

```
Out[9]: [1, 1, 5, 5, 4]
```

Comprobamos que efectivamente el código anterior funciona

```
In [10]: lista[2].count("A")
```

```
Out[10]: 5
```

Calculamos el porcentaje de nucleótidos G + C en los ADN diana. Esto se conoce como el contenido de GC.

```
In [11]: por_g_mas_c = []
for i in lista:
    por_g_mas_c.append((i.count("G") + i.count("C")) / 23)

por_g_mas_c[0:5]
```

```
Out[11]: [0.6956521739130435,
0.6956521739130435,
0.5652173913043478,
0.6086956521739131,
0.6956521739130435]
```

Un contenido de GC entre el 40% y el 60% es favorable, mientras que un contenido de GC por encima del 80% o por debajo del 35% es ineficiente. Por lo que, si en la lista anterior el porcentaje se encuentra entre 0.4 y 0.6, esto codificará el número 1. Si el porcentaje no está dentro de ese rango, se codificará el número 0.

```
In [12]: con_gc = []
for i in por_g_mas_c:
    if 0.40 < i < 0.60:
        con_gc.append(1)
    else:
        con_gc.append(0)

con_gc[0:6]
```

```
Out[12]: [0, 0, 1, 0, 0, 0]
```

Construimos una lista con el nucleótido de la posición 18 de cada gRNA.

```
In [13]: num_18 = []
for i in range(len(lista)):
    num_18.append(lista[i][17])

num_18[0:5]
```

```
Out[13]: ['T', 'C', 'C', 'C', 'G']
```

Construimos una lista en la cual el nucleótido C en la posición 18 codifica el número 1 y los nucleótidos A, G y T codifican el número 0.

```
In [14]: num_18_bool = []
for i in num_18:
    if i == "C":
        num_18_bool.append(1)
    else:
```

```
num_18_bool.append(0)
```

```
num_18_bool[0:5]
```

```
Out[14]: [0, 1, 1, 1, 0]
```

Construimos una lista con el nucleótido de la posición 16 de cada gRNA.

```
In [15]: num_16 = []  
for i in range(len(lista)):  
    num_16.append(lista[i][15])  
  
num_16[0:5]
```

```
Out[15]: ['T', 'C', 'A', 'G', 'G']
```

Construimos una lista en la cual el nucleótido C en la posición 16 codifica el número 1 y los nucleótidos A, G y T codifican el número 0.

```
In [16]: num_16_bool = []  
for i in num_16:  
    if i == "C":  
        num_16_bool.append(1)  
    else:  
        num_16_bool.append(0)  
  
num_16_bool[0:5]
```

```
Out[16]: [0, 1, 0, 0, 0]
```

Creamos una base de datos, en la que se han añadido todas las features anteriormente descritas y sus valores para cada ADN diana. Visualizamos las 15 primeras filas de la base de datos.

```
In [17]: base = pd.DataFrame({"Lista": lista,  
                             "Nucleótido en 20": num_20,  
                             "G_20": num_20_bool,  
                             "Nucleótido en 21": num_21,  
                             "C_21": num_21_bool,  
                             "Número de A": num_a,  
                             "Porcentaje de G y C": por_g_mas_c,  
                             "Contenido de GC": con_gc,  
                             "Nucleótido en 18": num_18,  
                             "C_18": num_18_bool,  
                             "Nucleótido en 16": num_16,  
                             "C_16": num_16_bool,  
                             "modFreq": df.modFreq})  
  
base.head()
```

Out[17]:

	Lista	Nucleótido en 20	G_20	Nucleótido en 21	C_21	Número de A	Porcentaje de G y C	Contenido de GC	Nucleótido en 18	C_18	Nucleótido en 16	C_16	modFreq
0	GGCTGCTTTACCCGCTGTGGGGG	G	1	G	0	1	0.695652	0	T	0	T	0	0.874864
1	TCCGGGTTGGCCTTCCACTGGGG	G	1	G	0	1	0.695652	0	C	1	C	1	0.871393
2	CAGCATCCTTCGAAAGCTCTGG	C	0	T	0	5	0.565217	1	C	1	A	0	0.860592
3	CGGTAGAAGCAGGTAGTCTGGGG	G	1	G	0	5	0.608696	0	C	1	G	0	0.802276
4	CATCCCGCTGCCCCAGTGAAGG	A	0	A	0	4	0.695652	0	G	0	G	0	0.796638

Eliminamos las columnas que no vamos a utilizar al aplicar los algoritmos de Machine Learning: "Nucleótido en 20", "Nucleótido en 21", "Nucleótido en 18" y "Nucleótido en 16".

```
In [18]: base = base.drop("Nucleótido en 20", axis = 1)
base = base.drop("Nucleótido en 21", axis = 1)
base = base.drop("Nucleótido en 18", axis = 1)
base = base.drop("Nucleótido en 16", axis = 1)

base.head()
```

Out[18]:

	Lista	G_20	C_21	Número de A	Porcentaje de G y C	Contenido de GC	C_18	C_16	modFreq
0	GGCTGCTTTACCCGCTGTGGGGG	1	0	1	0.695652	0	0	0	0.874864
1	TCCGGGTTGGCCTTCCACTGGGG	1	0	1	0.695652	0	1	1	0.871393
2	CAGCATCCTTCGAAAGCTCTGG	0	0	5	0.565217	1	1	0	0.860592
3	CGGTAGAAGCAGGTAGTCTGGGG	1	0	5	0.608696	0	1	0	0.802276
4	CATCCCGCTGCCCCAGTGAAGG	0	0	4	0.695652	0	0	0	0.796638

Construimos una matriz (X) con los valores de las features: "G_20", "C_21", "Número de A", "Contenido de GC", "C_18" y "C_16". También construimos un vector "target" con los valores de la columna "modFreq".

```
In [19]: X = base[["G_20", "C_21", "Número de A", "Contenido de GC", "C_18", "C_16"]]
y = base["modFreq"]
```

```
In [20]: from sklearn.model_selection import train_test_split
```

```
In [21]: X_train, X_test, y_train, y_test = train_test_split(X, y)
```

Empezamos a realizar pruebas con los algoritmos de Machine Learning. Primero, importamos un modelo de regresión lineal.

```
In [22]: from sklearn.linear_model import LinearRegression
```

```
In [23]: modelo = LinearRegression()
modelo.fit(X_train, y_train)
```

Out[23]:

▼ LinearRegression

LinearRegression()

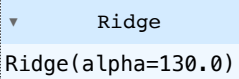
```
In [24]: modelo.score(X_test, y_test)
```

```
Out[24]: 0.03960358524315999
```

Ahora, importamos otro modelo de regresión conocido como "Ridge".

```
In [25]: from sklearn.linear_model import Ridge
```

```
In [26]: modelo1 = Ridge(alpha=130.0)  
modelo1.fit(X_train, y_train)
```

```
Out[26]:  Ridge(alpha=130.0)
```

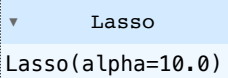
```
In [27]: modelo1.score(X_test, y_test)
```

```
Out[27]: 0.03559779601115842
```

Por último, importamos otro modelo de regresión llamado "Lasso".

```
In [28]: from sklearn.linear_model import Lasso
```

```
In [29]: modelo2 = Lasso(alpha=10.0)  
modelo2.fit(X_train, y_train)
```

```
Out[29]:  Lasso(alpha=10.0)
```

```
In [30]: modelo2.score(X_test, y_test)
```

```
Out[30]: -0.004562944633035748
```