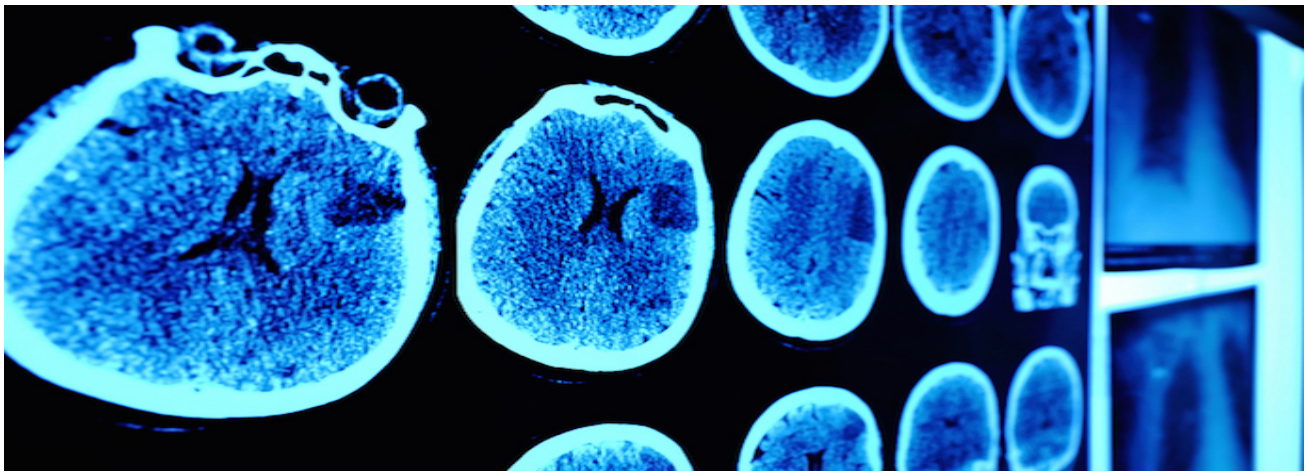


CONSTRUYENDO UN MODELO SIMPLE DE APRENDIZAJE AUTOMÁTICO SOBRE DATOS DE CÁNCER DE MAMA

14/05/2019

CÁNCER DE MAMA

SITUACIÓN MUNDIAL Y NACIONAL

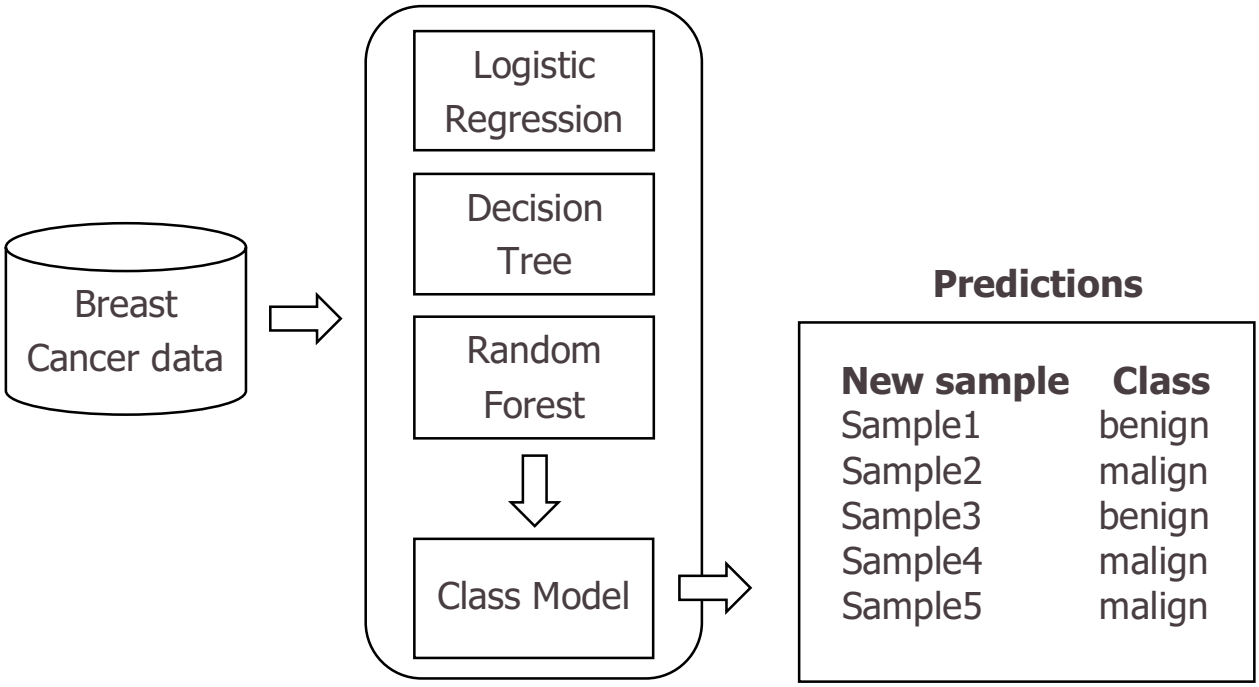
En el 2016, según estimaciones de la Agencia para la Investigación del Cáncer (IARC), un 16% de las muertes mundiales fueron debido al cancer. Más aún, de acuerdo a los datos del World Cancer Research Fundation (web), el cáncer de mama es el segundo cáncer más frecuente a nivel mundial y el primero en el caso de las mujeres. Sólo en el 2018 se presentaron alrededor de 2 millones de nuevos casos en todo el mundo. Sin duda uno de los problemas de salúd mundial más importantes.

En Chile, la situación no es muy diferente. El cáncer de mama es también la primera causa de muerte entre las mujeres chilenas con más de 1.300 decesos al año (Minsal), es decir, un promedio de 3 muertes diarias. Diferentes esfuerzos se han realizado por obtener una detección temprana de la enfermedad, ya que esto mejora significativamente la sobrevida de los pacientes. Del 2017 al 2018, las mamografías por Fonasa aumentaron un 12% llegando a un total de 14.098, logrando diagnosticar un total de 5.528 personas con cáncer de mama, de las cuales 1.367 eran mujeres sobre 60 años.

Una herramienta que permita una alta precisión en la clasificación de los tumores benignos/malignos puede ayudar al diagnóstico temprano, a evitar tratamientos innecesarios, asignar el tratamiento adecuado a cada paciente y en consecuencia mejorar la utilización de recursos. Técnicas como el Aprendizaje Automático (Machine Learning) han sido utilizados para crear modelos que puedan ayudar a la clasificación de enfermedades a partir de datos históricos.

OBJETIVO

Este ejercicio tiene como objetivo mostrar como a través de datos históricos de cáncer de mama podemos crear y comparar modelos de aprendizaje automático que nos permitan clasificar nuevas muestras de tejido en benignas o malignas, como se muestra en la figura 1. Para ello, compararemos el desempeño de 3 modelos: Logistic Regression (Regresión Logística), Decision Tree Classifier (Árboles de Decisión) y Random Forest Classifier (Bosques Aleatorios), utilizando el lenguaje de programación Python a través de la aplicación web de fuente abierta Jupyter Notebook.



ESTRUCTURA

Voy a estructurar el ejercicio en los siguientes apartados:

1. Información general de los datos
2. Instalar e Importar librerías en Python
3. Importar y preparar los datos
4. Explorar los datos y comprobar supuestos
5. Dividir el conjunto de datos
6. Configurar y testear los modelos
7. Predecir nuevas muestras con el mejor modelo
8. Consejos para mejorar el desempeño de los modelos
9. Conclusiones

INFORMACIÓN GENERAL DE LOS DATOS

Para crear nuestro modelo que nos ayude a clasificar muestras como benignas o malignas, utilizaremos datos públicos creado por el Dr. William H. Wolberg, médico del hospital de la Universidad de Wisconsin en Madison, Wisconsin, Estados Unidos. La base de datos la podemos descargar del siguiente link. Cada fila de la base de datos representa una muestra de tejido y cada columna, a excepción de la primera y la última, representa una variable o característica de cada muestra. La primera columna nos indica el ID de cada muestra y la última denota la característica de resultado, es decir, si la muestra fue benigna o maligna, representadas por los valores 2 y 4 respectivamente.

INFORMACIÓN DE LAS VARIABLES:

- **Sample code number:** ID de la muestra e.g. 1000025
- **Clump Thickness:** 1 - 10
- **Uniformity of Cell Size:** 1 - 10
- **Uniformity of Cell Shape:** 1 - 10
- **Marginal Adhesion:** 1 - 10
- **Single Epithelial Cell Size:** 1 - 10
- **Bare Nuclei:** 1 - 10
- **Bland Chromatin:** 1 - 10
- **Normal Nucleoli:** 1 - 10
- **Mitoses:** 1 - 10
- **Class:** (2: benignos, 4: malignos)

INSTALAR E IMPORTAR LIBRERIAS EN PYTHON

Primero debemos asegurar que contamos con todas las librerías a utilizar en Python. El siguiente código muestra como hacerlo directamente de Jupyter Notebook utilizando pip install.

```
#-- Instalar las librerias a utilizar
! pip install pandas
! pip install numpy
! pip install seaborn
! pip install matplotlib
! pip install sklearn
! pip install Plotly
! pip install cufflinks
```

Una vez terminada la instalación de todas las librerías, tenemos que importarlas para poder hacer uso de ellas.

```
#-- Manipulación de datos
import pandas as pd
import numpy as np

#-- Visualización de datos
import seaborn as sns
import matplotlib.pyplot as plt
from pylab import rcParams
import plotly.plotly as py
import plotly.tools as tls
import plotly.graph_objs as go
import cufflinks as cf
from bokeh.plotting import figure, output_file, show, output_notebook
from bokeh.models import ColumnDataSource
from bokeh.palettes import Spectral6
from bokeh.models import HoverTool

#-- Aprendizaje automático
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split

#-- Evaluación de los modelos
from sklearn.metrics import accuracy_score, precision_score, recall_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
```

Luego utilizamos ciertos parámetros para tener un mejor estilo en las visualizaciones posteriores. Adicionalmente, utilizaremos diferentes librerías para visualizar datos, para el caso de Plotly debemos establecer una conexión API. Para ello, solo debes crearte una cuenta gratuita en su página web y generar una API KEY (link).

```
#-- Formatos para una mejor visualización
%matplotlib inline
rcParams['figure.figsize'] = 10, 8
sns.set(style="ticks", color_codes=True)

#-- Estableciendo la conexión API de Plotly
tls.set_credentials_file(username='rmansilla', api_key='1Rp4uRujNLFPwdNWuKhW')
```

IMPORTAR Y PREPARAR LOS DATOS

Una vez que descargamos la base de datos, debemos importarla a nuestro Jupyter Notebook y definir correctamente los nombres de todas las variables. Para chequear que todo este en orden podemos utilizar la función **head()** de pandas para visualizar las primeras cinco filas de la base de datos.

```
#-- Importar la base de datos
df = pd.read_csv('breast-cancer-wisconsin.data.csv',header=None)

#-- Definir el nombre de las columnas
df.columns = ['sample_number','clump_thickness','uniformity_cell_size','uniformity_cell_shape'
              ,'adhesion','cell_size','bare_nuclei','bland_chromatin','normal_nucleoli',
              'mitoses','class']

#-- Desplegar las primeras 5 filas
df.head()
```

	sample_number	clump_thickness	uniformity_cell_size	uniformity_cell_shape	adhesion	cell_size	bare_nuclei	bland_chromatin	normal_nucleoli	mitoses	class
0	1000025	5	1	1	1	2	1	3	1	1	2
1	1002945	5	4	4	5	7	10	3	2	1	2
2	1015425	3	1	1	1	2	2	3	1	1	2
3	1016277	6	8	8	1	3	4	3	7	1	2
4	1017023	4	1	1	3	2	1	3	1	1	2

Para poder implementar correctamente ciertos modelos de aprendizaje automático, debemos verificar que nuestra base de datos no contenga datos erróneos. Para eso comenzaremos por obtener información general de los datos utilizando la función **info()**. Esta función entrega información relevante, por ejemplo, podemos notar que la base de datos cuenta con un total de 699 muestras sin valores nulos para cada una de las 11 columnas. Lo interesante es que, de las 11 variables, 10 son numéricas y 1 categórica. Esta última variable categórica, 'bare_nuclei', llama particularmente la atención porque cuando desplegamos las primeras 5

filas de los datos, pudimos notar que todas las variables eran numéricas. Esto nos hace pensar que posiblemente la variable 'bare_nuclei' contenga alguna celda con contenido categórico.

```
#-- Información general
print(df.info())

#-- Número de filas y columnas
print('\n')
print("Numbero de muestras:", df.shape[0])
print("Numbero de variables:", df.shape[1])
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 699 entries, 0 to 698
Data columns (total 11 columns):
sample_number             699 non-null int64
clump_thickness           699 non-null int64
uniformity_cell_size      699 non-null int64
uniformity_cell_shape     699 non-null int64
adhesion                  699 non-null int64
cell_size                 699 non-null int64
bare_nuclei               699 non-null object
bland_chromatin           699 non-null int64
normal_nucleoli           699 non-null int64
mitoses                   699 non-null int64
class                     699 non-null int64
dtypes: int64(10), object(1)
memory usage: 60.1+ KB
None


Numbero de muestras: 699
Numbero de variables: 11
```

No siempre los valores nulos son espacios en blanco, sino que también pueden ser celdas que contengas símbolos como: '?', '#', '$', etc. El siguiente código nos ayuda a detectar si existen celdas con símbolos que pueden estar impidiendo transforma la variable 'bare_nuclei' en una variable numérica.

```
#-- Verifacar si existen celdas con el simbolo '?'
print(df[df == '?'].count())

#-- Verifacar si existen celdas con el simbolo '#'
print(df[df == '#'].count())
```

```
sample_number               0
clump_thickness             0
uniformity_cell_size        0
uniformity_cell_shape       0
adhesion                    0
cell_size                   0
bare_nuclei                16
bland_chromatin             0
normal_nucleoli             0
mitoses                     0
class                       0
dtype: int64
sample_number              0
clump_thickness            0
uniformity_cell_size       0
uniformity_cell_shape      0
adhesion                   0
cell_size                  0
bare_nuclei                0
bland_chromatin            0
normal_nucleoli            0
mitoses                    0
class                      0
dtype: int64
```

El resultado nos indica que efectivamente la variable 'bare_nuclei' contienen un total de 16 celdas con el símbolo '?'. Para poder deshacernos de los símbolos '?', tenemos que reemplazarlos por valores nulos y luego aplicar la función **fillna(method='ffill')** que llenará hacia adelante cualquier valor nulo con el valor del último elemento no nulo de la columna correspondiente. A continuación, transformamos la columna 'bare_nuclei' en numérica con la función **to_numeric()** y luego a tipo entero con **astype()**.

```
#-- Reemplazar el simbolo '?' por NaN
df[df=='?']=np.NaN

#-- Aplicamos el médoto 'ffill'
df = df.fillna(method='ffill')
```

```
#-- Convirtiendo la columna 'bare_nuclei' a numérica (int)
df['bare_nuclei'] = pd.to_numeric(df['bare_nuclei'])
df['bare_nuclei'] = df['bare_nuclei'].astype(int)
```

Dado que la variable 'sample_number' solo nos indica el ID de cada muestra, la podemos eliminar con **drop()**, ya que no aportará ninguna información relevante para nuestros modelos. Finalmente, chequeamos que todo este en orden.

```
#-- Elminar la columna sample_number
df = df.drop(['sample_number'], axis = 1)
```

```
#-- Verificar que todo esté en orden
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 699 entries, 0 to 698
Data columns (total 10 columns):
clump_thickness          699 non-null int64
uniformity_cell_size     699 non-null int64
uniformity_cell_shape    699 non-null int64
adhesion                 699 non-null int64
cell_size                699 non-null int64
bare_nuclei              699 non-null int64
bland_chromatin          699 non-null int64
normal_nucleoli          699 non-null int64
mitoses                  699 non-null int64
class                    699 non-null int64
dtypes: int64(10)
memory usage: 54.7 KB
```

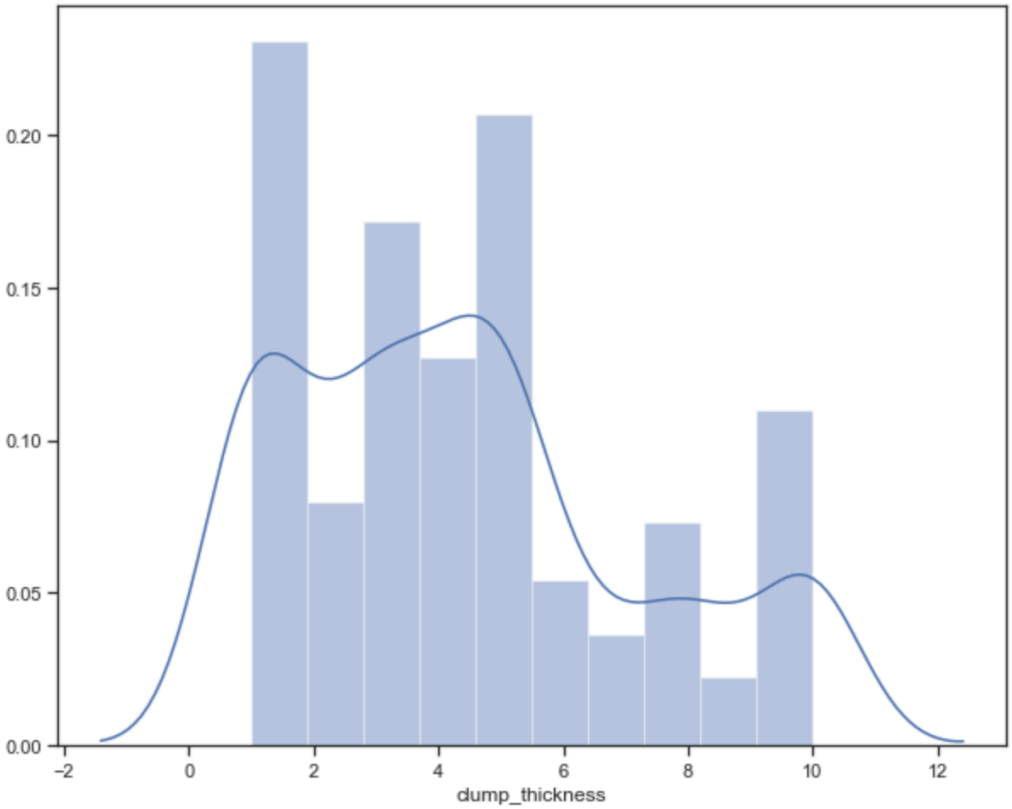
EXPLORAR LOS DATOS Y COMPROBAR SUPUESTOS

La visualización de datos es uno de los aspectos más relevantes para alguien que trabaja con datos, ya que ayuda a comprender patrones, relaciones e información en los datos de una forma más rápida, así como también facilita la tarea de compartir los resultados encontrados con otras personas. Python cuenta con variadas librerías de visualización de datos, en este ejercicio usaremos 4 (Matplotlib, Seaborn, Bokeh y Plotly), que nos ayudarán a visualizar distintas tareas.

ANÁLISIS UNIVARIABLE

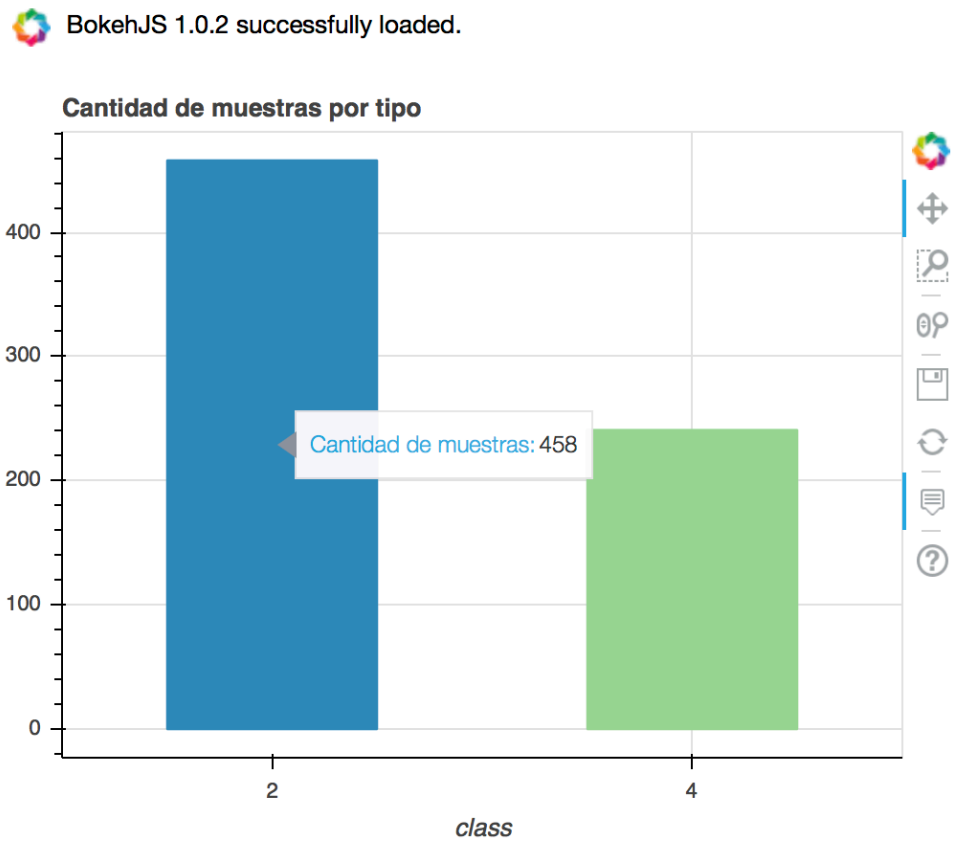
Un histograma de cada una de las variables nos ayudará a entender la distribución de cada variable. En este caso solo analizaremos la variable 'clump_thickness'. A simple vista, utilizando la librería Seaborn, podemos notar que la variable 'clump_thickness' no tiene una distribución normal, más bien la podríamos asociar a una distribución multimodal. Más

adelante mostraremos como realizar un diagrama de dispersión entre las variables para entender cómo se relacionan las variables entre sí.



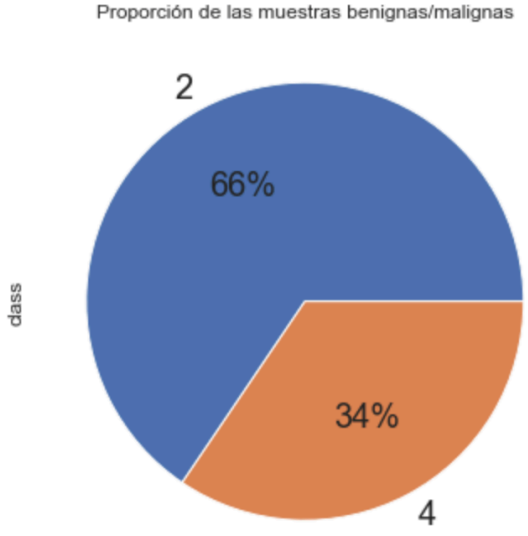
La variable 'class' es nuestra variable objetivo para la cual queremos crear un modelo de predicción. Mediante un gráfico de barras, utilizando la librería Bokeh podemos visualizar la cantidad de muestras benignas y malignas de manera interactiva, es decir, si posamos el cursor sobre las barras nos indicará la cantidad. Además, podemos realizar zoom, guardar la imagen, mover el gráfico y otras opciones que se pueden personalizar. Podemos notar entonces que nuestra variable objetivo es binaria (2: benigna y 4: maligna), de las cuales 458 muestras son benignas y 241 malignas. Esto es relevante, ya que definirá el tipo de problema al que nos vemos enfrentado, que en este caso sería un problema de clasificación (benigno o maligno).

```
#-- Gráfico de barras
bar_plot_b(df,'class','class','count','Cantidad de muestras por tipo')
```



En términos de proporción, un gráfico de torta nos muestra que un 66% de las muestras corresponden a muestras benignas y un 34% a muestras malignas. Información importante, ya que queremos evitar entrenar nuestros modelos en bases de datos desbalanceadas, como por ejemplo 90/10.

```
#-- Gráfico de torta
pie_chart(df,'class','Proporción de las muestras benignas/malignas')
```



RESUMEN ESTADÍSTICO

Algunos cálculos estadísticos, como la mediana, la moda, la media, valor mínimo y máximo ayudan a resumir un conjunto de datos, con el fin de comunicar la mayor cantidad de información de una manera más sencilla. Esto nos permite tener una mejor comprensión de nuestros datos y sacar conclusiones rápidas al respecto. Podemos utilizar diferentes funciones en Python para realizar distintos cálculos estadísticos. Sin embargo, con la función **describe()** podemos obtener rápidamente un resumen estadístico. Como podemos ver, todas las variables predictoras se encuentran en la misma escala, entre 1 y 10, por lo que no es necesario estandarizar los datos. Adicionalmente, podemos notar que a simple vista no existen valores extremos, pero eso lo corroboraremos más adelante.

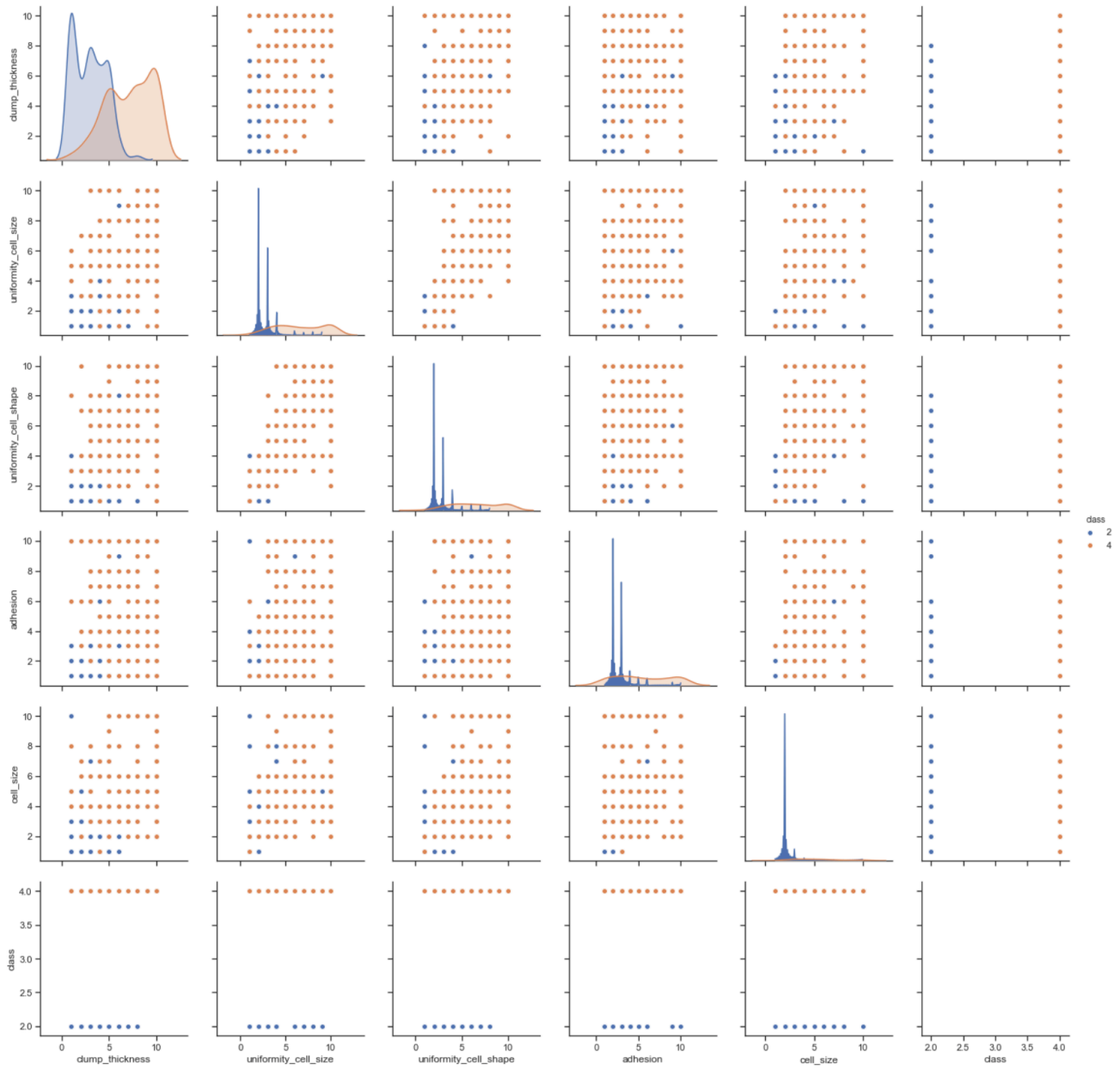
	clump_thickness	uniformity_cell_size	uniformity_cell_shape	adhesion	cell_size	bare_nuclei	bland_chromatin	normal_nucleoli	mitoses
count	699.000000	699.000000	699.000000	699.000000	699.000000	699.000000	699.000000	699.000000	699.000000
mean	4.417740	3.134478	3.207439	2.806867	3.216023	3.529328	3.437768	2.866953	1.589413
std	2.815741	3.051459	2.971913	2.855379	2.214300	3.635260	2.438364	3.053634	1.715078
min	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
25%	2.000000	1.000000	1.000000	1.000000	2.000000	1.000000	2.000000	1.000000	1.000000
50%	4.000000	1.000000	1.000000	1.000000	2.000000	1.000000	3.000000	1.000000	1.000000
75%	6.000000	5.000000	5.000000	4.000000	4.000000	6.000000	5.000000	4.000000	1.000000
max	10.000000	10.000000	10.000000	10.000000	10.000000	10.000000	10.000000	10.000000	10.000000

ANÁLISIS MULTIVARIABLE

Un análisis multivariable nos ayuda a comprender como se relacionan las variables entre sí. Para ello, utilizaremos la función de Seaborn **pairplot()** para desplegar un diagrama de dispersión entre las variables, la cual también en la diagonal nos entrega la distribución de cada variable. Para una mejor visualización solo haremos este análisis con algunas de las variables, eliminando para este caso 'bare_nuclei', 'bland_chromatin', 'normal_nucleoli' y 'mitoses'. Rápidamente podemos observar que las variables, 'uniformity_cell_size', 'uniformity_cell_shape', 'adhesion' y 'cell_size' tienen un sesgo a la derecha concentrado entre los valores 1 y 5. Anteriormente, en el análisis univariable, nos dimos cuenta que la variable 'clump_thickness' se podía asociar más a una distribución multimodal. Sin embargo, ahora al estar diferenciados por color con la variable objetivo 'class', podemos notar que las muestras benignas tienen un sesgo a la derecha mientras que las muestras malignas a la izquierda.

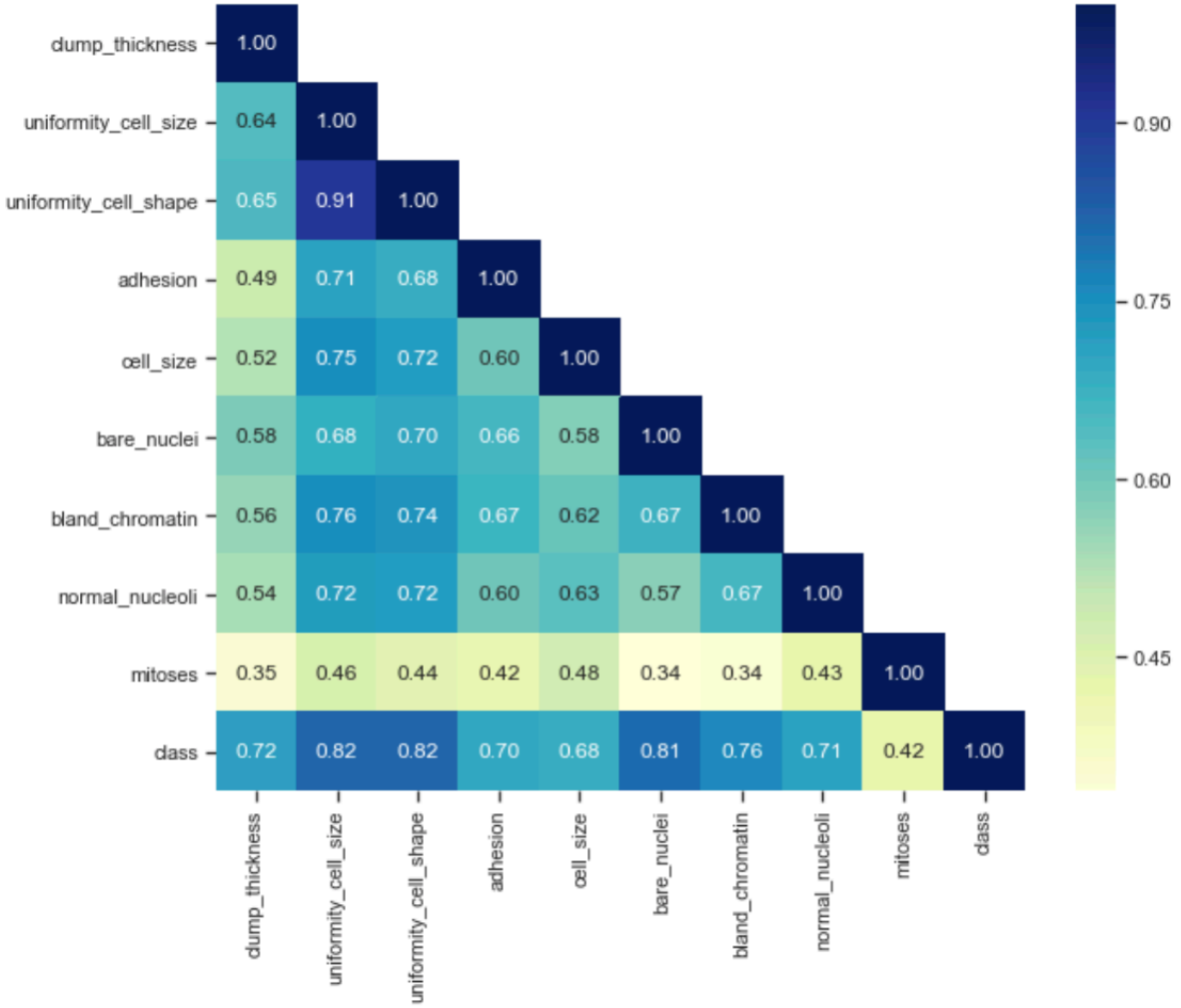
```
#-- Pairplot
df_select = df.drop(['bare_nuclei','bland_chromatin','normal_nucleoli','mitoses'], axis = 1)

sns.pairplot(df_select, height= 3, aspect= 1, hue='class')
```



Una matriz de correlación en forma de mapa de calor es una forma visual muy útil para comprender la correlación entre las variables. A primera vista se puede observar que la variable objetivo 'class' tiene una fuerte correlación con cada una de las variables predictoras, a excepción de la variable 'mitosis' con la cual existe una correlación leve. Con respecto a las variables predictoras, llama la atención la fuerte correlación entre las variables 'uniformity_cell_size' y 'uniformity_cell_size' que podría indicar multicolinealidad, es decir, que básicamente comparten la misma información.

```
#-- Matriz de correlación
df_corr = df.corr()
mask = np.triu(df_corr, k=1)
sns.heatmap(df_corr, cmap= 'YlGnBu', annot=True, fmt=".2f", mask=mask, )
```

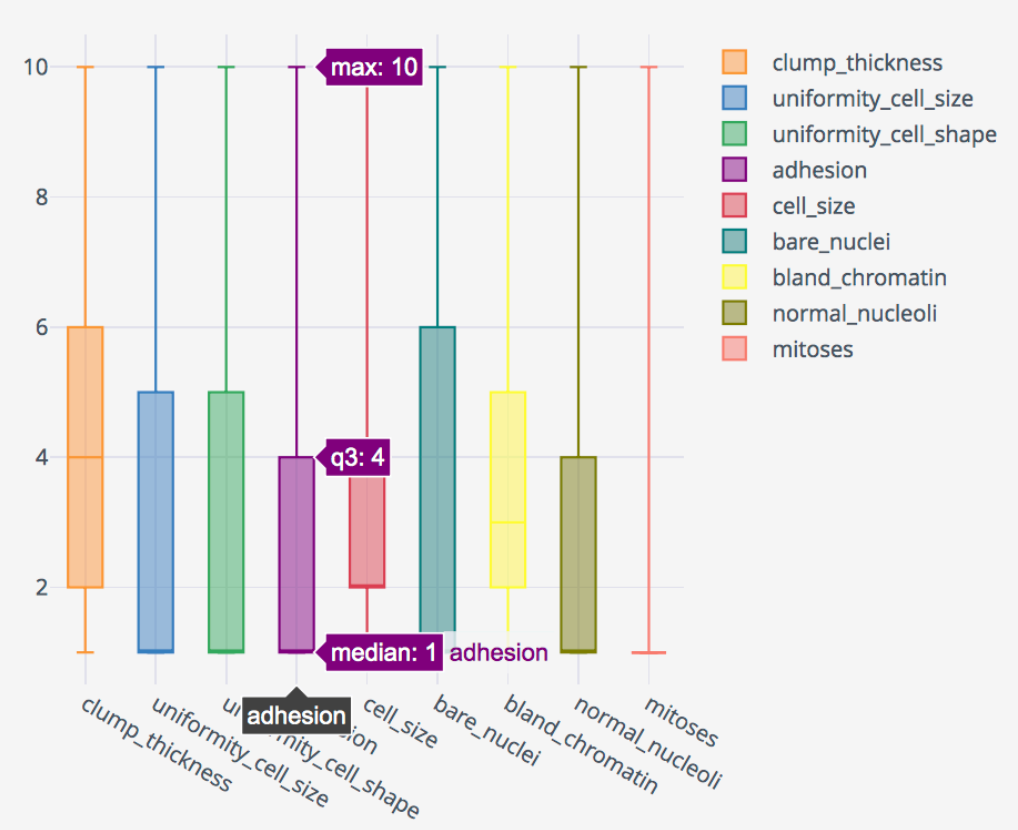


VERIFICAR SUPUESTOS

Más adelante explicaremos por que vamos a utilizar ciertos modelos de aprendizaje automático, por el momento sólo mencionaré que utilizaré Logistic Regression como uno de los modelos. Para ello es necesario verificar que los datos cumplan los supuestos del modelo, ya que así nos aseguramos que el modelo se ajustará mejor a los datos. Primero, verificamos si existen datos suficientes para cada una de las variables predictoras. Segundo, que además no existan datos nulos. Nuevamente la función **info()** es una forma rápida de validar ambos supuestos, en donde podemos notar que en cada una de las variables hay 699 datos sin valores nulos, un valor que nos permite utilizar Logistic Regression.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 699 entries, 0 to 698
Data columns (total 10 columns):
clump_thickness           699 non-null int64
uniformity_cell_size      699 non-null int64
uniformity_cell_shape     699 non-null int64
adhesion                  699 non-null int64
cell_size                 699 non-null int64
bare_nuclei               699 non-null int64
bland_chromatin           699 non-null int64
normal_nucleoli           699 non-null int64
mitoses                   699 non-null int64
class                     699 non-null int64
dtypes: int64(10)
memory usage: 54.7 KB
```

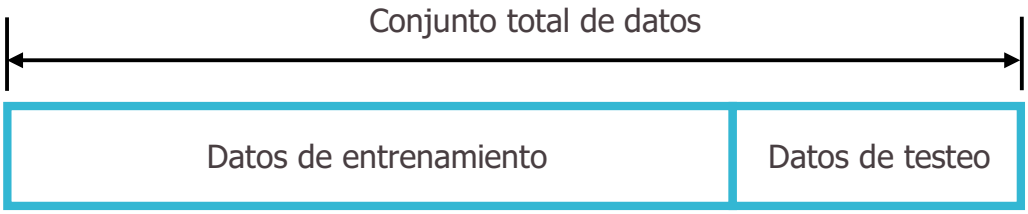
Tercero, tenemos que verificar que no existan valores extremos o outliers en las variables predictoras que puedan afectar el rendimiento del modelo. Plotly, es otra librería de visualización de datos interactiva la cual usaremos para crear un diagrama de caja, una manera simple para representar gráficamente una serie de datos numéricos a través de sus cuartiles y detectar posibles outliers. El diagrama de caja nos reafirma que para todas las variables los datos se encuentran entre 1 y 10 sin la presencia de outliers o valores extremos.



Cuarto, es necesario verificar que la variable objetivo es binaria, lo cual ya confirmamos en el análisis univariable con el gráfico de barras. Finalmente, es recomendable que no exista una alta correlación entre las variables predictoras. De acuerdo a lo mostrado anteriormente por la matriz de correlación en forma de mapa de calor, se pudo observar la mayoría de las variables están fuertemente correlacionadas, lo que nos lleva a pensar que probablemente Logistic Regression no termine siendo el modelo más preciso.

DIVIDIR EL CONJUNTO DE DATOS

Uno de los problemas más comunes en los modelos de aprendizaje automático es overfitting o sobreajuste, en otras palabras, sobreentrenar nuestro modelo con datos conocidos para que se ajuste lo más posible a nuestro set de datos, pero que falle al predecir situaciones distintas a las del entrenamiento. Una buena práctica para evitar overfiting, es dividir nuestro conjunto de datos en aquellos que utilizaremos para entrenar y en aquellos que usaremos para testear los modelos.



La librería SciKit-Learn cuenta con una función en Python que divide los datos por nosotros. Primero separamos nuestra variable objetivo (y) de las variables predictoras (X). Luego, usamos la función **train_test_split()** para dividir aleatoriamente nuestros datos. Normalmente, se trabaja con un 80% de los datos para entrenamiento y 20% para testeo. El resultado es 559 datos que se usarán para entrenar los modelos y 140 datos para testear cada uno de ellos.

```
#-- Separando la variable objetivo 'class' con las variables predictoras
y = df['class']
X = df.drop('class', axis = 1)
```

```
#-- Diviendo los datos de entrenamiento y testeo
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=0)

#-- Desplegar el detalle de entrenamiento y testeo
print( "Predictor - Training : ", X_train.shape, "Predictor - Testing : ", X_test.shape )
```

```
Predictor - Training :  (559, 9) Predictor - Testing :  (140, 9)
```

CONFIGURAR Y TESTEAR LOS MODELOS

Esta es la etapa más emocionante en la aplicación de un modelo de aprendizaje automático a cualquier conjunto de datos. También se conoce como selección de algoritmo, entre los muchos que existen, para predecir el mejor resultado. Como mencionamos en los objetivos, en este ejercicio solo compararemos 3 modelos: Logistic Regression, Decision Tree Classifier y Random Forest Classifier. Existen otros algoritmos que se pueden utilizar para este ejercicio tales como: K-Nearest Neighbor, Support Vector Machine, Kernel SVM, Naïve Bayes, Neural network, entre otros.

Entre las ventajas de usar Decision Tree podemos mencionar que es un algoritmo de fácil interpretación, excelente para explorar datos, requiere muy poca limpieza, es robusto antes valores nulos, es flexible y no-paramétrico. Por su parte Random Forest es un modelo con un gran desempeño en datos reales, muy robusto a outliers, escalable y que naturalmente puede modelar fronteras de decisión no-lineales gracias a su estructura jerárquica. Sin embargo, ambos modelos son susceptibles al sobre entrenamiento sino se tiene cuidado con la parametrización del modelo. Logistic Regression, por su parte, es uno de los algoritmos más utilizado, es rápido, limpio y su resultado es de fácil interpretación. Como desventaja, podemos mencionar que generalmente termina siendo un modelo muy simple para datos del mundo real, no tiene un buen desempeño cuando los valores nulos no son bien tratados y naturalmente no maneja más de una variable de resultado.

Logistic Regression

De la librería SciKit-Learn importamos el algoritmo Logistic Regression. Luego entrenamos el modelos con los conjuntos de entrenamiento (y_train y X_train) y lo testeamos con el conjunto de testeo (X_test). A continuación, para medir el rendimiento de los 3 modelos vamos a calcular la precisión del modelo (accuracy) mediante la función **accuracy_score()**. Con la función **confusion_matrix()** calcularemos una matriz de confusión, que es muy útil para entender que cantidad de los datos fueron predecidos correcta y erroneamente. Finalmente, mediante la función **classification_report()** obtendremos precision y recall (sensitivity), otras metricas que se utilizan para medir el desempeño de los modelos. Las siguientes formulas ayudan a entender mejor estas métricas:

- Accuracy = (TP+TN)/(TP+TN+FP+FN)
- Precision = TN/(TP+FP)
- Recall o Sensitivity = TP/(TP+FN)

En donde TP = Nº verdaderos positivos, TN = Nº verdaderos negativos, FP = Nº falsos positivos y FN = Nº falsos negativos. Dependerá del tipo de problema al que nos enfrentemos para saber cual de estas métricas es más relevante para evaluar nuestros modelos.

Para nuestro modelo de Logistic Regression obtuvimos un accuracy del 97%. De la matriz de confusión podemos notar que 83 y 53 son los datos predecidos correctamente, mientras que por otro lado tenemos 2 falsos-positivos y 2 falsos-negativos. Finalmetne, obtuvimos una precision promedio del 97% y un recall tambien del 97%.

```
#-- Entrenamos el modelo
lr = LogisticRegression(random_state = 0)

lr.fit(X_train,y_train)

#-- Testeamos el modelo
lr_pred = lr.predict(X_test)

#-- Desplegamos los resultados del modelo
print('* Logistic Regression accuracy: ',accuracy_score(y_test, lr_pred))
print('\n')
print('* Matriz de Confusión:')
print(confusion_matrix(y_test, lr_pred))
print('\n')
print('* Informe de Clasificación :')
print(classification_report(y_test, lr_pred))
```

```
* Logistic Regression accuracy:  0.9714285714285714


* Matriz de Confusión:
[[83  2]
 [ 2 53]]


* Informe de Clasificación :
             precision    recall  f1-score   support

          2       0.98      0.98      0.98        85
          4       0.96      0.96      0.96        55

avg / total       0.97      0.97      0.97       140
```

Decision Tree Classifier

En el caso de nuestro modelo de Decision Tree, podemos notar una menor accuracy de 93%. La matriz de confusión muestra 81 verdaderos positivos, 49 verdaderos negativos, 4 falsos negativos y 6 falsos positivos. Por último, obtuvimos una precision y recall del 93%.

```
#-- Entrenamos el modelo
dt = DecisionTreeClassifier(random_state = 0)

dt.fit(X_train,y_train)

#-- Testeamos el modelo
dt_pred = dt.predict(X_test)

#-- Desplegamos los resultados del modelo
print('* Decision Tree accuracy: ',accuracy_score(y_test, dt_pred))
print('\n')
print('* Matriz de Confusión:')
print(confusion_matrix(y_test, dt_pred))
print('\n')
print('* Informe de Clasificación :')
print(classification_report(y_test, dt_pred))
```

```
* Random Forest accuracy:  0.9285714285714286


* Matriz de Confusión:
[[81  4]
 [ 6 49]]


* Informe de Clasificación :
             precision    recall  f1-score   support

          2       0.93      0.95      0.94        85
          4       0.92      0.89      0.91        55

avg / total       0.93      0.93      0.93       140
```

Random Forest Classifier

Entrenando y testeando nuestro último modelo Random Forest obtuvimos un accuracy del 98% . La matriz de confusión, por su parte, muestra 83 verdaderos positivos, 54 verdaderos negativos, 2 falsos negativos y solo 1 falso positivo. Por último obtuvimos una precision y recall del 98%.

```
#-- Entrenamos el modelo
rf = RandomForestClassifier(random_state = 0)

rf.fit(X_train,y_train)

#-- Testeamos el modelo
rf_pred = rf.predict(X_test)

#-- Desplegamos los resultados del modelo
print('* Random Forest accuracy: ',accuracy_score(y_test, rf_pred))
print('\n')
print('* Matriz de Confusión:')
print(confusion_matrix(y_test, rf_pred))
print('\n')
print('* Informe de Clasificación :')
print(classification_report(y_test, rf_pred))
```

```
* Random Forest accuracy:  0.9785714285714285


* Matriz de Confusión:
[[83  2]
 [ 1 54]]


* Informe de Clasificación :
             precision    recall  f1-score   support

          2       0.99      0.98      0.98        85
          4       0.96      0.98      0.97        55

avg / total       0.98      0.98      0.98       140
```

Finalmente hemos construido nuestros 3 modelos. Para este ejercicio una métrica muy relevante sería sensitivity, ya que es más grave tener falsos negativos que falsos positivo. Sin embargo, tampoco queremos tener gente en tratamiento que no tengan la enfermedad. Por lo tanto, elegir la mejor métrica aveces dependerá de cada equipo evaluador. Para este ejercicio nos centraremos en las métricas sensitivy y accuracy. Teniendo esto en cuenta, es posible concluir que Random Forest es el algoritmo que mejor predice nuestras muestras con un 98% de sensitivity y accuracy.

PREDECIR NUEVAS MUESTRAS CON EL MEJOR MODELO

Dado que el entrenamiento de nuestro modelo Random Forest fue el que entregó mejores resultados, lo usaremos para predecir nuevos datos. Hagamos cuenta que llegan dos nuevas muestras al laboratorio con las siguientes características:

Muestra 1		**Muestra 2**	
clump_thickness	8	clump_thickness	2
uniformity_cell_size	10	uniformity_cell_size	1
uniformity_cell_shape	10	uniformity_cell_shape	3
adhesion	8	adhesion	6
cell_size	7	cell_size	3
bare_nuclei	6	bare_nuclei	10
bland_chromatin	9	bland_chromatin	4
normal_nucleoli	7	normal_nucleoli	2
mitoses	2	mitoses	10

Si ingresamos cada nueva muestra a nuestro modelo de predicción obtendremos que la primera muestra la predice como 4, es decir, como maligna. Por el contrario, la segunda el modelo la predice como benigna (valor 2).

```
new_sample_1 = np.array([8,10,10,8,7,10,9,7,2]).reshape(1, -1)
new_sample_2 = np.array([2,1,3,6,3,10,4,2,10]).reshape(1, -1)
new_sample_1=rf.predict(new_sample_1)
new_sample_2=rf.predict(new_sample_2)
print("La predicción para la nueva muestra 1 es: ",new_sample_1[0])
print("La predicción para la nueva muestra 2 es: ",new_sample_2[0])
```

```
La predicción para la nueva muestra 1 es:  4
La predicción para la nueva muestra 2 es:  2
```

Es importante recordar, que la mayoría de los modelos de aprendizaje automático mejoran su rendimiento a medida que cuentan con más datos para entrenar. Por consiguiente, es importante ir alimentando las bases de datos con toda nueva información.

CONSEJOS PARA MEJORAR LOS MODELOS

En este ejercicio hemos aplicado algunas buenas prácticas que mejoran el desempeño de los modelos. Sin embargo, Los modelos de aprendizaje automático siempre pueden ser mejorados. Algunos consejos para mejorar nuestros modelos son:

1. Recolectar más datos de entrenamiento.
2. Seleccionar aquellas variables con el mayor valor predictivo.
3. Realizar ingeniería de variables, es decir, crear y/o manipular variables que puedan mejorar el modelo.
4. Buscar los mejores hiper-parametros para el modelo específico.
5. Entrenar y testear otros modelos.

CONCLUSIONES

Finalmente, hemos logrado construir un modelo de aprendizaje automático que es capaz de predecir nuevas muestras de tejido en benignas o malignas. Para lograrlo, hemos pasado por una serie de etapas, entre ellas comenzamos con una preparación simple de los datos, eliminando caracteres innecesarios, reemplazándolas por valores más adecuados y modificándolas al tipo de dato correspondiente. Luego, exploramos los datos, lo cual nos ayudó a comprender las variables y la relación entre ellas. Antes de pasar a construir nuestros modelos, verificamos algunos supuestos. Una vez finalizado lo anterior construimos,

entrenamos y testeamos nuestros 3 modelos: Logistic Regression, Decision Tree y Random Forest. Para evaluar el rendimiento de los modelos utilizamos diferentes métricas, tales como accuracy, precisión y sensitivity. Random Forest fue nuestro modelo con mejor desempeño logrando acertadamente el 98% de los datos de testeo. Para finalizar, realizamos el ejercicio de simular la llegada de nuevas muestras de tejido para que nuestro modelo final prediga si las muestras son benignas o malignas.

Esta fue una aplicación sencilla de modelos de aprendizaje automático para un conjunto de datos en particular. Todos los conjuntos de datos son distintos, por lo que siempre para elegir nuestro mejor modelo, primero debemos analizar los datos y luego aplicar nuestros modelos de aprendizaje automático eligiendo aquel con mejor desempeño de acuerdo a la métrica específica para cada problema.

Siéntete libre de hacer preguntas si tienes alguna duda a través de mi LinkedIn o correo: robertomansilla.lobos@gmail.com.

Si quieres obtener el código completo solicítamelo por un mensaje a través de LinkedIn.

¡Espero que hayas disfrutado del artículo, así como yo disfruté haciéndolo!