GUIÓN DE PRÁCTICAS DE REDES NEURONALES ARTIFICIALES (TSCAO): Parte I

MªCarmen Pegalajar Jiménez

09/01/2025

Dpto Ciencias de la Computación e Inteligencia Artificial
Universidad de Granada

INTRODUCCIÓN

Una de las bibliotecas de Python más potentes y fáciles de usar para desarrollar y evaluar modelos de aprendizaje profundo es Keras; En ella se encuentran las bibliotecas de computación numérica Theano y TensorFlow. La ventaja de esta es principalmente que podemos trabajar con redes neuronales artificiales de una manera fácil.

En esta práctica aprenderemos:

- Cómo usar Python y sus bibliotecas para comprender, explorar y visualizar sus datos,
- Cómo preprocesar sus datos: aprenderemos cómo dividir los datos en conjunto de entrenamiento y test y cómo normalizar los datos,
- Cómo construir perceptrones multicapa para tareas de clasificación,
- Cómo compilar y ajustar los datos a estos modelos,
- Cómo usar tu modelo para predecir valores objetivo, y
- Cómo validar los modelos que has construido.
- Por último, también verás cómo puedes construir un modelo para las tareas de regresión, y aprenderás cómo ajustar el modelo que has construido.

ANTES DE COMENZAR

Las librerías necesarias para crear nuestra red neuronal, si tenemos ya instalado Python en nuestro ordenador, serán fundamentalmente Keras y TensorFlow.

Para instalar Keras y TensorFlow sólo hace falta poner el siguiente comando en la consola:

conda install tensorflow

conda install keras

1. CLASIFICACIÓN DE VINOS

Para esta práctica, se utilizará el conjunto de datos de calidad del vino que puede encontrar en el conjunto de datos de calidad del vino en UCI. Idealmente, podríamos realizar un aprendizaje profundo en conjuntos de datos más grandes, pero para el propósito de esta práctica, se utilizará uno más pequeño. Esto se debe principalmente a que el objetivo es comenzar con la biblioteca y familiarizarse con el funcionamiento de las redes neuronales.

El archivo de descripción de datos enumera las 12 variables que se incluyen en los datos. A continuación, se expone una descripción breve de cada variable:

1. **Acidez fija:** los ácidos son las principales propiedades del vino y contribuyen en gran medida al sabor del vino. Por lo general, la acidez total se divide en dos grupos: los ácidos volátiles y los ácidos no volátiles o fijos. Entre los ácidos fijos que puedes encontrar en los vinos están los siguientes: tartárico, málico, cítrico y succínico.
2. **Acidez volátil**: la acidez volátil es básicamente el proceso de convertir el vino en vinagre. En los Estados Unidos, los límites legales de la acidez volátil son 1.2 g / L para el vino tinto de mesa y 1.1 g / L para el vino blanco de mesa.
3. **El ácido cítrico** es uno de los ácidos fijos que encontrarás en los vinos.
4. **El azúcar** residual generalmente se refiere al azúcar restante después de que la fermentación se detiene.
5. **Los cloruros** pueden ser un contribuyente significativo a la salinidad en el vino.
6. **Dióxido de azufre libre**: se dice que la parte del dióxido de azufre que se agrega a un vino y que se pierde en él está ligada, mientras que la parte activa se dice que está libre. El enólogo siempre intentará que la mayor proporción de azufre libre se una.
7. **El dióxido de azufre total** es la suma del dióxido de azufre unido y libre (SO2). Aquí, se expresa en mg /. Existen límites legales para los niveles de azufre en los vinos: en la UE, los vinos tintos solo pueden tener 160 mg / L, mientras que los vinos blancos y rosados pueden tener aproximadamente 210 mg / L. Los vinos dulces pueden tener 400 mg / L. Para los EE. UU., Los límites legales se establecen en 350mg / L, y para Australia
8. **La densidad** se usa generalmente como una medida de la conversión de azúcar a alcohol.
9. **El pH o el potencial del hidrógeno** es una escala numérica para especificar la acidez o basicidad del vino. Las soluciones con un pH inferior a 7 son ácidas, mientras que las soluciones con un pH superior a 7 son básicas. Con un pH de 7, el agua pura es neutra. La mayoría de los vinos tienen un pH entre 2.9 y 3.9 y, por lo tanto, son ácidos.
10. **Los sulfatos** son para el vino como el gluten es para la comida. Es posible que ya conozca los sulfitos por los dolores de cabeza que pueden causar. Son una parte habitual de la

vinificación en todo el mundo y se consideran necesarias.

11. **Alcohol:** el vino es una bebida alcohólica pero el porcentaje de alcohol puede variar de un vino a otro. Esta variable está incluida en el conjunto de datos, donde se expresa en % vol.
12. **Calidad**: los expertos en vinos calificaron la calidad del vino entre 0 (muy malo) y 10 (muy excelente). El número final es la mediana de al menos tres evaluaciones realizadas por esos mismos expertos en vinos.

Cargando los datos

Esto se puede hacer fácilmente con la biblioteca de manipulación de datos de Python *Pandas*. Para ello se importa el paquete bajo su alias, *pd*.

A continuación, utiliza la *read_csv()* función para leer en los archivos CSV en los que se almacenan los datos. Además, se usa el argumento *sep* para especificar que el separador, en este caso, es un punto y coma y no una coma.

```
# Import pandas
import pandas as pd

# Read in white wine data
white = pd.read_csv('winequality-white.csv', sep=';')

# Read in red wine data
red = pd.read_csv('winequality-red.csv', sep=';')
```

Exploración de datos

Una de las primeras cosas que probablemente quieras hacer es comenzar por obtener una vista rápida de ambos marcos de datos, y verificar si la carga de los datos tuvo éxito, para ello tenemos que verificar si los datos contienen todas las variables que en el archivo de descripción se mencionaba. Además del número de variables, también debemos verificar la calidad de la importación de datos: ¿son correctos los tipos de datos? ¿Están todas las filas? ¿Hay algún valor nulo que debe tener en cuenta al limpiar los datos?

```
# Print info on white wine
print(white.info())

# Print info on red wine
print(red.info())

# First rows of `red`
red.head()

# Last rows of `white`
white.tail()

# Take a sample of 5 rows of `red`
red.head(5)

# Describe `white`
white.describe()

# Double check for null values in `red`
pd.isnull(red)
```

Algunas funciones de pandas como: *head()*, *tail()* y *sample()* son geniales porque brindan una forma rápida de inspeccionar los datos.

A continuación, *describe()* ofrece algunas estadísticas resumidas sobre los datos que pueden ayudar a evaluar la calidad de los datos. Puedes ver que algunas de las variables tienen una gran diferencia en sus valores máximos y mínimos.

Por último, se puede verificar la presencia de valores nulos con la ayuda de *isnull()*. Esta es una función que siempre puede ser útil cuando todavía se tiene dudas después de haber leído los resultados *info()*.

Consejo : también verificar si los datos del vino contienen valores nulos.

Ahora si ya se ha inspeccionado los datos para ver si la importación tuvo éxito y es correcta, es hora de profundizar un poco más.

Visualizando los datos

Una forma de hacer esto es mirar la distribución de algunas de las variables del conjunto de datos y hacer diagramas de dispersión para ver posibles correlaciones.

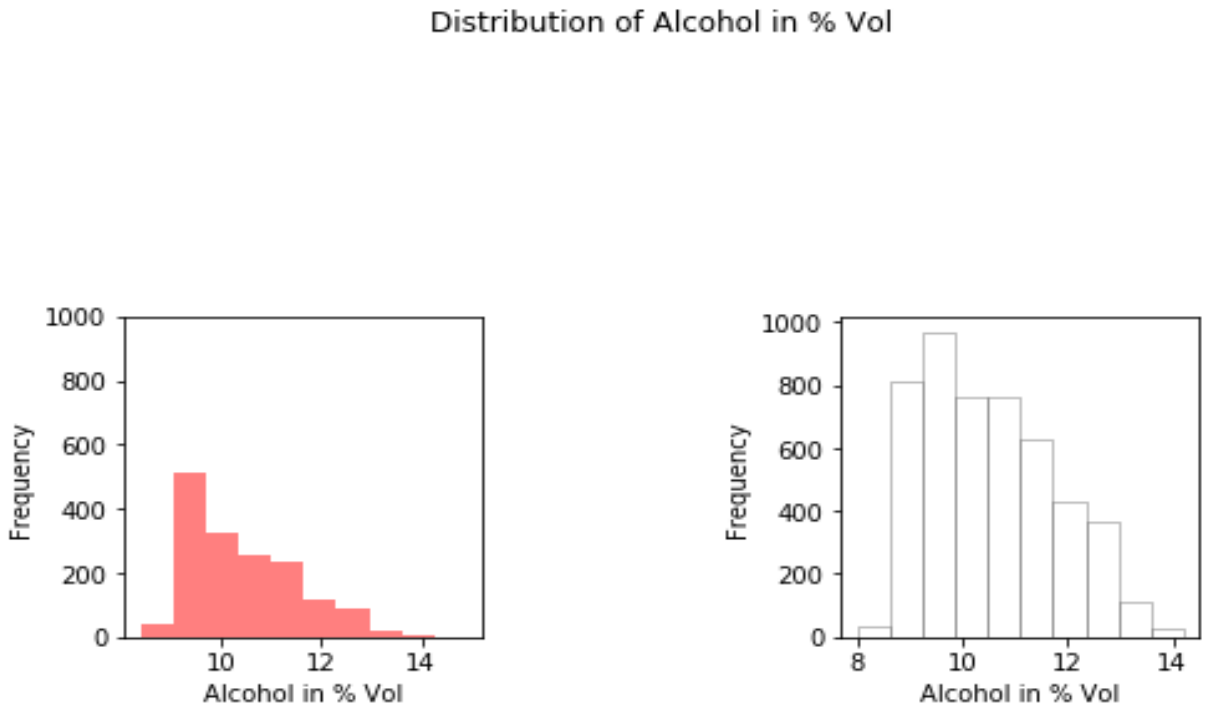
Alcohol

Una variable que puedes encontrar interesante a primera vista es *alcohol*. Probablemente sea una de las primeras variables que llame tu atención cuando inspecciones el conjunto de datos de vinos. Puede visualizar las distribuciones con cualquier biblioteca de visualización de datos, pero en este caso, vamos a utilizar *matplotlib* para trazar las distribuciones rápidamente:

```
import matplotlib.pyplot as plt

fig, ax = plt.subplots(1, 2)

ax[0].hist(red.alcohol, 10, facecolor='red', alpha=0.5, label="Red wine")
ax[1].hist(white.alcohol, 10, facecolor='white', ec="black", lw=0.5, alpha=0.5, label="White wine")

fig.subplots_adjust(left=0, right=1, bottom=0, top=0.5, hspace=0.05, wspace=1)
ax[0].set_ylim([0, 1000])
ax[0].set_xlabel("Alcohol in % Vol")
ax[0].set_ylabel("Frequency")
ax[1].set_xlabel("Alcohol in % Vol")
ax[1].set_ylabel("Frequency")
#ax[0].legend(loc='best')
#ax[1].legend(loc='best')
fig.suptitle("Distribution of Alcohol in % Vol")

plt.show()
import numpy as np
print(np.histogram(red.alcohol, bins=[7,8,9,10,11,12,13,14,15]))
print(np.histogram(white.alcohol, bins=[7,8,9,10,11,12,13,14,15]))
```

Como puede verse en la imagen que aparece a continuación, se observa que los niveles de alcohol entre el vino tinto y el blanco son básicamente los mismos: tienen alrededor del 9% de alcohol. Por supuesto, también hay una cantidad considerable de observaciones que tienen un 10% o 11% de porcentaje de alcohol.

Hay que tener en cuenta que puede verificarse esto dos veces si usa la función *histogram()* del paquete *numpy* para calcular el histograma de los datos asociado al vino blanco y tinto (rojo), de la siguiente manera:

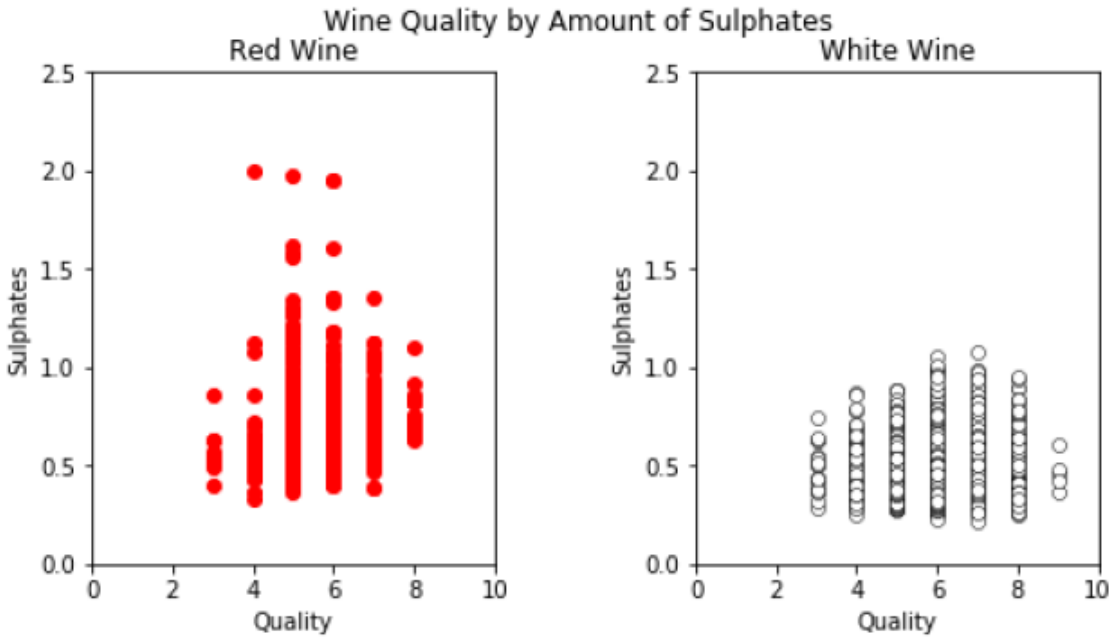


Sulfatos

A continuación, una cosa que puede interesar es la relación entre los sulfatos y la calidad del vino. Como se ha dicho anteriormente los sulfatos pueden causar dolores de cabeza a las personas, y puede ser interesante saber si esto influye en la calidad del vino. Tal vez esto afecta las calificaciones para el vino tinto?

```
#SULFATOS
import matplotlib.pyplot as plt

fig, ax = plt.subplots(1, 2, figsize=(8, 4))

ax[0].scatter(red['quality'], red["sulphates"], color="red")
ax[1].scatter(white['quality'], white['sulphates'], color="white", edgecolors="black", lw=0.5)

ax[0].set_title("Red Wine")
ax[1].set_title("White Wine")
ax[0].set_xlabel("Quality")
ax[1].set_xlabel("Quality")
ax[0].set_ylabel("Sulphates")
ax[1].set_ylabel("Sulphates")
ax[0].set_xlim([0,10])
ax[1].set_xlim([0,10])
ax[0].set_ylim([0,2.5])
ax[1].set_ylim([0,2.5])
fig.subplots_adjust(wspace=0.5)
fig.suptitle("Wine Quality by Amount of Sulphates")

plt.show()
```

Como se puede ver en la imagen a continuación, el vino tinto parece contener más sulfatos que el vino blanco, que tiene menos sulfatos por encima de 1 g / . Para el vino blanco, solo parece haber un par de excepciones que caen justo por encima de 1 g / , mientras que esto definitivamente es más para los vinos tintos. Esto podría explicar el dicho general de que el vino tinto causa dolores de cabeza, pero ¿qué pasa con la calidad?



Puede verse claramente que hay vino blanco con una cantidad relativamente baja de sulfatos que obtiene una puntuación de 9, pero por lo demás, es difícil interpretar los datos correctamente en este momento.

Por supuesto, debe tener en cuenta que la diferencia en las observaciones también podría afectar los gráficos y cómo podría interpretarlos.

Acidez

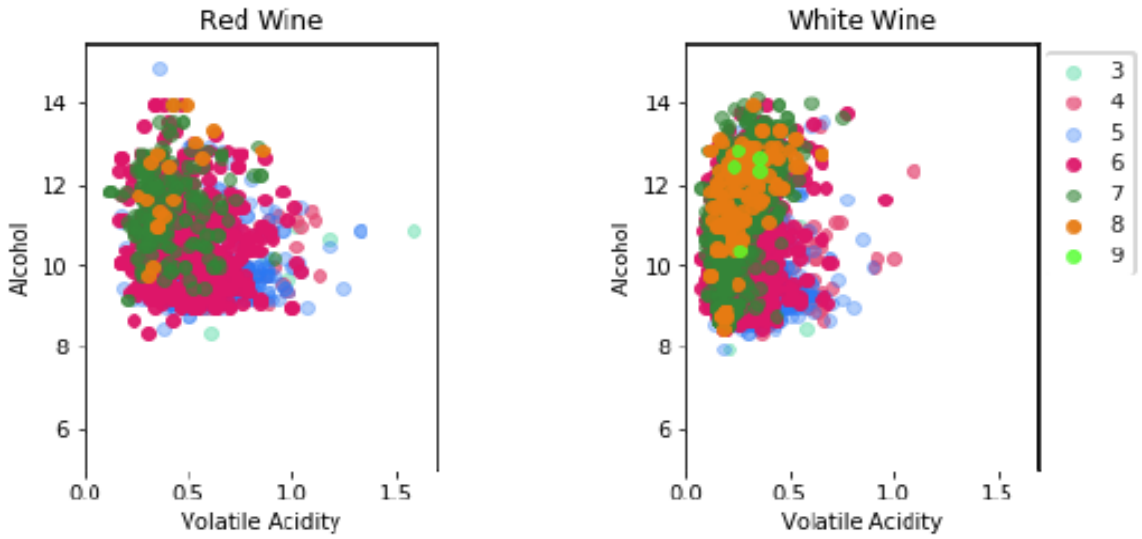
Además de los sulfatos, la acidez es una de las principales características vitales del vino necesarias para lograr vinos de calidad. Los grandes vinos a menudo equilibran la acidez, el tanino, el alcohol y la dulzura. Algunas observaciones indican que en cantidades de 0.2 a 0.4 g / L, la acidez volátil no afecta la calidad de un vino. Sin embargo, a niveles más altos, la acidez volátil puede dar al vino una sensación táctil aguda y vinagre. La acidez extremadamente volátil significa un vino muy defectuoso.

Pongamos a prueba los datos y hagamos un diagrama de dispersión que represente el alcohol versus la acidez volátil. Los puntos asociados a los datos deben colorearse de acuerdo con su clasificación:

```
#ACIDEZ

import matplotlib.pyplot as plt
import numpy as np

np.random.seed(570)

redlabels = np.unique(red['quality'])
whitelabels = np.unique(white['quality'])

import matplotlib.pyplot as plt
fig, ax = plt.subplots(1, 2, figsize=(8, 4))
redcolors = np.random.rand(6,4)
whitecolors = np.append(redcolors, np.random.rand(1,4), axis=0)

for i in range(len(redcolors)):
    redy = red['alcohol'][red.quality == redlabels[i]]
    redx = red['volatile acidity'][red.quality == redlabels[i]]
    ax[0].scatter(redx, redy, c=redcolors[i])
for i in range(len(whitecolors)):
    whitey = white['alcohol'][white.quality == whitelabels[i]]
    whitex = white['volatile acidity'][white.quality == whitelabels[i]]
    ax[1].scatter(whitex, whitey, c=whitecolors[i])

ax[0].set_title("Red Wine")
ax[1].set_title("White Wine")
ax[0].set_xlim([0,1.7])
ax[1].set_xlim([0,1.7])
ax[0].set_ylim([5,15.5])
ax[1].set_ylim([5,15.5])
ax[0].set_xlabel("Volatile Acidity")
ax[0].set_ylabel("Alcohol")
ax[1].set_xlabel("Volatile Acidity")
ax[1].set_ylabel("Alcohol")
#ax[0].legend(redlabels, loc='best', bbox_to_anchor=(1.3, 1))
ax[1].legend(whitelabels, loc='best', bbox_to_anchor=(1.3, 1))
#fig.suptitle("Alcohol - Volatile Acidity")
fig.subplots_adjust(top=0.85, wspace=0.7)

plt.show()
```

Hay que tener en cuenta que los colores de esta imagen se eligen aleatoriamente con la ayuda del módulo de Numpy *random*. Siempre se puede cambiar ésto, pasando una lista a las variables redcolors o whitecolors. Sin embargo, asegurate de que sean iguales (excepto en 1 porque los datos del vino blanco tienen únicamente un valor *quality* más que los datos del vino tinto), de lo contrario, sus leyendas no coincidirán.

Mira el gráfico completo aquí:



En la imagen de arriba, puede verse por ejemplo que: la mayoría de los vinos con etiqueta 8 tienen niveles de acidez volátiles de 0.5 o menos, pero si tiene o no un efecto en la calidad también es difícil de decir, ya que todos los puntos de datos están muy densamente agrupados hacia un lado del gráfico.

Hasta ahora, se ha examinado los datos del vino blanco y del vino tinto por separado. Las dos parecen diferir un poco cuando observas algunas de las variables de cerca, y en otros casos, las dos parecen ser muy similares. ¿Crees que podría haber una manera de clasificar las entradas en función de sus variables en vino blanco o tinto?

Preprocesamiento de Datos

¡Ahora que hemos explorado los datos, es hora de actuar sobre las ideas que han podido surgir y resultados obtenidos! Comencemos a preprocesar los datos y comencemos a construir la red neuronal !

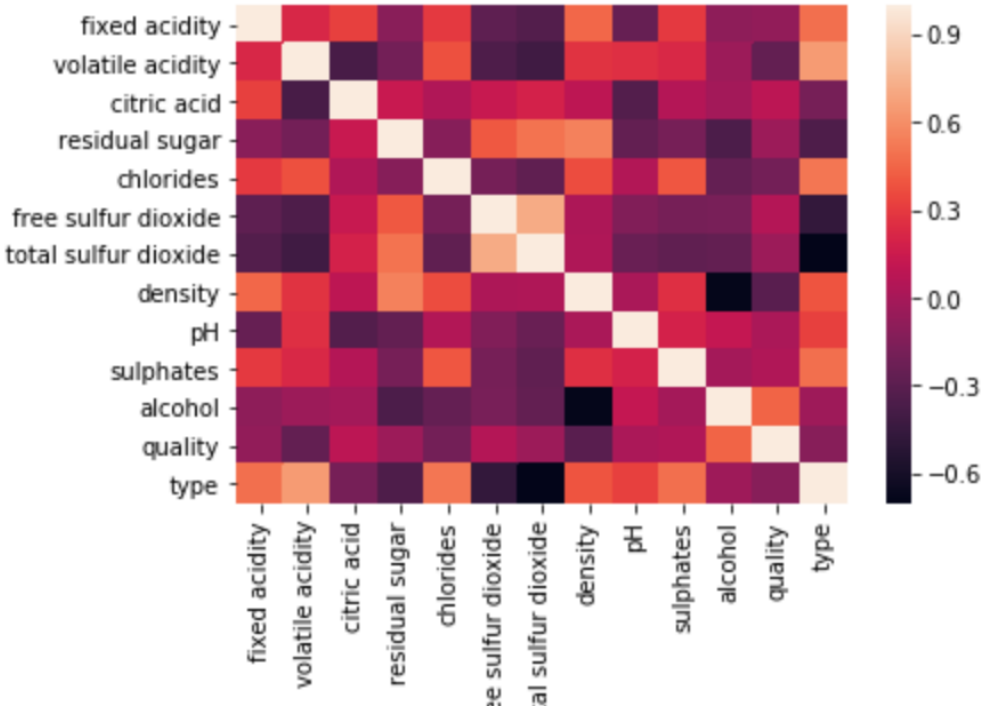
```
# Add `type` column to `red` with value 1
red['type'] = 1

# Add `type` column to `white` with value 0
white['type'] = 0

# Append `white` to `red`
wines = red.append(white, ignore_index=True)

import seaborn as sns

corr = wines.corr()
sns.heatmap(corr,
            xticklabels=corr.columns.values,
            yticklabels=corr.columns.values)
#sns.plt.show()

# Import `train_test_split` from `sklearn.model_selection`
```

Se establece *ignore_index* a *True* en este caso ya no se desea mantener las etiquetas de índice de *white* cuando se está añadiendo los datos al conjunto *red*: lo que se pretende es que las etiquetas continúen desde donde se dejaron en *red*, y las etiquetas del índice que no están duplicados en ambos conjuntos unirse en uno nuevo.

Matriz de Correlación

Ahora que tenemos el conjunto de datos completo, es una buena idea también hacer una exploración rápida de datos; De esta manera recopilaremos aún más información sobre el conjunto de datos.

Dado que puede ser algo difícil interpretar gráficos, también es una buena idea trazar una matriz de correlación. Esto proporcionará información de manera más rápida sobre qué variables se correlacionan:



Como era de esperar, hay algunas variables que se correlacionan, como *density* y *residual sugar*. También *volatile acidity*, y *type* están más estrechamente conectadas de lo que inicialmente podríamos habernos imaginado mirando a los dos conjuntos de datos por separado, y era esperable que *free sulfur dioxide* y *total sulfur dioxide* iban a estar correlacionadas.

Conjuntos de Entrenamiento y Test

Cuando se habla de **datos desequilibrados** en un problema de clasificación normalmente se refiere a un problema, donde las clases no están representadas igualmente, es decir no tienen exactamente el mismo número de casos en cada clase, pero una pequeña diferencia a menudo no es importante. Por tanto, es necesario asegurarse que las dos clases de vino están presentes en el modelo que estamos construyendo. Además, la cantidad de instancias de los dos tipos de vino debe ser más o menos igual para que no favorezca a una u otra clase en sus predicciones.

En este caso, parece haber un desequilibrio, pero esto no será un problema en este caso. Posteriormente, al evaluar el modelo, si tiene un rendimiento inferior, se puede recurrir al submuestreo o al sobremuestreo para ocultar la diferencia en las observaciones.

Por ahora, importe el `train_test_split` de `sklearn.model_selection` para asignar los datos y las

etiquetas de destino para las variables *X* e *y* . Verás que necesitas “aplanar” la gama de etiquetas de destino con el fin de estar totalmente preparado para utilizar la *X* y la *y* como variables de entrada para la función *train_test_split()*.

```
from sklearn.model_selection import train_test_split

# Specify the data
X=wines.iloc[:,0:11]

# Specify the target labels and flatten the array
y=np.ravel(wines.type)

# Split the data up in train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)
```

Ya estamos preparados para construir nuestra primera red neuronal, pero todavía hay una cosa con la que tenemos que tener cuidado ¿Recuerdas lo que viste al mirar los resúmenes de los conjuntos de datos *white* y *red?..* Algunos de los valores estaban bastante separados o muy distintos. Puede tener sentido hacer alguna normalización aquí.

Estandarizar los datos: Normalización

La normalización (estardarización) es una forma de lidiar con estos valores que están tan separados. El paquete scikit-learn le ofrece una forma excelente y rápida de estandarizar sus datos: ¡importe el módulo *StandardScaler* y *sklearn.preprocessing* y estarás listo para escalar su conjunto de entrenamiento y test!

```
# Import `StandardScaler` from `sklearn.preprocessing`
from sklearn.preprocessing import StandardScaler

# Define the scaler
scaler = StandardScaler().fit(X_train)

# Scale the train set
X_train = scaler.transform(X_train)

# Scale the test set
X_test = scaler.transform(X_test)
```

Ahora que los datos están preprocesados, podemos pasar al trabajo real: construir tu propia red neuronal para clasificar vinos.

Datos del modelo

Antes de comenzar a construir el modelo, volvamos a nuestra pregunta original: ¿se puede predecir si un vino es rojo o blanco al observar sus propiedades químicas, como la acidez volátil o los sulfatos?

Como únicamente tenemos dos clases: blanco y rojo, vamos a hacer **una clasificación binaria**. Codificaremos rojo como 1 y blanco como 0.

Un tipo de red que funciona bien en tal problema es un **Perceptrón Multicapa**. Este tipo de red neuronal es completamente conectada. Eso significa que se busca construir una serie de capas completamente conectadas para resolver este problema. En cuanto a la función de activación que se usará, es mejor usar una de las más comunes aquí con el fin de familiarizarse con Keras y las redes neuronales, que es la función de activación *relu*.

Ahora, ¿cómo comienzas a construir tu Perceptrón multicapa? Una forma rápida de comenzar es usar el modelo secuencial de Keras: es una "pila" lineal de capas. Puede crearse fácilmente el modelo pasando una lista de instancias de capa al constructor, que se configura ejecutando *model = Sequential()*.

A continuación, recordemos la estructura del perceptrón multicapa: tiene una capa de entrada, algunas capas ocultas y una capa de salida. Cuando creas tu modelo, es importante tener en cuenta que la primera capa necesita conocer la forma de entrada. el modelo necesitará saber qué forma de entrada va a tener y es por esto que utilizaremos los argumentos *input_shape*, *input_dim*, *input_length*, o *batch_size*.

En este caso, tendrá que usar una capa *Dense*, que es una capa completamente conectada. Las capas densas aplican la siguiente operación: *output = activation(dot(input, kernel) + bias)*. Hay que tener en cuenta que, sin la función de activación, su capa densa consistiría sólo en dos operaciones lineales: un producto de puntos y una suma.

En la primera capa, el argumento *activation* toma el valor *relu*. A continuación, también verás que *input_shape* se ha definido. Esta es la operación de *input* que acabamos de ver: el modelo toma como matrices de entrada de forma *(11,)*, o *(*, 11)*. Por último, verás que la primera capa intermedia tiene 12 como primer valor en el argumento *units* de *Dense()*, y que será la

dimensionalidad del espacio de salida de esta capa intermedia y que en realidad son 12 unidades ocultas. Esto significa que el modelo generará conjuntos de formas *(*, 12)*: esta es la dimensionalidad del espacio de salida de esta primera capa oculta.

En realidad *units*, representa la fórmula *kernel* anterior o la matriz de pesos, compuesta de todos los pesos dados a todos los nodos de entrada, creados por la capa. Hay que tener en cuenta que no incluye ningún sesgo en el ejemplo, ya que no se ha incluido el argumento *use_bias* y se configuró *TRUE*. Introducir este argumento también es una posibilidad pero nosotros no lo haremos.

A continuación hay otra capa intermedia que también usa la función *relu* de activación. La salida de esta capa serán conjuntos de 8 valores *(*,8)*(segunda capa oculta de 8 neuronas)

La construcción de la red está finalizando con una capa *Dense* de tamaño 1. La capa final también utilizará una función de activación sigmoidea para que su salida sea realmente una probabilidad; Esto significa que dará como resultado una puntuación entre 0 y 1, que indica la probabilidad de que la muestra tenga el objetivo "1", o la probabilidad de que el vino sea rojo.

```
# Import `Sequential` from `keras.models`
from keras import Sequential

# Import `Dense` from `keras.layers`
from keras.layers import Dense


# Initialize the constructor
model = Sequential()
# Add an input layer
model.add(Dense(12, activation='relu', input_shape=(11,)))

# Add one hidden layer
model.add(Dense(8, activation='relu'))

# Add an output layer
model.add(Dense(1, activation='sigmoid'))
```

En general, verás que hay dos decisiones clave que hay que tomar para construir arquitectura del modelo: cuántas capas va a usar y cuántas "unidades ocultas" elegirás para cada capa.

En este caso, se seleccionaron *8* unidades ocultas para la capa oculta del modelo. **Si se utilizaran más unidades ocultas de las necesarias, la red podrá aprender representaciones más complejas, pero también sería una operación más costosa que puede ser propensa a un sobreajuste**.

Recuerda que el sobreajuste ocurre cuando el modelo es demasiado complejo: describirá un error o ruido aleatorio y no la relación subyacente que se necesita describir. En otras palabras, los datos de entrenamiento se modelan demasiado bien y los datos de test no tan bien...

Ten en cuenta que cuando no tiene tantos datos de entrenamiento disponibles, debería preferir usar una red pequeña con muy pocas capas ocultas (generalmente solo una, como en el ejemplo anterior).

Si desea obtener información sobre el modelo que acaba de crear, puede utilizar el atributo *output_shape* o la función *summary()*, entre otras. Algunas de los más básicos se enumeran a continuación.

Intenta ejecutarlos para ver qué resultados obtienes exactamente y qué te dicen sobre el modelo que acabas de crear:

```
# Model output shape
model.output_shape

# Model summary
model.summary()

# Model config
model.get_config()
```

Compilar y ajustar el modelo

A continuación, es hora de compilar el modelo creado y ajustarlo a los datos: utilizar *compile()*y *fit()*:

```
# List all weight tensors
model.get_weights()
model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

model.fit(X_train, y_train,epochs=20, batch_size=1, verbose=1)
```

Al compilar, se entrena el modelo con el optimizador *adam* y la función de pérdida *binary_crossentropy*. Además, también puede controlar la precisión durante el entrenamiento pasando al argumento *metrics* el parámetro *['accuracy']*

Tanto *optimizer* como *loss* son dos argumentos que se requieren si se desea compilar el modelo. Algunos de los algoritmos de optimización más populares utilizados son el Descenso de gradiente estocástico (SGD), ADAM y RMSprop. Dependiendo del algoritmo que elija, deberá ajustar ciertos parámetros, como la velocidad de aprendizaje o el término momentum (impulso). La elección de una función de error depende de la tarea que se tenga entre manos: por ejemplo, para un problema de regresión, generalmente usará el error cuadrático medio (MSE). Como puede ver en este ejemplo, se utilizó *binary_crossentropy*, el problema de clasificación binaria para determinar si un vino es rojo o blanco. Por último, con la clasificación de varias clases, hará uso de *categorical_crossentropy*.

Después, puede entrenar el modelo durante 20 épocas o iteraciones sobre todas las muestras en los conjuntos *X_train* y *y_train*, en lotes de 1 muestra. También puede especificar el argumento *verbose*. Al configurarlo *1*, indica que desea ver el registro de la barra de progreso.

En otras palabras, se debe entrenar el modelo durante un número específico de épocas o exposiciones del conjunto de datos de entrenamiento. Una época es un solo paso a través de todo el conjunto de entrenamiento, seguido de una prueba del conjunto de verificación. El tamaño de lote que se especifique en el código anterior define el número de muestras que se propagarán a través de la red. Además, al hacer esto, se optimiza la eficiencia porque se asegura de no cargar demasiados patrones de entrada en la memoria al mismo tiempo.

Predecir valores

¡Pongamos tu modelo en uso! Puedes hacer predicciones para las etiquetas del conjunto de pruebas con él. Simplemente usa *predict()* y pasa el conjunto de prueba para predecir las etiquetas de los datos. En este caso, el resultado se almacena en *y_pred*:

```
y_pred = model.predict(X_test)
```

Antes evaluar el modelo, ya podemos tener una idea rápida de la precisión al comparar *y_pred* e *y_test* visualizándolos, por ejemplo:

```
y_pred[:10]
```

```
Out[41]:

array([[1.3768673e-05],
       [9.9999702e-01],
       [2.6822090e-07],
       [0.0000000e+00],
       [0.0000000e+00],
       [1.0000000e+00],
       [0.0000000e+00],
       [0.0000000e+00],
       [1.0000000e+00],
       [3.5762787e-07]], dtype=float32)
```

```
In [52]:

y_pred=(y_pred>0.5)
```

```
In [50]:

y_test
```

```
Out[50]:

array([0, 1, 0, ..., 0, 0, 0], dtype=int64)
```

Evaluar nuestra Red Neural Artificial

Ahora que hemos creado nuestra red y la hemos utilizado para hacer predicciones sobre datos que aún no había visto, es hora de evaluar su rendimiento. Se puede comparar visualmente las predicciones con las etiquetas de prueba reales (*y_test*), o se puede usar todo tipo de métricas para determinar el rendimiento real. En este caso, usarás *evaluate()* para hacer esto. Pasar los datos de prueba y las etiquetas de prueba y, si quieres, coloca el argumento *verbose* a 1. Verás que aparecen más registros cuando haga esto.

```
score = model.evaluate(X_test, y_test,verbose=1)

print(score)
```

En este caso, probarás algunas técnicas básicas de evaluación de clasificación, como:

- La matriz de confusión, que es un desglose de las predicciones en una tabla que muestra las predicciones correctas y los tipos de predicciones incorrectas realizadas. Idealmente, únicamente verás números en la diagonal, lo que significa que todas sus predicciones fueron correctas.

- La precisión es una medida de la exactitud de un clasificador. Cuanto mayor sea la precisión, más preciso será el clasificador.

```
# Import the modules from `sklearn.metrics`
from sklearn.metrics import confusion_matrix, precision_score, recall_score, f1_score, cohen_kappa_score

# Confusion matrix
confusion=confusion_matrix(y_test, y_pred)
print(confusion)

# Precision
precision_score(y_test, y_pred)
```

```
# Confusion matrix
confusion_matrix(y_test, y_pred)
```

```
Out[55]:

array([[1585,    3],
       [  10,  547]], dtype=int64)
```

```
# Precision
precision_score(y_test, y_pred)
```

```
Out[56]:

0.9945454545454545
```

- La recuperación (*recall*) es una medida de la integridad de un clasificador. Cuanto mayor sea su valor, más casos cubre el clasificador.

```
# Recall
recall_score(y_test, y_pred)
```

```
Out[57]:

0.9820466786355476
```

- La puntuación *F1* o la puntuación F es un promedio ponderado de precisión y recuperación.

```
# F1 score
f1_score(y_test,y_pred)
```

```
Out[58]:

0.988256549232159
```

- El valor de Kappa de Cohen es la precisión de la clasificación normalizada por el

desequilibrio de las clases en los datos.

```
# Cohen's kappa
cohen_kappa_score(y_test, y_pred)
```

Out[59]:

```
0.9841725720350888
```

ALGUNAS PRUEBAS MÁS

Prueba las siguientes ideas, recoge qué resultados te generan, observa cuál es su efecto y saca conclusiones (se recomienda construir una tabla de resultados donde finalmente se recoja toda la información asociada a la red y a los resultados obtenidos, y así se puedan derivar de ella las conclusiones)

- Usaste 1 capa oculta. Intenta usar 2 o 3 capas ocultas. Recoge datos en una tabla y comenta
- Usa capas con más o menos unidades ocultas. Recoge datos en una tabla y comenta.
- Cambia la función de activación en los casos anteriores. En lugar de *relu*, intente usar la *tanh* función de activación y vea cuál es el resultado. Recoge los resultados en las tablas anteriores y coméntalo.
- Toma la columna *quality* como las etiqueta objetivo, transformando ahora el problema a un problema multiclase. Aplica lo aprendido y extrae resultados y conclusiones. Recoge resultados igual que en los casos anteriores.

1) Elige un nuevo problema de clasificación, y aplica todo lo aprendido anteriormente. Recógelo todo en un documento. Finalmente escoge el mejor modelo que hayas encontrado junto con las conclusiones obtenidas.