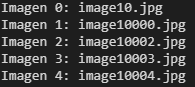
Descripción de los datos empleado

El punto de partida consta de un banco de 5983 imágenes y un archivo Data.csv.

En lo referente a las imágenes se cumple que:

* Nombre: image**#N**.jpg-🡪 **#N**: número de imagen.
* Imagen a color RGB.
* Dimensiones distintas.

Ejemplo:



En lo referente al archivo Data.csv, se clasifican las imágenes en 4 grupos:

'Food', 'Attire', 'Decorationandsignage', 'misc'.

Ejemplo:

|  |  |
| --- | --- |
| **Image** | **Class** |
| image7042.jpg | Food |
| image3327.jpg | misc |
| image10335.jpg | Attire |
| image8019.jpg | Food |
| image2128.jpg | Attire |
| image1106.jpg | misc |
| image6750.jpg | Food |

Partiendo de estos datos y con el uso del lenguaje de programación Python, se realizará el entrenamiento y posterior comparación de modelos de aprendizaje automático aplicados a la clasificación de imágenes.

Partida: 5983 Imágenes Clasificadas.

El primer paso, antes de entrenar cualquier modelo, es preparar las variables que vamos a introducir como entrada.

**Tratamiento: Entradas**

Objetivo:

x: entrada del modelo, matriz de bits de una imagen.

t: salida del modelo, clasificación entrada.

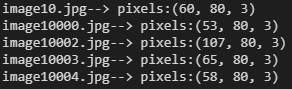
x 🡪 🡪t

MODELO

PASO 1:

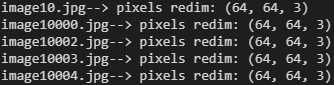
Las imágenes no tienen la misma dimensión, por lo que se redimensionan para que todas tengan la misma: 64x64 Pixeles.

Imágenes originales:





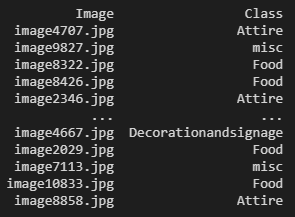
Imágenes redimensionadas:



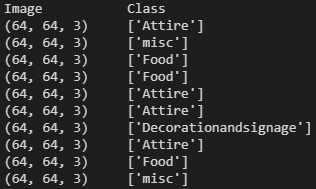
PASO 2:

La clasificación de la imagen que se representa en el archivo Data.csv se asocia el nombre de una imagen con una clase. En nuestro caso nos interesa asociar la matriz de pixeles de cada imagen con su clase.

imagen.jpg 🡪clase

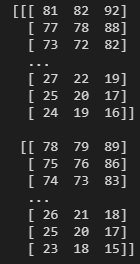


Matriz píxeles 🡪clase



PASO 3:

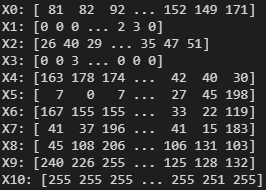
Las matrices de pixeles no se pueden introducir como entradas a los modelos directamente, hay que realizar un proceso de aplanamiento de la matriz de forma que se representen todos los pixeles de la imagen en un único vector.

(64,64,3) (1, 12288)

🡪 

De esta forma para las 5983 imágenes tendremos todos los vectores de entrada de nuestro modelo.

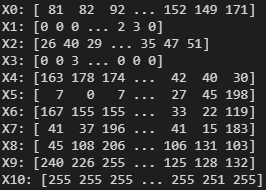
En el siguiente ejemplo se muestran las primeras 11 entradas:



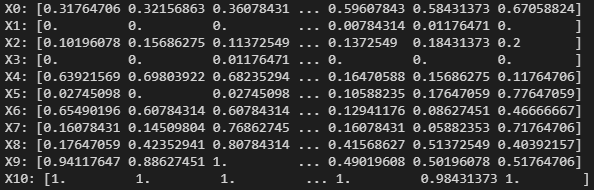
PASO 4:

Para miminizar el tiempo de entrenamiento y debido a que los datos tienen una variación de 0 a 255 ya que son de 8 bits, se opta por normalizar las entradas de forma que varien entre 0 y 1 y de esta forma lograr que el resultado del entrenamiento sea mas preciso.

Sin normalizar



Normalizado

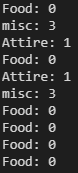


PASO 5:

Para facilitar el entrenamiento de los modelos, las clases que tienen un formato del tipo “string” se asocian con un número.

'Food': 0, 'Attire': 1, 'Decorationandsignage':2, 'misc':3

Ejemplo:

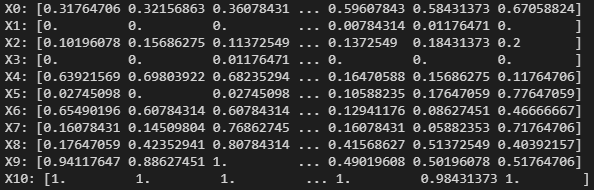


De esta forma obtenemos que:



Finalmente ya podemos realizar los entrenamientos de los modelos con el uso de nuestras entradas salidas previamente tratadas:

X normalizado t numérica

MODELO

Experimentos realizados

En el siguiente estudio se analizaran los resultados de distintos modelos de aprendizaje automático supervisado aplicados a la clasificación de imágenes.

Cabe destacar que el aprendizaje es supervisado porque además de las entradas del modelo se indican las salidas del mismo.

**Modelos de aprendizaje supervisado aplicados a clasificación de imágenes**

**Regresión logística vs Regresión lineal**

**Modelo líneal: Regresión lineal**

* Se producen problemas especialmente cuando las clases están desbalanceadas.
* Problemas cuando hay clases no homogéneas (datos atípicos).
* Por lo general, no es un método muy adecuado para la clasificación.
* Método de mínimos cuadrados.

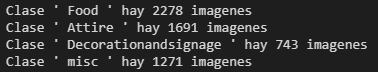
**Modelo no línea: Regresión logística**

* En general es el método más robusto ya que no es sensible a casos atípicos.
* Para solucionar el problema del método de mínimos cuadrados, la regresión logística propone el uso de una función logística o sigmoide.

**¿Qué modelo de regresión utilizar?**

Analizamos nuestros datos:

Como vemos en la siguiente imagen, partimos de datos desbalanceados, ya que el número de datos en las clases difieren entre ellos.



El caso más crítico es el de **‘Food‘** en el que tenemos **2278** imágenes frente a las **743** de **‘Decorationanddsignage’**.

El modelo de regresión escogido las condiciones de nuestro problema es regresión logística.

**Regresión Logistica:**

Con el uso de la función LogisticRegression disponible en la librería sklearn, se realizó el entrenamiento del modelo de regresión logística.

A tener en cuenta:

**Logistic Regression (aka logit, MaxEnt) classifier**

“In the multiclass case, the training algorithm uses the one-vs-rest (OvR) scheme if the ‘multi\_class’ option is set to ‘ovr’, and uses the cross-entropy loss if the ‘multi\_class’ option is set to ‘multinomial’. (Currently the ‘multinomial’ option is supported only by the ‘lbfgs’, ‘sag’, ‘saga’ and ‘newton-cg’ solvers.)”

**solver*{‘newton-cg’, ‘lbfgs’, ‘liblinear’, ‘sag’, ‘saga’}, default=’lbfgs’***

“For multiclass problems, only ‘newton-cg’, ‘sag’, ‘saga’ and ‘lbfgs’ handle multinomial loss; ‘liblinear’ is limited to one-versus-rest schemes”

**multi\_class*{‘auto’, ‘ovr’, ‘multinomial’}, default=’auto’***

***“***If the option chosen is ‘ovr’, then a binary problem is fit for each label. For ‘multinomial’ the loss minimised is the multinomial loss fit across the entire probability distribution, even when the data is binary. ‘multinomial’ is unavailable when solver=’liblinear’. ‘auto’ selects ‘ovr’ if the data is binary, or if solver=’liblinear’, and otherwise selects ‘multinomial’”

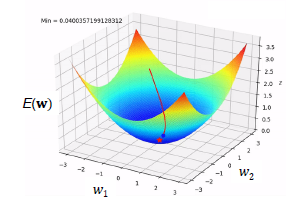
***max\_iterint, default=100***

“Maximum number of iterations taken for the solvers to converge.”

Teniendo en cuenta lo anterior y a que nuestros datos no son binarios en la opción multi\_class escogeremos 'multinomial' y como solver 'lbfgs' que es compatible con este tipo de clasificación.

**Sintaxis:**

El objetivo de las iteraciones es lograr que nuestro modelo converja: mínimo error /mejores pesos de entrenamiento de nuestro modelo.

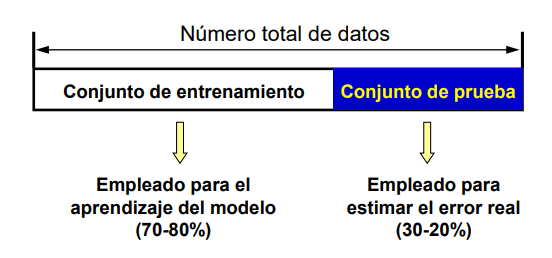


En nuestro caso escogimos 10000 iteraciones para llegar a la convergencia.

LogisticRegression(penalty='none', solver = 'lbfgs', max\_iter=10000,multi\_class='multinomial')

**Estrategia de evaluación de modelos**

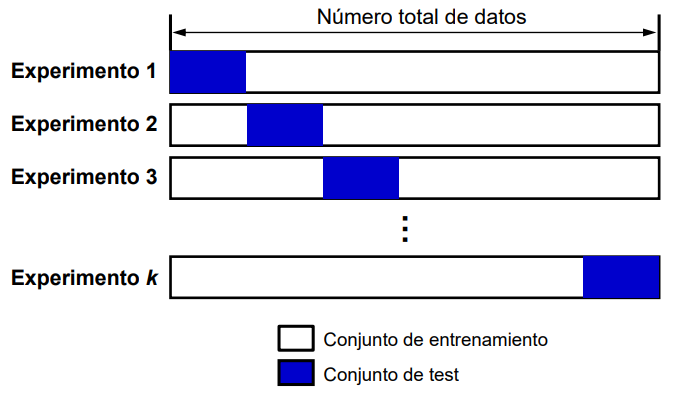
No podemos dividir los datos en dos subconjuntos, entrenamiento y prueba.



Ya que:

* No disponemos de una parte importante como conjunto de pruebas para nuestro modelo.
* Se corre el riesgo de que al dividir los datos para Train/Test estén desbalanceados por lo que el entrenamiento del modelo y su posterior evaluación sobre los datos de Test no sea fiable.

Se opta por realizar **K-fold cross-validation**:

Todas las muestras del conjunto de datos se usan alguna vez para entrenar o como parte del conjunto de prueba, esto soluciona la problemática anterior.

Para ello se utiliza la función cross\_validate, que nos permite obtener los parámetros significativos de los modelos estudiados.

Los parámetros evaluados son los siguientes:

Exactitud (accuaracy): tasa de acierto global del sistema

Roc Auc (roc\_auc\_ovo): El AUC calcula el área bajo la curva ROC y, por lo tanto, su valor varía entre 0 (peor caso) y 1 (mejor caso). El AUC indica cómo de bien se separan las probabilidades de las clases positivas de las negativas.

F1 macro: es la media armónica entre la precisión y la sensibilidad, tiene la ventaja que su métrica es mas relvante si las clases son desbalanceadas.

Presición (precision\_macro):

Recall\_macro:

['accuracy','roc\_auc\_ovo','f1\_macro', 'precision\_macro', 'recall\_macro']

análisis de los resultados

conclusiones obtenidas