



❖ **Asignatura:**

Aprendizaje Inteligente

❖ **Catedrático:**

FRANCISCO JAVIER LUNA ROSAS

❖ **Tema:**

Aprendizaje Supervisado y No Supervisado

❖ **Equipo:**

Michelle Stephanie Adame

Andrés Eloy Escobedo Esparza

❖ **Carrera:**

I.C.I.

❖ **Semestre:**

6º

Aprendizaje No supervisado.

Los algoritmos de Aprendizaje no Supervisados infieren patrones de un conjunto de datos sin referencia a resultados conocidos o etiquetados. A diferencia del Aprendizaje Supervisado, los métodos de Aprendizaje no Supervisado no se pueden aplicar directamente a un problema de regresión o clasificación porque no tiene idea de cuáles pueden ser los valores de los datos de salida, lo que hace imposible que entrene el algoritmo de la forma en que lo haría normalmente. En cambio, el aprendizaje sin supervisión puede utilizarse para descubrir la estructura subyacente de los datos.

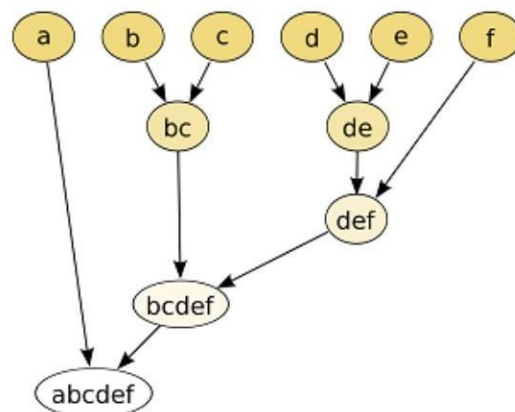
Los algoritmos de Aprendizaje no Supervisados te permiten realizar tareas de procesamiento más complejas en comparación con el Aprendizaje Supervisado. Sin embargo, el aprendizaje sin supervisión puede ser más impredecible en comparación con otros métodos de aprendizaje naturales. Los algoritmos de Aprendizaje no Supervisados se utilizan para agrupar los datos no estructurados según sus similitudes y patrones distintos en el conjunto de datos. El término "no supervisado" se refiere al hecho de que el algoritmo no está guiado como el algoritmo de Aprendizaje Supervisado.

Como previamente se explicó la diferencia entre el aprendizaje no supervisado y supervisado es la diferencia en el dataset y en los algoritmos. Para nuestro examen por la parte del aprendizaje no supervisado usamos agrupación jerárquica:

➤ Algoritmo: Cluster Jerárquico.

El Clustering Jerárquico (agrupamiento jerárquico o *Hierarchical Clustering* en inglés), es un método de data mining para agrupar datos (en minería de datos a estos grupos de datos se les llama clústers). El algoritmo de clúster jerárquico agrupa los datos basándose en la distancia entre cada uno y buscando que los datos que están dentro de un clúster sean los más similares entre sí.

En una representación gráfica los elementos quedan anidados en jerarquías con forma de árbol. Lo mejor para explicarlo es una imagen. Así que para ilustrar mejor este tema de agrupación en categorías voy a retomar un ejemplo gráfico muy difundido – y a la vez es el más descriptivo que he encontrado – que es el que exponen en la Wikipedia. En la primera

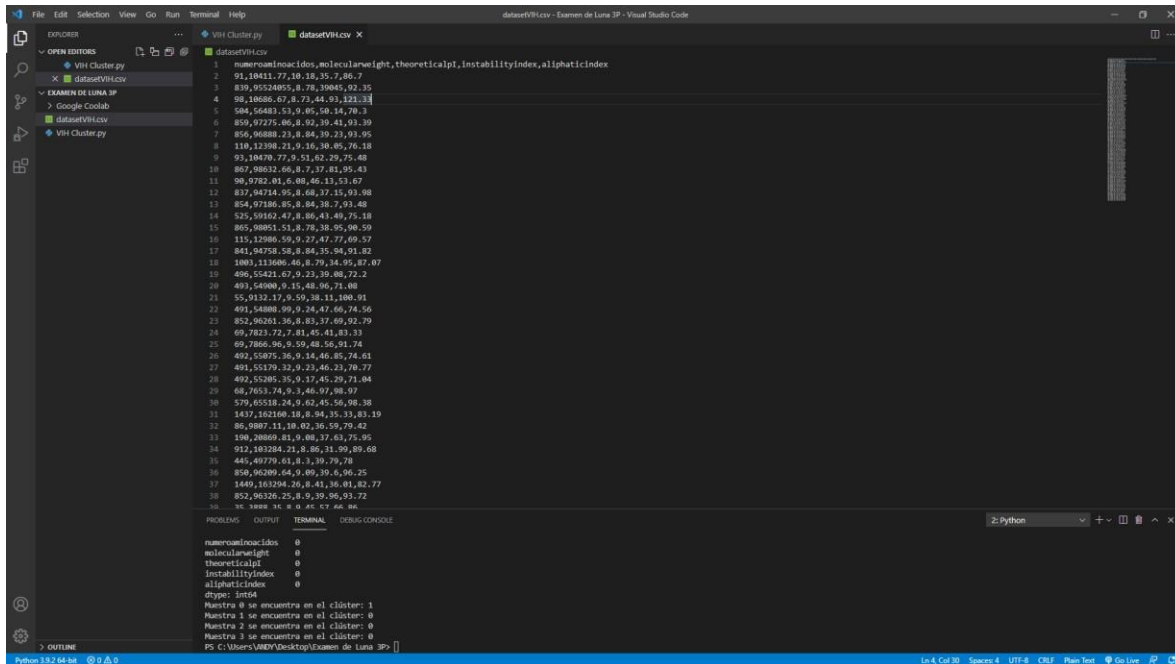


Cluster jerárquico en forma de árbol

imagen vemos cómo están distribuidos los datos y a qué distancia se encuentran unos de otros. En la segunda, vemos un ejemplo de clustering jerárquico dónde los datos se agrupan en función de la distancia (en este caso distancia euclidiana) entre ellos.

➤ Explicación:

En el caso de no supervisado implementamos el dataset de manera que no estuviera etiquetado de que solamente vinieran los datos, para esto colocamos lo que viene siendo:



```
datasetVH.csv
1  Numerical values, molecular weight, theoretical pI, instability index, aliphatic index
2  91,10411,77,10,10,2,06,7
3  839,95524055,8,78,10045,92,35
4  98,18086,67,8,72,44,93,121,33
5  504,56481,53,9,65,50,14,70,2
6  859,97275,86,8,92,39,41,93,39
7  856,96888,23,8,84,39,23,93,95
8  119,12398,21,9,16,30,85,76,18
9  93,10478,77,9,53,62,29,75,48
10  867,98632,66,8,7,37,81,95,43
11  98,9782,81,6,88,46,13,53,67
12  837,94714,95,8,68,17,15,91,98
13  854,97186,85,8,88,38,7,93,48
14  525,59162,47,8,88,43,49,75,18
15  865,98051,53,8,78,18,95,90,59
16  115,12986,59,9,27,42,72,69,57
17  841,94758,58,8,84,35,54,91,82
18  1003,113686,46,8,79,34,95,87,87
19  496,55421,67,9,23,39,88,72,2
20  493,54006,9,15,48,96,71,88
21  55,9132,17,9,59,38,11,100,91
22  491,54888,99,9,24,47,66,74,56
23  852,96261,36,8,83,37,59,92,79
24  69,7823,22,7,81,45,41,83,13
25  69,7886,96,9,59,48,56,91,74
26  492,55875,36,9,14,46,85,74,61
27  402,55179,32,9,23,46,27,70,77
28  492,55285,35,9,17,45,29,71,84
29  68,7653,74,9,3,46,97,98,97
30  579,65518,24,9,62,45,56,98,38
31  1437,162168,18,8,94,35,33,83,19
32  86,9887,11,10,82,36,59,79,42
33  190,28869,81,9,88,17,63,75,95
34  912,103284,21,8,86,31,79,89,68
35  445,48770,61,8,1,36,79,78
36  858,96269,64,9,89,39,6,96,25
37  1449,163294,26,8,41,36,81,82,77
38  852,96266,25,8,9,39,96,83,72
39  95,9886,9,8,8,45,97,98,98
```

```
numercial values 0
molecular weight 0
theoretical pI 0
instability index 0
aliphatic index 0
dtype: int64
Mantra 0 se encuentra en el clúster: 1
Mantra 1 se encuentra en el clúster: 0
Mantra 2 se encuentra en el clúster: 0
Mantra 3 se encuentra en el clúster: 0
PS C:\Users\VINNY\Desktop\Examen de Luna 3P>
```

Teniendo nuestro dataset sin etiquetas y sacando la información de los virus de las paginas que el profesor nos dio con la cual una pagina es para sacar su cadena del virus y la otra para obtener información acerca de ese virus las cuales vienen en la parte de bibliografías siendo los puntos número 4 y 5.

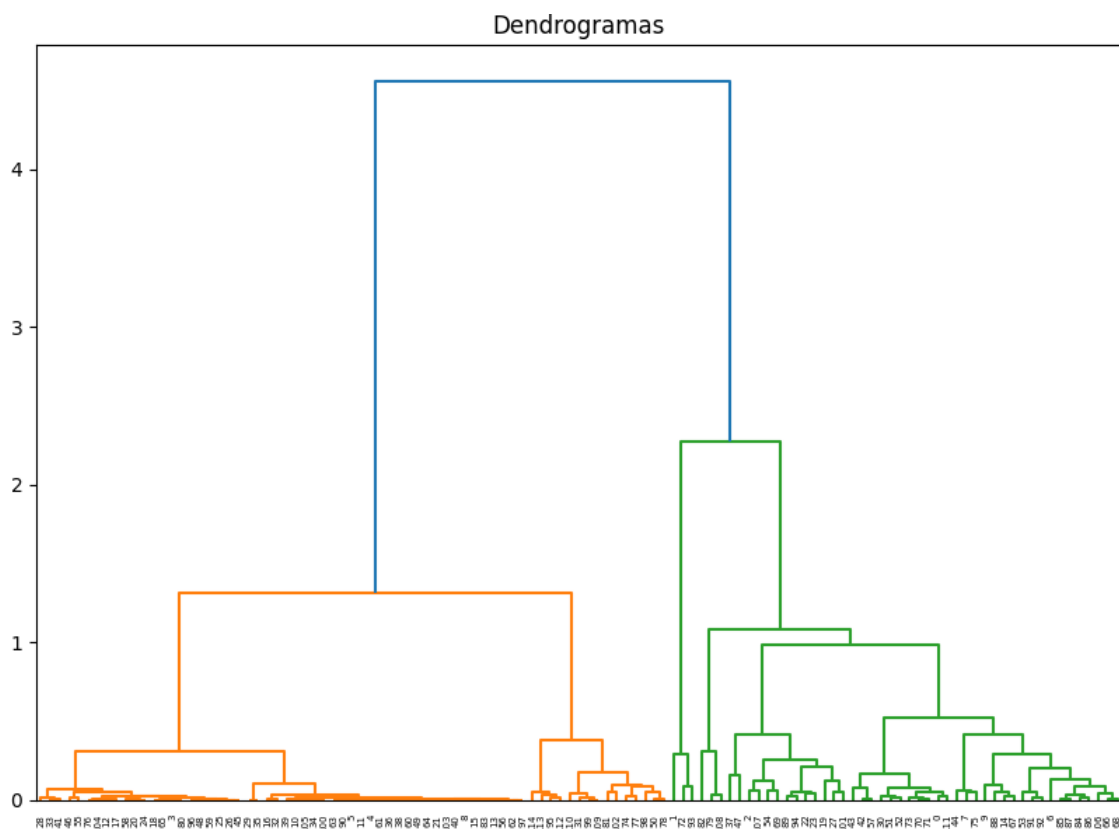
```
VIH Cluster.py X datasetVIH.csv
1
2 Algoritmo Agrupamiento Jerárquico
3
4 import pandas as pd
5 import matplotlib.pyplot as plt
6 ### CARGAR LOS DATOS ###
7 data = pd.read_csv('datasetVIH.csv')
8 ### ANALIZAR LOS DATOS ###
9 #Conocer la forma de los datos
10 print(data.shape)
11 #Conocer los datos nulos
12 print(data.isnull().sum())
13 #Conocer el formato de los datos
14 data.dtypes
15 #Se seleccionan unos datos al azar para posteriormente verificar el clúster
16 #al que pertenecen
17 indices = [1, 71, 75, 110]
18 muestras = pd.DataFrame(data.loc[indices],
19                          columns = data.keys()).reset_index(drop = True)
20
21 ### PROCESAMIENTO DE LOS DATOS ###
22 #Eliminamos las columnas de región y canal
23 data = data.drop(['molecularweight'], axis = 1)
24 muestras = muestras.drop(['molecularweight'], axis = 1)
25 #Se realiza el escalamiento de los datos
26 from sklearn import preprocessing
27 data_escalada = preprocessing.Normalizer().fit_transform(data)
28 muestras_escalada = preprocessing.Normalizer().fit_transform(muestras)
29
30 ### ANÁLISIS DE MACHINE LEARNING ###
31 #Se determina las variables a evaluar
32 X = data_escalada
33 #Se gráfica el dendrograma para obtener el número de clúster
34 import scipy.cluster.hierarchy as shc
35 plt.figure(figsize=(10, 7))
36 plt.title("Dendrogramas")
37 dendrograma = shc.dendrogram(shc.linkage(X, method = 'ward'))
38 #Obtenido el número de clúster se procede a definir los clústeres
39 from sklearn.cluster import AgglomerativeClustering
40 #Se define el algoritmo junto con el valor de K
41 algoritmo = AgglomerativeClustering(n_clusters = 2,
42                                     affinity='euclidean', linkage='ward')
43 #Se entrena el algoritmo
44 algoritmo.fit(X)
45 pred1 = algoritmo.fit_predict(X)
46 #Utilicemos los datos de muestras y verifiquemos en que cluster se encuentran
47 muestra_prediccion = algoritmo.fit_predict(muestras_escalada)
48 for i, pred in enumerate(muestra_prediccion):
49     print( "Muestra", i, "se encuentra en el clúster:", pred)
50 plt.show()
```

Aquí en esta parte del código es donde se carga el dataset y se muestran algunos datos de este como con que tipo estamos trabajando de variable, damos a conocer los datos nulos, la forma de los datos entre otras cosas.

Al igual seleccionamos unos datos al azar para que nuestro algoritmo trabaje en cuanto a ellos. Posterior a esto realizamos un escalamiento de datos que lo que hace es que los vamos a procesar para que estos estén de la mejor manera posible.

```
#Se determina las variables a evaluar
X = data_escalada
#Se gráfica el dendrograma para obtener el número de clúster
import scipy.cluster.hierarchy as shc
plt.figure(figsize=(10, 7))
plt.title("Dendrogramas")
dendrograma = shc.dendrogram(shc.linkage(X, method = 'ward'))
#Obtenido el número de clúster se procede a definir los clústeres
from sklearn.cluster import AgglomerativeClustering
#Se define el algoritmo junto con el valor de K
algoritmo = AgglomerativeClustering(n_clusters = 2,
                                     affinity='euclidean', linkage='ward')
#Se entrena el algoritmo
algoritmo.fit(X)
pred1 = algoritmo.fit_predict(X)
#Utilicemos los datos de muestras y verifiquemos en que cluster se encuentran
muestra_prediccion = algoritmo.fit_predict(muestras_escalada)
for i, pred in enumerate(muestra_prediccion):
    print( "Muestra", i, "se encuentra en el clúster:", pred)
plt.show()
```

Colocamos el dendrograma donde nos mostrará los clusters donde fueron agrupados los datos (en este caso se usaron 2 clusters).



```
numeroaminoacidos    0
molecularweight       0
theoreticalpI         0
instabilityindex      0
aliphaticindex        0
dtype: int64
Muestra 0 se encuentra en el clúster: 1
Muestra 1 se encuentra en el clúster: 0
Muestra 2 se encuentra en el clúster: 0
Muestra 3 se encuentra en el clúster: 0
```

Estos vienen siendo los resultados de nuestro algoritmo de aprendizaje no supervisado.

Aprendizaje Supervisado.

Podría definirse como un tipo de aprendizaje en IA en el que un algoritmo es entrenado con variables que incluyen los valores que queremos predecir; a estos valores conocidos se les llama "etiquetas" y se usan también para la evaluación del modelo. El aprendizaje supervisado se puede subdividir en dos tipos: clasificación y regresión.

En cuanto a clasificación, el objetivo es predecir las etiquetas de clase categóricas de nuevos registros, con base en observaciones pasadas. Dependiendo de la etiqueta, se puede decir que la clasificación es binaria o multiclase. Cuando solo existen dos clases (es decir, la etiqueta es discreta) se trata de clasificación binaria, y si existen más de dos clases entonces es clasificación multiclase.

Respecto a regresión, se trata del proceso estadístico predictivo en el que el modelo intenta predecir un valor continuo (como ventas, precio, calificaciones) mediante la relación entre variables dependientes e independientes. Es decir, se encuentra una ecuación en la que se sustituyen los valores de las variables y como resultado se obtiene el valor a predecir.

➤ Algoritmo: Gaussian Naive Bayes.

Un clasificador de Naive Bayes asume que la presencia o ausencia de una característica particular no está relacionada con la presencia o ausencia de cualquier otra característica, dada la clase variable. Por ejemplo, una fruta puede ser considerada como una manzana si es roja, redonda y de alrededor de 7 cm de diámetro. Un clasificador de Naive Bayes considera que cada una de estas características contribuye de manera independiente a la probabilidad de que esta fruta sea una manzana, independientemente de la presencia o ausencia de las otras características.

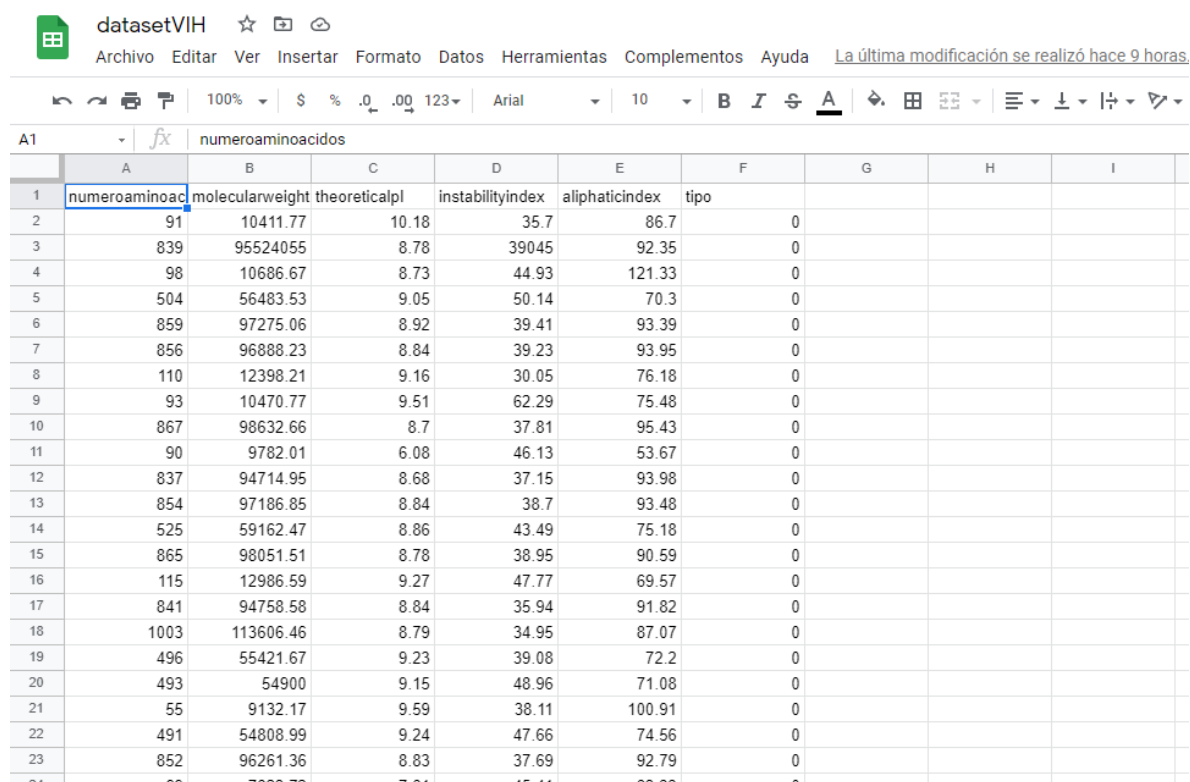
Para otros modelos de probabilidad, los clasificadores de Naive Bayes se pueden entrenar de manera muy eficiente en un entorno de aprendizaje supervisado. En muchas aplicaciones prácticas, la estimación de parámetros para los modelos Naive Bayes utiliza el método de máxima verosimilitud, en otras palabras, se puede trabajar con el modelo de Naive Bayes sin aceptar probabilidad bayesiana o cualquiera de los métodos bayesianos.

Una ventaja del clasificador de Naive Bayes es que solo se requiere una pequeña cantidad de datos de entrenamiento para estimar los parámetros (las medias y las varianzas de las variables) necesarias para la clasificación. Como las variables independientes se asumen, solo es necesario determinar las varianzas de las variables de cada clase y no toda la matriz de covarianza.

➤ Explicación:

Para el aprendizaje supervisado lo hicimos en Google colab la cual es una herramienta de la empresa de Google que nos permite usar una computadora remota en la cual es más rápida ya que tiene una gran capacidad para correr este tipo de algoritmos.

En este caso el dataset se realizo de la misma manera que el anterior dataset, pero en este hay un ligero cambio y es que si etiquetemos los datos (poner a cuál virus pertenece) los tipos 0 pertenecen al R5 y los tipos 1 son del X4 dado a esto mostraremos una captura de pantalla del dataset usado para este algoritmo(el algoritmo usado fue el de Gaussian NB):



| | A | B | C | D | E | F | G | H | I |
|----|-------------------|-----------------|---------------|------------------|----------------|------|---|---|---|
| | numeroaminoacidos | molecularweight | theoreticalpl | instabilityindex | aliphaticindex | tipo | | | |
| 1 | 91 | 10411.77 | 10.18 | 35.7 | 86.7 | 0 | | | |
| 2 | 839 | 95524055 | 8.78 | 39045 | 92.35 | 0 | | | |
| 3 | 98 | 10686.67 | 8.73 | 44.93 | 121.33 | 0 | | | |
| 4 | 504 | 56483.53 | 9.05 | 50.14 | 70.3 | 0 | | | |
| 5 | 859 | 97275.06 | 8.92 | 39.41 | 93.39 | 0 | | | |
| 6 | 856 | 96888.23 | 8.84 | 39.23 | 93.95 | 0 | | | |
| 7 | 110 | 12398.21 | 9.16 | 30.05 | 76.18 | 0 | | | |
| 8 | 93 | 10470.77 | 9.51 | 62.29 | 75.48 | 0 | | | |
| 9 | 867 | 98632.66 | 8.7 | 37.81 | 95.43 | 0 | | | |
| 10 | 90 | 9782.01 | 6.08 | 46.13 | 53.67 | 0 | | | |
| 11 | 837 | 94714.95 | 8.68 | 37.15 | 93.98 | 0 | | | |
| 12 | 854 | 97186.85 | 8.84 | 38.7 | 93.48 | 0 | | | |
| 13 | 525 | 59162.47 | 8.86 | 43.49 | 75.18 | 0 | | | |
| 14 | 865 | 98051.51 | 8.78 | 38.95 | 90.59 | 0 | | | |
| 15 | 115 | 12986.59 | 9.27 | 47.77 | 69.57 | 0 | | | |
| 16 | 841 | 94758.58 | 8.84 | 35.94 | 91.82 | 0 | | | |
| 17 | 1003 | 113606.46 | 8.79 | 34.95 | 87.07 | 0 | | | |
| 18 | 496 | 55421.67 | 9.23 | 39.08 | 72.2 | 0 | | | |
| 19 | 493 | 54900 | 9.15 | 48.96 | 71.08 | 0 | | | |
| 20 | 55 | 9132.17 | 9.59 | 38.11 | 100.91 | 0 | | | |
| 21 | 491 | 54808.99 | 9.24 | 47.66 | 74.56 | 0 | | | |
| 22 | 852 | 96261.36 | 8.83 | 37.69 | 92.79 | 0 | | | |
| 23 | 60 | 7022.72 | 7.84 | 45.44 | 82.22 | 0 | | | |
| 24 | 60 | 7022.72 | 7.84 | 45.44 | 82.22 | 0 | | | |

Cabe destacar que una de las condiciones para usar Google colab es que con los archivos que estemos trabajando tienen que estar en Google drive con la cuenta que estas usando el Google colab.

```
Copia_de_VIH.ipynb
Archivo Editar Ver Insertar Entorno de ejecución Herramientas Ayuda Se guardaron todos los cambios

+ Código + Texto

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import colors
import seaborn as sb
import glob
import keras as k

%matplotlib inline
plt.rcParams['figure.figsize'] = (16, 9)
plt.style.use('ggplot')

from sklearn.preprocessing import LabelEncoder, MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.naive_bayes import GaussianNB
from sklearn.feature_selection import SelectKBest
from keras.layers import Dense
from keras.models import Sequential, load_model
from sklearn.metrics import plot_confusion_matrix

[2] #Importante ejecutar esto para poder hacer uso de tf en colab
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()
```

Cargamos las librerías que van a hacer usadas en nuestro algoritmo.

```
[3] from google.colab import drive
drive.mount('/content/drive')
```

Mounted at /content/drive

```
df = pd.read_csv(r"/content/drive/My Drive/datasetVIH.csv")
df.describe()
```

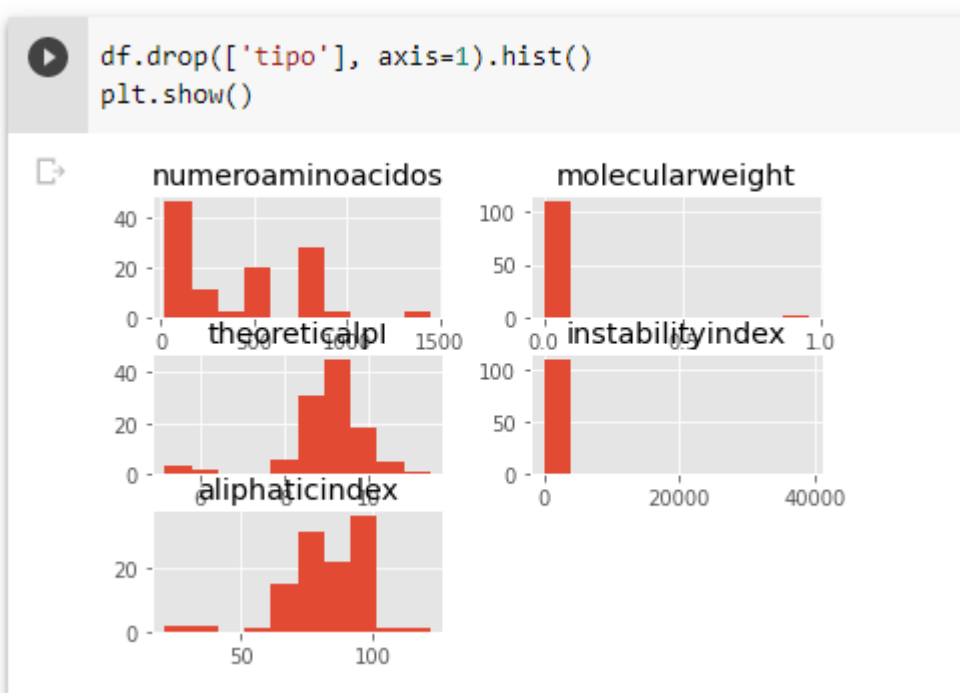
| | numeroaminoacidos | molecularweight | theoreticalpI | instabilityindex | aliphaticindex | tipo |
|-------|-------------------|-----------------|---------------|------------------|----------------|------------|
| count | 111.000000 | 1.110000e+02 | 111.000000 | 111.000000 | 111.000000 | 111.000000 |
| mean | 411.540541 | 9.058096e+05 | 9.024775 | 400.264234 | 82.031261 | 0.369369 |
| std | 350.216447 | 9.062494e+06 | 0.960468 | 3701.745914 | 14.854156 | 0.484823 |
| min | 22.000000 | 1.926150e+03 | 5.170000 | 24.070000 | 21.670000 | 0.000000 |
| 25% | 95.500000 | 1.092315e+04 | 8.780000 | 37.850000 | 74.585000 | 0.000000 |
| 50% | 217.000000 | 2.820407e+04 | 9.090000 | 41.740000 | 83.190000 | 0.000000 |
| 75% | 839.000000 | 9.473676e+04 | 9.520000 | 47.580000 | 92.570000 | 1.000000 |
| max | 1449.000000 | 9.552406e+07 | 11.450000 | 39045.000000 | 121.330000 | 1.000000 |

Le damos permiso a Google colab que acceda a nuestra nube para obtener el dataset y nos muestra una descripción básica acerca de los datos con los que estaremos trabajando.

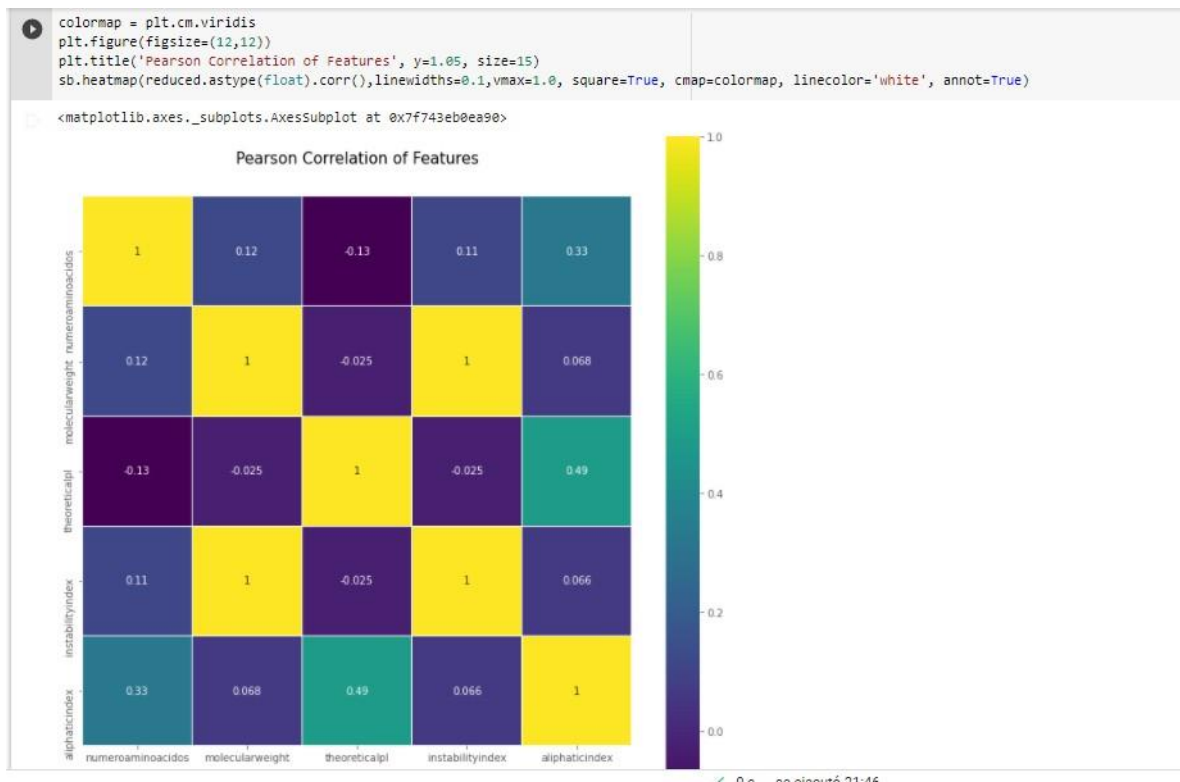

```
df.head(30)
```

| | numeroaminoacidos | molecularweight | theoreticalpI | instabilityindex | aliphaticindex | tipo |
|----|-------------------|-----------------|---------------|------------------|----------------|------|
| 0 | 91 | 10411.77 | 10.18 | 35.70 | 86.70 | 0 |
| 1 | 839 | 95524055.00 | 8.78 | 39045.00 | 92.35 | 0 |
| 2 | 98 | 10686.67 | 8.73 | 44.93 | 121.33 | 0 |
| 3 | 504 | 56483.53 | 9.05 | 50.14 | 70.30 | 0 |
| 4 | 859 | 97275.06 | 8.92 | 39.41 | 93.39 | 0 |
| 5 | 856 | 96888.23 | 8.84 | 39.23 | 93.95 | 0 |
| 6 | 110 | 12398.21 | 9.16 | 30.05 | 76.18 | 0 |
| 7 | 93 | 10470.77 | 9.51 | 62.29 | 75.48 | 0 |
| 8 | 867 | 98632.66 | 8.70 | 37.81 | 95.43 | 0 |
| 9 | 90 | 9782.01 | 6.08 | 46.13 | 53.67 | 0 |
| 10 | 837 | 94714.95 | 8.68 | 37.15 | 93.98 | 0 |
| 11 | 854 | 97186.85 | 8.84 | 38.70 | 93.48 | 0 |

Se nos muestra los primeros 30 datos.



Unas graficas tipo histograma acerca de los datos con los que se están trabajando.



Aquí se puede observar una correlación entre los datos que obtenemos esto se hace para saber qué datos son los que usaremos para el algoritmo para entrenarlo ya que algunos datos se puede obtener mejor información que otros.

```

X=df.drop(['tipo'], axis=1)
y=df['tipo']

best=SelectKBest(k=2)
X_new = best.fit_transform(X, y)
X_new.shape
selected = best.get_support(indices=True)
print(X.columns[selected])

```

```

Index(['numeroaminoacidos', 'aliphaticindex'], dtype='object')

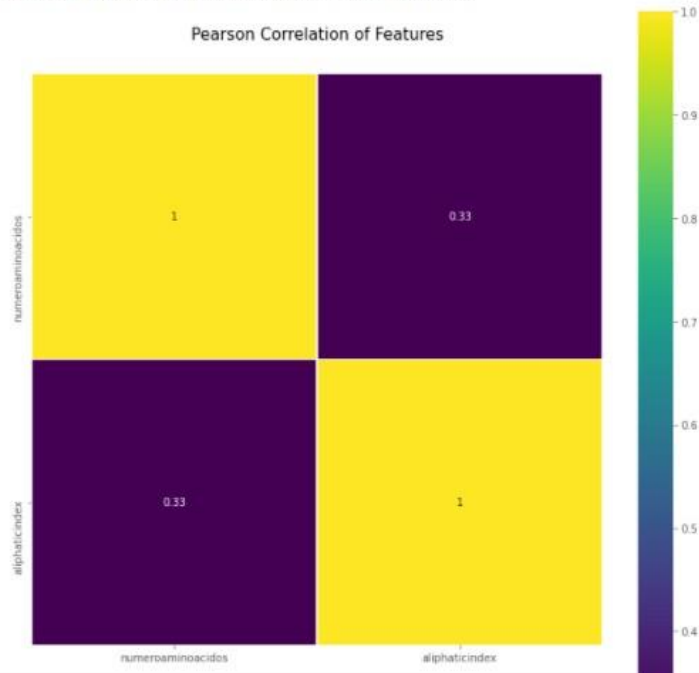
```

Usaremos los datos numeroaminoacidos y aliphaticindex.

```
used_features = x.columns[selected]

colormap = plt.cm.viridis
plt.figure(figsize=(12,12))
plt.title('Pearson Correlation of Features', y=1.05, size=15)
sb.heatmap(df[used_features].astype(float).corr(),linewidths=0.1,vmax=1.0, square=True, cmap=colormap, linecolor='white', annot=True)
```

<matplotlib.axes._subplots.AxesSubplot at 0x7f744730d1d0>



La correlación entre solamente esos dos datos.

```
[34] # Definimos nuestro entrenamiento y nuestro test.
      X_train, X_test = train_test_split(df, test_size=0.33, random_state=67)
      y_train = X_train["tipo"]
      y_test = X_test["tipo"]
```

```
[35] # Instantiate the classifier
      gnb = GaussianNB()
      # Train classifier
      gnb.fit(
          X_train[used_features].values,
          y_train
      )
      y_pred = gnb.predict(X_test[used_features])

      print('Precisión en el set de Entrenamiento: {:.2f}'
            .format(gnb.score(X_train[used_features], y_train)))
      print('Precisión en el set de Test: {:.2f}'
            .format(gnb.score(X_test[used_features], y_test)))
```

```
Precisión en el set de Entrenamiento: 0.64
Precisión en el set de Test: 0.65
```

```
▶ from sklearn.metrics import confusion_matrix
   from sklearn.svm import SVC
   confusion_matrix(y_test,y_pred)
```

```
array([[22,  3],
       [10,  2]])
```

Entrenamos el modelo nos muestra su precisión tanto en el entrenamiento como en la prueba. Para poder mejorar esta precisión hace falta agregar más datos y posiblemente mas variables para poder hacerlo de una mejor manera.

Y nos muestra su matriz de confusión.

Explicación del Análisis Comparativo de Modelos de Aprendizaje Supervisado y No Supervisado.

Después de realizar la practica podemos comparar nuestros resultados.

En primer lugar con el algoritmo el clúster jerárquico separa en dos agrupamientos:

```
numeroaminoacidos    0
molecularweight       0
theoreticalpI         0
instabilityindex      0
aliphaticindex        0
dtype: int64
Muestra 0 se encuentra en el clúster: 1
Muestra 1 se encuentra en el clúster: 0
Muestra 2 se encuentra en el clúster: 0
Muestra 3 se encuentra en el clúster: 0
```

Tenemos el cluster '0' y el cluster '1' donde se agrupan las cuatro muestras.

Esto corresponde a los resultados del aprendizaje no supervisado.

Vemos que la tendencia en las muestras fue agruparse en el clúster cero.

Por otro lado, la matriz de confusión del aprendizaje supervisado arrojo los siguientes resultados:

| | R5 | X4 |
|----|----|----|
| R5 | 21 | 9 |
| X4 | 10 | 2 |

En el aprendizaje no supervisado, como trabajamos con agrupamiento jerárquico aquí no tenemos etiquetas, lo único que podemos ver es que tienden a irse al cluster '0' que corresponde al X4. Por otro lado en el Gaussiano si tenemos la matriz de confusión para comparar. Podemos ver que adivino correctamente 21 de R5 y erro en 9. Luego atino en X4 en 10, y solo erro en 2.

Fuentes de Bibliografía.

- [1] D. (2019, 14 julio). *Clustering Jerárquico - Agrupar elementos con minería de datos*. ESTRATEGIAS DE TRADING. <https://estrategiastrading.com/clustering-jerarquico/>
- [2] D. (2019, 14 julio). *Clustering Jerárquico - Agrupar elementos con minería de datos*. ESTRATEGIAS DE TRADING. <https://estrategiastrading.com/clustering-jerarquico/>
- [3] Webb, G. I.; Boughton, J.; Wang, Z. (2005). "Not So Naive Bayes: Aggregating One-Dependence Estimators". *Machine Learning*. 58 (1): 5–24.
doi:10.1007/s10994-005-4258-6.
- [4] NCBI - WWW Error Blocked Diagnostic. (2015). NCBI.
<https://misuse.ncbi.nlm.nih.gov/error/abuse.shtml>
- [5] ProtParam. (2021). ExPASy - ProtParam tool. <https://web.expasy.org/protparam/>