

APRENDE MACHINE LEARNING

TEORÍA +
PRÁCTICA
PYTHON

ESCRITO POR
JUAN IGNACIO BAGNATO

BASADO EN
EL CONTENIDO DEL BLOG



Aprende Machine Learning en Español

Teoría + Práctica Python

Juan Ignacio Bagnato

Este libro está a la venta en <http://leanpub.com/aprendeml>

Esta versión se publicó en 2020-07-19



Este es un libro de [Leanpub](#). Leanpub anima a los autores y publicadoras con el proceso de publicación. [Lean Publishing](#) es el acto de publicar un libro en progreso usando herramientas sencillas y muchas iteraciones para obtener retroalimentación del lector hasta conseguir el libro adecuado.

© 2020 Juan Ignacio Bagnato

¡Tuitea sobre el libro!

Por favor ayuda a Juan Ignacio Bagnato hablando sobre el libro en [Twitter](#)!

El hashtag sugerido para este libro es [#aprendeML](#).

Descubre lo que otra gente dice sobre el libro haciendo clic en este enlace para buscar el hashtag en Twitter:

[#aprendeML](#)

Índice general

Nota Inicial	1
Version 1.5	1
Extras	1
Repositorio	1
Feedback	2
¿Qué es el Machine Learning?	3
Definiendo Machine Learning	3
Una Definición Técnica	3
Diagrama de Venn	3
Aproximación para programadores	4
Resumen	5
Instalar el Ambiente de Desarrollo Python	6
¿Por qué instalar Python y Anaconda en mi ordenador?	6
1. Descargar Anaconda	6
2. Instalar Anaconda	7
3. Iniciar y Actualizar Anaconda	7
4. Actualizar librería scikit-learn	10
5. Instalar librerías para Deep Learning	11
Resumen	11
Análisis Exploratorio de Datos con Pandas en Python	13
¿Qué es el EDA?	13
EDA deconstruido	13
¿Qué sacamos del EDA?	14
Técnicas para EDA	15
Un EDA de pocos minutos con Pandas	15
Más cosas! (que se suelen hacer):	24
Resumen	24
Regresión Lineal con Python	26
¿Qué es la regresión lineal?	26
¿Cómo funciona el algoritmo de regresión lineal en Machine Learning?	27

ÍNDICE GENERAL

Un Ejercicio Práctico	27
Predecir cuántas veces será compartido un artículo de Machine Learning.	28
Regresión Lineal con Python y SKLearn	31
Visualicemos la Recta	33
Predicción en regresión lineal simple	33
Regresión Lineal Múltiple en Python	34
Visualizar un plano en 3 Dimensiones en Python	35
Predicción con el modelo de Múltiples Variables	37
Resumen	37
Regresión Logística	39
Introducción	39
Ejercicio de Regresión Logística en Python	39
Regresión Logística con SKLearn:	40
Visualización de Datos	42
Creamos el Modelo de Regresión Logística	44
Validación de nuestro modelo	45
Reporte de Resultados del Modelo	46
Clasificación de nuevos valores	47
Resumen	48
Arbol de Decisión	49
¿Qué es un árbol de decisión?	49
¿Cómo funciona un árbol de decisión?	50
Arbol de Decisión con Scikit-Learn paso a paso	51
Predicción del “Billboard 100”: ¿Qué artista llegará al número uno del ranking?	51
Obtención de los datos de entrada	51
Análisis Exploratorio Inicial	52
Balanceo de Datos: Pocos artistas llegan al número uno	56
Preparamos los datos	58
Mapeo de Datos	60
Buscamos la profundidad para el árbol de decisión	65
Visualización del árbol de decisión	67
Análisis	68
Predicción de Canciones al Billboard 100	69
Resumen	70
Datos desbalanceados	72
Problemas de clasificación con Clases desequilibradas	72
¿Cómo nos afectan los datos desbalanceados?	73
Métricas y Confusion Matrix	73
Vamos al Ejercicio con Python!	76
Análisis exploratorio	76
Estrategias para el manejo de Datos Desbalanceados:	79

ÍNDICE GENERAL

Probando el Modelo sin estrategias	79
Estrategia: Penalización para compensar	81
Estrategia: Subsampling en la clase mayoritaria	83
Estrategia: Oversampling de la clase minoritaria	84
Estrategia: Combinamos resampling con Smote-Tomek	86
Estrategia: Ensamble de Modelos con Balanceo	87
Resultados de las Estrategias	88
Resumen	89
Random Forest, el poder del Ensamble	91
¿Cómo surge Random Forest?	91
¿Cómo funciona Random Forest?	91
¿Por qué es aleatorio?	92
Ventajas y Desventajas del uso de Random Forest	92
Vamos al Código Python	92
Creamos el modelo y lo entrenamos	93
Los Hiperparámetros más importantes	94
Evaluamos resultados	94
Comparamos con el Baseline	96
Resumen	96
K-Means	97
Cómo funciona K-Means	97
Casos de Uso de K-Means	97
Datos de Entrada para K-Means	98
El Algoritmo K-means	98
Elegir el valor de K	99
Ejemplo K-Means con Scikit-learn	99
Agrupar usuarios Twitter de acuerdo a su personalidad con K-means	100
Visualización de Datos	103
Definimos la entrada	104
Obtener el valor K	106
Ejecutamos K-Means	106
Clasificar nuevas muestras	113
Resumen	113
K-Nearest-Neighbor	115
¿Qué es el algoritmo k-Nearest Neighbor ?	115
¿Dónde se aplica k-Nearest Neighbor?	115
Pros y contras	116
¿Cómo funciona kNN?	116
Un ejemplo k-Nearest Neighbor en Python	116
El Ejercicio: App Reviews	117
Un poco de Visualización	119

ÍNDICE GENERAL

Preparamos las entradas	121
Usemos k-Nearest Neighbor con Scikit Learn	121
Precisión del modelo	121
Y ahora, la gráfica que queríamos ver!	122
Elegir el mejor valor de k	125
Clasificar ó Predecir nuevas muestras	126
Resumen	127
Clasificación de Imágenes en Python	128
Ejercicio: Clasificar imágenes de deportes	128
Vamos al código Python	131
1- Importar librerías	131
2-Cargar las imágenes	132
3- Crear etiquetas y clases	133
4-Creamos sets de Entrenamiento y Test, Validación y Preprocesar	134
5 - Creamos la red (Aquí la Magia)	135
6-Entrenamos la CNN	137
7-Resultados de la clasificación	140
Resumen	141
¿Cómo funcionan las Convolutional Neural Networks?	143
Muchas imágenes	143
Pixeles y neuronas	144
Convoluciones	145
Filtro: conjunto de kernels	146
La función de Activación	148
Subsampling	148
Subsampling con Max-Pooling	149
¿Ya terminamos? NO: ahora más convoluciones!!	150
Conectar con una red neuronal “tradicional”	152
¿Y cómo aprendió la CNN a “ver”??: Backpropagation	153
Comparativa entre una red neuronal “tradicional” y una CNN	153
Arquitectura básica	154
No incluido	154
Resumen	155
Detección de Objetos con Python	156
¿En qué consiste la detección YOLO?	157
El proyecto Propuesto: Detectar personajes de Lego	157
Crea un dataset: Imágenes y Anotaciones	159
El lego dataset	161
El código Python	161
Leer el Dataset	162
Train y Validación	163

ÍNDICE GENERAL

Data Augmentation	163
Crear la Red de Clasificación	164
Crear la Red de Detección	165
Generar las Anclas	168
Entrenar la Red!	169
Revisar los Resultados	170
Probar la Red	170
Resumen	176
Conjuntos de Train, Test y Validación	177
Motor de Recomendación	178
Naive Bayes	179
Overfitting y la Generalización del conocimiento	180
Redes Neuronales	181
Deep Learning	182
Tensorflow y Keras	183
Coche Robot que conduce solo	184
Series Temporales	185
Crea tu propio servicio de Machine Learning	186

Nota Inicial

Si has adquirido ó descargado este ejemplar, primero que nada **quiero agradecerte**.

Este libro es un trabajo en progreso, por lo que con tu ayuda podré ir completando con el paso de las siguientes semanas. Ten en cuenta que lo estoy desarrollando en mis tiempos libres, entre el trabajo, un master, cuidar de mis hijos y *una Pandemia* de contexto...

Escribir cada Artículo me lleva desde la idea inicial en mi cabeza, investigar e informarme del tema, crear un ejercicio original en código Python, crear el conjunto de datos, testear, crear las gráficas y... redactar el texto del propio artículo! (y alguna cosilla más: editar, revisar, corregir, enlaces, difundir, pull-push, etc.)

Todos los artículos serán versiones corregidas, actualizadas y mejoradas de los originales que podrás encontrar en el blog [Aprende Machine Learning](#)¹

Espero que sigas en contacto conmigo y poder anunciar -dentro de no mucho tiempo- que la publicación está terminada!

Version 1.5

Agregados en esta versión:

- Correcciones
- Clasificación de Imágenes (ejercicio Python)
- Qué son las Redes Neuronales Convolucionales (teoría)

Extras

No olvides descargar los extras que iré subiendo.

Por el momento tienes a disposición el “Lego-dataset” para el ejercicio de detección de Objetos en un archivo zip de 170 MB con más de 300 imágenes y sus anotaciones.

Repositorio

El código completo y las Jupyter Notebooks las podrás ver y descargar desde [mi repositorio Github](#)²

¹<https://www.aprendemachinelearning.com/>

²<https://github.com/jbagnato/machine-learning/>

Feedback

Todos los comentarios para mejorar son bienvenidos, por lo que sientete libre de enviarme sugerencias, correcciones ó lo que sea por las vías que ofrece LeanPub ó por Twitter en @jbagnato ó por el [formulario de contacto del blog](#)³

³<https://www.aprendemachinelearning.com/contacto/>

¿Qué es el Machine Learning?

Veamos algunas definiciones existentes para intentar dar comprensión a esta revolucionaria materia.

Definiendo Machine Learning

El Machine Learning -traducido al Español como Aprendizaje Automático- es un subcampo de la Inteligencia Artificial que busca resolver el “cómo construir programas de computadora que mejoran automáticamente adquiriendo experiencia”.

Esta definición indica que el programa que se crea con ML no necesita que el programador indique explícitamente las reglas que debe seguir para lograr su tarea si no que este mejora automáticamente.

Grandes volúmenes de datos están surgiendo de diversas fuentes en los últimos años y el Aprendizaje Automático relacionado al campo estadístico consiste en extraer y reconocer patrones y tendencias para comprender qué nos dicen los datos. Para ello, se vale de algoritmos que pueden procesar Gygas y/o Terabytes y obtener información útil.

Una Definición Técnica

Durante mi cursada de Aprendizaje Automático en Coursera, encontré la siguiente definición técnica:

A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.

La experiencia E hace referencia a grandes volúmenes de datos recolectados (muchas veces el Big Data) para la toma de decisiones T y la forma de medir su desempeño P para comprobar que mejoran con la adquisición de más experiencia.

Diagrama de Venn

Drew Conway creó un simpático diagrama de Venn en el que inerrelaciona diversos campos. Aquí copio su versión al Español traducida por mi:



Diagrama de Venn

En esta aproximación al ML, podemos ver que es una intersección entre conocimientos de Matemáticas y Estadística con Habilidades de Hackeo del programador.

Aproximación para programadores

Los programadores sabemos que los algoritmos de búsqueda pueden tomar mucho tiempo en resolverse y que cuanto mayor sea el espacio de búsqueda crecerán exponencialmente las posibilidades de combinación de una respuesta óptima, haciendo que los tiempos de respuesta tiendan al infinito o que tomen más tiempo de lo que un ser humano tolerar (por quedarse sin vida o por impaciencia).

Para poder resolver este tipo de problemas surgen soluciones de tipo heurísticas que intentan dar «intuición» al camino correcto a tomar para resolver un problema. Estos pueden obtener buenos resultados en tiempos menores de procesamiento, pero muchas veces su intuición es arbitraria y pueden llegar a fallar.

Los algoritmos de ML intentan utilizar menos recursos para «entrenar» grandes volúmenes de datos e ir aprendiendo por sí mismos. Podemos subdividir el ML en 2 grandes categorías: Aprendizaje Supervisado o Aprendizaje No Supervisado.

Entre los Algoritmos más utilizados en Inteligencia Artificial encontramos:

- Arboles de Decisión
- Regresión Lineal
- Regresión Logística
- k Nearest Neighbor
- PCA / Principal Component Analysis
- SVM
- Gaussian Naive Bayes
- K-Means
- Redes Neuronales Artificiales
- Aprendizaje Profundo ó Deep Learning

Una mención especial a las Redes Neuronales Artificiales

Una mención distintiva merecen las RNAs ya que son algoritmos que utilizan un comportamiento similar a las neuronas humanas y su capacidad de sinopsis para la obtención de resultados, interrelacionándose diversas capas de neuronas para darle mayor poder.

Aunque estos códigos existen desde hace más de 70 años, en la última década han evolucionado notoriamente (en paralelo a la mayor capacidad tecnológica de procesamiento, memoria RAM y disco, la nube, etc.) y están logrando impresionantes resultados para analizar textos y síntesis de voz, traducción de idiomas, procesamiento de lenguaje natural, visión artificial, análisis de riesgo, clasificación y predicción y la creación de motores de recomendación.

Resumen

El Machine Learning es una nueva herramienta clave que posibilitará el desarrollo de un futuro mejor para el hombre brindando inteligencia a robots, coches y hogares. Las Smart Cities, el IOT ya se están volviendo una realidad y también las aplicaciones de Machine Learning en Asistentes como Siri, las recomendaciones de Netflix o Sistemas de Navegación en Drones. Para los ingenieros o informáticos es una disciplina fundamental para ayudar a crear y transitar este nuevo futuro.

Instalar el Ambiente de Desarrollo Python

Para programar tu propia Máquina de Inteligencia Artificial necesitarás tener listo tu ambiente de desarrollo local, en tu computadora de escritorio o portatil. En este capitulo explicaremos una manera sencilla de obtener Python y las librerías necesarias para programar como un Científico de Datos y poder utilizar los algoritmos más conocidos de Machine Learning.

¿Por qué instalar Python y Anaconda en mi ordenador?

Python es un lenguaje sencillo, rápido y liviano y es ideal para aprender, experimentar, practicar y trabajar con machine learning, redes neuronales y aprendizaje profundo.

Utilizaremos la Suite gratuita de Anaconda que nos facilitará la tarea de instalar el ambiente e incluye las Jupyter Notebooks, que es una aplicación web que nos ayudará a hacer ejercicios paso a paso en Machine Learning, visualizacion de datos y escribir comentarios tal como si se tratase de un cuaderno de notas de la universidad.

Esta Suite es multiplataforma y se puede utilizar para Windows, Linux y Macintosh.

Agenda

Nuestra agenda de hoy incluye:

1. Descargar Anaconda
2. Instalar Anaconda
3. Iniciar y Actualizar Anaconda
4. Actualizar paquete scikit-learn
5. Instalar Librerías para Deep Learning

Comencemos!

1. Descargar Anaconda

Veamos como descargar Anaconda a nuestro disco y obtener esta suite científica de Python

Nos dirigimos a la Home de Anaconda e iremos a la [sección de Download⁴](#) (descargas)
Elegimos nuestra plataforma: Windows, Mac o Linux

The screenshot shows the 'Anaconda Installers' page. It has three main sections: 'Windows' (with icons for 32-bit and 64-bit), 'MacOS' (with icons for 32-bit and 64-bit), and 'Linux' (with icons for 32-bit and 64-bit). Each section lists Python versions (3.7 and 2.7) and their corresponding graphical and command-line installers.

Plataforma	Versión Python	Tipo de Instalador	Tamaño
Windows	Python 3.7	64-Bit Graphical Installer	466 MB
		32-Bit Graphical Installer	423 MB
	Python 2.7	64-Bit Graphical Installer	413 MB
MacOS	Python 3.7	64-Bit Graphical Installer	442 MB
		64-Bit Command Line Installer	430 MB
Linux	Python 3.7	64-Bit (x86) Installer	522 MB
		64-Bit (Power8 and Power9) Installer	276 MB
	Python 2.7	64-Bit (x86) Installer	477 MB
Linux	Python 2.7	64-Bit (Power8 and Power9) Installer	295 MB

Atención: Elegir la versión de Python 3.7 (y no la de 2.7) y seleccionar el instalador Gráfico (Graphical Installer)

Con esto guardaremos en nuestro disco duro unos 460MB (según sistema operativo) y obtendremos un archivo con el nombre similar a Anaconda3-5.1.10-MacOSX-x86_64.pkg

2. Instalar Anaconda

En este paso instalaremos la app en nuestro sistema. (Deberá tener permisos de Administrador si instala para todos los usuarios).

Ejecutamos el archivo que descargamos haciendo doble click.

Se abrirá un Típico Wizard de instalación.

Seguiremos los pasos, podemos seleccionar instalación sólo para nuestro usuario, seleccionar la ruta en disco donde instalaremos y listo.

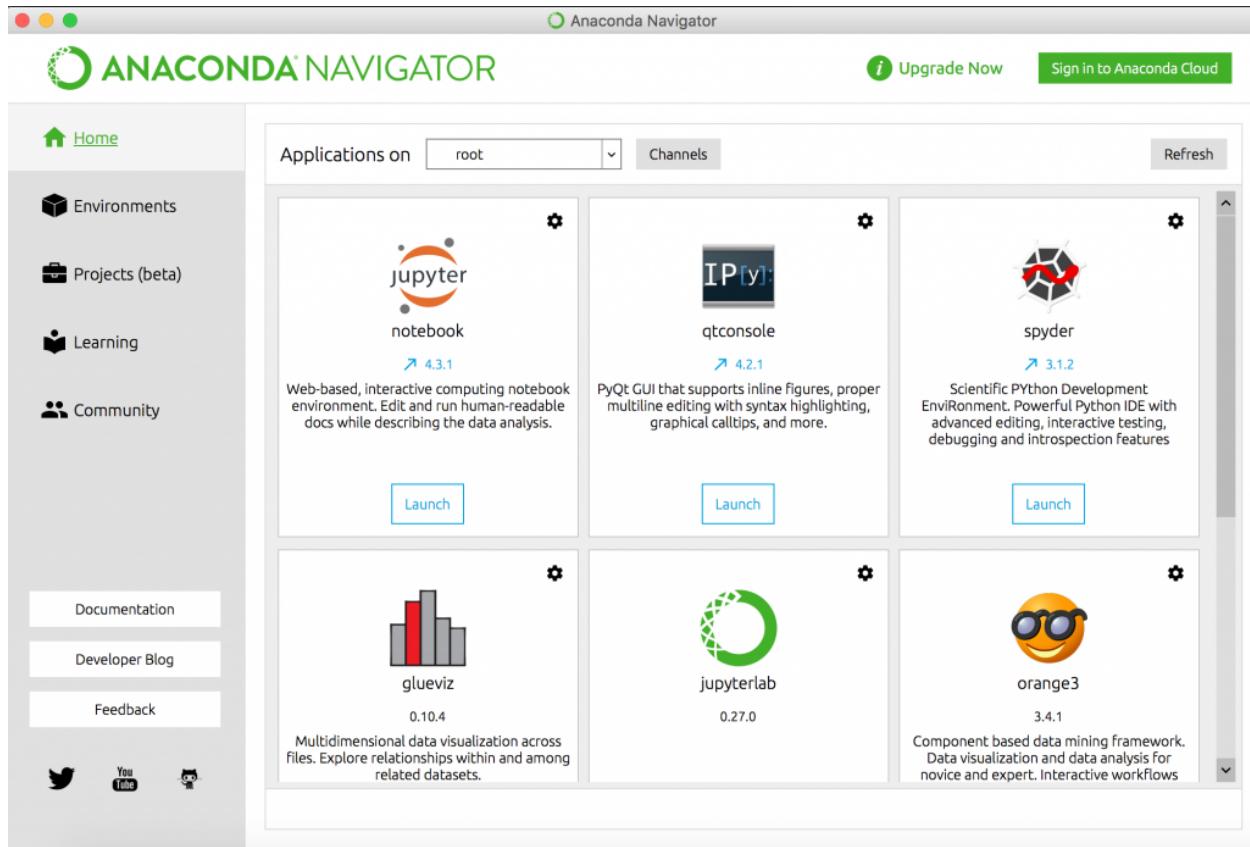
Al instalarse el tamaño total podrá superar 1Gb en disco.

3. Iniciar y Actualizar Anaconda

En este paso comprobaremos que se haya instalado correctamente y verificar tener la versión más reciente.

⁴<https://www.anaconda.com/products/individual#download>

Anaconda viene con una suite de herramientas gráficas llamada Anaconda Navigator. Iniciemos la aplicación y veremos una pantalla como esta:



Entre otros vemos que podemos lanzar las Jupyter Notebooks!.

Para comprobar la instalación abrimos una Terminal de Mac/Linux/Ubuntu o la Línea de Comandos de Windows.

Escribimos

```
1 $ conda -V
```

y obtenemos la versión

```
1 conda 4.3.30
```

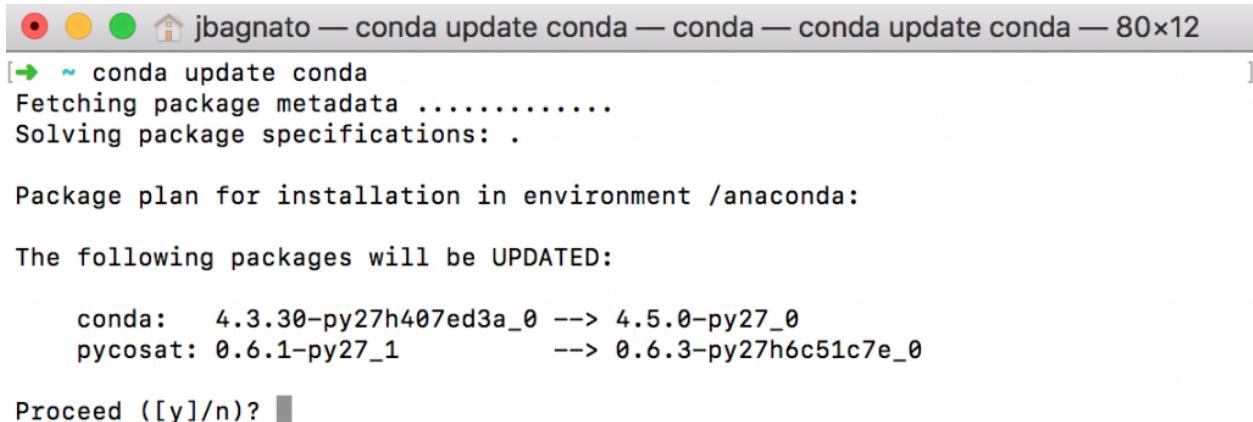
luego tipeamos

```
1 $ python -V
```

y verificamos la versión de Python de nuestro sistema.

Para asegurarnos de tener la versión más reciente de la suite ejecutaremos

```
1 $ conda update conda
```



The screenshot shows a terminal window titled 'jbagnato — conda update conda — conda — conda update conda — 80x12'. The command '\$ conda update conda' is run, followed by package metadata fetching and solving. A package plan for installation in environment '/anaconda' is shown, listing 'conda' and 'pycosat' as updated packages. The user is prompted with 'Proceed ([y]/n)?'.

```
[~] ~ conda update conda
Fetching package metadata .....
Solving package specifications: .

Package plan for installation in environment /anaconda:

The following packages will be UPDATED:

  conda: 4.3.30-py27h407ed3a_0 --> 4.5.0-py27_0
  pycosat: 0.6.1-py27_1           --> 0.6.3-py27h6c51c7e_0

Proceed ([y]/n)?
```

debemos poner 'y' para actualizar y se descargarán. Luego ejecutamos

```
1 $ conda update anaconda
```

Para confirmar que todo funciona bien, crearemos un archivo de texto para escribir un breve script de python. Nombra al archivo versiones.py y su contenido será:

```
1 # scipy
2 import scipy
3 print('scipy: %s' % scipy.__version__)
4 # numpy
5 import numpy
6 print('numpy: %s' % numpy.__version__)
7 # matplotlib
8 import matplotlib
9 print('matplotlib: %s' % matplotlib.__version__)
10 # pandas
11 import pandas
12 print('pandas: %s' % pandas.__version__)
13 # statsmodels
14 import statsmodels
15 print('statsmodels: %s' % statsmodels.__version__)
16 # scikit-learn
17 import sklearn
18 print('sklearn: %s' % sklearn.__version__)
```

En la linea de comandos, en el mismo directorio donde está el archivo escribiremos:

```
1 $ python versiones.py
```

y deberemos ver una salida similar a esta:

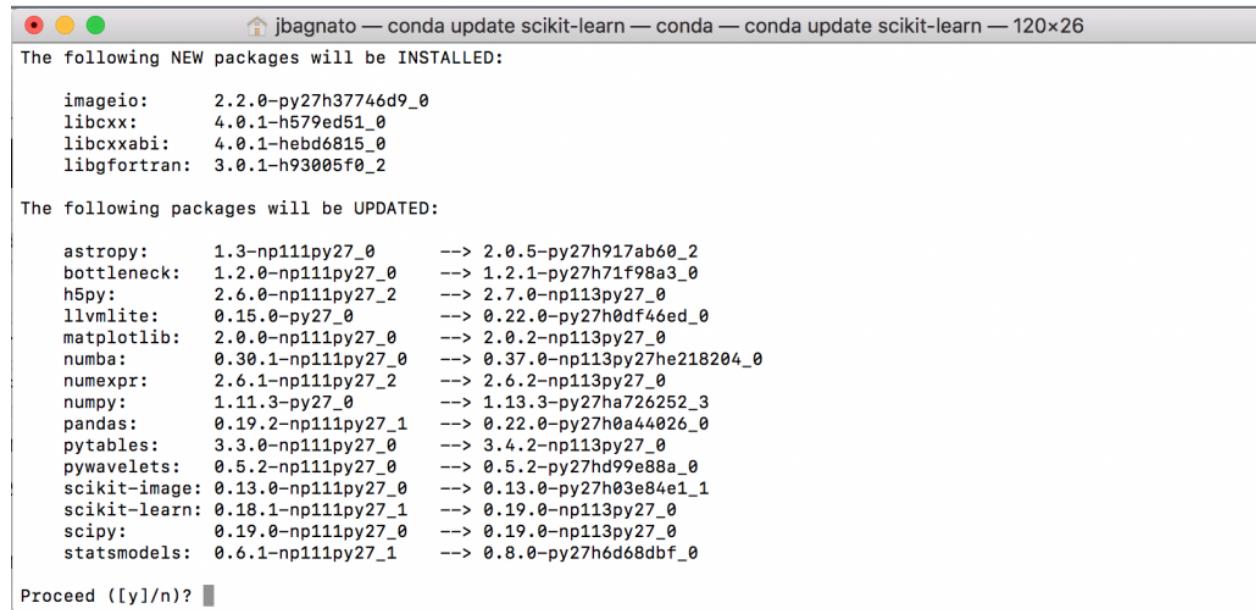
```
1 scipy: 0.18.1
2 numpy: 1.12.1
3 matplotlib: 1.5.3
4 pandas: 0.19.2
5 statsmodels: 0.8.0
6 sklearn: 0.18.1
```

4. Actualizar libreria scikit-learn

En este paso actualizaremos la librería más usada para Machine Learning en python llamada SciKit Learn

En la Terminal escribiremos

```
1 $ conda update scikit-learn
```



```
jbagnato — conda update scikit-learn — conda — conda update scikit-learn — 120x26
The following NEW packages will be INSTALLED:
imageio:      2.2.0-py27h37746d9_0
libcxx:        4.0.1-h579ed51_0
libcxxabi:     4.0.1-hebd6815_0
libgfortran:   3.0.1-h93005f0_2

The following packages will be UPDATED:
astropy:       1.3-np11ipy27_0    --> 2.0.5-py27h917ab60_2
bottleneck:    1.2.0-np11ipy27_0  --> 1.2.1-py27h71f98a3_0
h5py:          2.6.0-np11ipy27_2  --> 2.7.0-np113py27_0
llvmlite:      0.15.0-py27_0    --> 0.22.0-py27h0df46ed_0
matplotlib:    2.0.0-np11ipy27_0  --> 2.0.2-np113py27_0
numba:         0.30.1-np11ipy27_0 --> 0.37.0-np113py27he218204_0
numexpr:       2.6.1-np11ipy27_2  --> 2.6.2-np113py27_0
numpy:         1.11.3-py27_0    --> 1.13.3-py27ha726252_3
pandas:        0.19.2-np11ipy27_1 --> 0.22.0-py27h0a44026_0
pytables:      3.3.0-np11ipy27_0  --> 3.4.2-np113py27_0
pywavelets:    0.5.2-np11ipy27_0  --> 0.5.2-py27hd99e88a_0
scikit-image:  0.13.0-np11ipy27_0 --> 0.13.0-py27h03e84e1_1
scikit-learn:   0.18.1-np11ipy27_1 --> 0.19.0-np113py27_0
scipy:         0.19.0-np11ipy27_0  --> 0.19.0-np113py27_0
statsmodels:   0.6.1-np11ipy27_1  --> 0.8.0-py27h6d68dbf_0

Proceed ([y]/n)?
```

Deberemos confirmar la actualización poniendo 'y' en la terminal.

Podemos volver a verificar que todo es correcto ejecutando

```
1 $ python versiones.py
```

5. Instalar librerías para Deep Learning

En este paso instalaremos las librerías utilizadas para Aprendizaje profundo. Específicamente serán keras y la famosa y querida Tensorflow de Google.

Para ello ejecutaremos en nuestra línea de comandos

```
1 $ conda install -c conda-forge tensorflow  
  
1 $ pip install keras
```

Y crearemos un nuevo script para probar que se instalaron correctamente. Le llamaremos `versiones_deep.py` y tendrá las siguientes líneas:

```
1 # tensorflow  
2 import tensorflow  
3 print('tensorflow: %s' % tensorflow.__version__)  
4 # keras  
5 import keras  
6 print('keras: %s' % keras.__version__)
```

Ejecutamos en línea de comandos

```
1 $ python versiones_deep.py
```

en la terminal y veremos la salida:

```
1 tensorflow: 1.0.1  
2 Using TensorFlow backend.  
3 keras: 2.0.2
```

Ya tenemos nuestro ambiente de desarrollo preparado para el combate,

Resumen

Para nuestra carrera en Machine Learning, el enfrentamiento con Big Data y el perfeccionamiento como Data Scientist necesitamos un buen entorno en el que programar y cacharrear -lease, probar cosas y divertirse-. Para ello contamos con la suite de herramientas gratuitas de Anaconda que nos ofrece un entorno amable y sencillo en el que crear nuestras máquinas en código Python.

Otros artículos de interés (en inglés)

Usar Anaconda Navigator⁵

Instalar Pip⁶

Instalar Tensorflow⁷

Instalación de Keras⁸

⁵<https://docs.anaconda.com/anaconda/navigator/>

⁶<https://recursospython.com/guias-y-manuales/instalacion-y-utilizacion-de-pip-en-windows-linux-y-os-x/>

⁷<https://www.tensorflow.org/install/>

⁸<https://keras.io/#installation>

Análisis Exploratorio de Datos con Pandas en Python

Veremos de qué se trata este paso inicial tan importante y necesario para comenzar un proyecto de Machine Learning. Aprendamos en qué consiste el EDA y qué técnicas utilizar. Veamos un ejemplo práctico y la manipulación de datos con Python utilizando la librería *Pandas* para analizar y Visualizar la información en pocos minutos.

¿Qué es el EDA?

Eda es la sigla en inglés para *Exploratory Data Analysis* y consiste en una de las primeras tareas que tiene que desempeñar el Científico de Datos. Es cuando revisamos por primera vez los datos que nos llegan, por ejemplo un archivo CSV que nos entregan y deberemos intentar comprender “¿de qué se trata?”, vislumbrar posibles patrones y reconociendo distribuciones estadísticas que puedan ser útiles en el futuro.

OJO!, lo ideal es que tengamos un objetivo que nos hayan “adjuntado” con los datos, que indique lo que se quiere conseguir a partir de esos datos. Por ejemplo, nos pasan un excel y nos dicen “Queremos predecir ventas⁹ a 30 días”, ó “Clasificar¹⁰ casos malignos/benignos de una enfermedad”, “Queremos identificar audiencias¹¹ que van a realizar re-compra de un producto”, “queremos hacer pronóstico¹² de fidelización de clientes/abandonos”, “Quiero detectar casos de fraude¹³ en mi sistema en tiempo real”.

EDA deconstruido

Al llegar un archivo, lo primero que deberíamos hacer es intentar responder:

- ¿Cuántos registros hay?
 - ¿Son demasiado pocos?
 - ¿Son muchos y no tenemos Capacidad (CPU+RAM) suficiente para procesarlo?
 - ¿Están todas las filas completas ó tenemos campos con valores nulos?
 - En caso que haya demasiados nulos: ¿Queda el resto de información inútil?

⁹<https://www.aprendemachinelearning.com/pronostico-de-ventas-redes-neuronales-python-embeddings/>

¹⁰<https://www.aprendemachinelearning.com/arbol-de-decision-en-python-clasificacion-y-prediccion/>

¹¹<https://www.aprendemachinelearning.com/k-means-en-python-paso-a-paso/>

¹²<https://www.aprendemachinelearning.com/regresion-logistica-con-python-paso-a-paso/>

¹³<https://www.aprendemachinelearning.com/clasificacion-con-datos-desbalanceados/>

- ¿Que datos son discretos y cuales continuos?
- Muchas veces sirve obtener el tipo de datos: texto, int, double, float
- Si es un problema de tipo supervisado:
 - ¿Cuál es la columna de “salida”? ¿binaria, multiclas?
 - ¿Esta balanceado el conjunto salida?
- ¿Cuales parecen ser features importantes? ¿Cuales podemos descartar?
- ¿Siguen alguna distribución?
- ¿Hay correlación entre features (características)?
- En [problemas de NLP¹⁴](#) es frecuente que existan categorías repetidas ó mal tipeadas, ó con mayusculas/minúsculas, singular y plural, por ejemplo “Abogado” y “Abogadas”, “avogado” pertenecerían todos a un mismo conjunto.
- ¿Estamos ante un problema dependiente del tiempo? Es decir un [TimeSeries¹⁵](#).
- Si fuera un problema de [Visión Artificial¹⁶](#): ¿Tenemos suficientes muestras de cada clase y variedad, para poder hacer generalizar un modelo de Machine Learning?
- ¿[Cuales son los Outliers¹⁷](#)? (unos pocos datos aislados que difieren drásticamente del resto y “contaminan” ó desvían las distribuciones)
 - Podemos eliminarlos? es importante conservarlos?
 - son errores de carga o son reales?
- ¿Tenemos posible sesgo de datos? (por ejemplo perjudicar a [clases minoritarias¹⁸](#) por no incluirlas y que el modelo de ML discrimine)

Puede ocurrir que tengamos set de datos incompletos y debamos pedir a nuestro cliente/proveedor ó interesado que nos brinde mayor información de los campos, que aporte más conocimiento ó que corrija campos.

[¿Qué son los conjuntos de Train, Test y Validación en Machine Learning?¹⁹](#)

También puede que nos pasen múltiples fuentes de datos, por ejemplo un csv, un excel y el acceso a una base de datos. Entonces tendremos que hacer un paso previo de unificación de datos.

¿Qué sacamos del EDA?

El EDA será entonces una primer aproximación a los datos, ATENCIóN, si estamos mas o menos bien preparados y suponiendo una muestra de datos “suficiente”, puede que en “unas horas” tengamos ya varias conclusiones como por ejemplo:

¹⁴<https://www.aprendemachinelearning.com/procesamiento-del-lenguaje-natural-nlp/>

¹⁵<https://www.aprendemachinelearning.com/pronostico-de-series-temporales-con-redes-neuronales-en-python/>

¹⁶<https://www.aprendemachinelearning.com/como-funcionan-las-convolutional-neural-networks-vision-por-ordenador/>

¹⁷<https://www.aprendemachinelearning.com/deteccion-de-outliers-en-python-anomalia/>

¹⁸<https://www.aprendemachinelearning.com/clasificacion-con-datos-desbalanceados/>

¹⁹<https://www.aprendemachinelearning.com/sets-de-entrenamiento-test-validacion-cruzada/>

- Esto que quiere hacer el cliente CON ESTOS DATOS es una locura imposible! (esto ocurre la mayoría de las veces jeje)
- No tenemos datos suficientes ó son de muy mala calidad, pedir más al cliente.
- Un [modelo de tipo Arbol²⁰](#) es lo más recomendado usar
 - (reemplazar Arbol, por el tipo de modelo que hayamos descubierto como mejor opción!)
- No hace falta usar Machine Learning para resolver lo que pide el cliente. (ESTO ES MUY IMPORTANTE!)
- Es todo tan aleatorio que no habrá manera de detectar patrones
- Hay datos suficientes y de buena calidad como para seguir a la próxima etapa.

A estas alturas podemos saber si nos están pidiendo algo viable ó si necesitamos más datos para comenzar.

Repto: el EDA debe tomar horas, ó puede que un día, pero la idea es poder sacar algunas conclusiones rápidas para contestar al cliente si podemos seguir o no con su propuesta.

Luego del EDA, suponiendo que seguimos adelante podemos tomarnos más tiempo y analizar en mayor detalle los datos y [avanzar a nuevas etapas para aplicar modelos de Machine Learning²¹](#).

Técnicas para EDA

Vamos a lo práctico!, ¿Que herramientas tenemos hoy en día? La verdad es que como cada conjunto de datos suele ser único, el EDA se hace bastante “a mano”, pero podemos seguir diversos pasos ordenados para intentar acercarnos a ese objetivo que nos pasa el cliente en pocas horas.

A nivel programación y como venimos utilizando Python, encontramos a la conocida librería Pandas, que nos ayudará a manipular datos, leer y transformarlos.

Otra de las técnicas que más nos ayudaran en el EDA es visualización de datos (que también podemos hacer con Pandas).

Finalmente podemos decir que nuestra Intuición -basada en Experiencia previa, no en coronadas- y nuestro conocimiento de casos similares también nos pueden aportar pistas para saber si estamos ante datos de buena calidad. Por ejemplo si alguien quiere hacer reconocimiento de imágenes de tornillos y tiene 25 imágenes y con muy mala resolución podremos decir que no tenemos muestras suficientes -dado nuestro conocimiento previo de este campo-.

Vamos a la práctica!

Un EDA de pocos minutos con Pandas

Vamos a hacer un ejemplo en pandas de un EDA bastante sencillo pero con fines educativos.

Vamos a leer un csv directamente desde una URL de GitHub que contiene información geográfica básica de los países del mundo y vamos a jugar un poco con esos datos.

²⁰<https://www.aprendemachinelearning.com/arbol-de-decision-en-python-clasificacion-y-prediccion/>

²¹<https://www.aprendemachinelearning.com/7-pasos-machine-learning-construir-maquina/>

```

1 import pandas as pd
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import statsmodels.api as sm
5
6 url = 'https://raw.githubusercontent.com/lorey/list-of-countries/master/csv/countrye\
7 s.csv'
8 df = pd.read_csv(url, sep=";")
9 print(df.head(5))

```

	alpha_2	alpha_3	area	capital	continent	currency_code	\
0	AD	AND	468.0	Andorra la Vella	EU	EUR	
1	AE	ARE	82880.0	Abu Dhabi	AS	AED	
2	AF	AFG	647500.0	Kabul	AS	AFN	
3	AG	ATG	443.0	St. John's	NaN	XCD	
4	AI	AIA	102.0	The Valley	NaN	XCD	

	currency_name	equivalent_fips_code	fips	geoname_id	languages	\
0	Euro		NaN	AN	3041565	ca
1	Dirham		NaN	AE	290557	ar-AE,fa,en,hi,ur
2	Afghani		NaN	AF	1149361	fa-AF,ps,uz-AF,tk
3	Dollar		NaN	AC	3576396	en-AG
4	Dollar		NaN	AV	3573511	en-AI

	name	neighbours	numeric	phone	population	\
0	Andorra	ES,FR	20	376	84000	
1	United Arab Emirates	SA,OM	784	971	4975593	
2	Afghanistan	TM,CN,IR,TJ,PK,UZ	4	93	29121286	
3	Antigua and Barbuda		Nan	28 +1-268	86754	
4	Anguilla		Nan	660 +1-264	13254	

	postal_code_format	postal_code_regex	tld
0	AD###	^(?:AD)*(\d{3})\$.ad
1	NaN	NaN	.ae
2	NaN	NaN	.af
3	NaN	NaN	.ag
4	NaN	NaN	.ai

Veamos los datos básicos que nos brinda pandas:

Nombre de columnas

```

1 print('Cantidad de Filas y columnas:',df.shape)
2 print('Nombre columnas:',df.columns)

```

```
Cantidad de Filas y columnas: (252, 19)
Nombre columnas: Index(['alpha_2', 'alpha_3', 'area', 'capital', 'continent', 'currency_code',
   'currency_name', 'eqivalent_fips_code', 'fips', 'geoname_id',
   'languages', 'name', 'neighbours', 'numeric', 'phone', 'population',
   'postal_code_format', 'postal_code_regex', 'tld'],
  dtype='object')
```

Columnas, nulos y tipo de datos

```
1 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 252 entries, 0 to 251
Data columns (total 19 columns):
alpha_2           251 non-null object
alpha_3           252 non-null object
area              252 non-null float64
capital           246 non-null object
continent         210 non-null object
currency_code     251 non-null object
currency_name     251 non-null object
eqivalent_fips_code 1 non-null object
fips               249 non-null object
geoname_id        252 non-null int64
languages          249 non-null object
name               252 non-null object
neighbours         165 non-null object
numeric            252 non-null int64
phone              247 non-null object
population         252 non-null int64
postal_code_format 154 non-null object
postal_code_regex  152 non-null object
tld                250 non-null object
dtypes: float64(1), int64(3), object(15)
memory usage: 37.5+ KB
```

En esta salida vemos las columnas, el total de filas y la cantidad de filas sin nulos. También los tipos de datos.

descripción estadística de los datos numéricos

```
1 df.describe()
```

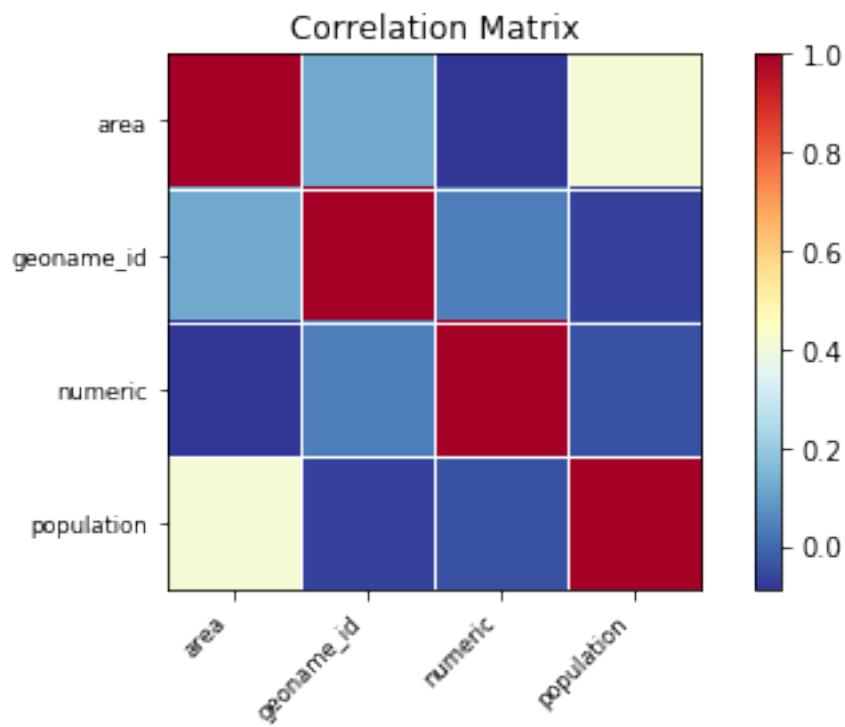
	area	geoname_id	numeric	population
count	2.520000e+02	2.520000e+02	252.000000	2.520000e+02
mean	5.952879e+05	2.427870e+06	434.309524	2.727679e+07
std	1.904818e+06	1.632093e+06	254.663139	1.164127e+08
min	0.000000e+00	4.951800e+04	0.000000	0.000000e+00
25%	1.098000e+03	1.163774e+06	217.000000	1.879528e+05
50%	6.489450e+04	2.367967e+06	436.000000	4.268583e+06
75%	3.622245e+05	3.478296e+06	652.500000	1.536688e+07
max	1.710000e+07	8.505033e+06	894.000000	1.330044e+09

Pandas filtra las features numéricas y calcula datos estadísticos que pueden ser útiles: cantidad, media, desvío estándar, valores máximo y mínimo.

Verifiquemos si hay correlación entre los datos

```

1 corr = df.set_index('alpha_3').corr()
2 sm.graphics.plot_corr(corr, xnames=list(corr.columns))
3 plt.show()
```



En este caso vemos baja correlación entre las variables. Dependiendo del algoritmo que utilicemos podría ser una buena decisión eliminar features que tuvieran alta correlación

Cargamos un segundo archivo csv para ahondar en el crecimiento de la población en los últimos años, filtramos a España y visualizamos

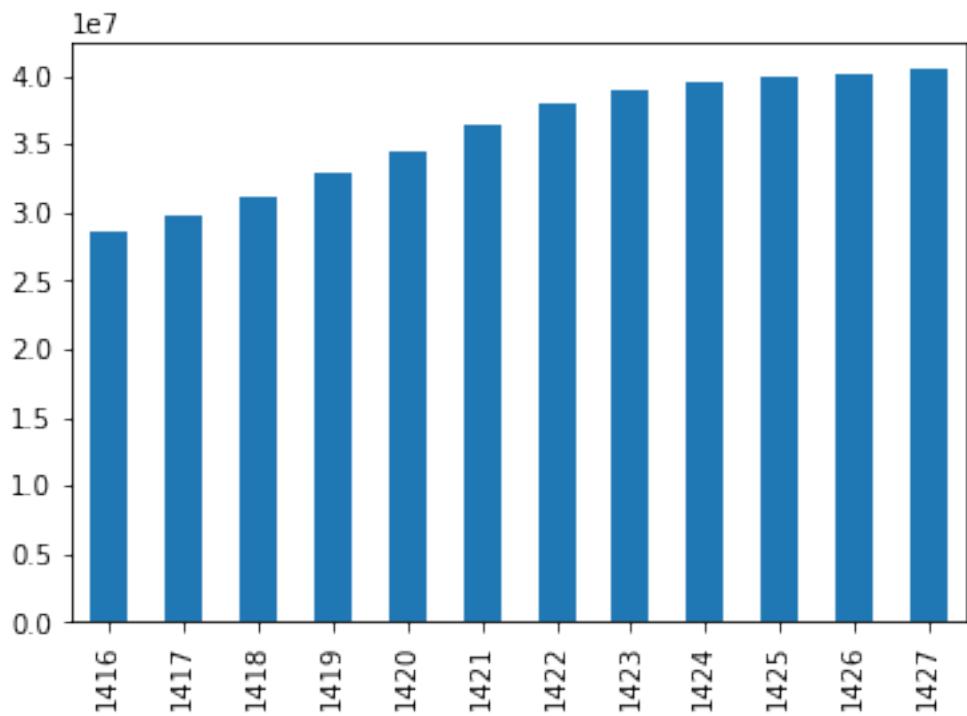
```

1 url = 'https://raw.githubusercontent.com/DrueStaples/Population_Growth/master/countr\
2 ies.csv'
3 df_pop = pd.read_csv(url)
4 print(df_pop.head(5))
5 df_pop_es = df_pop[df_pop["country"] == 'Spain' ]
6 print(df_pop_es.head())
7 df_pop_es.drop(['country'],axis=1)[['population']].plot(kind='bar')

```

	country	year	population
0	Afghanistan	1952	8425333
1	Afghanistan	1957	9240934
2	Afghanistan	1962	10267083
3	Afghanistan	1967	11537966
4	Afghanistan	1972	13079460

	country	year	population
1416	Spain	1952	28549870
1417	Spain	1957	29841614
1418	Spain	1962	31158061
1419	Spain	1967	32850275
1420	Spain	1972	34513161



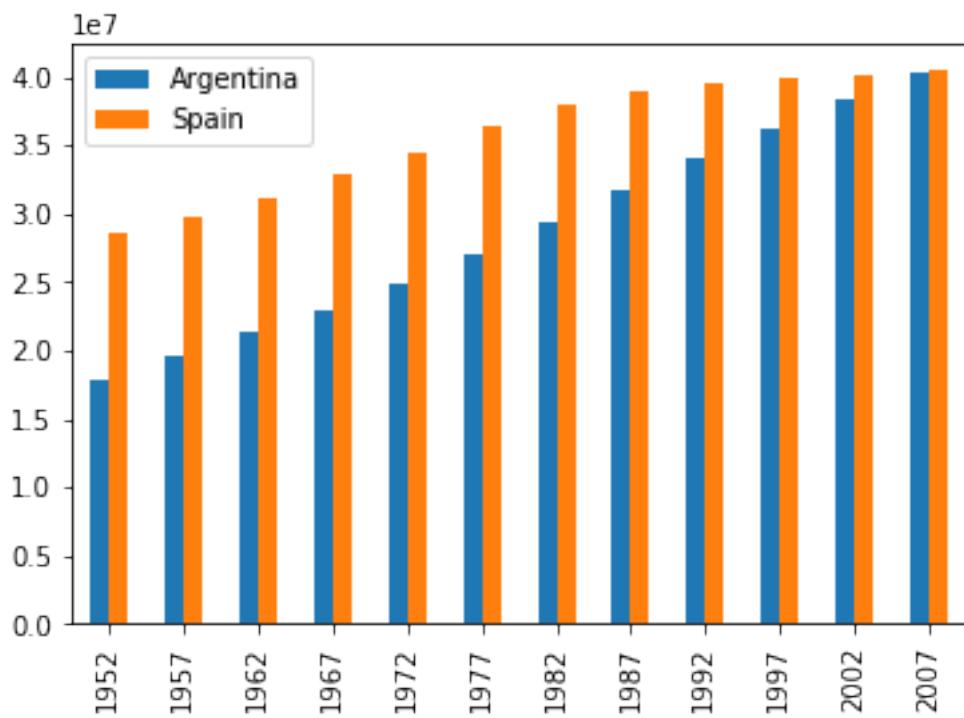
Crecimiento de la Población de España. El eje x no está establecido y aparece un id de fila.

Hagamos la comparativa con otro país, por ejemplo con el crecimiento poblacional en Argentina

```

1 df_pop_ar = df_pop[(df_pop["country"] == 'Argentina')]
2
3 anios = df_pop_es['year'].unique()
4 pop_ar = df_pop_ar['population'].values
5 pop_es = df_pop_es['population'].values
6
7 df_plot = pd.DataFrame({'Argentina': pop_ar,
8                         'Spain': pop_es},
9                         index=anios)
10 df_plot.plot(kind='bar')

```



Gráfica comparativa de crecimiento poblacional entre España y Argentina entre los años 1952 al 2007

Ahora filtremos todos los países hispano-hablantes

```

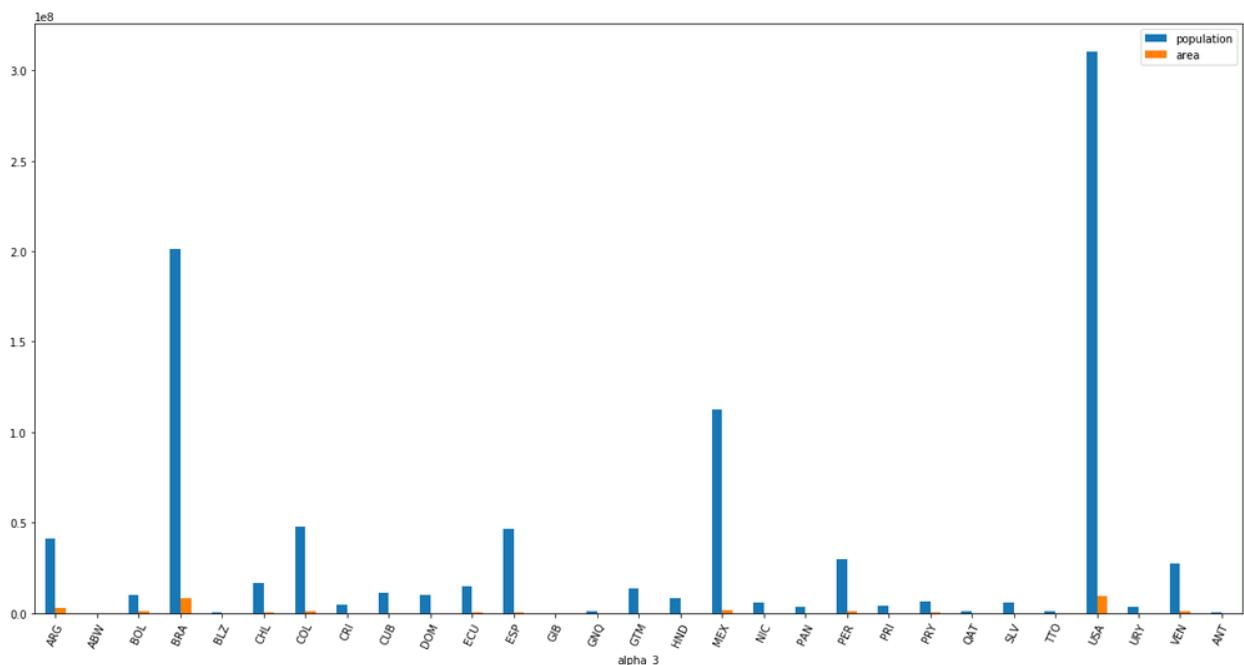
1 df_espanol = df.replace(np.nan, '', regex=True)
2 df_espanol = df_espanol[ df_espanol['languages'].str.contains('es') ]
3 df_espanol

```

alpha_2	alpha_3	area	capital	continent	currency_code	currency_name	equivalent_fips_code	fips	geoname_id	languages	name
9	AR	ARG	2766890.0	Buenos Aires	SA	ARS	Peso	AR	3865483	es-AR,en,it,de,fr,gn	Argentina
13	AW	ABW	193.0	Oranjestad		AWG	Guilder	AA	3577279	nl-AW,es,en	Aruba
28	BO	BOL	1098580.0	Sucre	SA	BOB	Boliviano	BL	3923057	es-BO,qu,ay	Bolivia
30	BR	BRA	8511965.0	Brasilia	SA	BRL	Real	BR	3469034	pt-BR,es,en,fr	Brazil SR,
36	BZ	BLZ	22966.0	Belmopan		BZD	Dollar	BH	3582678	en-BZ,es	Belize
45	CL	CHL	756950.0	Santiago	SA	CLP	Peso	CI	3895114	es-CL	Chile
48	CO	COL	1138910.0	Bogota	SA	COP	Peso	CO	3686110	es-CO	Colombia
49	CR	CRI	51100.0	San Jose		CRC	Colon	CS	3624060	es-CR,en	Costa Rica
50	CU	CUB	110860.0	Havana		CUP	Peso	CU	3562981	es-CU	Cuba
60	DO	DOM	48730.0	Santo Domingo		DOP	Peso	DR	3508796	es-DO	Dominican Republic
62	EC	ECU	283560.0	Quito	SA	USD	Dollar	EC	3658394	es-EC	Ecuador
67	ES	ESP	504782.0	Madrid	EU	EUR	Euro	SP	2510769	es-ES,ca,gl,eu,oc	Spain

Visualizamos...

```
1 df_espanol.set_index('alpha_3')[['population', 'area']].plot(kind='bar', rot=65, figsize\\
2 e=(20,10))
```



Vamos a hacer **detección de Outliers**²², (con fines educativos) en este caso definimos como límite superior (e inferior) la media más (menos) “*2 veces la desviación estándar*” que muchas veces es tomada como máximos de tolerancia.

²²<https://www.aprendemachinelearning.com/deteccion-de-outliers-en-python-anomalia/>

```

1 anomalies = []
2
3 # Funcion ejemplo para detección de outliers
4 def find_anomalies(data):
5     # Set upper and lower limit to 2 standard deviation
6     data_std = data.std()
7     data_mean = data.mean()
8     anomaly_cut_off = data_std * 2
9     lower_limit = data_mean - anomaly_cut_off
10    upper_limit = data_mean + anomaly_cut_off
11    print(lower_limit.iloc[0])
12    print(upper_limit.iloc[0])
13
14    # Generate outliers
15    for index, row in data.iterrows():
16        outlier = row # # obtener primer columna
17        # print(outlier)
18        if (outlier.iloc[0] > upper_limit.iloc[0]) or (outlier.iloc[0] < lower_limit\
19 .iloc[0]):
20            anomalies.append(index)
21    return anomalies
22
23 find_anomalies(df_espanol.set_index('alpha_3')[['population']])

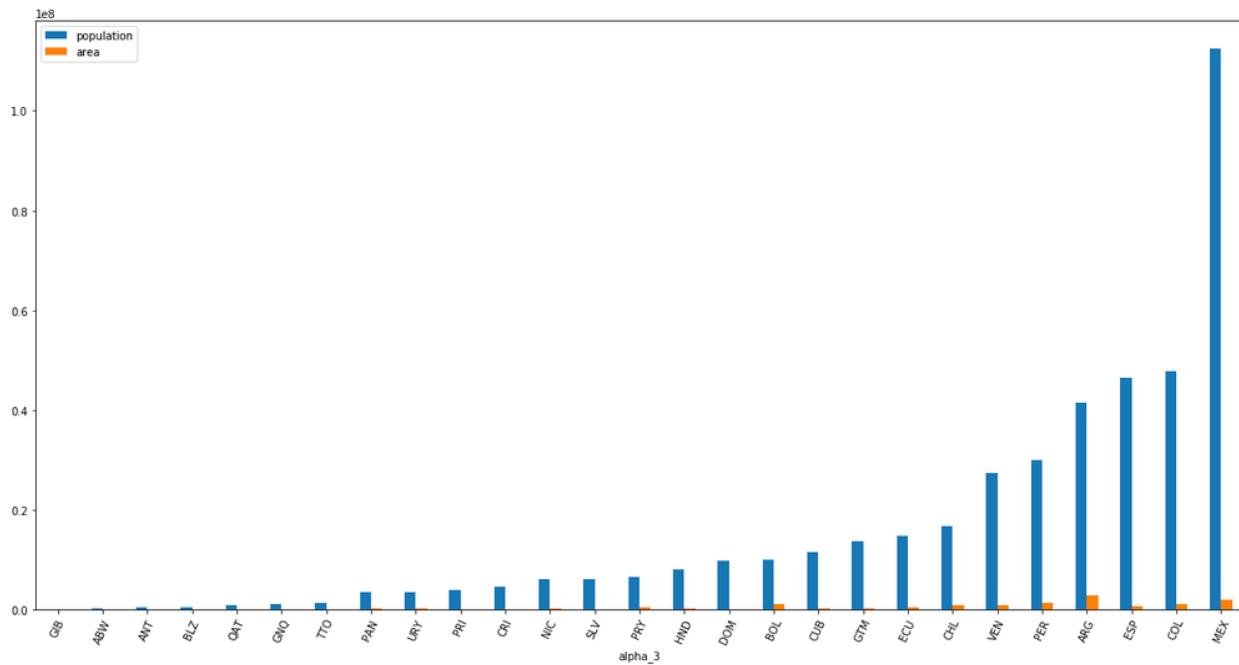
```

Detectamos como outliers a Brasil y a USA. Los eliminamos y graficamos ordenado por población de menor a mayor.

```

1 # Quitemos BRA y USA por ser outliers y volvamos a graficar:
2 df_espanol.drop([30,233], inplace=True)
3 df_espanol.set_index('alpha_3')[['population','area']].sort_values(["population"]).p\
4 lot(kind='bar',rot=65,figsize=(20,10))

```



Así queda nuestra gráfica sin outliers :)

En pocos minutos hemos podido responder: cuántos datos tenemos, si hay nulos, los tipos de datos (entero, float, string), la correlación, hicimos visualizaciones, comparativas, manipulación de datos, detección de outliers y volver a graficar. ¡No está nada mal, no?

Más cosas! (que se suelen hacer):

Otras pruebas y gráficas que se suelen hacer son:

- Si hay datos categóricos, agruparlos, contabilizarlos y ver su relación con las clases de salida
- gráficas de distribución en el tiempo, por ejemplo si tuviéramos ventas, para tener una primera impresión sobre su estacionalidad.
- Rankings del tipo “10 productos más vendidos” ó “10 ítems con más referencias por usuario”.
- Calcular importancia de Features y descartar las menos útiles.

Resumen

Vimos un repaso sobre qué es y cómo lograr hacer un Análisis Exploratorio de Datos en pocos minutos. Su importancia es sobre todo la de darnos un vistazo sobre la calidad de datos que tenemos y hasta puede determinar la continuidad o no de un proyecto.

Siempre dependerá de los datos que tengamos, en cantidad y calidad y por supuesto nunca deberemos dejar de tener en vista **EL OBJETIVO**, el propósito que buscamos lograr. Siempre debemos apuntar a lograr eso con nuestras acciones.

Como resultado del EDA si determinamos continuar, pasaremos a una etapa en la que ya preprocesaremos los datos pensando en la entrada a un modelo (ó modelos!) de Machine Learning.

Recursos

Puedes descargar la notebook relacionada con este artículo desde aquí:

- Descargar notebook ejemplo EDA para Machine Learning²³ (GitHub)

BONUS track: Notebook sobre manipulación de datos con Pandas

Como Bonus.... te dejo una notebook con los Casos más comunes de uso de Manipulación de datos con Pandas!

- Descargar Notebook Educativa sobre uso de Panda²⁴s

Más Recursos

Estos son otros artículos relacionados que pueden ser de tu interés:

- EDA House Prices Data (python)²⁵
- ML project in Python²⁶
- EDA example in Python²⁷
- EDA Tutorial²⁸

²³https://github.com/jbagnato/machine-learning/blob/master/Ejercicio_EDA.ipynb

²⁴https://github.com/jbagnato/machine-learning/blob/master/Manipulacion_datos_pandas.ipynb

²⁵<https://www.hackerearth.com/practice/machine-learning/machine-learning-projects/python-project/tutorial/>

²⁶<https://machinelearningmastery.com/machine-learning-in-python-step-by-step/>

²⁷https://www.activestate.com/blog/exploratory-data-analysis-using-python/?utm_campaign=exploratory-data-analysis-blog&utm_medium=referral&utm_source=kdnuggets&utm_content=2019-08-07-kdnuggets-article

²⁸<https://www.datacamp.com/community/tutorials/exploratory-data-analysis-python>

Regresión Lineal con Python

¿Qué es la regresión lineal?

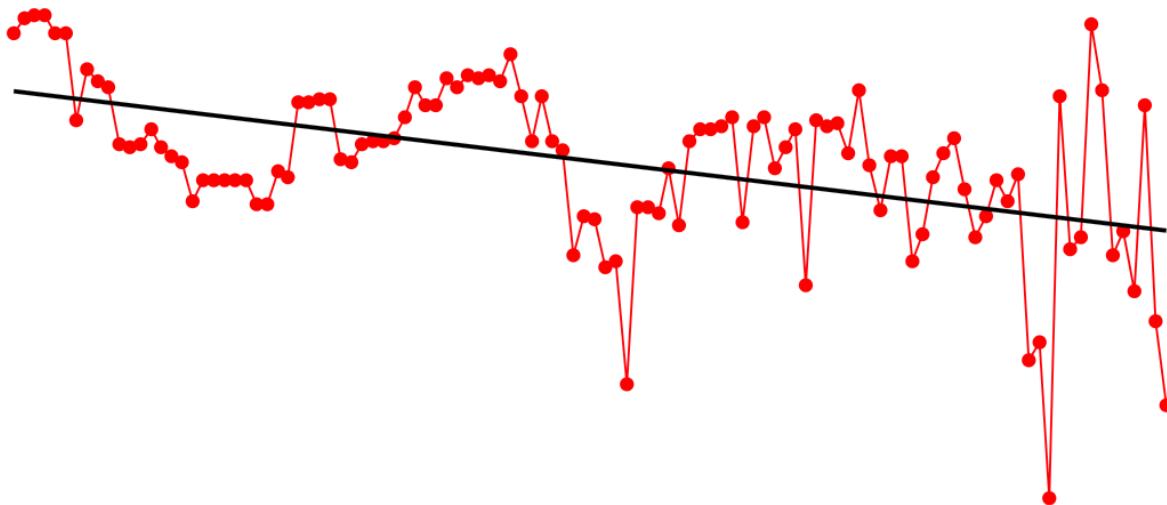
La [regresión lineal²⁹](#) es un [algoritmo³⁰](#) de [aprendizaje supervisado³¹](#) que se utiliza en Machine Learning y en estadística. En su versión más sencilla, lo que haremos es “dibujar una recta” que *nos indicará la tendencia* de un conjunto de datos continuos (si fueran discretos, utilizaríamos [Regresión Logística³²](#)). En estadísticas, *regresión lineal es una aproximación para modelar la relación entre una variable escalar dependiente “y” y una o mas variables explicativas nombradas con “X”*.

Recordemos rápidamente la fórmula de la recta:

$$Y = mX + b$$

Donde Y es el resultado, X es la variable, m la pendiente (o coeficiente) de la recta y b la constante o también conocida como el “punto de corte con el eje Y” en la gráfica (cuando X=0)

The development in Pizza prices in Denmark from 2009 to 2018



Aquí vemos un ejemplo donde vemos datos recabados sobre los precios de las pizzas en Dinamarca (los puntos en rojo) y la linea negra es la tendencia. Esa es la línea de regresión que buscamos que el algoritmo aprenda y calcule sólo.

²⁹https://es.wikipedia.org/wiki/Regresi%C3%B3n_lineal

³⁰<http://www.aprendemachinelearning.com/principales-algoritmos-usados-en-machine-learning/>

³¹<http://www.aprendemachinelearning.com/aplicaciones-del-machine-learning/#supervisado>

³²<http://www.aprendemachinelearning.com/regresion-logistica-con-python-paso-a-paso/>

¿Cómo funciona el algoritmo de regresión lineal en Machine Learning?

Recordemos que los [algoritmos de Machine Learning Supervisados³³](#), aprenden por sí mismos y -en este caso- a obtener automáticamente esa “recta” que buscamos con la tendencia de predicción. Para hacerlo se mide el error con respecto a los puntos de entrada y el valor “Y” de salida real. El algoritmo deberá minimizar el coste de una función de [error cuadrático³⁴](#) y esos coeficientes corresponderán con la recta óptima. Hay diversos métodos para conseguir minimizar el coste. Lo más común es utilizar una versión vectorial y la llamada [Ecuación Normal³⁵](#) que nos dará un resultado directo.

NOTA: cuando hablo de “recta” es en el caso particular de regresión lineal simple. Si hubiera más variables, hay que generalizar el término.

Un Ejercicio Práctico

En este ejemplo cargaremos un [archivo .csv de entrada³⁶](#) obtenido por [webscraping³⁷](#) que contiene diversas URLs a artículos sobre Machine Learning de algunos sitios muy importantes como [Techcrunch³⁸](#) o [KDnuggets³⁹](#) y como características de entrada -las columnas- tendremos:

- **Title:** Titulo del Artículo
- **url:** ruta al artículo
- **Word count:** la cantidad de palabras del artículo,
- **# of Links:** los enlaces externos que contiene,
- **# of comments:** cantidad de comentarios,
- **# Images video:** suma de imágenes (o videos),
- **Elapsed days:** la cantidad de días transcurridos (al momento de crear el archivo)
- **# Shares:** nuestra columna de salida que será la “cantidad de veces que se compartió el artículo”.

A partir de las características de un artículo de machine learning intentaremos predecir, cuantas veces será compartido en Redes Sociales. Haremos una primer predicción de [regresión lineal simple⁴⁰](#) -con una sola variable predictora- para poder graficar en 2 dimensiones (ejes X e Y) y luego un ejemplo de [regresión Lineal Múltiple](#), en la que utilizaremos 3 dimensiones (X,Y,Z) y predicciones.

NOTA: el archivo .csv contiene mitad de datos reales, y otra mitad los generé de manera aleatoria, por lo que las predicciones que obtendremos no serán reales. Intentaré en el futuro hacer webscrapping de los enlaces que me faltaban y reemplazar los resultados por valores reales.

³³<http://www.aprendemachinelearning.com/aplicaciones-del-machine-learning/#supervisado>

³⁴https://es.wikipedia.org/wiki/Error_cuadr%C3%A1tico_medio

³⁵[https://en.wikipedia.org/wiki/Linear_least_squares_\(mathematics\)#Derivation_of_the_normal_equations](https://en.wikipedia.org/wiki/Linear_least_squares_(mathematics)#Derivation_of_the_normal_equations)

³⁶http://www.aprendemachinelearning.com/articulos_ml/

³⁷<http://www.aprendemachinelearning.com/ejemplo-web-scraping-python-ibex35-bolsa-valores/>

³⁸<https://techcrunch.com/tag/machine-learning/>

³⁹<https://www.kdnuggets.com>

⁴⁰https://en.wikipedia.org/wiki/Simple_linear_regression

Requerimientos para hacer el Ejercicio

Para realizar este ejercicio, crearemos una [Jupyter notebook⁴¹](#) con código Python y la librería Scikit-Learn muy utilizada en Data Science. Recomendamos utilizar la suite de [Anaconda⁴²](#). Podrás descargar los archivos de [entrada csv⁴³](#) o visualizar la [notebook online⁴⁴](#).

Predecir cuántas veces será compartido un artículo de Machine Learning.

Regresión lineal simple en Python (con 1 variable)

Aquí vamos con nuestra notebook! Comencemos por importar las librerías que utilizaremos:

```

1 # Imports necesarios
2 import numpy as np
3 import pandas as pd
4 import seaborn as sb
5 import matplotlib.pyplot as plt
6 %matplotlib inline
7 from mpl_toolkits.mplot3d import Axes3D
8 from matplotlib import cm
9 plt.rcParams['figure.figsize'] = (16, 9)
10 plt.style.use('ggplot')
11 from sklearn import linear_model
12 from sklearn.metrics import mean_squared_error, r2_score

```

Leemos el archivo csv y lo cargamos como un dataset de Pandas. Y vemos su tamaño

```

1 #cargamos los datos de entrada
2 data = pd.read_csv("./articulos_ml.csv")
3 #veamos cuantas dimensiones y registros contiene
4 data.shape

```

Nos devuelve (161,8) Veamos esas primeras filas:

⁴¹<http://data-speaks.luca-d3.com/2018/03/python-para-todos-2-jupyter-notebook.html>

⁴²<https://www.anaconda.com/download/>

⁴³http://www.aprendemachinelearning.com/articulos_ml/

⁴⁴https://github.com/jbagnato/machine-learning/blob/master/Ejercicio_Regresion_Lineal.ipynb

```

1 #son 161 registros con 8 columnas. Veamos los primeros registros
2 data.head()

```

	Title	url	Word count	# of Links	# of comments	# Images video	Elapsed days	# Shares
0	What is Machine Learning and how do we use it ...	https://blog.signals.network/what-is-machine-l...	1888	1	2.0	2	34	200000
1	10 Companies Using Machine Learning in Cool Ways	NaN	1742	9	NaN	9	5	25000
2	How Artificial Intelligence Is Revolutionizing...	NaN	962	6	0.0	1	10	42000
3	Dbrain and the Blockchain of Artificial Intell...	NaN	1221	3	NaN	2	68	200000
4	Nasa finds entire solar system filled with eig...	NaN	2039	1	104.0	4	131	200000 ⁴⁵

Se ven algunos campos con valores NaN (nulos) por ejemplo algunas urls o en comentarios. Veamos algunas estadísticas básicas de nuestros datos de entrada:

```

1 # Ahora veamos algunas estadísticas de nuestros datos
2 data.describe()

```

	Word count	# of Links	# of comments	# Images video	Elapsed days	# Shares
count	161.000000	161.000000	129.000000	161.000000	161.000000	161.000000
mean	1808.260870	9.739130	8.782946	3.670807	98.124224	27948.347826
std	1141.919385	47.271625	13.142822	3.418290	114.337535	43408.006839
min	250.000000	0.000000	0.000000	1.000000	1.000000	0.000000
25%	990.000000	3.000000	2.000000	1.000000	31.000000	2800.000000
50%	1674.000000	5.000000	6.000000	3.000000	62.000000	16458.000000
75%	2369.000000	7.000000	12.000000	5.000000	124.000000	35691.000000
max	8401.000000	600.000000	104.000000	22.000000	1002.000000	350000.000000 ⁴⁶

Aquí vemos que la media de palabras en los artículos es de 1808. El artículo más corto tiene 250 palabras y el más extenso 8401. Intentaremos ver con nuestra relación lineal, si hay una correlación entre la cantidad de palabras del texto y la cantidad de Shares obtenidos. Hacemos una visualización en general de los datos de entrada:

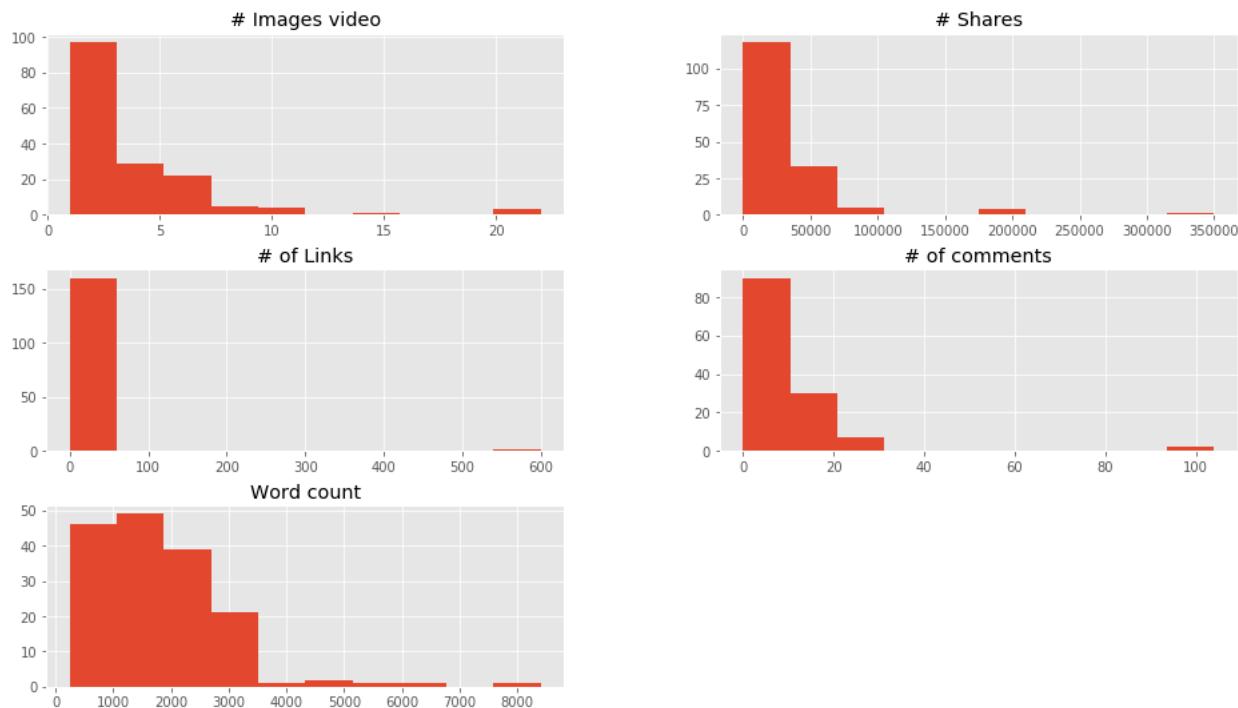
```

1 # Visualizamos rápidamente las características de entrada
2 data.drop(['Title','url', 'Elapsed days'],1).hist()
3 plt.show()

```

⁴⁵http://www.aprendemachinelearning.com/wp-content/uploads/2018/05/reg_lineal_filas_iniciales.png

⁴⁶http://www.aprendemachinelearning.com/wp-content/uploads/2018/05/reg_lineal_stats_base.png



47

En estas gráficas vemos entre qué valores se concentran la mayoría de registros. Vamos a filtrar los datos de cantidad de palabras para quedarnos con los registros con menos de 3500 palabras y también con los que tengan Cantidad de compartidos menos a 80.000. Lo gratificaremos pintando en azul los puntos con menos de 1808 palabras (la media) y en naranja los que tengan más.

```

1 # Vamos a RECORTAR los datos en la zona donde se concentran más los puntos
2 # esto es en el eje X: entre 0 y 3.500
3 # y en el eje Y: entre 0 y 80.000
4 filtered_data = data[(data['Word count'] <= 3500) & (data['# Shares'] <= 80000)]
5
6 colores=['orange','blue']
7 tamanios=[30,60]
8
9 f1 = filtered_data['Word count'].values
10 f2 = filtered_data['# Shares'].values
11
12 # Vamos a pintar en colores los puntos por debajo y por encima de la media de Cantidad de Palabras
13 asignar=[]
14 for index, row in filtered_data.iterrows():
15     if(row['Word count']>1808):
16         asignar.append(colores[0])
17     else:
18         asignar.append(colores[1])

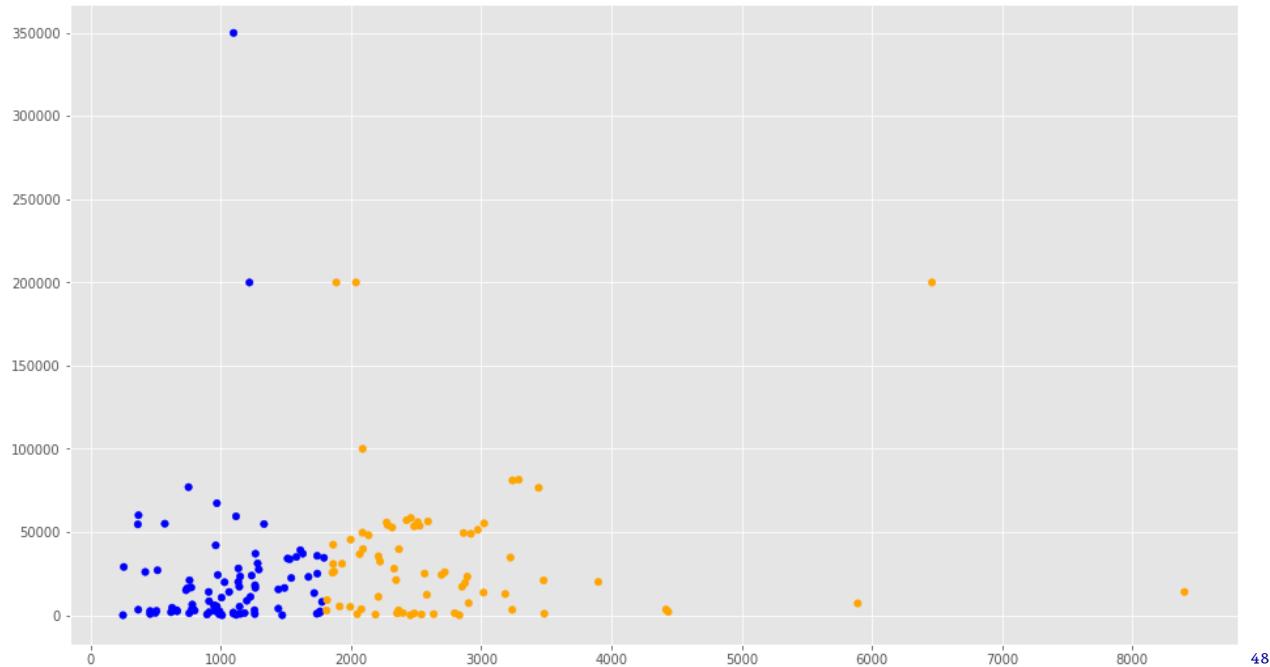
```

⁴⁷http://www.aprendemachinelearning.com/wp-content/uploads/2018/05/reg_lin_visualiza_entradas.png

```

19         asignar.append(colores[1])
20
21 plt.scatter(f1, f2, c=asignar, s=tamanios[0])
22 plt.show()

```



Regresión Lineal con Python y SKLearn

Vamos a crear nuestros datos de entrada por el momento sólo Word Count y como etiquetas los # Shares. Creamos el objeto LinearRegression y lo hacemos “encajar” (entrenar) con el método fit(). Finalmente imprimimos los coeficientes y puntajes obtenidos.

```

1 # Asignamos nuestra variable de entrada X para entrenamiento y las etiquetas Y.
2 dataX = filtered_data[["Word count"]]
3 X_train = np.array(dataX)
4 y_train = filtered_data['# Shares'].values
5
6 # Creamos el objeto de Regresión Lineal
7 regr = linear_model.LinearRegression()
8
9 # Entrenamos nuestro modelo
10 regr.fit(X_train, y_train)

```

⁴⁸http://www.aprendemachinelearning.com/wp-content/uploads/2018/05/reg_lineal_grafica_pal_vs_shares.png

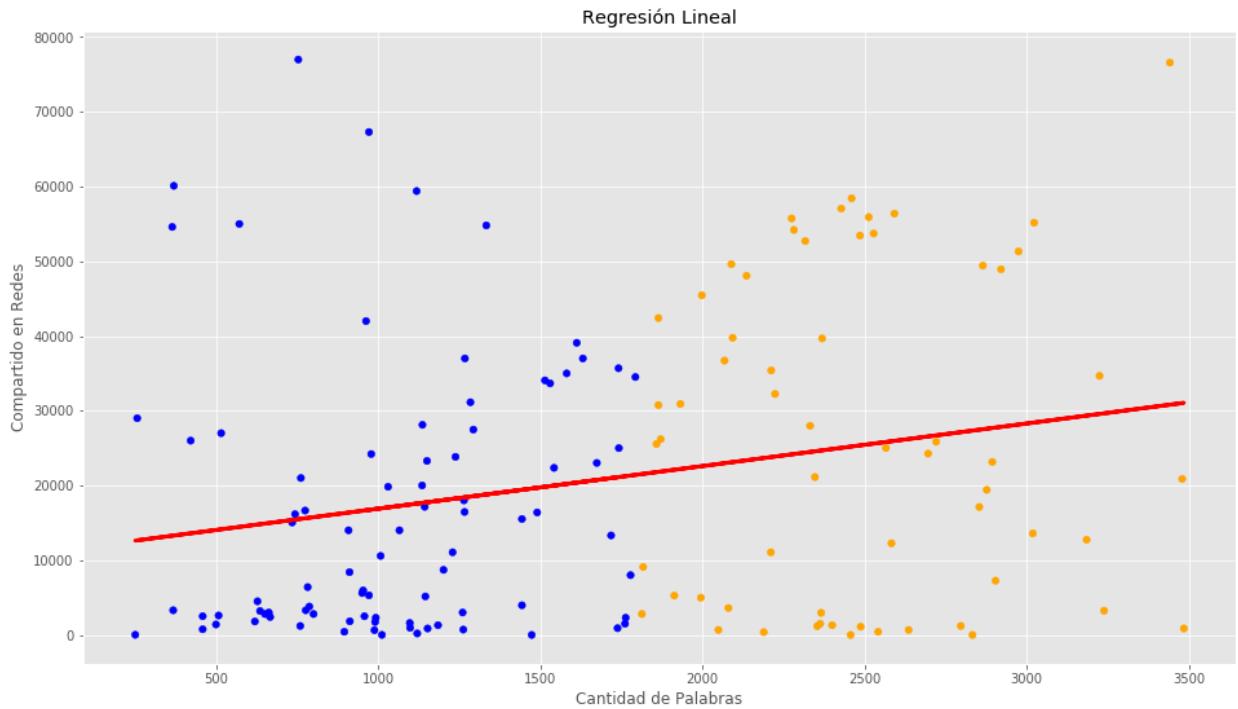
```
11
12 # Hacemos las predicciones que en definitiva una línea (en este caso, al ser 2D)
13 y_pred = regr.predict(X_train)
14
15 # Veamos los coeficientes obtenidos, En nuestro caso, serán la Tangente
16 print('Coefficients: \n', regr.coef_)
17 # Este es el valor donde corta el eje Y (en X=0)
18 print('Independent term: \n', regr.intercept_)
19 # Error Cuadrado Medio
20 print("Mean squared error: %.2f" % mean_squared_error(y_train, y_pred))
21 # Puntaje de Varianza. El mejor puntaje es un 1.0
22 print('Variance score: %.2f' % r2_score(y_train, y_pred))
```

```
1 Coefficients: [5.69765366]
2 Independent term: 11200.303223074163
3 Mean squared error: 372888728.34
4 Variance score: 0.06
```

De la ecuación de la recta $y = mX + b$ nuestra pendiente “m” es el coeficiente 5,69 y el término independiente “b” es 11200. Tenemos un Error Cuadrático medio enorme... por lo que en realidad este modelo no será muy bueno ;) Pero estamos aprendiendo a usarlo, que es lo que nos importa ahora :) Esto también se ve reflejado en el puntaje de Varianza que debería ser cercano a 1.0.

Visualicemos la Recta

Veamos la recta que obtuvimos:



49

Predicción en regresión lineal simple

Vamos a intentar probar nuestro algoritmo, suponiendo que quisiéramos predecir cuántos “compartir” obtendrá un artículo sobre ML de 2000 palabras

```

1 #Vamos a comprobar:
2 # Quiero predecir cuántos "Shares" voy a obtener por un artículo con 2.000 palabras,
3 # según nuestro modelo, hacemos:
4 y_Dosmil = regr.predict([[2000]])
5 print(int(y_Dosmil))

```

Nos devuelve una predicción de 22595 “Shares” para un artículo de 2000 palabras.

⁴⁹http://www.aprendemachinelearning.com/wp-content/uploads/2018/05/reg_lineal_recta_1_variable.png

Regresión Lineal Múltiple en Python

(o “Regresión con Múltiples Variables”)

Vamos a extender el ejercicio utilizando más de una variable de entrada para el modelo. Esto le da **mayor poder** al algoritmo de Machine Learning, pues de esta manera podremos obtener predicciones más complejas. Nuestra “ecuación de la Recta”, ahora pasa a ser:

$$Y = b + m_1 X_1 + m_2 X_2 + \dots + m(n) X(n)$$

y deja de ser una recta) **En nuestro caso, utilizaremos 2 “variables predictivas” para poder graficar en 3D**, pero recordar que *para mejores predicciones podemos utilizar más de 2 entradas* y prescindir del grafico. Nuestra primer variable seguirá siendo la **cantidad de palabras** y la segunda variable será la **suma de 3 columnas de entrada**: la cantidad de enlaces, comentarios y cantidad de imágenes. Vamos a programar!

```

1 #Vamos a intentar mejorar el Modelo, con una dimensión más:
2 # Para poder graficar en 3D, haremos una variable nueva que será la suma de los enla\
3 ces, comentarios e imágenes
4 suma = (filtered_data["# of Links"] + filtered_data['# of comments'].fillna(0) + fil\
5 tered_data['# Images video'])
6
7 dataX2 = pd.DataFrame()
8 dataX2["Word count"] = filtered_data["Word count"]
9 dataX2["suma"] = suma
10 XY_train = np.array(dataX2)
11 z_train = filtered_data['# Shares'].values

```

Nota: hubiera sido mejor **aplicar PCA** para reducción de dimensiones⁵⁰, manteniendo la información más importante de todas

Ya tenemos nuestras 2 variables de entrada en XY_train y **nuestra variable de salida pasa de ser “Y” a ser el eje “Z”**. Creamos un nuevo objeto de Regresión lineal con SKLearn pero esta vez tendrá las dos dimensiones que entrenar: las que contiene XY_train. Al igual que antes, imprimimos los coeficientes y puntajes obtenidos:

⁵⁰<http://www.aprendemachinelearning.com/comprende-principal-component-analysis/>

```
1 # Creamos un nuevo objeto de Regresión Lineal
2 regr2 = linear_model.LinearRegression()
3
4 # Entrenamos el modelo, esta vez, con 2 dimensiones
5 # obtendremos 2 coeficientes, para graficar un plano
6 regr2.fit(XY_train, z_train)
7
8 # Hacemos la predicción con la que tendremos puntos sobre el plano hallado
9 z_pred = regr2.predict(XY_train)
10
11 # Los coeficientes
12 print('Coefficients: \n', regr2.coef_)
13 # Error cuadrático medio
14 print("Mean squared error: %.2f" % mean_squared_error(z_train, z_pred))
15 # Evaluamos el puntaje de varianza (siendo 1.0 el mejor posible)
16 print('Variance score: %.2f' % r2_score(z_train, z_pred))

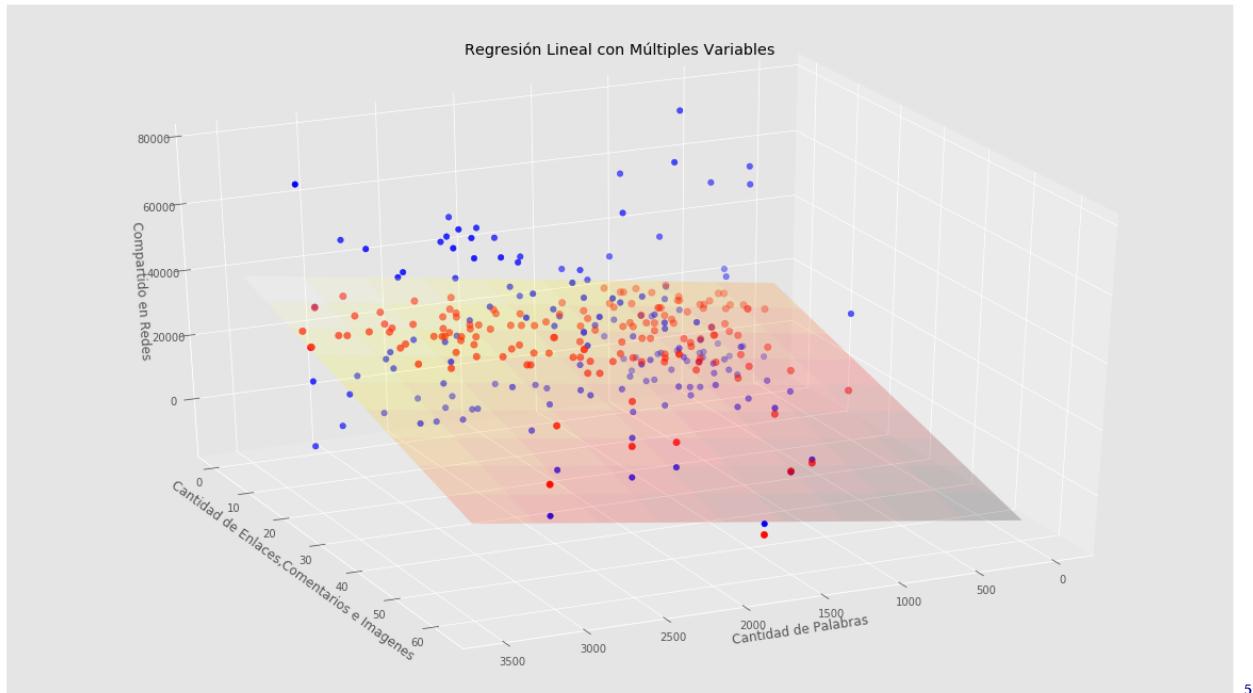
1 Coefficients: [ 6.63216324 -483.40753769]
2 Mean squared error: 352122816.48
3 Variance score: 0.11
```

Como vemos, obtenemos 2 coeficientes (cada uno correspondiente a nuestras 2 variables predictivas), pues ahora lo que graficamos no será una linea si no, **un plano en 3 Dimensiones**. El error obtenido sigue siendo grande, aunque algo mejor que el anterior y el puntaje de Varianza mejora casi el doble del anterior (aunque sigue siendo muy malo, muy lejos del 1).

Visualizar un plano en 3 Dimensiones en Python

Graficaremos nuestros puntos de las características de entrada en color azul y los puntos proyectados en el plano en rojo. Recordemos que en esta gráfica, el eje Z corresponde a la “altura” y representa la cantidad de Shares que obtendremos.

```
1 fig = plt.figure()
2 ax = Axes3D(fig)
3
4 # Creamos una malla, sobre la cual graficaremos el plano
5 xx, yy = np.meshgrid(np.linspace(0, 3500, num=10), np.linspace(0, 60, num=10))
6
7 # calculamos los valores del plano para los puntos x e y
8 nuevoX = (regr2.coef_[0] * xx)
9 nuevoY = (regr2.coef_[1] * yy)
10
11 # calculamos los correspondientes valores para z. Debemos sumar el punto de intercep\
12 ción
13 z = (nuevoX + nuevoY + regr2.intercept_)
14
15 # Graficamos el plano
16 ax.plot_surface(xx, yy, z, alpha=0.2, cmap='hot')
17
18 # Graficamos en azul los puntos en 3D
19 ax.scatter(XY_train[:, 0], XY_train[:, 1], z_train, c='blue', s=30)
20
21 # Graficamos en rojo, los puntos que
22 ax.scatter(XY_train[:, 0], XY_train[:, 1], z_pred, c='red', s=40)
23
24 # con esto situamos la "camara" con la que visualizamos
25 ax.view_init(elev=30., azim=65)
26
27 ax.set_xlabel('Cantidad de Palabras')
28 ax.set_ylabel('Cantidad de Enlaces, Comentarios e Imagenes')
29 ax.set_zlabel('Compartido en Redes')
30 ax.set_title('Regresión Lineal con Múltiples Variables')
```



Podemos rotar el gráfico para apreciar el plano desde diversos ángulos modificando el valor del parámetro `azim` en `view_init` con números de 0 a 360.

Predicción con el modelo de Múltiples Variables

Veamos ahora, que predicción tendremos para un artículo de 2000 palabras, con 10 enlaces, 4 comentarios y 6 imágenes.

```

1 # Si quiero predecir cuántos "Shares" voy a obtener por un artículo con:
2 # 2000 palabras y con enlaces: 10, comentarios: 4, imágenes: 6
3 # según nuestro modelo, hacemos:
4
5 z_Dosmil = regr2.predict([[2000, 10+4+6]])
6 print(int(z_Dosmil))

```

Esta predicción nos da 20518 y probablemente sea un poco mejor que nuestra predicción anterior con 1 variables.

Resumen

Hemos visto cómo utilizar SKLearn en Python para crear modelos de Regresión Lineal con 1 o múltiples variables. En nuestro ejercicio no tuvimos una gran confianza en las predicciones. Por

⁵¹http://www.aprendemachinelearning.com/wp-content/uploads/2018/05/regresion_lineal_3d_plano.png

ejemplo en nuestro primer modelo, con 2000 palabras nos predice que podemos tener 22595 pero el margen de error haciendo raíz del error cuartico medio es más menos 19310. Es decir que escribiendo un artículo de 2000 palabras lo mismo tenemos 3285 Shares que 41905. En este caso usamos este modelo para aprender a usarlo y habrá que ver en otros casos en los que sí nos brinde predicciones acertadas. Para mejorar nuestro modelo, deberíamos utilizar más dimensiones y encontrar datos de entrada mejores.

Atención: también es posible, que no exista ninguna relación fuerte entre nuestras variables de entrada y el éxito en Shares del artículo... Esto fue un experimento!

Recursos y enlaces

- Descarga la [Jupyter Notebook⁵²](#) y el [archivo de entrada csv⁵³](#)
- ó puedes [visualizar online⁵⁴](#)
- o ver y descargar desde mi cuenta [github⁵⁵](#)

Otros enlaces con Artículos sobre Regresión Lineal (en Inglés)

- [Introduction to Linear Regression using python⁵⁶](#)
- [Linear Regression using Python SkLearn⁵⁷](#)
- [Linear Regression Detailed View⁵⁸](#)
- [How do you solve a linear regression problem in python⁵⁹](#)
- [Python tutorial on LinearRegression with Batch Gradient Descent⁶⁰](#)

⁵²http://www.aprendemachinelearning.com/ejercicio_Regresion_lineal/

⁵³http://www.aprendemachinelearning.com/articulos_ml/

⁵⁴https://github.com/jbagnato/machine-learning/blob/master/Ejercicio_Regresion_Lineal.ipynb

⁵⁵<https://github.com/jbagnato/machine-learning>

⁵⁶<https://aktechthoughts.wordpress.com/2018/03/26/introduction-to-linear-regression-using-python/>

⁵⁷<https://dzone.com/articles/linear-regression-using-python-scikit-learn>

⁵⁸<https://towardsdatascience.com/linear-regression-detailed-view-ea73175f6e86>

⁵⁹<http://lineardata.net/how-do-you-solve-a-linear-regression-machine-learning-problem-in-python/>

⁶⁰<http://ozzieliu.com/2016/02/09/gradient-descent-tutorial/>

Regresión Logística

Introducción

Utilizaremos algoritmos de Machine Learning en Python para resolver un problema de Regresión Logística⁶¹. A partir de un conjunto de datos de entrada (características), nuestra salida será discreta (y no continua) por eso utilizamos Regresión Logística (y no Regresión Lineal⁶²). La Regresión Logística es un Algoritmo Supervisado⁶³ y se utiliza para clasificación⁶⁴.

Vamos a clasificar problemas con dos posibles estados “SI/NO”: binario o un número finito de “etiquetas” o “clases”: múltiple.

Algunos Ejemplos de Regresión Logística son:

- Clasificar si el correo que llega es Spam o No es Spam.
- Dados unos resultados clínicos de un tumor clasificar en “Benigno” o “Maligno”.
- El texto de un artículo a analizar es: Entretenimiento, Deportes, Política ó Ciencia.
- A partir de historial bancario conceder un crédito o no.

Confiaremos en la implementación del paquete Scikit-learn de Python para ponerlo en práctica.

Ejercicio de Regresión Logística en Python

Para el ejercicio he creado un archivo csv⁶⁵ con datos de entrada a modo de ejemplo para clasificar si el usuario que visita un sitio web usa como sistema operativo Windows, Macintosh o Linux. Nuestra información de entrada son 4 características que tomé de una web que utiliza Google Analytics y son:

- Duración de la visita en Segundos
- Cantidad de Páginas Vistas durante la Sesión
- Cantidad de Acciones del usuario (click, scroll, uso de checkbox, sliders,etc)
- Suma del Valor de las acciones (cada acción lleva asociada una valoración de importancia)

⁶¹https://en.wikipedia.org/wiki/Logistic_regression

⁶²<http://www.aprendemachinelearning.com/regresion-lineal-en-espanol-con-python/>

⁶³<http://www.aprendemachinelearning.com/aplicaciones-del-machine-learning/#supervisado>

⁶⁴<http://www.aprendemachinelearning.com/principales-algoritmos-usados-en-machine-learning/>

⁶⁵http://www.aprendemachinelearning.com/wp-content/uploads/2017/11/usuarios_win_mac_lin.csv

Como la salida es discreta, asignaremos los siguientes valores a las etiquetas:

- 0 - Windows
- 1 - Macintosh
- 2 -Linux

La muestra es pequeña: son 170 registros para poder comprender el ejercicio, pero recordemos que para conseguir buenos resultados siempre es mejor contar con un número abundante de datos que darán mayor exactitud a las predicciones y evitarán [problemas de overfitting u underfitting⁶⁶](#).

Requerimientos técnicos

Para ejecutar el código necesitas tener instalado Python 3.6+ y varios paquetes usados comúnmente en Data Science, en este caso scikit-learn, pandas, seaborn y numpy.

Para este ejemplo he creado un block de notas Jupyter donde se muestra paso a paso el ejercicio. Se puede descargar [desde aquí⁶⁷](#) o se puede seguir online desde el [Jupyter Notebook Viewer⁶⁸](#).

Regresión Logística con SKLearn:

Identificar Sistema Operativo de los usuarios

Para comenzar hacemos los Import necesarios con los paquetes que utilizaremos en el Ejercicio.

```
1 import pandas as pd
2 import numpy as np
3 from sklearn import linear_model
4 from sklearn import model_selection
5 from sklearn.metrics import classification_report
6 from sklearn.metrics import confusion_matrix
7 from sklearn.metrics import accuracy_score
8 import matplotlib.pyplot as plt
9 import seaborn as sb
10 %matplotlib inline
```

Leemos el archivo csv (por sencillez, se considera que estará en el mismo directorio que el archivo de notebook .ipynb) y lo asignamos mediante Pandas a la variable **dataframe**. Mediante el método **dataframe.head()** vemos en pantalla los 5 primeros registros.

⁶⁶<http://www.aprendemachinelearning.com/que-es-overfitting-y-underfitting-y-como-solucionarlo/>

⁶⁷http://www.aprendemachinelearning.com/wp-content/uploads/2017/11/Regresion_logistica.ipynb

⁶⁸http://nbviewer.jupyter.org/github/jbagnato/machine-learning/blob/master/Regresion_logistica.ipynb

```

1 dataframe = pd.read_csv(r"usuarios_win_mac_lin.csv")
2 dataframe.head()

```

	duracion	paginas	acciones	valor	clase
0	7.0	2	4	8	2
1	21.0	2	6	6	2
2	57.0	2	4	4	2
3	101.0	3	6	12	2
4	109.0	2	6	12	2

A continuación llamamos al método `dataframe.describe()` que nos dará algo de información estadística básica de nuestro set de datos. La Media, el desvío estándar, valores mínimo y máximo de cada característica.

```
1 dataframe.describe()
```

	duracion	paginas	acciones	valor	clase
count	170.000000	170.000000	170.000000	170.000000	170.000000
mean	111.075729	2.041176	8.723529	32.676471	0.752941
std	202.453200	1.500911	9.136054	44.751993	0.841327
min	1.000000	1.000000	1.000000	1.000000	0.000000
25%	11.000000	1.000000	3.000000	8.000000	0.000000
50%	13.000000	2.000000	6.000000	20.000000	0.000000
75%	108.000000	2.000000	10.000000	36.000000	2.000000
max	898.000000	9.000000	63.000000	378.000000	2.000000

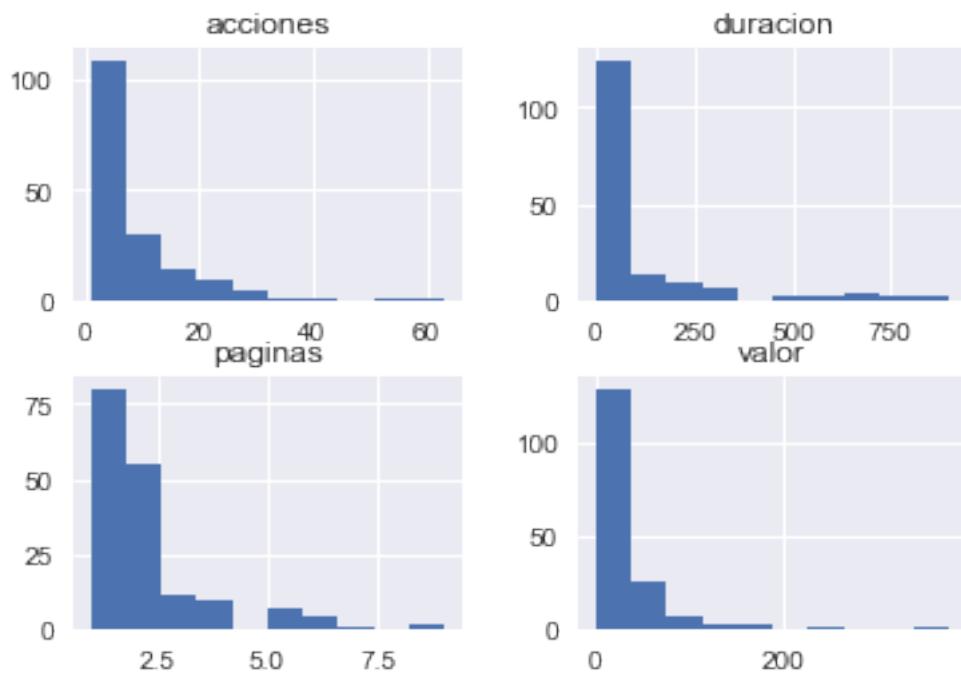
Luego analizaremos cuantos resultados tenemos de cada tipo usando la función groupby y vemos que tenemos 86 usuarios “Clase 0”, es decir Windows, 40 usuarios Mac y 44 de Linux.

```
1 print(dataframe.groupby('clase').size())  
  
clase  
0    86  
1    40  
2    44  
dtype: int64
```

Visualización de Datos

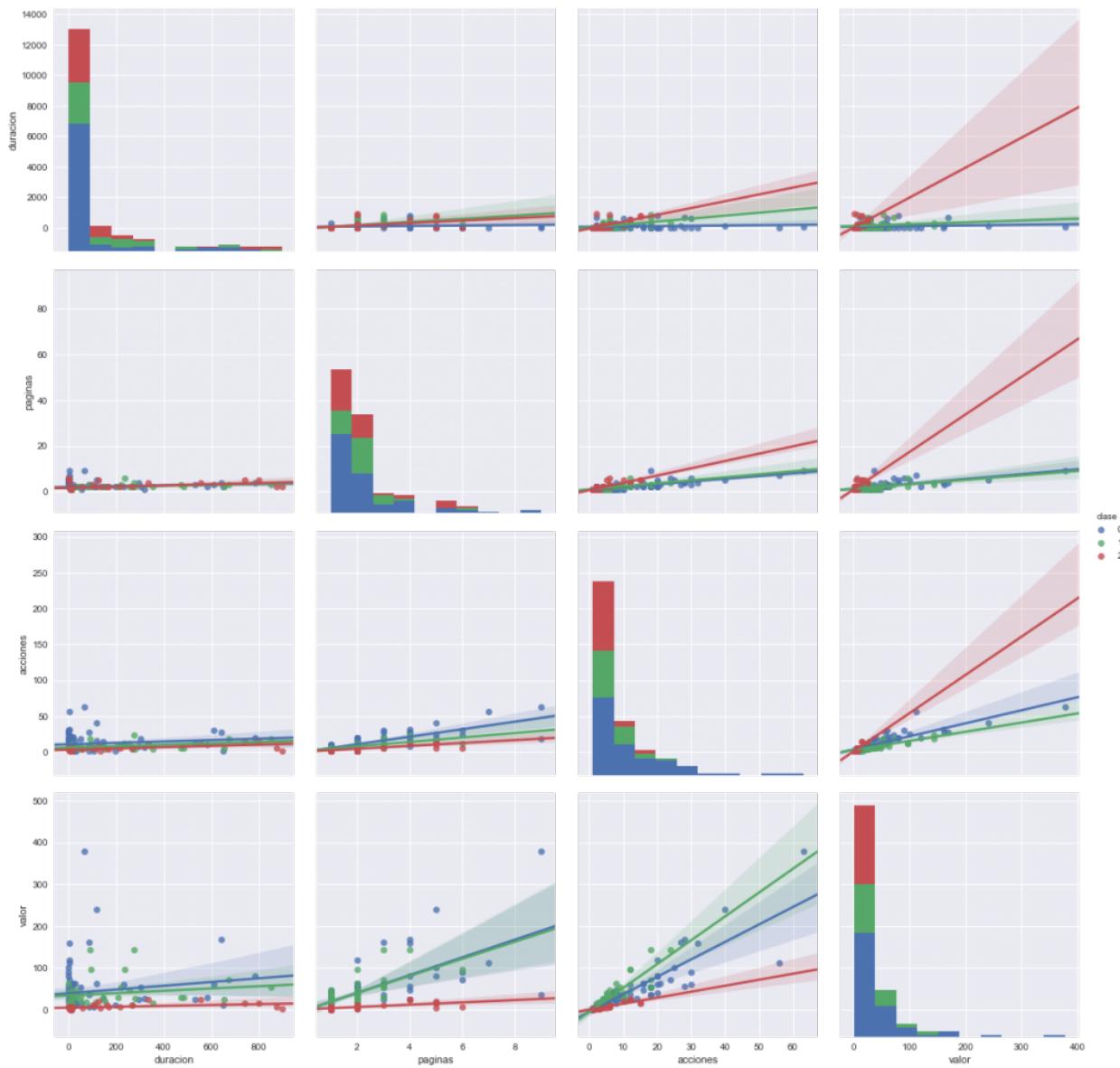
Antes de empezar a procesar el conjunto de datos, vamos a hacer unas visualizaciones que muchas veces nos pueden ayudar a comprender mejor las características de la información con la que trabajamos y su correlación. Primero visualizamos en formato de historial los cuatro Features de entrada con nombres “duración”, “páginas”, “acciones” y “valor” podemos ver gráficamente entre qué valores se comprenden sus mínimos y máximos y en qué intervalos concentran la mayor densidad de registros.

```
1 dataframe.drop(['clase'],1).hist()  
2 plt.show()
```



Y también podemos interrelacionar las entradas de a pares, para ver como se concentran linealmente las salidas de usuarios por colores: Sistema Operativo Windows en azul, Macintosh en verde y Linux en rojo.

```
1 sb.pairplot(dataframe.dropna(), hue='clase', size=4, vars=["duracion", "paginas", "acciones", "valor"], kind='reg')
```



Creamos el Modelo de Regresión Logística

Ahora cargamos las variables de las 4 columnas de entrada en X excluyendo la columna “clase” con el método `drop()`. En cambio agregamos la columna “clase” en la variable y. Ejecutamos `X.shape` para comprobar la dimensión de nuestra matriz con datos de entrada de 170 registros por 4 columnas.

```
1 X = np.array(dataframe.drop(['clase'],1))
2 y = np.array(dataframe['clase'])
3 X.shape
```

(170, 4) Y creamos nuestro modelo y hacemos que se ajuste (fit) a nuestro conjunto de entradas X y salidas 'y'.

```
1 model = linear_model.LogisticRegression()
2 model.fit(X,y)
```

Una vez compilado nuestro modelo, le hacemos clasificar todo nuestro conjunto de entradas X utilizando el método “predict(X)” y revisamos algunas de sus salidas y vemos que coincide con las salidas reales de nuestro archivo csv.

```
1 predictions = model.predict(X)
2 print(predictions)[0:5]

1 [2 2 2 2 2]
```

Y confirmamos cuan bueno fue nuestro modelo utilizando model.score() que nos devuelve la precisión media de las predicciones, en nuestro caso del 77%.

```
1 model.score(X,y)

1 0.77647058823529413
```

Validación de nuestro modelo

Una buena práctica en Machine Learning es la de subdividir nuestro conjunto de datos de entrada en un set de entrenamiento y otro para validar el modelo (que no se utiliza durante el entrenamiento y por lo tanto la máquina desconoce). Esto evitará problemas en los que nuestro algoritmo pueda fallar por “**sobregeneralizar” el conocimiento**⁶⁹. Para ello, dividimos nuestros datos de entrada en forma aleatoria (mezclados) utilizando 80% de registros para entrenamiento y 20% para validar.

⁶⁹<http://www.aprendemachinelearning.com/que-es-overfitting-y-underfitting-y-como-solucionarlo/>

```

1 validation_size = 0.20
2 seed = 7
3 X_train, X_validation, Y_train, Y_validation = model_selection.train_test_split(X, y\
4 , test_size=validation_size, random_state=seed)

```

Volvemos a compilar nuestro modelo de Regresión Logística pero esta vez sólo con 80% de los datos de entrada y calculamos el nuevo scoring que ahora nos da 74%.

```

1 name='Logistic Regression'
2 kfold = model_selection.KFold(n_splits=10, random_state=seed)
3 cv_results = model_selection.cross_val_score(model, X_train, Y_train, cv=kfold, scor\
4 ing='accuracy')
5 msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
6 print(msg)

```

```
1 Logistic Regression: 0.743407 (0.115752)
```

Y ahora hacemos las predicciones -en realidad clasificación- utilizando nuestro “cross validation set”, es decir del subconjunto que habíamos apartado. En este caso vemos que los aciertos fueron del 85% pero hay que tener en cuenta que el tamaño de datos era pequeño.

```

1 predictions = model.predict(X_validation)
2 print(accuracy_score(Y_validation, predictions))

```



```
1 0.852941176471
```

Finalmente vemos en pantalla la “matriz de confusión” donde muestra cuantos resultados equivocados tuvo de cada clase (los que no están en la diagonal), por ejemplo predijo 3 usuarios que eran Mac como usuarios de Windows y predijo a 2 usuarios Linux que realmente eran de Windows.

Reporte de Resultados del Modelo

```
1 print(confusion_matrix(Y_validation, predictions))
```

[[16	0	2]
		3	3	0]
		0	0	10]
]]

También podemos ver el reporte de clasificación con nuestro conjunto de Validación. En nuestro caso vemos que se utilizaron como “soporte” 18 registros windows, 6 de mac y 10 de Linux (total de 34 registros). Podemos ver la precisión con que se acertaron cada una de las clases y vemos que por ejemplo de Macintosh tuvo 3 aciertos y 3 fallos (0.5 recall). La valoración que de aquí nos conviene tener en cuenta es la de F1-score⁷⁰, que tiene en cuenta la precisión y recall. El promedio de F1 es de 84% lo cual no está nada mal.

```
1 print(classification_report(Y_validation, predictions))
```

	precision	recall	f1-score	support
0	0.84	0.89	0.86	18
1	1.00	0.50	0.67	6
2	0.83	1.00	0.91	10
avg / total	0.87	0.85	0.84	34

Clasificación de nuevos valores

Como último ejercicio, vamos a inventar los datos de entrada de navegación de un usuario ficticio que tiene estos valores:

- Tiempo Duración: 10
- Páginas visitadas: 3
- Acciones al navegar: 5
- Valoración: 9

Lo probamos en nuestro modelo y vemos que lo clasifica como un usuario tipo 2, es decir, de Linux.

```
1 X_new = pd.DataFrame({'duracion': [10], 'paginas': [3], 'acciones': [5], 'valor': [9]}
2 })
3 model.predict(X_new)
```

⁷⁰https://en.wikipedia.org/wiki/F1_score

```
1 array([2])
```

Los invito a jugar y variar estos valores para obtener usuarios de tipo Windows o Macintosh.

Resumen

Durante este artículo vimos cómo crear un modelo de Regresión Logística en Python para poder clasificar el Sistema Operativo de usuarios a partir de sus características de navegación en un sitio web. A partir de este ejemplo, se podrá extender a otro tipos de tareas que pueden surgir durante nuestro trabajo en el que deberemos clasificar resultados en valores discretos. Si tuviéramos que predecir valores continuos, deberemos aplicar [Regresión Lineal⁷¹](#).

Recuerda descargar los archivos para realizar el Ejercicio:

- [Archivo de Entrada csv⁷²](#) (su nombre es usuarios_win_mac_lin.csv)
- [Notebook Jupyter Python⁷³](#) (clic derecho y “descargar archivo como...”)
- OPCIÓN 2: Se puede ver online en [Jupyter Notebook Viewer⁷⁴](#)
- OPCIÓN 3: Se puede [visualizar y descargar el Notebook⁷⁵](#) y el [csv⁷⁶](#) desde mi cuenta de [Github⁷⁷](#)

⁷¹<http://www.aprendemachinelearning.com/regresion-lineal-en-espanol-con-python/>

⁷²http://www.aprendemachinelearning.com/wp-content/uploads/2017/11/usuarios_win_mac_lin.csv

⁷³http://www.aprendemachinelearning.com/wp-content/uploads/2017/11/Regresion_logistica.ipynb

⁷⁴http://nbviewer.jupyter.org/github/jbagnato/machine-learning/blob/master/Regresion_logistica.ipynb

⁷⁵https://github.com/jbagnato/machine-learning/blob/master/Regresion_logistica.ipynb

⁷⁶https://github.com/jbagnato/machine-learning/blob/master/usuarios_win_mac_lin.csv

⁷⁷<https://github.com/jbagnato/machine-learning>

Arbol de Decisión

En este capítulo describiremos en qué consisten y cómo funcionan los árboles de decisión utilizados en [Aprendizaje Automático⁷⁸](#) y nos centraremos en un divertido ejemplo en Python en el que analizaremos a los cantantes y bandas que lograron un puesto número uno en las listas de [Billboard Hot 100⁷⁹](#) e intentaremos predecir quién será el próximo [Ed Sheeran⁸⁰](#) a fuerza de Inteligencia Artificial.

Realizaremos Gráficas que nos ayudarán a visualizar los datos de entrada y un grafo para interpretar el árbol que crearemos con el paquete Scikit-Learn. Comencemos!

¿Qué es un árbol de decisión?

Los arboles de decisión son representaciones gráficas de posibles soluciones a una decisión basadas en ciertas condiciones, es uno de los algoritmos de [aprendizaje supervisado⁸¹](#) más utilizados en [machine learning⁸²](#) y pueden realizar tareas de clasificación o regresión (acrónimo del inglés [CART⁸³](#)).

La comprensión de su funcionamiento suele ser simple y a la vez muy potente. Utilizamos mentalmente estructuras de árbol de decisión constantemente en nuestra vida diaria sin darnos cuenta:

¿Llueve? ⇒ lleva paraguas.

¿Soleado? ⇒ lleva gafas de sol.

¿estoy cansado? ⇒ toma café.

Son Decisiones del tipo IF THIS, THEN THAT

Los árboles de decisión *tienen un primer nodo llamado raíz* (root) y luego se descomponen el resto de atributos de entrada en dos ramas (podrían ser más, pero no nos meteremos en eso ahora) planteando una condición que puede ser cierta o falsa. **Se bifurca cada nodo en 2 y vuelven a subdividirse hasta llegar a las hojas que son los nodos finales** y que equivalen a respuestas a la solución: Si/No, Comprar/Vender, o lo que sea que estemos clasificando. Otro ejemplo son los populares juegos de adivinanza:

1. ¿Animal ó vegetal? -Animal
2. ¿Tiene cuatro patas? -Si
3. ¿Hace guau? -Si
4. -Es un perro!

⁷⁸<http://www.aprendemachinelearning.com/que-es-machine-learning/>

⁷⁹<https://www.billboard.com/charts/hot-100>

⁸⁰<https://www.billboard.com/music/ed-sheeran>

⁸¹<http://www.aprendemachinelearning.com/aplicaciones-del-machine-learning/#supervisado>

⁸²<http://www.aprendemachinelearning.com/principales-algoritmos-usados-en-machine-learning/>

⁸³https://en.wikipedia.org/wiki/Classification_and_regression_tree

¿Qué necesidad hay de usar el Algoritmo de Arbol?

Supongamos que tenemos atributos como Género con valores “hombre ó mujer” y edad en rangos: “menor de 18 ó mayor de 18” para tomar una decisión. Podríamos crear un árbol en el que dividamos primero por género y luego subdividir por edad. Ó **podría ser al revés**: primero por edad y luego por género. El algoritmo es quien *analizando los datos y las salidas -por eso es supervisado!* decidirá la mejor forma de hacer las divisiones (*splits*) entre nodos. Tendrá en cuenta de qué manera lograr una predicción (clasificación ó regresión) con mayor probabilidad de acierto. Parece sencillo, no? Pensemos que si tenemos 10 atributos de entrada cada uno con 2 o más valores posibles, **las combinaciones para decidir el mejor árbol serían cientos ó miles...** Esto ya no es un trabajo para hacer artesanalmente. Y ahí es donde este algoritmo cobra importancia, pues él nos devolverá el árbol óptimo para la toma de decisión más acertada desde un punto de vista probabilístico.

¿Cómo funciona un árbol de decisión?

Para obtener el árbol óptimo y valorar cada subdivisión entre todos los árboles posibles y conseguir el nodo raiz y los subsiguientes, el algoritmo **deberá medir de alguna manera las predicciones logradas** y valorarlas para comparar de entre todas y obtener la mejor. Para medir y valorar, utiliza diversas funciones, siendo las más conocidas y usadas los “[Indice gini⁸⁴](#)” y “[Ganancia de información⁸⁵](#)” que utiliza la denominada “[entropía⁸⁶](#)”. La división de nodos continuará hasta que **lleguemos a la profundidad máxima** posible del árbol ó **se limiten los nodos a una cantidad mínima** de muestras en cada hoja. A continuación describiremos muy brevemente cada una de las estrategias nombradas:

Indice Gini:

Se utiliza para atributos con valores continuos (precio de una casa). Esta función de coste mide el “grado de impureza” de los nodos, es decir, **cuán desordenados o mezclados quedan los nodos una vez divididos**. *Deberemos minimizar ese GINI index.*

Ganancia de información:

Se utiliza para atributos categóricos (como en hombre/mujer). Este criterio intenta estimar la información que aporta cada atributo basado en la “[teoría de la información⁸⁷](#)”. Para medir la aleatoriedad de incertidumbre de un valor aleatorio de una variable “X” se define la [Entropía⁸⁸](#). Al obtener la medida de entropía de cada atributo, podemos calcular la ganancia de información del árbol. *Deberemos maximizar esa ganancia.*

⁸⁴https://en.wikipedia.org/wiki/Decision_tree_learning#Gini_impurity

⁸⁵https://en.wikipedia.org/wiki/Information_gain_in_decision_trees

⁸⁶[https://es.wikipedia.org/wiki/Entrop%C3%ADa_\(informaci%C3%B3n\)](https://es.wikipedia.org/wiki/Entrop%C3%ADa_(informaci%C3%B3n))

⁸⁷https://es.wikipedia.org/wiki/Teor%C3%ADa_de_la_informaci%C3%B3n

⁸⁸[https://es.wikipedia.org/wiki/Entrop%C3%ADa_\(informaci%C3%B3n\)](https://es.wikipedia.org/wiki/Entrop%C3%ADa_(informaci%C3%B3n))

Arbol de Decisión con Scikit-Learn paso a paso

Para este ejercicio me propuse crear un set de datos original e intentar que sea divertido a la vez que explique de forma clara el funcionamiento del árbol. Comencemos:

Requerimientos para hacer el Ejercicio

Para realizar este ejercicio, utilizaremos una [Jupyter notebook⁸⁹](#) con código python y la librería Scikit-learn. Podrás descargar los archivos de entrada csv o [visualizar la notebook online⁹⁰](#) (al final del artículo los enlaces).

Predicción del “Billboard 100”: ¿Qué artista llegará al número uno del ranking?

A partir de atributos de cantantes y de un histórico de canciones que alcanzaron entrar al Billboard 100 (U.S.) en 2013 y 2014 crearemos un árbol que nos permita intentar predecir si un nuevo cantante podrá llegar a número uno.

Obtención de los datos de entrada

Utilicé un [código python para hacer webscraping⁹¹](#) de una web pública “Ultimate Music Database” con información histórica del Billboard que obtuve de este artículo: “[Analyzing billboard 100⁹²](#)”. Luego completé atributos utilizando la [API de Deezer⁹³](#) (duración de las canciones), la [API de Gracenote⁹⁴](#) (género y ritmo de las canciones). Finalmente agregué a mano varias fechas de nacimiento de artistas utilizando la Wikipedia que no había conseguido con la Ultimate Music Database. Algunos artistas quedaron sin completar su fecha de nacimiento y con valor 0. Veremos como superar este obstáculo tratando los datos. Para empezar importemos las librerías que utilizaremos y revisemos sus atributos de entrada:

⁸⁹<http://data-speaks.luca-d3.com/2018/03/python-para-todos-2-jupyter-notebook.html>

⁹⁰https://github.com/jbagnato/machine-learning/blob/master/Ejercicio_Arbol_de_Decision.ipynb

⁹¹<http://www.aprendemachinelearning.com/ejemplo-web-scraping-python-ibex35-bolsa-valores/>

⁹²<http://mikekling.com/analyzing-the-billboard-hot-100/>

⁹³<https://developers.deezer.com/api>

⁹⁴<https://developer.gracenote.com/web-api>

```

1 # Imports needed for the script
2 import numpy as np
3 import pandas as pd
4 import seaborn as sb
5 import matplotlib.pyplot as plt
6 %matplotlib inline
7 plt.rcParams['figure.figsize'] = (16, 9)
8 plt.style.use('ggplot')
9 from sklearn import tree
10 from sklearn.metrics import accuracy_score
11 from sklearn.model_selection import KFold
12 from sklearn.model_selection import cross_val_score
13 from IPython.display import Image as PImage
14 from subprocess import check_call
15 from PIL import Image, ImageDraw, ImageFont

```

Si te falta alguna de ellas, recuerda que puedes instalarla con el entorno Anaconda o con la herramienta Pip.

Análisis Exploratorio Inicial

Ahora veamos cuantas columnas y registros tenemos:

```
1 artists_billboard.shape
```

Esto nos devuelve (635,11) es decir que tenemos 11 columnas (features) y 635 filas de datos. Vamos a echar un ojo a los primeros registros para tener una mejor idea del contenido:

```
1 artists_billboard.head()
```

	id	title	artist	mood	tempo	genre	artist_type	chart_date	durationSeg	top	anioNacimiento
0	0	Small Town Throwdown	BRANTLEY GILBERT featuring JUSTIN MOORE & THOM...	Brooding	Medium Tempo	Traditional	Male	20140628	191.0	0	1975.0
1	1	Bang Bang	JESSIE J, ARIANA GRANDE & NICKI MINAJ	Energizing	Medium Tempo	Pop	Female	20140816	368.0	0	1989.0
2	2	Timber	PITBULL featuring KE\$HA	Excited	Medium Tempo	Urban	Mixed	20140118	223.0	1	1993.0
3	3	Sweater Weather	THE NEIGHBOURHOOD	Brooding	Medium Tempo	Alternative & Punk	Male	20140104	206.0	0	1989.0
4	4	Automatic	MIRANDA LAMBERT	Yearning	Medium Tempo	Traditional	Female	20140301	232.0	0	0.0

95

Vemos que tenemos: Titulo de la canción, artista, “mood” ó *estado de ánimo* de esa canción, tempo,

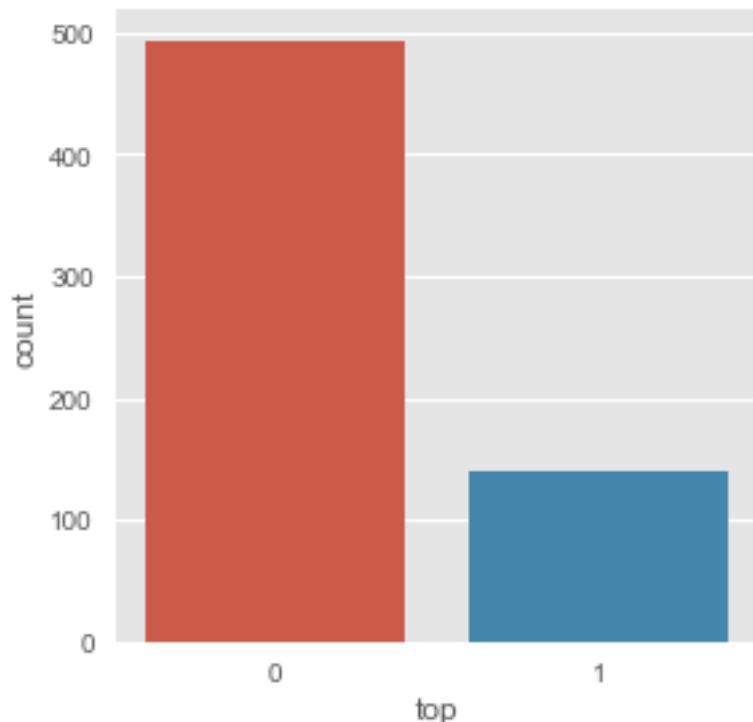
⁹⁵http://www.aprendemachinelearning.com/wp-content/uploads/2018/04/billboard_dataset_head.png

género, Tipo de artista, fecha en que apareció en el billboard (por ejemplo 20140628 equivale al 28 de junio de 2014), la columna TOP será nuestra etiqueta, en la que aparece 1 si llegó al número uno de Billboard ó 0 si no lo alcanzó y el año de Nacimiento del artista. Vemos que muchas de las columnas contienen información categórica. La columna *durationSeg* contiene la duración en segundos de la canción, siendo un valor continuo pero que nos convendrá pasar a categórico más adelante. Vamos a realizar algunas visualizaciones para comprender mejor nuestros datos. Primero, agrupemos registros para ver cuántos alcanzaron el número uno y cuantos no:

```
1 artists_billboard.groupby('top').size()
```

nos devuelve: top 0 494 1 141 Es decir que tenemos 494 canciones que no alcanzaron la cima y a 141 que alcanzaron el número uno. Esto quiere decir que *tenemos una cantidad DESBALANCEADA⁹⁶ de etiquetas con 1 y 0. Lo tendremos en cuenta al momento de crear el árbol.*

Visualizamos esta diferencia:



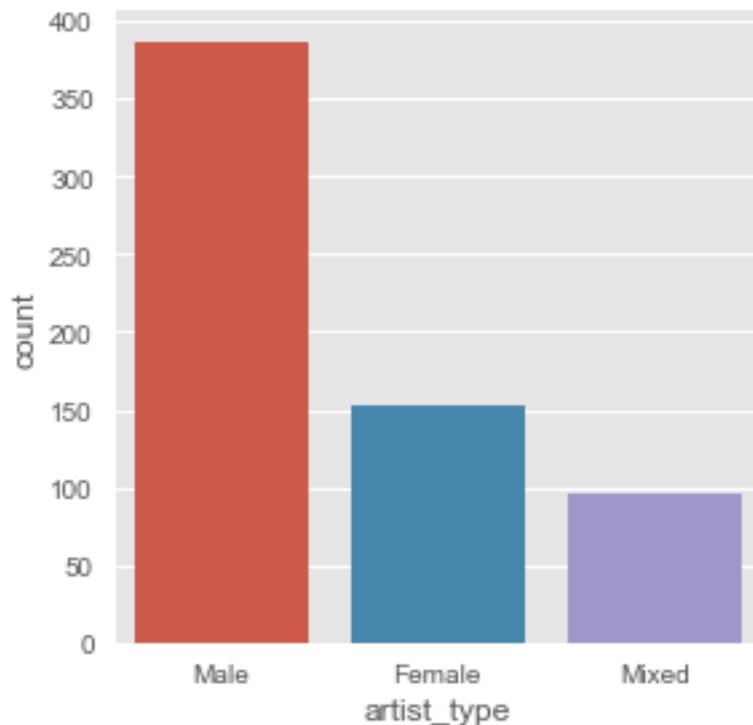
Nuestras etiquetas que indican 0-No llegó al Top y 1-Llegó al número uno Billboard están desbalanceadas.

Deberemos resolver este inconveniente[/caption]

Veamos cuántos registros hay de tipo de artista, “mood”, tempo y género de las canciones:

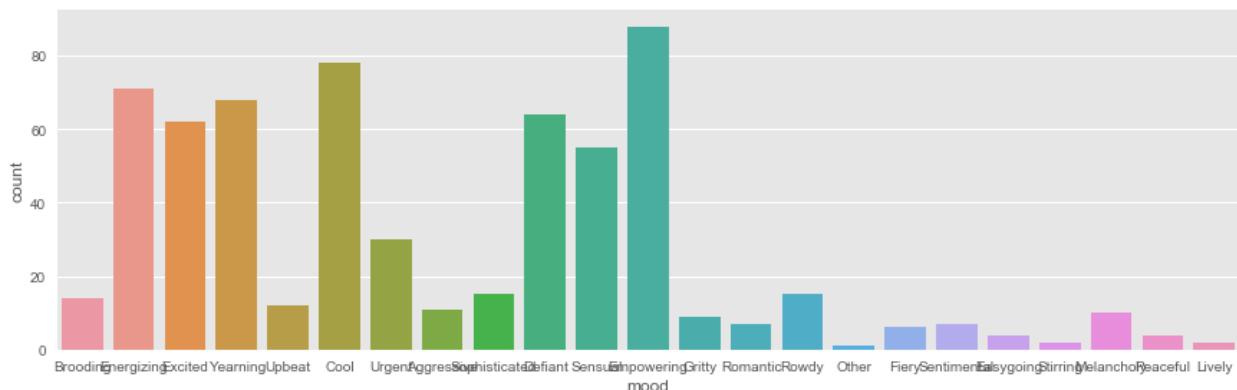
⁹⁶<http://www.aprendemachinelearning.com/clasificacion-con-datos-desbalanceados/>

```
1 sb.factorplot('artist_type', data=artists_billboard, kind="count")
```



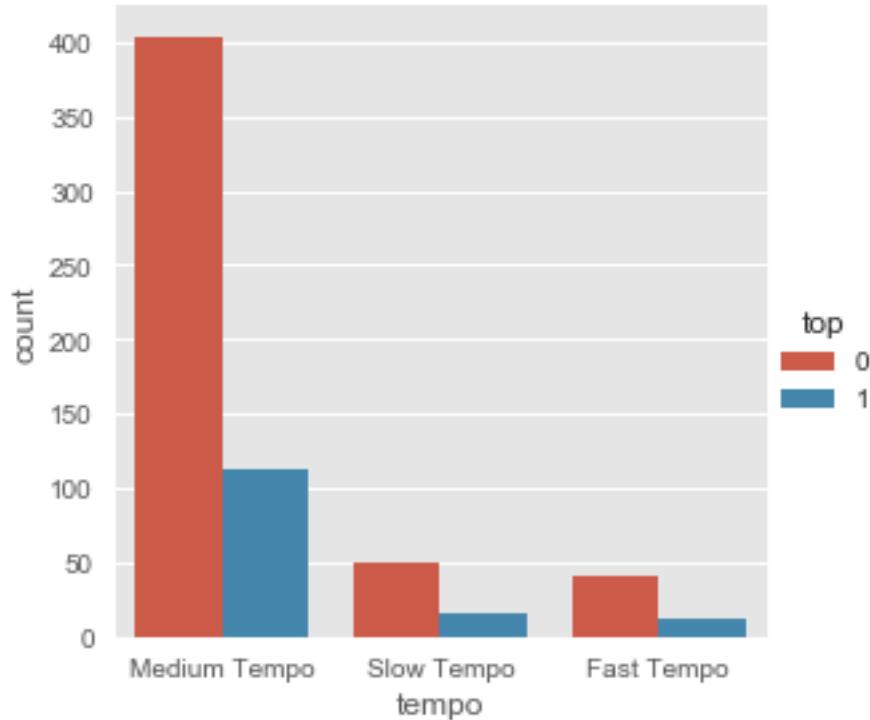
Aquí vemos que tenemos más del doble de artistas masculinos que femeninos y unos 100 registros de canciones mixtas

```
1 sb.factorplot('mood', data=artists_billboard, kind="count", aspect=3)
```



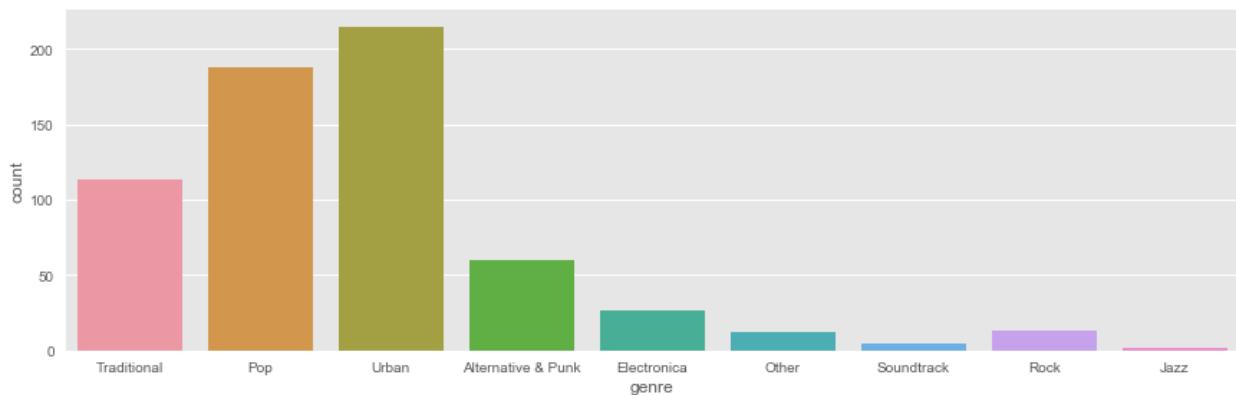
Vemos que de 23 tipos de Mood, destacan 7 con picos altos. Además notamos que algunos estados de ánimo son similares.

```
1 sb.factorplot('tempo', data=artists_billboard, hue='top', kind="count")
```



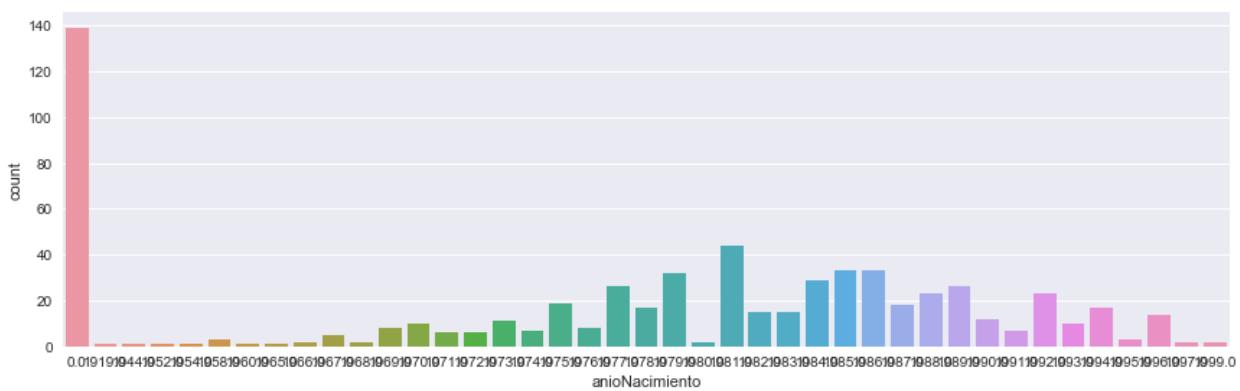
En esta gráfica vemos que hay 3 tipos de Tempo: Medium, Slow y Fast. Evidentemente predominan los tiempos Medium y también es donde encontramos más canciones que hayan alcanzado el Top 1 (en azul)

```
1 sb.factorplot('genre', data=artists_billboard, kind="count", aspect=3)
```



Entre los géneros musicales destacan Urban y Pop, seguidos de Tradicional. Veamos ahora que pasa al visualizar los años de nacimiento de los artistas:

```
1 sb.factorplot('anioNacimiento', data=artists_billboard, kind="count", aspect=3)
```



Aquí notamos algo raro: en el año “cero” tenemos cerca de 140 registros...

Como se ve en la gráfica tenemos cerca de 140 canciones de las cuales **desconocemos el año de nacimiento del artista**. El resto de años parecen concentrarse entre 1979 y 1994 (a ojo). Más adelante trataremos estos registros.

Balanceo de Datos: Pocos artistas llegan al número uno

Como dijimos antes, no tenemos “equilibrio” en la cantidad de etiquetas top y “no-top” de las canciones. Esto se debe a que en el transcurso de un año, **apenas unas 5 o 6 canciones logran el primer puesto** y se mantienen durante varias semanas en ese puesto. Cuando inicialmente extraje las canciones, utilicé 2014 y 2015 y tenía apenas a 11 canciones en el top de Billboard y 494 que no llegaron. Para intentar equilibrar los casos positivos agregué solamente los TOP de los años 2004 al 2013. Con eso conseguí los valores que tenemos en el archivo csv: son 494 “no-top” y 141 top. A pesar de esto sigue estando desbalanceado, y podríamos seguir agregando sólo canciones TOP de años previos, pero utilizaremos un parámetro (**class_weight**) del algoritmo de árbol de decisión para compensar esta diferencia.

En el capítulo “[Clasificación con Datos Desbalanceados](#)⁹⁷ teuento todas las estrategias para equilibrar las clases.

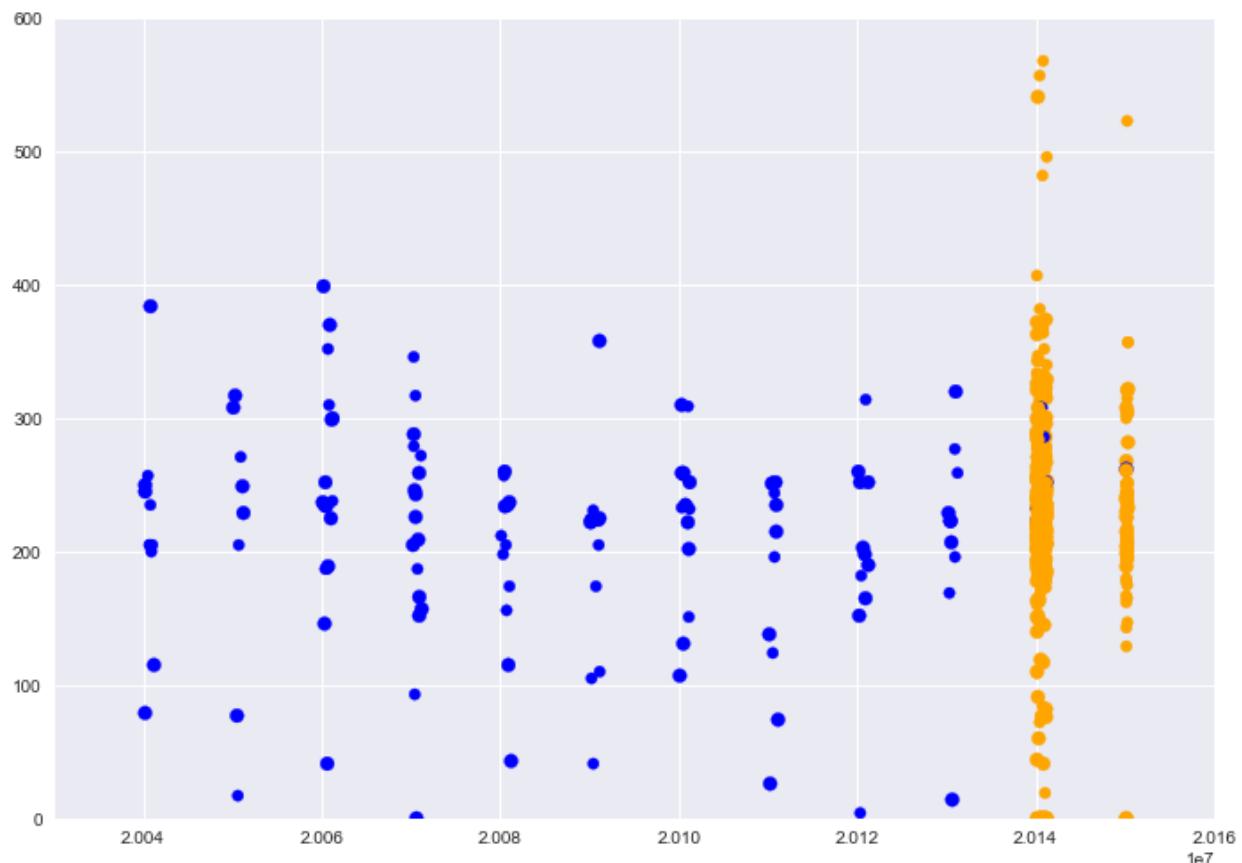
Visualicemos los top y no top de acuerdo a sus fechas en los Charts:

⁹⁷<http://www.aprendemachinelearning.com/clasificacion-con-datos-desbalanceados/>

```

1 f1 = artists_billboard['chart_date'].values
2 f2 = artists_billboard['durationSeg'].values
3
4 colores=['orange','blue'] # si no estaban declarados previamente
5 tamanios=[60,40] # si no estaban declarados previamente
6
7 asignar=[]
8 asignar2=[]
9 for index, row in artists_billboard.iterrows():
10     asignar.append(colores[row['top']])
11     asignar2.append(tamanios[row['top']])
12
13 plt.scatter(f1, f2, c=asignar, s=tamanios)
14 plt.axis([20030101,20160101,0,600])
15 plt.show()

```



En nuestro conjunto de Datos, se agregaron canciones que llegaron al top (en azul) de años 2004 al 2013 para sumar a los apenas 11 que lo habían logrado en 2014-2015.

Preparamos los datos

Vamos a arreglar el problema de los años de nacimiento que están en cero. Realmente el “feature” o característica que queremos obtener es : “sabiendo el año de nacimiento del cantante, calcular qué edad tenía al momento de aparecer en el Billboard”. Por ejemplo un artista que nació en 1982 y apareció en los charts en 2012, tenía 30 años. Primero vamos a sustituir los ceros de la columna “anioNacimiento” por el valor None -que es es nulo en Python-.

```

1 def edad_fix(anio):
2     if anio==0:
3         return None
4     return anio
5
6 artists_billboard['anioNacimiento']=artists_billboard.apply(lambda x: edad_fix(x['an\\
7 ioNacimiento']), axis=1);

```

Luego vamos a calcular las edades en una nueva columna “edad_en_billboard” restando el año de aparición (los 4 primeros caracteres de chart_date) al año de nacimiento. En las filas que estaba el año en None, tendremos como resultado edad None.

```

1 def calcula_edad(anio,cuando):
2     cad = str(cuando)
3     momento = cad[:4]
4     if anio==0.0:
5         return None
6     return int(momento) - anio
7
8 artists_billboard['edad_en_billboard']=artists_billboard.apply(lambda x: calcula_eda\\
9 d(x['anioNacimiento'],x['chart_date']), axis=1);

```

Y finalmente asignaremos edades aleatorias a los registros faltantes: para ello, obtenemos el promedio de edad de nuestro conjunto (avg) y su desvío estándar (std) -por eso necesitábamos las edades en None- y pedimos valores random a la función que van desde avg - std hasta avg + std. En nuestro caso son edades de entre 21 a 37 años.

```

1 age_avg = artists_billboard['edad_en_billboard'].mean()
2 age_std = artists_billboard['edad_en_billboard'].std()
3 age_null_count = artists_billboard['edad_en_billboard'].isnull().sum()
4 age_null_random_list = np.random.randint(age_avg - age_std, age_avg + age_std, size=\
5 age_null_count)
6
7 conValoresNulos = np.isnan(artists_billboard['edad_en_billboard'])
8
9 artists_billboard.loc[np.isnan(artists_billboard['edad_en_billboard']), 'edad_en_bil\
10 lboard'] = age_null_random_list
11 artists_billboard['edad_en_billboard'] = artists_billboard['edad_en_billboard'].astype\
12 pe(int)
13 print("Edad Promedio: " + str(age_avg))
14 print("Desvió Std Edad: " + str(age_std))
15 print("Intervalo para asignar edad aleatoria: " + str(int(age_avg - age_std)) + " a \
16 " + str(int(age_avg + age_std)))

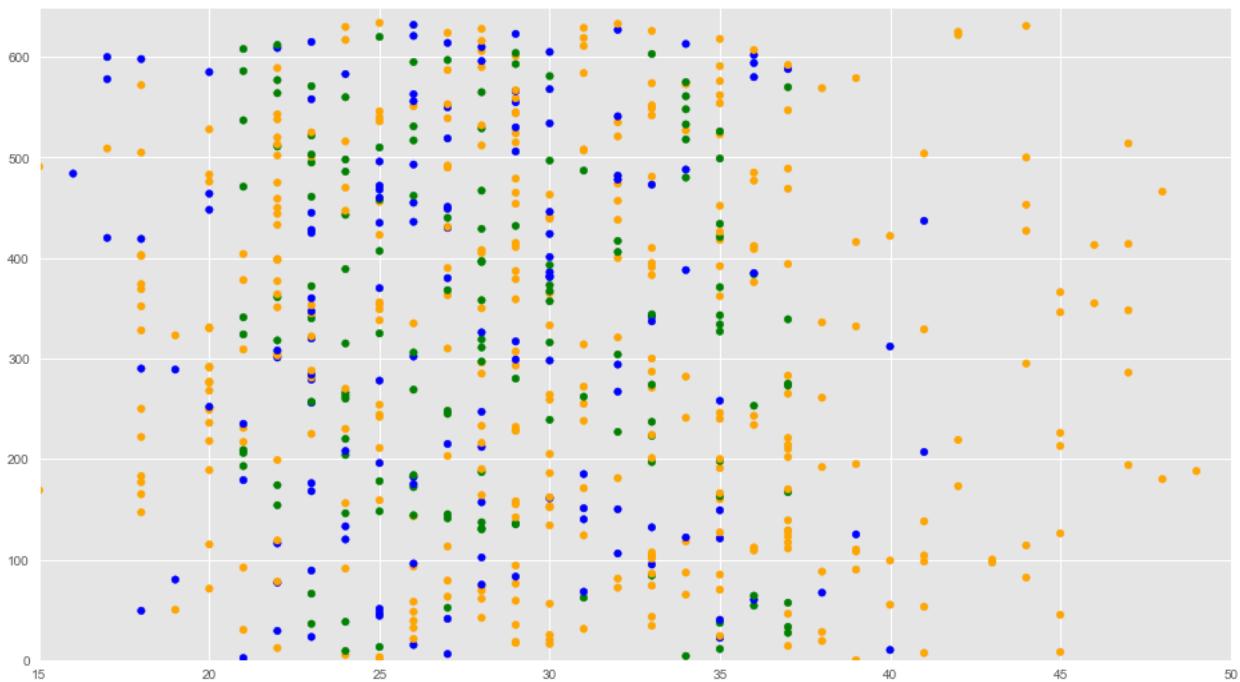
```

Si bien *lo ideal es contar con la información real*, y de hecho la podemos obtener buscando en Wikipedia (o en otras webs de música), quise mostrar otra vía para poder completar datos faltantes manteniendo los promedios de edades que teníamos en nuestro conjunto de datos. Podemos visualizar los valores que agregamos (en color verde) en el siguiente gráfico:

```

1 f1 = artists_billboard['edad_en_billboard'].values
2 f2 = artists_billboard.index
3
4 colores = ['orange', 'blue', 'green']
5
6 asignar=[]
7 for index, row in artists_billboard.iterrows():
8     if (conValoresNulos[index]):
9         asignar.append(colores[2]) # verde
10    else:
11        asignar.append(colores[row['top']])
12
13 plt.scatter(f1, f2, c=asignar, s=30)
14 plt.axis([15,50,0,650])
15 plt.show()

```



Mapeo de Datos

Vamos a transformar varios de los datos de entrada en valores categóricos. Las edades, las separamos en: menor de 21 años, entre 21 y 26, etc. las duraciones de canciones también, por ej. entre 150 y 180 segundos, etc. Para los estados de ánimo (mood) agrupé los que eran similares. El Tempo que puede ser lento, medio o rápido queda mapeado: 0-Rapido, 1-Lento, 2-Medio (por cantidad de canciones en cada tempo: el Medio es el que más tiene)

```

1 # Mood Mapping
2 artists_billboard['moodEncoded'] = artists_billboard['mood'].map( {'Energizing': 6,
3                     'Empowering': 6,
4                     'Cool': 5,
5                     'Yearning': 4, # anhelo, deseo, ansia
6                     'Excited': 5, #emocionado
7                     'Defiant': 3,
8                     'Sensual': 2,
9                     'Gritty': 3, #coraje
10                    'Sophisticated': 4,
11                    'Aggressive': 4, # provocativo
12                    'Fiery': 4, #caracter fuerte
13                    'Urgent': 3,
14                    'Rowdy': 4, #ruidoso alboroto

```

```

15      'Sentimental': 4,
16      'Easygoing': 1, # sencillo
17      'Melancholy': 4,
18      'Romantic': 2,
19      'Peaceful': 1,
20      'Brooding': 4, # melancolico
21      'Upbeat': 5, #optimista alegre
22      'Stirring': 5, #emocionante
23      'Lively': 5, #animado
24      'Other': 0, '' :0} ).astype(int)
25 # Tempo Mapping
26 artists_billboard['tempoEncoded'] = artists_billboard['tempo'].map( {'Fast Tempo': 0 \
27 , 'Medium Tempo': 2, 'Slow Tempo': 1, '' : 0} ).astype(int)
28 # Genre Mapping
29 artists_billboard['genreEncoded'] = artists_billboard['genre'].map( {'Urban': 4,
30                     'Pop': 3,
31                     'Traditional': 2,
32                     'Alternative & Punk': 1,
33                     'Electronica': 1,
34                     'Rock': 1,
35                     'Soundtrack': 0,
36                     'Jazz': 0,
37                     'Other': 0, '' :0}
38                     ).astype(int)
39 # artist_type Mapping
40 artists_billboard['artist_typeEncoded'] = artists_billboard['artist_type'].map( {'Female': 2, 'Male': 3, 'Mixed': 1, '' : 0} ).astype(int)
41
42 # Mapping edad en la que llegaron al billboard
43 artists_billboard.loc[ artists_billboard['edad_en_billboard'] <= 21, 'edadEncoded' ] \
44 = 0
45 artists_billboard.loc[(artists_billboard['edad_en_billboard'] > 21) & (artists_billb\
46 oard['edad_en_billboard'] <= 26), 'edadEncoded' ] = 1
47 artists_billboard.loc[(artists_billboard['edad_en_billboard'] > 26) & (artists_billb\
48 oard['edad_en_billboard'] <= 30), 'edadEncoded' ] = 2
49 artists_billboard.loc[(artists_billboard['edad_en_billboard'] > 30) & (artists_billb\
50 oard['edad_en_billboard'] <= 40), 'edadEncoded' ] = 3
51 artists_billboard.loc[ artists_billboard['edad_en_billboard'] > 40, 'edadEncoded' ] =\
52 4
53
54 # Mapping Song Duration
55 artists_billboard.loc[ artists_billboard['durationSeg'] <= 150, 'durationEncoded' ] =\
56 0

```

```

58 artists_billboard.loc[(artists_billboard['durationSeg'] > 150) & (artists_billboard[\
59 'durationSeg'] <= 180), 'durationEncoded'] = 1
60 artists_billboard.loc[(artists_billboard['durationSeg'] > 180) & (artists_billboard[\\
61 'durationSeg'] <= 210), 'durationEncoded'] = 2
62 artists_billboard.loc[(artists_billboard['durationSeg'] > 210) & (artists_billboard[\\
63 'durationSeg'] <= 240), 'durationEncoded'] = 3
64 artists_billboard.loc[(artists_billboard['durationSeg'] > 240) & (artists_billboard[\\
65 'durationSeg'] <= 270), 'durationEncoded'] = 4
66 artists_billboard.loc[(artists_billboard['durationSeg'] > 270) & (artists_billboard[\\
67 'durationSeg'] <= 300), 'durationEncoded'] = 5
68 artists_billboard.loc[ artists_billboard['durationSeg'] > 300, 'durationEncoded'] = 6

```

Finalmente obtenemos un nuevo conjunto de datos llamado artists_encoded con el que tenemos los atributos definitivos para crear nuestro árbol. Para ello, quitamos todas las columnas que no necesitamos con “drop”:

```

1 drop_elements = ['id', 'title', 'artist', 'mood', 'tempo', 'genre', 'artist_type', 'chart_d\\
2 ate', 'anioNacimiento', 'durationSeg', 'edad_en_billboard']
3 artists_encoded = artists_billboard.drop(drop_elements, axis = 1)

```

Como quedan los top en relación a los datos mapeados

Revisemos en tablas cómo se reparten los top=1 en los diversos atributos mapeados. Sobre la columna sum, estarán los top, pues al ser valor 0 o 1, sólo se sumarán los que sí llegaron al número 1.

```

1 artists_encoded[['moodEncoded', 'top']].groupby(['moodEncoded'], as_index=False).agg\
2 ([['mean', 'count', 'sum']])

```

	top		
	mean	count	sum
moodEncoded			
0	0.000000	1	0
1	0.000000	8	0
2	0.274194	62	17
3	0.145631	103	15
4	0.136986	146	20
5	0.294872	156	46
6	0.270440	159	43

La mayoría de top 1 los vemos en los estados de ánimo 5 y 6 con 46 y 43 canciones

```
1 artists_encoded[['artist_typeEncoded', 'top']].groupby(['artist_typeEncoded'], as_index=False).agg(['mean', 'count', 'sum'])
```

	top		
	mean	count	sum
artist_typeEncoded			
1	0.305263	95	29
2	0.320261	153	49
3	0.162791	387	63

Aquí están bastante repartidos, pero hay mayoría en tipo 3: artistas masculinos.

```
1 artists_encoded[['genreEncoded', 'top']].groupby(['genreEncoded'], as_index=False).agg(['mean', 'count', 'sum'])
```

	top		
	mean	count	sum
genreEncoded			
0	0.105263	19	2
1	0.070000	100	7
2	0.008850	113	1
3	0.319149	188	60
4	0.330233	215	71

Los géneros con mayoría son evidentemente los géneros 3 y 4 que corresponden con Urbano y Pop

```
1 artists_encoded[['tempoEncoded', 'top']].groupby(['tempoEncoded'], as_index=False).agg(['mean', 'count', 'sum'])
```

	top		
	mean	count	sum
tempoEncoded			
0	0.226415	53	12
1	0.246154	65	16
2	0.218569	517	113

El tempo con más canciones exitosas en el número 1 es el 2, tempo medio

```
1 artists_encoded[['durationEncoded', 'top']].groupby(['durationEncoded'], as_index=False)
2 .agg(['mean', 'count', 'sum'])
```

	top		
	mean	count	sum
durationEncoded			
0.0	0.295775	71	21
1.0	0.333333	30	10
2.0	0.212963	108	23
3.0	0.202381	168	34
4.0	0.232143	112	26
5.0	0.145455	55	8
6.0	0.208791	91	19

Están bastante repartidos en relación a la duración de las canciones

```
1 artists_encoded[['edadEncoded', 'top']].groupby(['edadEncoded'], as_index=False).agg\
2 ([['mean', 'count', 'sum']])
```

	top		
	mean	count	sum
edadEncoded			
0.0	0.257576	66	17
1.0	0.300613	163	49
2.0	0.260563	142	37
3.0	0.165899	217	36
4.0	0.042553	47	2

Edad con mayoría es la tipo 1 que comprende de 21 a 25 años.

Buscamos la profundidad para el árbol de decisión

Ya casi tenemos nuestro árbol. Antes de crearlo, vamos a buscar cuántos niveles de profundidad le asignaremos. Para ello, aprovecharemos la función de KFold que nos ayudará a crear varios subgrupos con nuestros datos de entrada para validar y valorar los árboles con diversos niveles de profundidad. De entre ellos, escogeremos el de mejor resultado.

Creamos el árbol y lo tuneamos

Para crear el árbol utilizamos de la [librería de sklearn tree.DecisionTreeClasifier](#)⁹⁸ pues buscamos un árbol de clasificación (no de Regresión). Lo configuramos con los parámetros:

- ** criterion=entropy** ó podría ser gini, pero utilizamos entradas categóricas
- **min_samples_split=20** se refiere a la cantidad mínima de muestras que debe tener un nodo para poder subdividir.
- **min_samples_leaf=5** cantidad mínima que puede tener una hoja final. Si tuviera menos, no se formaría esa hoja y “subiría” un nivel, su antecesor.
- **class_weight= IMPORTANTÍSIMO**: con esto [compensamos los desbalances](#)⁹⁹ que hubiera. En nuestro caso, como venía diciendo anteriormente, tenemos menos etiquetas de tipo top=1 (los artistas que llegaron al número 1 del ranking). Por lo tanto, le asignamos 3.5 de peso a la etiqueta 1 para compensar. El valor sale de dividir la cantidad de top=0 (son 494) con los top=1 (son 141).

NOTA: estos valores asignados a los parámetros fueron puestos luego de prueba y error (muchas veces visualizando el árbol, en el siguiente paso y retrocediendo a este).

⁹⁸<http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html>

⁹⁹<http://www.aprendemachinelearning.com/clasificacion-con-datos-desbalanceados/>

```

1 cv = KFold(n_splits=10) # Numero deseado de "folds" que haremos
2 accuracies = list()
3 max_attributes = len(list(artists_encoded))
4 depth_range = range(1, max_attributes + 1)
5
6 # Testearemos la profundidad de 1 a cantidad de atributos +1
7 for depth in depth_range:
8     fold_accuracy = []
9     tree_model = tree.DecisionTreeClassifier(criterion='entropy',
10                                              min_samples_split=20,
11                                              min_samples_leaf=5,
12                                              max_depth = depth,
13                                              class_weight={1:3.5})
14    for train_fold, valid_fold in cv.split(artists_encoded):
15        f_train = artists_encoded.loc[train_fold]
16        f_valid = artists_encoded.loc[valid_fold]
17
18        model = tree_model.fit(X = f_train.drop(['top'], axis=1),
19                               y = f_train["top"])
20        valid_acc = model.score(X = f_valid.drop(['top'], axis=1),
21                               y = f_valid["top"]) # calculamos la precision con el \
22 segmento de validacion
23     fold_accuracy.append(valid_acc)
24
25 avg = sum(fold_accuracy)/len(fold_accuracy)
26 accuracies.append(avg)
27
28 # Mostramos los resultados obtenidos
29 df = pd.DataFrame({"Max Depth": depth_range, "Average Accuracy": accuracies})
30 df = df[["Max Depth", "Average Accuracy"]]
31 print(df.to_string(index=False))

```

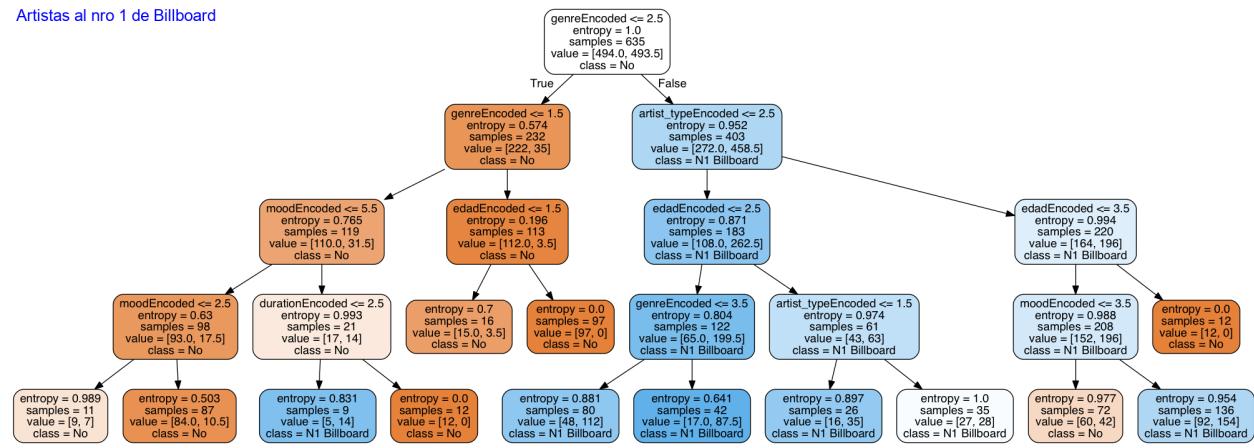
Max Depth	Average Accuracy
1	0.556101
2	0.556126
3	0.564038
4	0.648859
5	0.617386
6	0.614236
7	0.625124

Podemos ver que en 4 niveles de splits tenemos el score más alto, con casi 65%. Ahora ya sólo nos queda crear y visualizar nuestro árbol de 4 niveles de profundidad.

Visualización del árbol de decisión

Asignamos los datos de entrada y los parámetros que configuramos anteriormente con 4 niveles de profundidad. Utilizaremos la función de export_graphviz para crear un archivo de extensión .dot que luego convertiremos en un gráfico png para visualizar el árbol.

```
1 # Crear arrays de entrenamiento y las etiquetas que indican si llegó a top o no
2 y_train = artists_encoded['top']
3 x_train = artists_encoded.drop(['top'], axis=1).values
4
5 # Crear Arbol de decision con profundidad = 4
6 decision_tree = tree.DecisionTreeClassifier(criterion='entropy',
7                                         min_samples_split=20,
8                                         min_samples_leaf=5,
9                                         max_depth = 4,
10                                        class_weight={1:3.5})
11 decision_tree.fit(x_train, y_train)
12
13 # exportar el modelo a archivo .dot
14 with open(r"tree1.dot", 'w') as f:
15     f = tree.export_graphviz(decision_tree,
16                             out_file=f,
17                             max_depth = 7,
18                             impurity = True,
19                             feature_names = list(artists_encoded.drop(['top'], a\
20 xis=1)),
21                             class_names = ['No', 'N1 Billboard'],
22                             rounded = True,
23                             filled= True )
24
25 # Convertir el archivo .dot a png para poder visualizarlo
26 check_call(['dot', '-Tpng', r'tree1.dot', '-o', r'tree1.png'])
27 PImage("tree1.png")
```



Al fin nuestro preciado árbol aparece en pantalla!. Ahora tendremos que mirar y ver si lo podemos mejorar (por ejemplo tuneando los parámetros de entrada).

Análisis

En la gráfica vemos, un nodo raíz que hace una primer subdivisión por género y las salidas van a izquierda por True que sea menor a 2.5, es decir los géneros 0, 1 y 2 (eran los que menos top=1 tenían) y a derecha en False van los géneros 3 y 4 que eran Pop y Urban con gran cantidad de usuarios top Billboard. En el segundo nivel vemos que la cantidad de muestras (samples) queda repartida en 232 y 403 respectivamente. A medida que bajamos de nivel veremos que los valores de entropía se aproximan más a 1 cuando el nodo tiene más muestras top=1 (azul) y se acercan a 0 cuando hay mayoría de muestras Top=0 (naranja). En los diversos niveles veremos divisiones por tipo de artista , edad, duración y mood. También vemos algunas hojas naranjas que finalizan antes de llegar al último nivel: esto es porque alcanzan un nivel de entropía cero, o porque quedan con una cantidad de muestras menor a nuestro mínimo permitido para hacer split (20). Veamos cuál fue la precisión alcanzada por nuestro árbol:

```

1 acc_decision_tree = round(decision_tree.score(x_train, y_train) * 100, 2)
2 print(acc_decision_tree)
  
```

Nos da un valor de 64.88%. Notamos en que casi todas las hojas finales del árbol tienen samples mezclados sobre todo en los de salida para clasificar los top=1. Esto hace que se reduzca el score. Pongamos a prueba nuestro algoritmo

Predicción de Canciones al Billboard 100

Vamos a testear nuestro árbol con 2 artistas que entraron al billboard 100 en 2017: Camila Cabello¹⁰⁰ que llegó al numero 1 con la Canción Havana¹⁰¹ y Imagine Dragons¹⁰² con su canción Believer¹⁰³ que alcanzó un puesto 42 pero no llegó a la cima.

```

1 #predecir artista CAMILA CABELLO featuring YOUNG THUG
2 # con su canción Havana llego a numero 1 Billboard US en 2017
3
4 x_test = pd.DataFrame(columns=('top', 'moodEncoded', 'tempoEncoded', 'genreEncoded', '\
5 artist_typeEncoded', 'edadEncoded', 'durationEncoded'))
6 x_test.loc[0] = (1,5,2,4,1,0,3)
7 y_pred = decision_tree.predict(x_test.drop(['top'], axis = 1))
8 print("Prediccion: " + str(y_pred))
9 y_proba = decision_tree.predict_proba(x_test.drop(['top'], axis = 1))
10 print("Probabilidad de Acierto: " + str(round(y_proba[0][y_pred]* 100, 2))+"%")

```

Nos da que Havana llegará al top 1 con una probabilidad del 83%. Nada mal...

```

1 #predecir artista Imagine Dragons
2 # con su canción Believer llego al puesto 42 Billboard US en 2017
3
4 x_test = pd.DataFrame(columns=('top', 'moodEncoded', 'tempoEncoded', 'genreEncoded', '\
5 artist_typeEncoded', 'edadEncoded', 'durationEncoded'))
5 x_test.loc[0] = (0,4,2,1,3,2,3)
6 y_pred = decision_tree.predict(x_test.drop(['top'], axis = 1))
7 print("Prediccion: " + str(y_pred))
8 y_proba = decision_tree.predict_proba(x_test.drop(['top'], axis = 1))
9 print("Probabilidad de Acierto: " + str(round(y_proba[0][y_pred]* 100, 2))+"%")

```

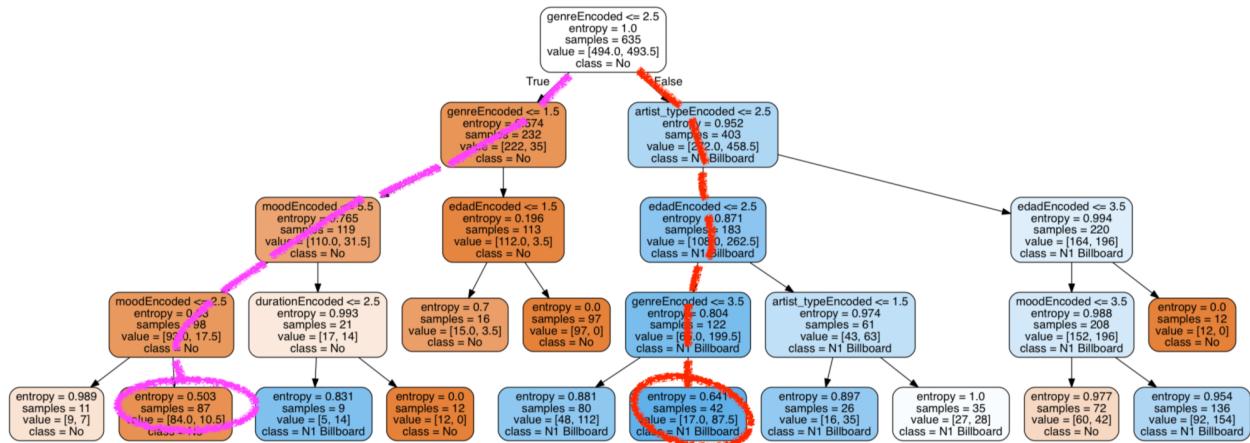
Nos da que la canción de Imagine Dragons NO llegará con una certeza del 88%. Otro acierto. Veamos los caminos tomados por cada una de las canciones:

¹⁰⁰<https://www.billboard.com/music/camila-cabello>

¹⁰¹<https://www.youtube.com/watch?v=BQ0mxQXmLsk>

¹⁰²<https://www.billboard.com/music/imagine-dragons>

¹⁰³<https://www.youtube.com/watch?v=7wtfhZwyrc>



Aquí vemos los caminos tomados por Havana en Rojo, que alcanzó el número 1 y el camino por Believer (en rosa) que no llegó.

Resumen

Pues hemos tenido un largo camino, para poder crear y generar nuestro árbol. Hemos revisado los datos de entrada, los hemos procesado, los pasamos a valores categóricos y generamos el árbol. Lo hemos puesto a prueba para validarla. Obtener un score de menos de 65% en el árbol no es un valor muy alto, pero tengamos en cuenta que nos pusimos una tarea bastante difícil de lograr: poder predecir al número 1 del Billboard y con un tamaño de muestras tan pequeño (635 registros) y desbalanceado¹⁰⁴. Ya quisieran las discográficas poder hacerlo :) Espero que hayan disfrutado de este artículo y si encuentran errores, comentarios, o sugerencias para mejorarlo, siempre son bienvenidas. Además pueden escribirme si tienen problemas en intentaré responder a la brevedad. Como siempre los invito a suscribirse al blog para seguir creciendo como comunidad de desarrolladores que estamos aprendiendo mediante ejemplos a crear algoritmos inteligentes.

Recursos y enlaces

- Descarga la Jupyter Notebook¹⁰⁵ y el archivo de entrada csv¹⁰⁶
 - ó puedes visualizar online¹⁰⁷
 - o ver y descargar desde github¹⁰⁸
 - Artículo Ejemplo Webscraping en Python¹⁰⁹
 - Cómo balancear tus set de datos¹¹⁰

¹⁰⁴<http://www.aprendemachinelearning.com/clasificacion-con-datos-desbalanceados/>

¹⁰⁵http://www.aprendemachinelearning.com/wp-content/uploads/2018/04/Ejercicio_Arbol_de_Decision.ipynb

http://www.aprendemachinelearning.com/wp-content/uploads/2018/04/artists_billboard_fix3.csv

¹⁰⁷https://github.com/jbagnato/machine-learning/blob/master/Ejercicio_Arbol_de_Decision.ipynb

¹⁰⁸ <https://github.com/jbagnato/machine-learning>

¹⁰⁹<http://www.aprendemachinelearning.com/ejemplo-web-scraping-python-ibex35-bolsa-valores/>

¹¹⁰<http://www.aprendemachinelearning.com/clasificacion-con-datos-desbalanceados/>

Otros enlaces con Artículos sobre Decisión Tree (en Inglés):

- [introduction-to-decision-trees-titanic-dataset¹¹¹](https://www.kaggle.com/dmilla/introduction-to-decision-trees-titanic-dataset)
- Decision Trees in Python¹¹²
- Decision Trees with Scikit learn¹¹³
- Building decision tree algorithm¹¹⁴

¹¹¹<https://www.kaggle.com/dmilla/introduction-to-decision-trees-titanic-dataset/notebook>

¹¹²<http://stackabuse.com/decision-trees-in-python-with-scikit-learn/>

¹¹³<https://napitupulu-jon.appspot.com/posts/decision-tree-ud.html>

¹¹⁴<http://dataaspirant.com/2017/02/01/decision-tree-algorithm-python-with-scikit-learn/>

Datos desbalanceados

Veremos qué son y cómo contrarrestar problemas con clases desbalanceadas.

Estrategias para resolver desequilibrio de datos en Python con la librería [imbalanced-learn¹¹⁵](#).

Agenda:

1. ¿Qué son las clases desequilibradas en un dataset?
 2. Métricas y Confusión Matrix
 3. Ejercicio con Python
 4. Estrategias
 5. Modelo sin modificar
 6. Penalización para compensar / Métricas
 7. Resampling y Muestras sintéticas
-
1. subsampling
 2. oversampling
 3. combinación
 8. Balanced Ensemble

Empecemos!

Problemas de clasificación con Clases desequilibradas

En los [problemas de clasificación¹¹⁶](#) en donde tenemos que etiquetar por ejemplo entre “spam” o “not spam” ó entre múltiples categorías (coche, barco, avión) solemos encontrar que en nuestro conjunto de datos de entrenamiento contamos con que alguna de las clases de muestra es una clase “minoritaria” es decir, de la cual tenemos muy poquitas muestras. Esto *provoca un desbalanceo en los datos* que utilizaremos para el entrenamiento de nuestra máquina.

Un caso evidente es en el área de Salud en donde solemos encontrar conjuntos de datos con miles de registros con pacientes “negativos” y unos pocos casos positivos es decir, que padecen la enfermedad que queremos clasificar.

Otros ejemplos suelen ser los de Detección de fraude donde tenemos muchas muestras de clientes “honestos” y pocos casos etiquetados como fraudulentos. Ó en un funnel de marketing, en donde por lo general tenemos un 2% de los datos de clientes que “compran” ó ejecutan algún tipo de acción (CTA) que queremos predecir.

¹¹⁵<https://imbalanced-learn.readthedocs.io/en/stable/>

¹¹⁶<https://www.aprendemachinelearning.com/aplicaciones-del-machine-learning/>

¿Cómo nos afectan los datos desbalanceados?

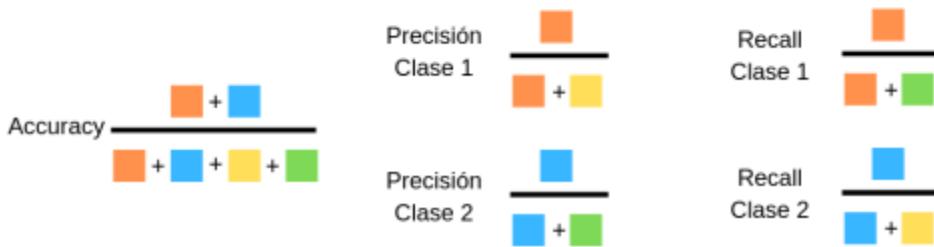
Por lo general afecta a los algoritmos en su proceso de generalización de la información y perjudicando a las clases minoritarias. Esto suena bastante razonable: si a una red neuronal le damos 990 de fotos de gatitos y sólo 10 de perros, no podemos pretender que logre diferenciar una clase de otra. Lo más probable que la red se limite a responder siempre “tu foto es un gato” puesto que así tuvo un acierto del 99% en su fase de entrenamiento.

Métricas y Confusion Matrix

Como decía, si medimos la efectividad de nuestro modelo por la cantidad de aciertos que tuvo, sólo teniendo en cuenta a la clase mayoritaria podemos estar teniendo **una falsa sensación de que el modelo funciona bien**.

Para poder entender esto un poco mejor, utilizaremos la llamada “Confusión matrix” que nos ayudará a comprender las salidas de nuestra máquina:

	Predicción Clase 1	Predicción Clase 2
Valor real Clase 1	Aciertos True Positive Clase 1	Fallos False Positive Clase 2
Valor real Clase 2	Fallos False Positive Clase 1	Aciertos True Positive Clase 2



117

Y de aqui salen nuevas métricas: precisión y recall

Veamos la Confusion matrix con el ejemplo de las predicciones de perro y gato.

¹¹⁷http://www.aprendemachinelearning.com/wp-content/uploads/2019/05/confusion_matix_example.png

	Predicción Gato	Predicción Perro
Valor real Gato	Aciertos 990	0
Valor real Perro	Fallos 10	0

$$\begin{array}{l}
 \text{Accuracy} = \frac{990 + 0}{990 + 0 + 10 + 0} \\
 \text{Precision Clase 1} = \frac{990}{990 + 10} \\
 \text{Recall Clase 1} = \frac{990}{990 + 0} \\
 \text{Precision Clase 2} = \frac{0}{0 + 0} \\
 \text{Recall Clase 2} = \frac{0}{0 + 10}
 \end{array}$$

Breve explicación de estas métricas:

La *Accuracy del modelo* es básicamente el número total de predicciones correctas dividido por el número total de predicciones. En este caso da 99% cuando no hemos logrado identificar ningún perro.

La *Precisión de una clase* define cuan confiable es un modelo en responder si un punto pertenece a esa clase. Para la clase gato será del 99% sin embargo para la de perro será 0%.

El *Recall de una clase* expresa cuan bien puede el modelo detectar a esa clase. Para gatos será de 1 y para perros 0.

*El F1 Score de una clase es dada por la media harmónica de precisión y recall *($2 \times \text{precision} \times \text{recall} / (\text{precision} + \text{recall})$) digamos que combina precisión y recall en una sola métrica. En nuestro caso daría cero para perros!.

Tenemos cuatro casos posibles para cada clase:

- **Alta precision y alto recall:** el modelo maneja perfectamente esa clase
- **Alta precision y bajo recall:** el modelo no detecta la clase muy bien, pero cuando lo hace es altamente confiable.
- **Baja precisión y alto recall:** La clase detecta bien la clase pero también incluye muestras de otras clases.
- **Baja precisión y bajo recall:** El modelo no logra clasificar la clase correctamente.

Cuando tenemos un dataset con desequilibrio, suele ocurrir que obtenemos un **alto valor de precisión en la clase Mayoritaria y un bajo recall en la clase Minoritaria**

Vamos al Ejercicio con Python!

Usaremos el set de datos [Credit Card Fraud Detection](https://www.kaggle.com/mlg-ulb/creditcardfraud) de la web de Kaggle¹¹⁸. Son 66 MB que al descomprimir ocuparán 150MB. Usaremos el archivo *creditcard.csv*. Este dataset consta de 285.000 filas con 31 columnas (features). Como la información es privada, no sabemos realmente que significan los features y están nombradas como V1, V2, V3, etc. excepto por las columnas Time y Amount (el importe de la transacción). Y nuestras clases son 0 y 1 correspondiendo con “transacción Normal” ó “Hubo Fraude”. Como podrán imaginar, el **set de datos está muy desequilibrado** y tendremos muy pocas muestras etiquetadas como fraude.

La notebook que acompaña este artículo puedes verla [en Github¹¹⁹](https://github.com/jbagnato/machine-learning/blob/master/Ejercicio_imbalanced_data.ipynb) y en los recursos, al final del artículo.

También debo decir que no nos centraremos tanto en [la elección del modelo¹²⁰](#) ni en su [configuración¹²¹](#) y [tuneo¹²²](#) si no que nos centraremos en aplicar las diversas estrategias para mejorar los resultados a pesar del desequilibrio de clases.

Instala la librería de Imbalanced Learn desde linea de comando con: ([toda la documentación en la web oficial imblearn¹²³](#))

```
1 pip install -U imbalanced-learn
```

Veamos el dataset

Análisis exploratorio

Haremos EDA para comprobar el desequilibrio entre las clases:

¹¹⁸<https://www.kaggle.com/mlg-ulb/creditcardfraud>

¹¹⁹https://github.com/jbagnato/machine-learning/blob/master/Ejercicio_imbalanced_data.ipynb

¹²⁰<http://www.aprendemachinelearning.com/principales-algoritmos-usados-en-machine-learning/>

¹²¹<http://www.aprendemachinelearning.com/7-pasos-machine-learning-construir-maquina/>

¹²²<http://www.aprendemachinelearning.com/12-consejos-utiles-para-aplicar-machine-learning/>

¹²³<https://imbalanced-learn.readthedocs.io/en/stable/>

```

1 import pandas as pd
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import seaborn as sns
5
6 from sklearn.metrics import confusion_matrix
7 from sklearn.metrics import classification_report
8 from sklearn.model_selection import train_test_split
9 from sklearn.linear_model import LogisticRegression
10 from sklearn.decomposition import PCA
11 from sklearn.tree import DecisionTreeClassifier
12
13 from pylab import rcParams
14
15 from imblearn.under_sampling import NearMiss
16 from imblearn.over_sampling import RandomOverSampler
17 from imblearn.combine import SMOTETomek
18 from imblearn.ensemble import BalancedBaggingClassifier
19
20 from collections import Counter

```

Luego de importar las librerías que usaremos, cargamos con pandas el dataframe y vemos las primeras filas:

```

1 df = pd.read_csv("creditcard.csv") # read in data downloaded to the local directory
2 df.head(n=5)

```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24
0	0.0	-1.358607	-0.072781	2.536347	1.378155	-0.338321	0.462368	0.239599	0.098698	0.363787	...	-0.016307	0.277838	-0.110474	0.066928
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078003	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.339846
2	1.0	-1.358354	-1.340163	1.773209	-0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247988	0.771579	0.899412	-0.669281
3	1.0	-0.968272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377458	-1.387024	...	-0.106300	0.005274	-0.190321	-1.175575
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.09431	0.798278	-0.137458	0.141267

5 rows × 31 columns

124

Veamos de cuantas filas tenemos y cuantas hay de cada clase:

```

1 print(df.shape)
2 print(pd.value_counts(df['Class'], sort = True))

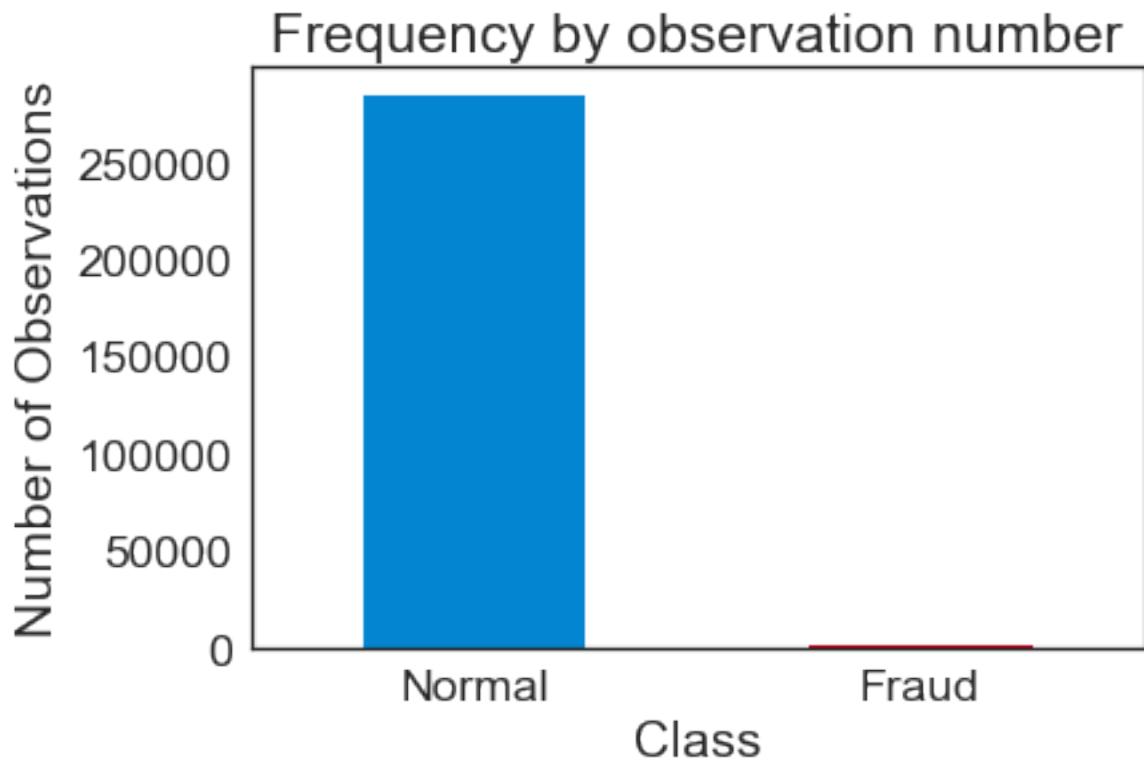
```

¹²⁴http://www.aprendemachinelearning.com/wp-content/uploads/2019/05/imbalance_dataframe.png

```
1 (284807, 31)
2
3 0 284315
4 1 492
5 Name: Class, dtype: int64
```

Vemos que son 284.807 filas y solamente 492 son la clase minoritaria con los casos de fraude. Representan el 0,17% de las muestras.

```
1 count_classes = pd.value_counts(df['Class'], sort = True)
2 count_classes.plot(kind = 'bar', rot=0)
3 plt.xticks(range(2), LABELS)
4 plt.title("Frequency by observation number")
5 plt.xlabel("Class")
6 plt.ylabel("Number of Observations");
```



¿Llegas a ver la mínima linea roja que representa los casos de Fraude? son muy pocas muestras!

¹²⁵http://www.aprendemachinelearning.com/wp-content/uploads/2019/05/imbalance_card_visualization.png

Estrategias para el manejo de Datos Desbalanceados:

Tenemos diversas estrategias para tratar de mejorar la situación. Las comentaremos brevemente y pasaremos a la acción (al código!) a continuación.

1. **Ajuste de Parámetros del modelo:** Consiste en ajustar parametros ó metricas del propio algoritmo para intentar equilibrar a la clase minoritaria penalizando a la clase mayoritaria durante el entrenamiento. Ejemplos on ajuste de peso en árboles, también en logisticregression tenemos el parámetro `class_weight= "balanced"` que utilizaremos en este ejemplo. No todos los algoritmos tienen estas posibilidades. En redes neuronales por ejemplo podríamos ajustar la métrica de Loss para que penalice a las clases mayoritarias.
2. **Modificar el Dataset:** podemos eliminar muestras de la clase mayoritaria para reducirlo e intentar equilibrar la situación. Tiene como “peligroso” que podemos prescindir de muestras importantes, que brindan información y por lo tanto empeorar el modelo. Entonces para seleccionar qué muestras eliminar, deberíamos seguir algún criterio. También podríamos agregar nuevas filas con los mismos valores de las clases minoritarias, por ejemplo cuadriplicar nuestras 492 filas. Pero esto no sirve demasiado y podemos llevar al modelo a caer en overfitting.
3. **Muestras artificiales:** podemos intentar crear muestras sintéticas (no idénticas) utilizando diversos algoritmos que intentan seguir la tendencia del grupo minoritario. Según el método, podemos mejorar los resultados. Lo peligroso de crear muestras sintéticas es que **podemos alterar la distribución** “natural” de esa clase y confundir al modelo en su clasificación.
4. **Balanced Ensemble Methods:** Utiliza las ventajas de hacer ensamble de métodos, es decir, entrenar diversos modelos y entre todos obtener el resultado final (por ejemplo “votando”) pero se asegura de tomar muestras de entrenamiento equilibradas.

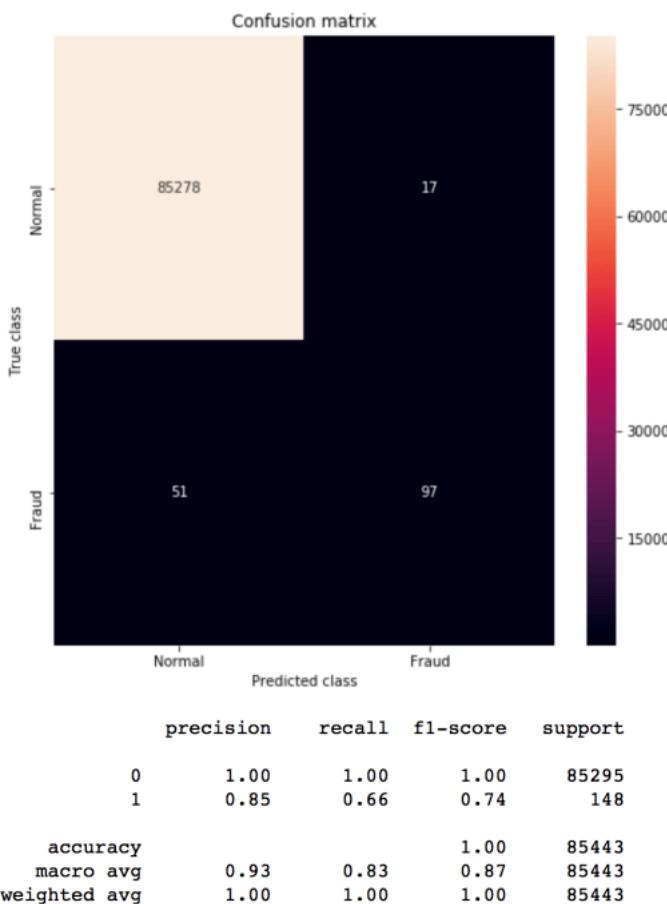
Aplicaremos estas técnicas de a una a nuestro código y veamos los resultados.

PERO... antes de empezar, ejecutaremos el modelo de [Regresión Logística¹²⁶](#) “desequilibrado”, para tener un “baseline”, es decir unas métricas contra las cuales podremos comparar y ver si mejoramos.

Probando el Modelo sin estrategias

¹²⁶<http://www.aprendemachinelearning.com/regresion-logistica-con-python-paso-a-paso/>

```
1 #definimos nuestras etiquetas y features
2 y = df['Class']
3 X = df.drop('Class', axis=1)
4 #dividimos en sets de entrenamiento y test
5 X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.7)
6
7 #creamos una función que crea el modelo que usaremos cada vez
8 def run_model(X_train, X_test, y_train, y_test):
9     clf_base = LogisticRegression(C=1.0, penalty='l2', random_state=1, solver="newton-c\
10 g")
11     clf_base.fit(X_train, y_train)
12     return clf_base
13
14 #ejecutamos el modelo "tal cual"
15 model = run_model(X_train, X_test, y_train, y_test)
16
17 #definimos función para mostrar los resultados
18 def mostrar_resultados(y_test, pred_y):
19     conf_matrix = confusion_matrix(y_test, pred_y)
20     plt.figure(figsize=(12, 12))
21     sns.heatmap(conf_matrix, xticklabels=LABELS, yticklabels=LABELS, annot=True, fmt\
22 ="d");
23     plt.title("Confusion matrix")
24     plt.ylabel('True class')
25     plt.xlabel('Predicted class')
26     plt.show()
27     print(classification_report(y_test, pred_y))
28
29 pred_y = model.predict(X_test)
30 mostrar_resultados(y_test, pred_y)
```



127

Aquí vemos la confusion matrix y en la clase 2 (es lo que nos interesa detectar) vemos 51 fallos y 97 aciertos dando un** recall de 0.66** y es el valor que queremos mejorar. También es interesante notar que en la columna de f1-score obtenemos muy buenos resultados PERO que realmente no nos deben engañar... pues están reflejando una realidad parcial. Lo cierto es que nuestro modelo no es capaz de detectar correctamente los casos de Fraude.

Estrategia: Penalización para compensar

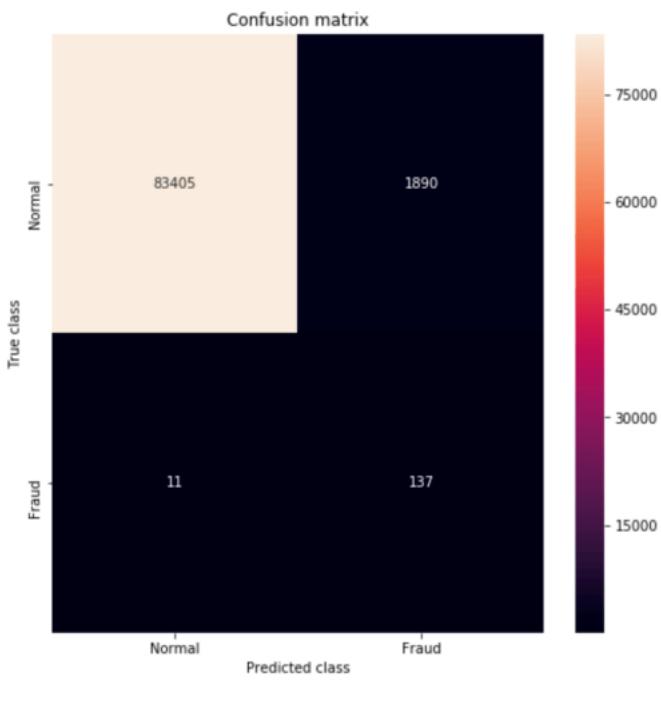
Utilizaremos un parámetro adicional en el modelo de Regresión logística en donde indicamos weight = “balanced” y con esto el algoritmo se encargará de equilibrar a la clase minoritaria durante el entrenamiento. Veamos:

¹²⁷http://www.aprendemachinelearning.com/wp-content/uploads/2019/05/confus_normal.png

```

1 def run_model_balanced(X_train, X_test, y_train, y_test):
2     clf = LogisticRegression(C=1.0,penalty='12',random_state=1,solver="newton-cg",cl\
3 ass_weight="balanced")
4     clf.fit(X_train, y_train)
5     return clf
6
7 model = run_model_balanced(X_train, X_test, y_train, y_test)
8 pred_y = model.predict(X_test)
9 mostrar_resultados(y_test, pred_y)

```



	precision	recall	f1-score	support
0	1.00	0.98	0.99	85295
1	0.07	0.93	0.13	148
accuracy			0.98	85443
macro avg	0.53	0.95	0.56	85443
weighted avg	1.00	0.98	0.99	85443

128

Ahora vemos una NOTABLE MEJORA! en la clase 2 -que indica si hubo fraude-, se han acertado 137 muestras y fallado en 11, dando un recall de 0.93 !! y sólo con agregar un parámetro al modelo ;) También notemos que en la columna de f1-score parecería que hubieran “empeorado” los resultados... cuando realmente estamos mejorando la detección de casos fraudulentos. Es cierto que aumentan los Falsos Positivos y se han etiquetado 1890 muestras como Fraudulentas cuando no lo eran... pero ustedes piensen... ¿qué prefiere la compañía bancaria? ¿tener que revisar esos casos manualmente ó fallar en detectar los verdaderos casos de fraude?

¹²⁸http://www.aprendemachinelearning.com/wp-content/uploads/2019/05/confus_balanced.png

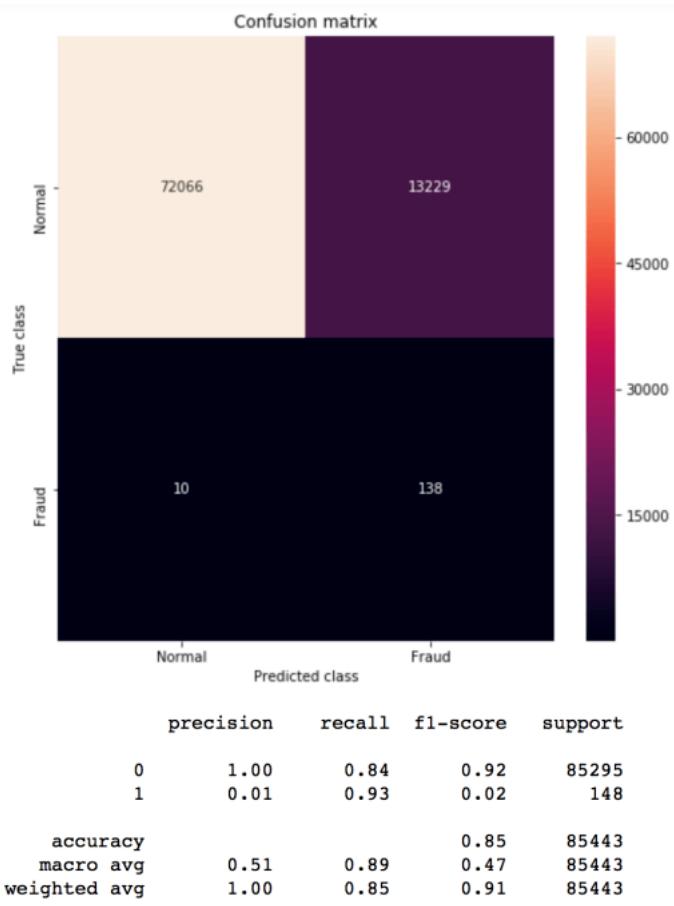
Sigamos con más métodos:

Estrategia: Subsampling en la clase mayoritaria

Lo que haremos es utilizar un algoritmo para reducir la clase mayoritaria. Lo haremos usando un algoritmo que hace similar al k-nearest neighbor para ir seleccionando cuales eliminar. Fijemonos que reducimos bestialmente de 199.020 muestras de clase cero (la mayoría) y pasan a ser 688. y Con esas muestras entrenamos el modelo.

```
1 us = NearMiss(ratio=0.5, n_neighbors=3, version=2, random_state=1)
2 X_train_res, y_train_res = us.fit_sample(X_train, y_train)
3
4 print ("Distribution before resampling {}".format(Counter(y_train)))
5 print ("Distribution after resampling {}".format(Counter(y_train_res)))
6
7 model = run_model(X_train_res, X_test, y_train_res, y_test)
8 pred_y = model.predict(X_test)
9 mostrar_resultados(y_test, pred_y)

1 Distribution before resampling Counter({0: 199020, 1: 344})
2 Distribution after resampling Counter({0: 688, 1: 344})
```



129

También vemos que obtenemos muy buen resultado con recall de 0.93 aunque a costa de que aumentaran los falsos positivos.

Estrategia: Oversampling de la clase minoritaria

En este caso, crearemos muestras nuevas “sintéticas” de la clase minoritaria. Usando RandomOverSampler. Y vemos que pasamos de 344 muestras de fraudes a 99.510.

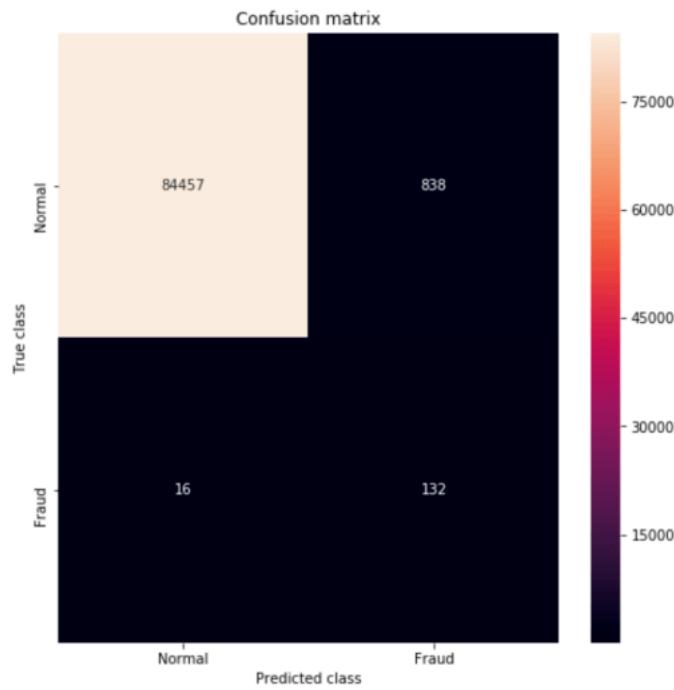
¹²⁹http://www.aprendemachinelearning.com/wp-content/uploads/2019/05/confus_subsampling.png

```

1 os = RandomOverSampler(ratio=0.5)
2 X_train_res, y_train_res = os.fit_sample(X_train, y_train)
3
4 print ("Distribution before resampling {}".format(Counter(y_train)))
5 print ("Distribution labels after resampling {}".format(Counter(y_train_res)))
6
7 model = run_model(X_train_res, X_test, y_train_res, y_test)
8 pred_y = model.predict(X_test)
9 mostrar_resultados(y_test, pred_y)

1 Distribution before resampling Counter({0: 199020, 1: 344})
2 Distribution after resampling Counter({0: 199020, 1: 99510})

```



	precision	recall	f1-score	support
0	1.00	0.99	0.99	85295
1	0.14	0.89	0.24	148
accuracy			0.99	85443
macro avg	0.57	0.94	0.62	85443
weighted avg	1.00	0.99	0.99	85443

130

Tenemos un 0.89 de recall para la clase 2 y los Falsos positivos son 838. Nada mal.

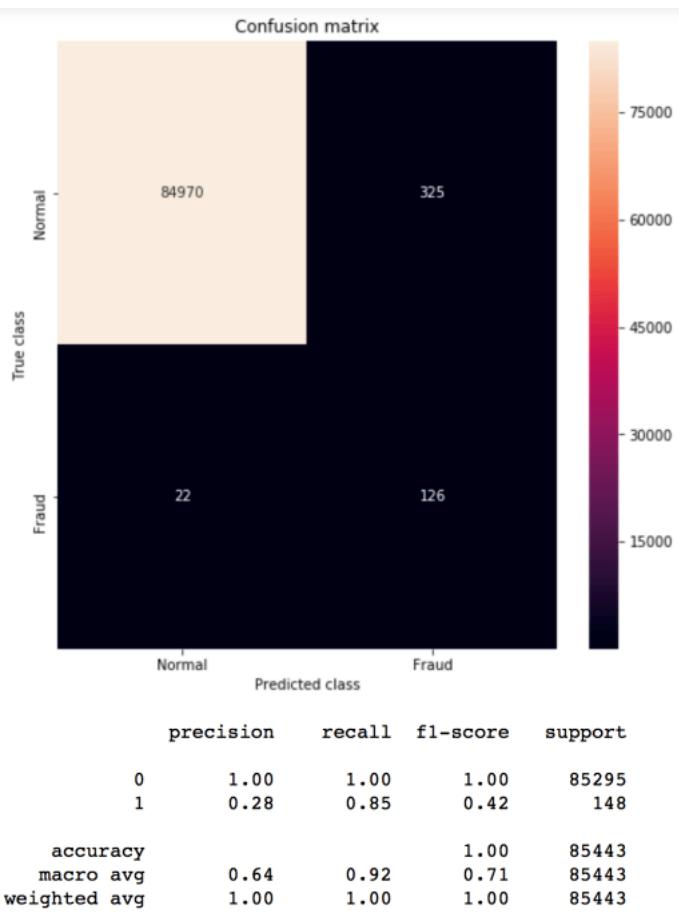
¹³⁰http://www.aprendemachinelearning.com/wp-content/uploads/2019/05/confus_oversampling.png

Estrategia: Combinamos resampling con Smote-Tomek

Ahora probaremos una técnica muy usada que consiste en **aplicar en simultáneo** un algoritmo de subsampling y otro de oversampling a la vez al dataset. En este caso usaremos SMOTE para oversampling: busca puntos vecinos cercanos y agrega puntos “en linea recta” entre ellos. Y usaremos Tomek para undersampling que quita los de distinta clase que sean nearest neighbor y deja ver mejor el decisión boundary (la zona límitrofe de nuestras clases).

```
1 os_us = SMOTETomek(ratio=0.5)
2 X_train_res, y_train_res = os_us.fit_sample(X_train, y_train)
3
4 print ("Distribution before resampling {}".format(Counter(y_train)))
5 print ("Distribution after resampling {}".format(Counter(y_train_res)))
6
7 model = run_model(X_train_res, X_test, y_train_res, y_test)
8 pred_y = model.predict(X_test)
9 mostrar_resultados(y_test, pred_y)

1 Distribution labels before resampling Counter({0: 199020, 1: 344})
2 Distribution after resampling Counter({0: 198194, 1: 98684})
```



En este caso seguimos teniendo bastante buen recall 0.85 de la clase 2 y vemos que los Falsos positivos de la clase 1 son bastante pocos, 325 (de 85295 muestras).

Estrategia: Ensamble de Modelos con Balanceo

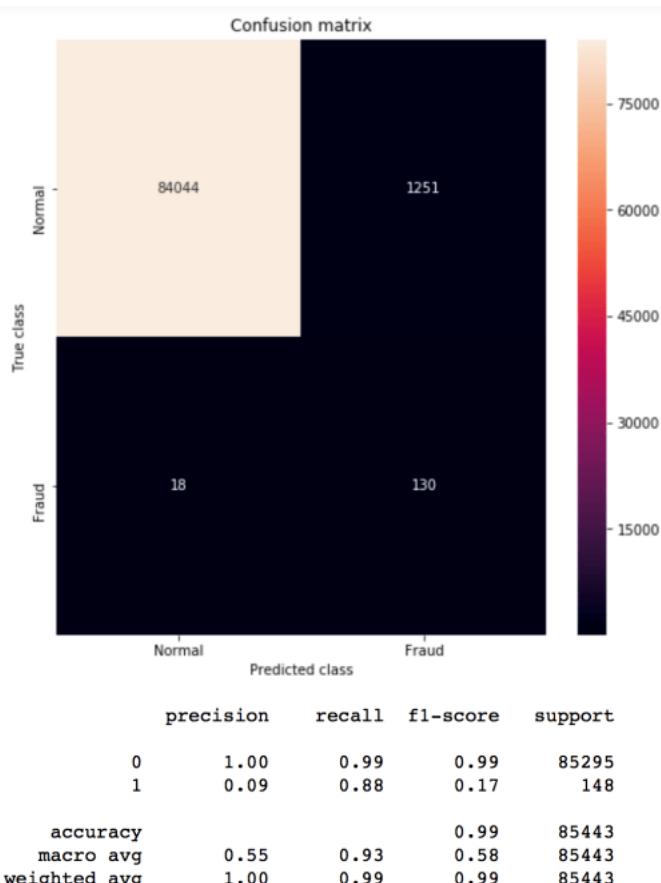
Para esta estrategia usaremos un Clasificador de Ensamble que utiliza Bagging y el modelo será un DecisionTree. Veamos como se comporta:

¹³¹<http://www.aprendemachinelearning.com/wp-content/uploads/2019/05/smote-tomek.png>

```

1 bbc = BalancedBaggingClassifier(base_estimator=DecisionTreeClassifier(),
2                                 sampling_strategy='auto',
3                                 replacement=False,
4                                 random_state=0)
5
6 #Train the classifier.
7 bbc.fit(X_train, y_train)
8 pred_y = bbc.predict(X_test)
9 mostrar_resultados(y_test, pred_y)

```



132

Tampoco está mal. Vemos siempre mejora con respecto al modelo inicial con un recall de 0.88 para los casos de fraude.

Resultados de las Estrategias

Veamos en una tabla, ordenada de mejor a peor los resultados obtenidos.

¹³²http://www.aprendemachinelearning.com/wp-content/uploads/2019/05/balance_ensemble.png

	algorithm	precision	recall	overall
1	Penalizacion	1.0	0.93	0.965
2	NearMiss Subsampling	1.0	0.93	0.965
3	Random Oversampling	1.0	0.89	0.945
5	Ensemble	1.0	0.88	0.940
4	Smote Tomek	1.0	0.85	0.925
0	Regresion Logística	1.0	0.66	0.830

133

Vemos que en nuestro caso las estrategias de Penalización y Subsampling nos dan el mejor resultado, cada una con un recall de 0.93.

Pero quedémonos con esto: **Con cualquiera de las técnicas que aplicamos MEJORAMOS el modelo inicial de Regresión logística**, que lograba un 0.66 de recall para la clase de Fraude. Y no olvidemos que hay un tremendo desbalance de clases en el dataset!

IMPORTANTE: esto no quiere decir que siempre hay que aplicar Penalización ó NearMiss Subsampling!, dependerá del caso, del desbalanceo y del modelo (en este caso usamos regresión logística, pero podría ser otro!).

Resumen

Es muy frecuente encontrarnos con datasets con clases desbalanceadas, de hecho... lo más raro sería encontrar datasets bien equilibrados.

Siempre que puedas **“Sal a la calle y consigue más muestras!”** pero la realidad es que a veces no es posible conseguir más datos de las clases minoritarias (como por ejemplo en Casos de Salud).

Vimos diversas estrategias a seguir para combatir esta problemática: eliminar muestras del set mayoritario, crear muestras sintéticas con algún criterio, ensamble y penalización.

Además revisamos la **Matriz de Confusión** y comprendimos que **las métricas pueden ser engañosas...** si miramos a nuestros aciertos únicamente, puede que pensemos que tenemos un buen clasificador, cuando realmente está fallando.

Recursos

- Notebook con todo el ejercicio y más cositas¹³⁴

¹³³http://www.aprendemachinelearning.com/wp-content/uploads/2019/05/imbalance_result.png

¹³⁴https://github.com/jbagnato/machine-learning/blob/master/Ejercicio_imbalanced_data.ipynb

- Descarga el dataset desde Kaggle¹³⁵
- Librería de Imbalanced-Learn¹³⁶

Enlaces de interés

- 8 tactics to combat imbalanced classes¹³⁷
- How to fix unbalanced dataset¹³⁸

¹³⁵<https://www.kaggle.com/mlg-ulb/creditcardfraud/data>

¹³⁶<https://imbalanced-learn.readthedocs.io/en/stable/>

¹³⁷<https://machinelearningmastery.com/tactics-to-combat-imbalanced-classes-in-your-machine-learning-dataset/>

¹³⁸<https://www.kdnuggets.com/2019/05/fix-unbalanced-dataset.html>

Random Forest, el poder del Ensamble

Luego del algoritmo de [árbol de Decisión¹³⁹](#), tu próximo paso es el de estudiar Random Forest. **Comprende qué és y cómo funciona** con un ejemplo práctico en Python.

Random Forest es un tipo de Ensamble en Machine Learning en donde combinaremos diversos árboles -ya veremos cómo y con qué características- y la salida de cada uno se contará como “*un voto*” y la opción más votada será la respuesta del Bosque Aleatorio.

Random Forest, al igual que el [árbol e decisión¹⁴⁰](#) es un [modelo de aprendizaje supervisado¹⁴¹](#) para [clasificación¹⁴²](#) (aunque también puede usarse para problemas de regresión).

¿Cómo surge Random Forest?

Uno de los problemas que aparecía con la creación de un árbol de decisión es que si le damos la profundidad suficiente, el árbol tiende a “memorizar” las soluciones en vez de generalizar el aprendizaje. Es decir, a [padecer de overfitting¹⁴³](#). La solución para evitar esto es la de crear muchos árboles y que trabajen en conjunto. Veamos cómo.

¿Cómo funciona Random Forest?

Random Forest funciona así:

- Seleccionamos **k** *features* (columnas) de las **m** totales (siendo **k** menor a **m**) y creamos un árbol de decisión con esas **k** características.
- Creamos **n** árboles variando siempre la cantidad de **k** *features* y también podríamos variar la cantidad de muestras que pasamos a esos árboles (esto es conocido como “*bootstrap sample*”)
- Tomamos **cada uno** de los **n** árboles y le pedimos que hagan una misma clasificación. Guardamos el resultado de cada árbol obteniendo **n** salidas.
- Calculamos los votos obtenidos para cada “clase” seleccionada y consideraremos a la más votada como la clasificación final de nuestro “bosque”.

¹³⁹<http://www.aprendemachinelearning.com/arbol-de-decision-en-python-clasificacion-y-prediccion/>

¹⁴⁰<http://www.aprendemachinelearning.com/principales-algoritmos-usados-en-machine-learning/>

¹⁴¹<http://www.aprendemachinelearning.com/aplicaciones-del-machine-learning/#supervisado>

¹⁴²<http://www.aprendemachinelearning.com/principales-algoritmos-usados-en-machine-learning/>

¹⁴³<http://www.aprendemachinelearning.com/que-es-overfitting-y-underfitting-y-como-solucionarlo/>

¿Por qué es aleatorio?

Contamos con una <>doble aleatoriedad>>: tanto en la selección del valor k de características para cada árbol como en la cantidad de muestras que usaremos para entrenar cada árbol creado.

Es curioso que para este algoritmo la aleatoriedad sea tan importante y de hecho es lo que lo “hace bueno”, pues le brinda flexibilidad suficiente como para poder obtener gran variedad de árboles y de muestras que en su conjunto aparentemente caótico, producen una salida concreta. Darwin estaría orgulloso ;)

Ventajas y Desventajas del uso de Random Forest

Vemos algunas de sus ventajas son:

- funciona bien -aún- sin ajuste de hiperparámetros¹⁴⁴
- funciona bien para problemas de clasificación y también de regresión.
- al utilizar múltiples árboles se reduce considerablemente el riesgo de overfitting¹⁴⁵
- se mantiene estable con nuevas muestras puesto que al utilizar cientos de árboles sigue prevaleciendo el promedio de sus votaciones.

Y sus desventajas:

- en algunos datos de entrada “particulares” random forest también puede caer en overfitting
- es mucho más “costo” de crear y ejecutar que “un sólo árbol” de decisión.
- Puede requerir muchísimo tiempo de entrenamiento
- OJO! Random Forest no funciona bien con datasets pequeños.
- Es muy difícil poder interpretar¹⁴⁶ los cientos? de árboles creados en el bosque, si quisieramos comprender y explicar a un cliente su comportamiento.

Vamos al Código Python

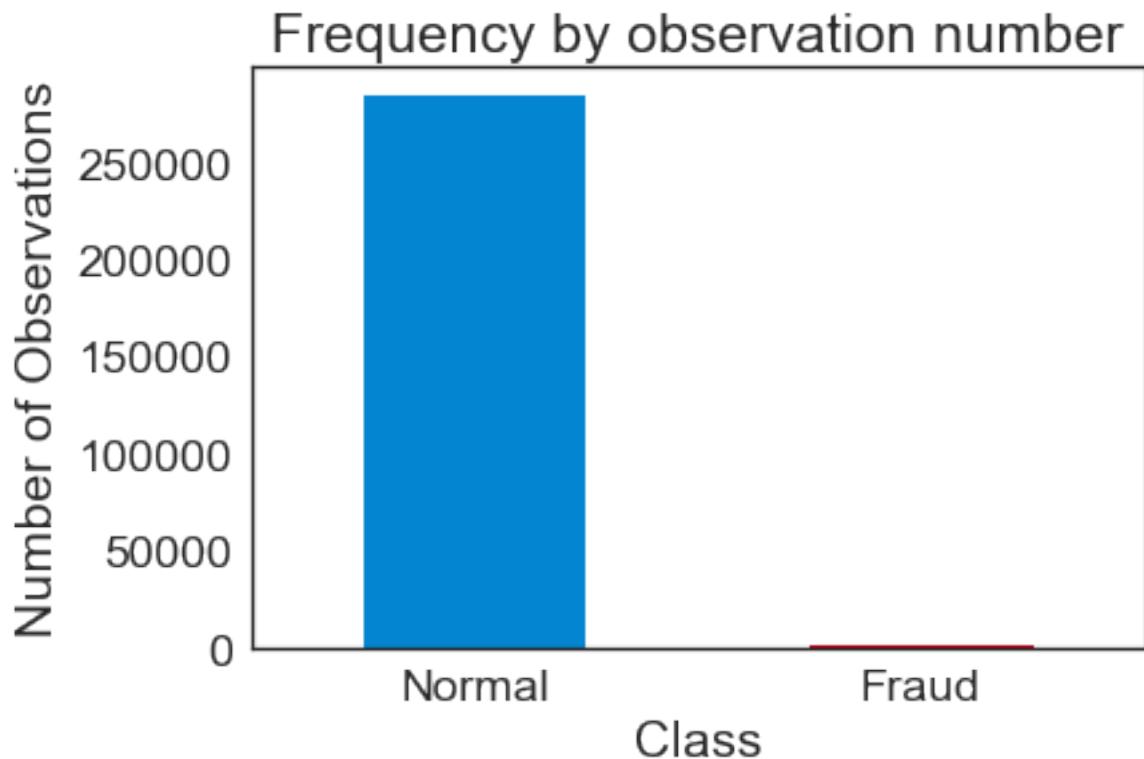
Continuaremos con el ejercicio propuesto en el artículo “desbalanceo de datos¹⁴⁷” en donde utilizamos el dataset de Kaggle con información de fraude en tarjetas de crédito. Cuenta con 284807 filas y 31 columnas de características. Nuestra salida será 0 si es un cliente “normal” o 1 si hizo uso fraudulento.

¹⁴⁴<http://www.aprendemachinelearning.com/7-pasos-machine-learning-construir-maquina/>

¹⁴⁵<http://www.aprendemachinelearning.com/que-es-overfitting-y-underfitting-y-como-solucionarlo/>

¹⁴⁶<http://www.aprendemachinelearning.com/interpretacion-de-modelos-de-machine-learning/>

¹⁴⁷<http://www.aprendemachinelearning.com/clasificacion-con-datos-desbalanceados/>



¿Llegas a ver la mínima linea roja que representa los casos de Fraude? son apenas 492 frente a más de 250.000 casos de uso normal.

Retomaremos el mejor caso que obtuvimos en el ejercicio anterior utilizando [Regresión Logística¹⁴⁸](#) y logrando un 98% de aciertos, pero recuerda también las métricas de F1, precisión y recall que eran las que realmente nos ayudaban a validar el modelo.

Creamos el modelo y lo entrenamos

Utilizaremos el modelo RandomForestClassifier de Scikit-Learn.

¹⁴⁸https://en.wikipedia.org/wiki/Logistic_regression

```
1 from sklearn.ensemble import RandomForestClassifier  
2  
3 # Crear el modelo con 100 arboles  
4 model = RandomForestClassifier(n_estimators=100,  
5                                bootstrap = True, verbose=2,  
6                                max_features = 'sqrt')  
7 # a entrenar!  
8 model.fit(X_train, y_train)
```

Luego de unos minutos obtendremos el modelo entrenado (en mi caso 1 minuto 30 segundos)

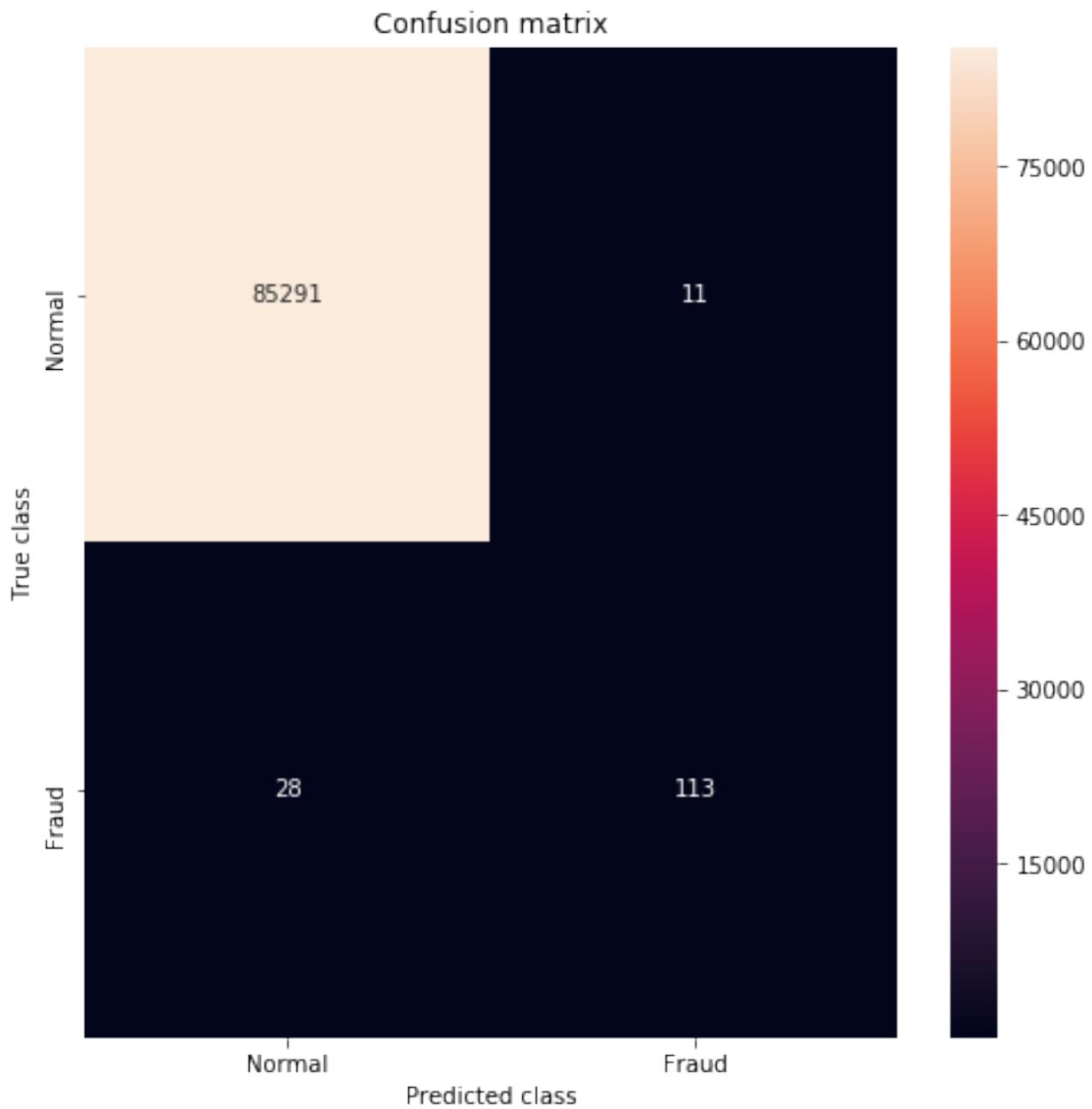
Los Hiperparámetros más importantes

Al momento de ajustar el modelo, debemos tener en cuenta los siguientes hiperparámetros. Estos nos ayudarán a que el bosque de mejores resultados para cada ejercicio. Recuerda que esto no se trata de “copiar y pegar”!

- **n_estimators**: será la cantidad de árboles que generaremos.
- **max_features**: la manera de seleccionar la cantidad máxima de features para cada árbol.
- **min_sample_leaf**: número mínimo de elementos en las hojas para permitir un nuevo split (división) del nodo.
- **oob_score**: es un método que emula el cross-validation en árboles y permite mejorar la precisión y evitar overfitting.
- **bootstrap**: para utilizar diversos tamaños de muestras para entrenar. Si se pone en falso, utilizará siempre el dataset completo.
- **n_jobs**: si tienes multiples cores en tu CPU, puedes indicar cuantos puede usar el modelo al entrenar para acelerar el entrenamiento.

Evaluamos resultados

Veamos la matriz de confusión y las métricas sobre el conjunto de test!!! (no confundir con el de training!!!)



Vemos muy buenos resultados, clasificando con error apenas 11 + 28 muestras.

	precision	recall	f1-score	support
0	1.00	1.00	1.00	85302
1	0.91	0.80	0.85	141
accuracy			1.00	85443
macro avg	0.96	0.90	0.93	85443
weighted avg	1.00	1.00	1.00	85443

Aquí podemos destacar que para la clase “minoritaria”, es decir la que detecta los casos de fraude tenemos un buen valor de recall (de 0.80) lo cual es un buen indicador! y el F1-score macro avg es de 0.93. Logramos construir un modelo de Bosque aleatorio que a pesar de tener un conjunto de datos de entrada muy desigual, logra buenos resultados.

Comparamos con el Baseline

Si comparamos estos resultados con los del algoritmo de Regresión Logística¹⁴⁹, vemos que el Random Forest nos dio mejores clasificaciones, menos falsos positivos¹⁵⁰ y mejores métricas en general.

Resumen

Avanzando en nuestro aprendizaje sobre diversos modelos que podemos aplicar a las problemáticas que nos enfrentamos, hoy sumamos a nuestro kit de herramientas¹⁵¹ el Random Forest, vemos que es un modelo sencillo, bastante rápido y si bien perdemos la interpretabilidad¹⁵² maravillosa que nos brindaba 1 sólo árbol de decisión, es el precio a pagar para evitar el overfitting¹⁵³ y para ganar un clasificador más robusto.

Los algoritmos Tree-Based¹⁵⁴ -en inglés- son muchos, todos parten de la idea principal de árbol de decisión¹⁵⁵ y la mejoran con diferentes tipos de ensambles y técnicas. Tenemos que destacar a 2 modelos que según el caso logran superar a las mismísimas redes neuronales¹⁵⁶! son XGboost y LightGBM. Si te parecen interesantes puede que en el futuro escribamos sobre ellos.

Recursos y Adicionales

Puedes descargar la notebook para este ejercicio desde mi cuenta de GitHub:

- Código en jupyter Notebook en GitHub¹⁵⁷
- Dataset de Kaggle¹⁵⁸

Otros artículos sobre Random Forest en inglés:

- Random Forest Simple Explanation¹⁵⁹
- An Implementation of Random Forest in Python¹⁶⁰

¹⁴⁹<http://www.aprendemachinelearning.com/regresion-logistica-con-python-paso-a-paso/>

¹⁵⁰<http://www.aprendemachinelearning.com/clasificacion-con-datos-desbalanceados/>

¹⁵¹<http://www.aprendemachinelearning.com/principales-algoritmos-usados-en-machine-learning/>

¹⁵²<http://www.aprendemachinelearning.com/interpretacion-de-modelos-de-machine-learning/>

¹⁵³<http://www.aprendemachinelearning.com/que-es-overfitting-y-underfitting-y-como-solucionarlo/>

¹⁵⁴<http://www.aprendemachinelearning.com/principales-algoritmos-usados-en-machine-learning/>

¹⁵⁵<http://www.aprendemachinelearning.com/arbol-de-decision-en-python-clasificacion-y-prediccion/>

¹⁵⁶<http://www.aprendemachinelearning.com/una-sencilla-red-neuronal-en-python-con-keras-y-tensorflow/>

¹⁵⁷https://github.com/jbagnato/machine-learning/blob/master/Ejercicio_Random_Forest.ipynb

¹⁵⁸<https://www.kaggle.com/mlg-ulb/creditcardfraud/data>

¹⁵⁹<https://medium.com/@williamkoehrsen/random-forest-simple-explanation-377895a60d2d>

¹⁶⁰<https://towardsdatascience.com/an-implementation-and-explanation-of-the-random-forest-in-python-77bf308a9b76>

K-Means

K-Means es un algoritmo [no supervisado¹⁶¹](#) de [Clustering¹⁶²](#). Se utiliza cuando tenemos un montón de datos **sin etiquetar**. El objetivo de este algoritmo es el de encontrar “K” grupos (clusters) entre los datos crudos. En este artículo repasaremos sus conceptos básicos y veremos un ejemplo paso a paso en python que podemos descargar.

Cómo funciona K-Means

El algoritmo trabaja iterativamente para asignar a cada “punto” (las filas de nuestro conjunto de entrada forman una coordenada) uno de los “K” grupos basado en sus características. Son agrupados en base a la similitud de sus features (las columnas). Como resultado de ejecutar el algoritmo tendremos:

- Los “centroids” de cada grupo que serán unas “coordenadas” de cada uno de los K conjuntos que se utilizarán para poder etiquetar nuevas muestras.
- Etiquetas para el conjunto de datos de entrenamiento. Cada etiqueta perteneciente a uno de los K grupos formados.

Los grupos se van definiendo de manera “orgánica”, es decir que se va ajustando su posición en cada iteración del proceso, hasta que converge el algoritmo. Una vez hallados los centroids deberemos analizarlos para ver cuales son sus características únicas, frente a la de los otros grupos. Estos grupos son las etiquetas que genera el algoritmo.

Casos de Uso de K-Means

El algoritmo de Clustering K-means es [uno de los más usados¹⁶³](#) para encontrar grupos ocultos, o sospechados en teoría sobre un conjunto de datos no etiquetado. Esto puede servir para confirmar -o desterrar- alguna teoría que teníamos asumida de nuestros datos. Y también puede ayudarnos a descubrir relaciones asombrosas entre conjuntos de datos, que de manera manual, no hubiéramos reconocido. Una vez que el algoritmo ha ejecutado y obtenido las etiquetas, será fácil clasificar nuevos valores o muestras entre los grupos obtenidos. Algunos casos de uso son:

- Segmentación por Comportamiento: relacionar el carrito de compras de un usuario, sus tiempos de acción e información del perfil.

¹⁶¹http://www.aprendemachinelearning.com/aplicaciones-del-machine-learning/#no_supervisado

¹⁶²<http://www.aprendemachinelearning.com/principales-algoritmos-usados-en-machine-learning/#clustering>

¹⁶³<http://www.aprendemachinelearning.com/principales-algoritmos-usados-en-machine-learning/>

- Categorización de Inventory: agrupar productos por actividad en sus ventas
- Detectar anomalías o actividades sospechosas: según el comportamiento en una web reconocer un troll -o un bot- de un usuario normal

Datos de Entrada para K-Means

Las “features” o características que utilizaremos como entradas para aplicar el algoritmo k-means deberán ser de valores numéricos, continuos en lo posible. En caso de valores categóricos (por ej. Hombre/Mujer o Ciencia Ficción, Terror, Novela,etc) se puede intentar pasarlo a valor numérico, pero no es recomendable pues no hay una “distancia real” -como en el caso de géneros de película o libros-. Además es recomendable que los valores utilizados estén normalizados, manteniendo una misma escala. En algunos casos también funcionan mejor datos porcentuales en vez de absolutos. No conviene utilizar features que estén correlacionados o que sean escalares de otros. Recordar los [7 pasos para el Aprendizaje Automático¹⁶⁴](#).

El Algoritmo K-means

El algoritmo utiliza una proceso iterativo en el que se van ajustando los grupos para producir el resultado final. Para ejecutar el algoritmo deberemos pasar como entrada el conjunto de datos y un valor de K. El conjunto de datos serán las características o features para cada punto. Las posiciones iniciales de los K centroids serán asignadas de manera aleatoria de cualquier punto del conjunto de datos de entrada. Luego se itera en dos pasos:

1- Paso de Asignación de datos En este paso, cada “fila” de nuestro conjunto de datos se asigna al centroide más cercano basado en la distancia cuadrada Euclídea. Se utiliza la siguiente fórmula (donde $dist()$ es la distancia Euclídea standard):

$$\underset{c_i \in C}{\operatorname{argmin}} dist(c_i, x)^2$$

2-Paso de actualización de Centroid En este paso los centroides de cada grupo son recalculados. Esto se hace tomando una media de todos los puntos asignados en el paso anterior.

$$c_i = \frac{1}{|S_i|} \sum_{x_i \in S_i} x_i$$

El algoritmo itera entre estos pasos hasta cumplir un criterio de detención:

* si no hay cambios en los puntos asignados a los grupos,

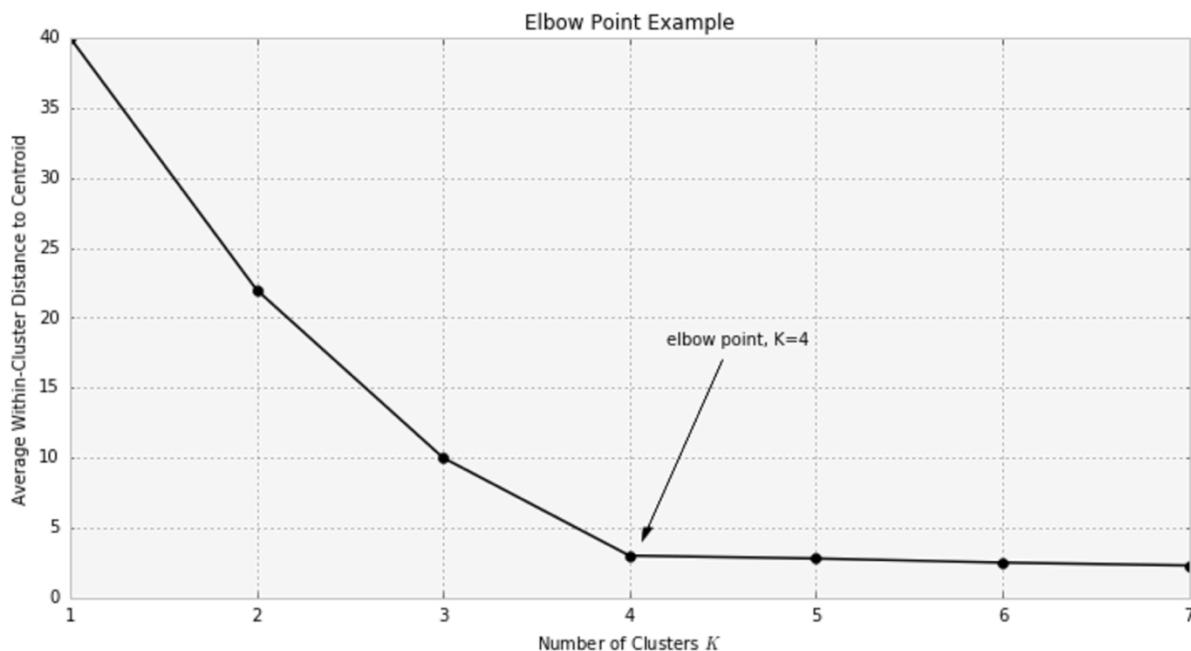
¹⁶⁴<http://www.aprendemachinelearning.com/7-pasos-machine-learning-construir-maquina/>

- o si la suma de las distancias se minimiza,
- o se alcanza un número máximo de iteraciones.

El algoritmo converge a un resultado que puede ser el óptimo local, por lo que será conveniente volver a ejecutar más de una vez con puntos iniciales aleatorios para confirmar si hay una salida mejor. Recuerden siempre seguir [los 7 pasos para construir IA¹⁶⁵](#)

Elegir el valor de K

Este algoritmo funciona pre-seleccionando un valor de K. Para encontrar el número de clusters en los datos, deberemos ejecutar el algoritmo para un rango de valores K, ver los resultados y comparar características de los grupos obtenidos. En general no hay un modo exacto de determinar el valor K, pero se puede estimar con aceptable precisión siguiendo la siguiente técnica: Una de las métricas usada para comparar resultados es la **distancia media entre los puntos de datos y su centroid**. Como el valor de la media diminuirá a medida de aumentemos el valor de K, deberemos utilizar la distancia media al centroide en función de K y entonrar el “punto codo”, donde la tasa de descenso se “afila”. Aquí vemos una gráfica a modo de ejemplo:



Ejemplo K-Means con Scikit-learn

Como ejemplo utilizaremos de entradas un conjunto de datos que obtuve de un proyecto propio, en el que se analizaban rasgos de la personalidad de usuarios de Twitter. He filtrado a 140 “famosos” del

¹⁶⁵<http://www.aprendemachinelearning.com/7-pasos-machine-learning-construir-maquina/>

mundo en diferentes areas: deporte, cantantes, actores, etc. Basado en una metodología de psicología conocida como “Ocean: The Big Five” tendemos como características de entrada:

- usuario (el nombre en Twitter)
- “op” = Openness to experience - grado de apertura mental a nuevas experiencias, curiosidad, arte
- “co” =Conscientiousness - grado de orden, prolijidad, organización
- “ex” = Extraversion - grado de timidez, solitario o participación ante el grupo social
- “ag” = Agreeableness - grado de empatía con los demás, temperamento
- “ne” = Neuroticism, - grado de neuroticismo, nervioso, irritabilidad, seguridad en sí mismo.
- Wordcount - Cantidad promedio de palabras usadas en sus tweets
- Categoría - Actividad laboral del usuario (actor, cantante, etc.)

Utilizaremos el algoritmo K-means para que agrupe estos usuarios -no por su actividad laboral- si no, por sus similitudes en la personalidad. Si bien tenemos 8 columnas de entrada, **sólo utilizaremos 3** en este ejemplo, de modo que podamos ver en un gráfico tridimensional -y sus proyecciones a 2D- los grupos resultantes. Pero para casos reales, podemos utilizar todas las dimensiones que necesitemos. Una de las hipótesis que podríamos tener es: “Todos los cantantes tendrán personalidad parecida” (y así con cada rubro laboral). Pues veremos si lo probamos, o por el contrario, los grupos no están relacionados necesariamente con la actividad de estas Celebridades.

Requerimientos para el Ejercicio

Necesitaremos tener Python 3.6. Mejor si tenemos instalada una suite como [Anaconda¹⁶⁶](https://www.anaconda.com/download/) o [Canopy¹⁶⁷](https://store.enthought.com/downloads/) (que funcionan en Windows, Mac y Linux). Puedes seguir este tutorial donde explico [cómo instalar tu ambiente de desarrollo¹⁶⁸](#). Crearemos un [Jupyter notebook¹⁶⁹](#) para seguir paso a paso el ejercicio e importaremos un [archivo de entrada csv¹⁷⁰](#). Utilizaremos los paquetes scikit-learn, pandas, matplotlib y numpy.

Agrupar usuarios Twitter de acuerdo a su personalidad con K-means

Implementando K-means en Python con Sklearn

Comenzaremos importando las librerías que nos asistirán para ejecutar el algoritmo y graficar.

¹⁶⁶<https://www.anaconda.com/download/>

¹⁶⁷<https://store.enthought.com/downloads/>

¹⁶⁸<http://www.aprendemachinelearning.com/installar-ambiente-de-desarrollo-python-anaconda-para-aprendizaje-automatico/>

¹⁶⁹http://www.aprendemachinelearning.com/wp-content/uploads/2018/03/Ejercicio_K_Means.ipynb

¹⁷⁰<http://www.aprendemachinelearning.com/wp-content/uploads/2018/03/analisis.csv>

```

1 import pandas as pd
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import seaborn as sb
5 from sklearn.cluster import KMeans
6 from sklearn.metrics import pairwise_distances_argmin_min
7
8 %matplotlib inline
9 from mpl_toolkits.mplot3d import Axes3D
10 plt.rcParams['figure.figsize'] = (16, 9)
11 plt.style.use('ggplot')

```

Importamos el archivo csv¹⁷¹ -para simplificar, suponemos que el archivo se encuentra en el mismo directorio que el notebook- y vemos los primeros 5 registros del archivo tabulados.

```

1 dataframe = pd.read_csv(r"analisis.csv")
2 dataframe.head()

```

	usuario	op	co	ex	ag	ne	wordcount	categoria
0	3gerardpique	34.297953	28.148819	41.948819	29.370315	9.841575	37.0945	7
1	aguerosergiokun	44.986842	20.525865	37.938947	24.279098	10.362406	78.7970	7
2	albertochicote	41.733854	13.745417	38.999896	34.645521	8.836979	49.2604	4
3	AlejandroSanz	40.377154	15.377462	52.337538	31.082154	5.032231	80.4538	2
4	alfredocasero1	36.664677	19.642258	48.530806	31.138871	7.305968	47.0645	4

También podemos ver una tabla de información estadística que nos provee Pandas dataframe:

```
1 dataframe.describe()
```

¹⁷¹<http://www.aprendemachinelearning.com/wp-content/uploads/2018/03/analisis.csv>

	op	co	ex	ag	ne	wordcount	categoria
count	140.000000	140.000000	140.000000	140.000000	140.000000	140.000000	140.000000
mean	44.414591	22.977135	40.764428	22.918528	8.000098	98.715484	4.050000
std	8.425723	5.816851	7.185246	7.657122	3.039248	44.714071	2.658839
min	30.020465	7.852756	18.693542	9.305985	1.030213	5.020800	1.000000
25%	38.206484	19.740299	36.095722	17.050993	6.086144	66.218475	2.000000
50%	44.507091	22.466718	41.457492	21.384554	7.839722	94.711400	3.500000
75%	49.365923	26.091606	45.197769	28.678867	9.758189	119.707925	7.000000
max	71.696129	49.637863	59.824844	40.583162	23.978462	217.183200	9.000000

El archivo contiene diferenciadas 9 categorías -actividades laborales- que son:

1. Actor/actriz
2. Cantante
3. Modelo
4. Tv, series
5. Radio
6. Tecnología
7. Deportes
8. Política
9. Escritor

Para saber cuantos registros tenemos de cada uno hacemos:

```
1 print(dataframe.groupby('categoria').size())
```

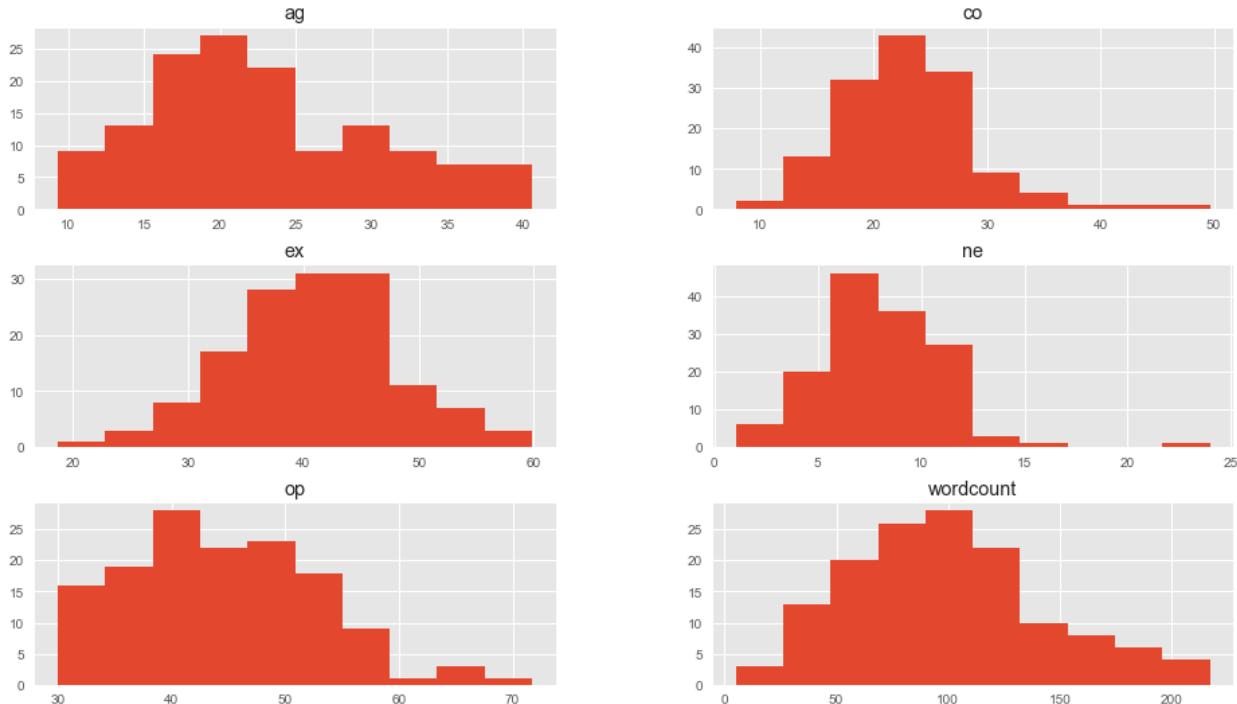
```
categoria
1    27
2    34
3     9
4    19
5     4
6     8
7    17
8    16
9     6
dtype: int64
```

Como vemos tenemos 34 cantantes, 27 actores, 17 deportistas, 16 políticos,etc.

Visualización de Datos

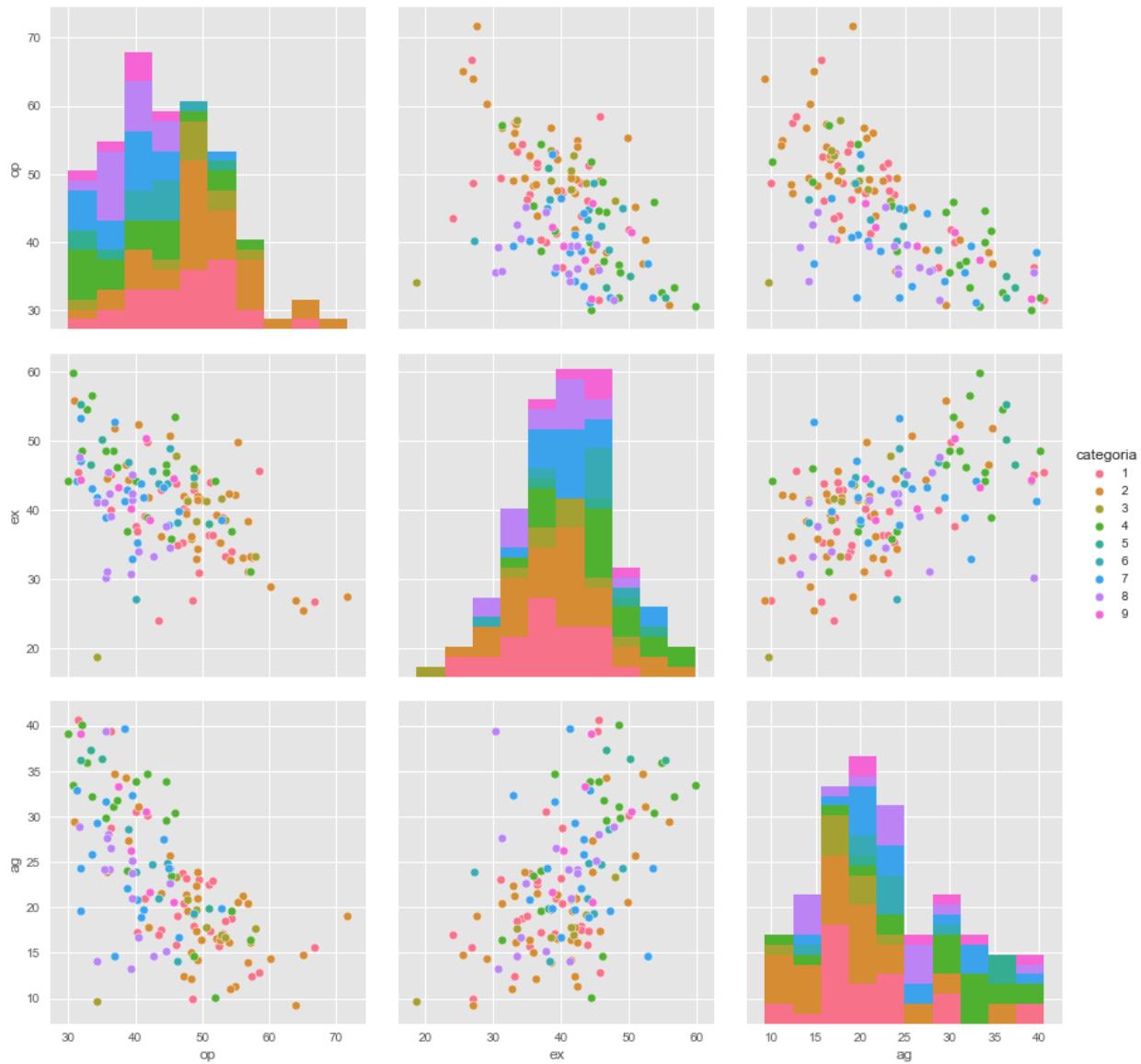
Veremos graficamente nuestros datos para tener una idea de la dispersión de los mismos:

```
1 dataframe.drop(['categoria'],1).hist()
2 plt.show()
```



En este caso seleccionamos 3 dimensiones: op, ex y ag y las cruzamos para ver si nos dan alguna pista de su agrupación y la relación con sus categorías.

```
1 sb.pairplot(dataframe.dropna(), hue='categoria', size=4, vars=["op", "ex", "ag"], kind='s\
2 catter')
```

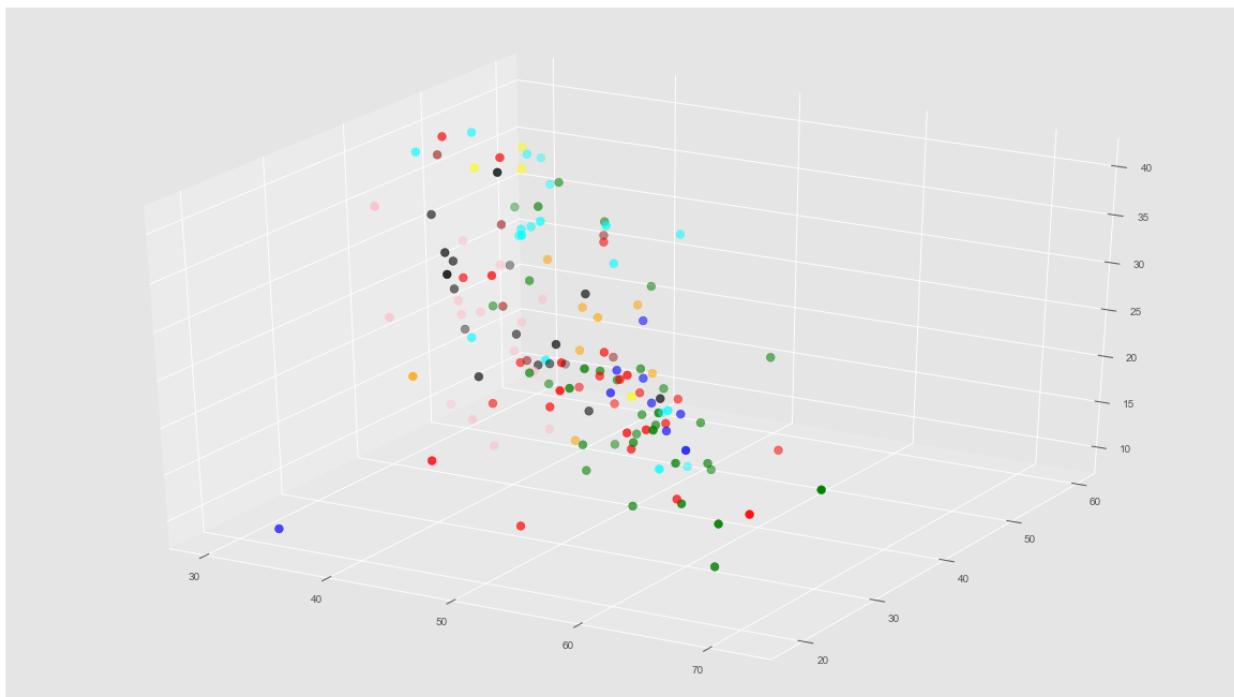


Revisando la gráfica no pareciera que ha ha algún tipo de agrupación o correlación entre los usuarios y sus categorías.

Definimos la entrada

Concretamos la estructura de datos que utilizaremos para alimentar el algoritmo. Como se ve, sólo cargamos las columnas op, ex y ag en nuestra variable X.

```
1 X = np.array(dataframe[["op", "ex", "ag"]])
2 y = np.array(dataframe['categoria'])
3 X.shape
4 ````python
5
6 ~~~
7 (140,3)
8 ~~~
9
10 Ahora veremos una gráfica en 3D con 9 colores representando las categorías.
11 ````python
12 fig = plt.figure()
13 ax = Axes3D(fig)
14 colores=['blue', 'red', 'green', 'blue', 'cyan', 'yellow', 'orange', 'black', 'pink', 'brown' \
15 , 'purple']
16 asignar=[]
17 for row in y:
18     asignar.append(colores[row])
19 ax.scatter(X[:, 0], X[:, 1], X[:, 2], c=asignar,s=60)
```

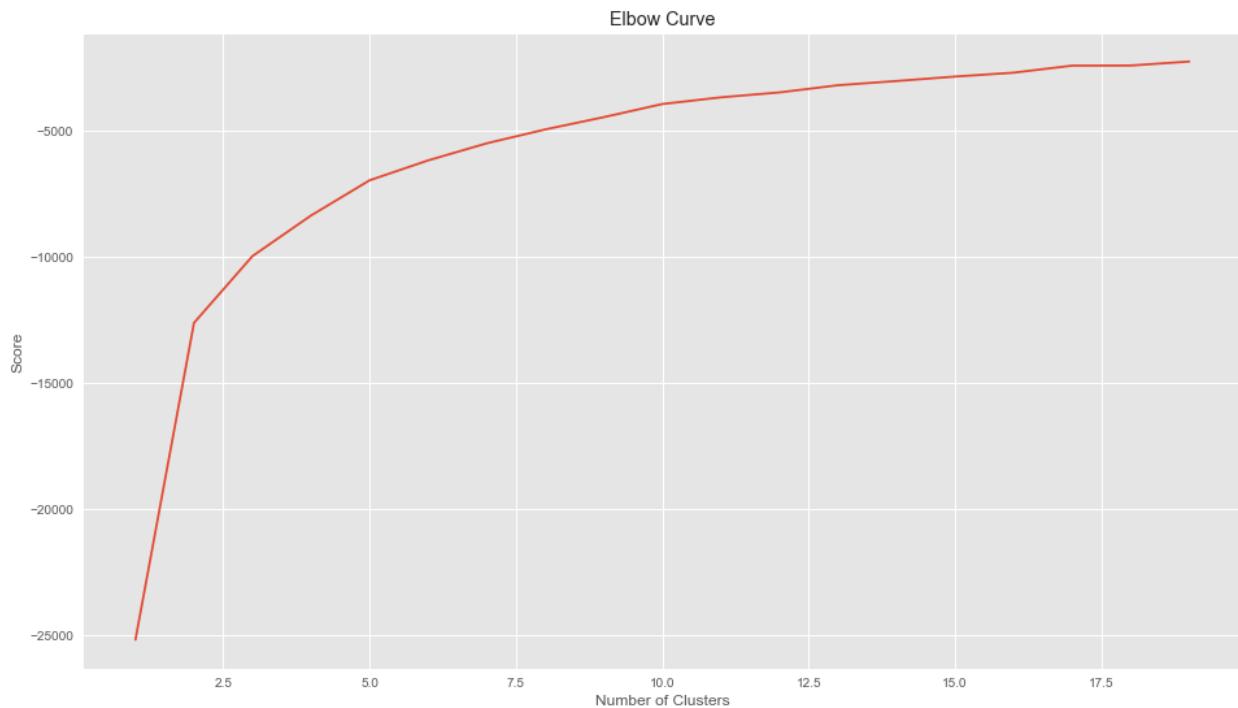


Veremos si con K-means, podemos “pintar” esta misma gráfica de otra manera, con clusters diferenciados.

Obtener el valor K

Vamos a hallar el valor de K haciendo una gráfica e intentando hallar el “punto de codo” que comentábamos antes. Este es nuestro resultado:

```
1 Nc = range(1, 20)
2 kmeans = [KMeans(n_clusters=i) for i in Nc]
3 kmeans
4 score = [kmeans[i].fit(X).score(X) for i in range(len(kmeans))]
5 score
6 plt.plot(Nc,score)
7 plt.xlabel('Number of Clusters')
8 plt.ylabel('Score')
9 plt.title('Elbow Curve')
10 plt.show()
```



Realmente la curva es bastante “suave”. Considero a 5 como un buen número para K. Según vuestro criterio podría ser otro.

Ejecutamos K-Means

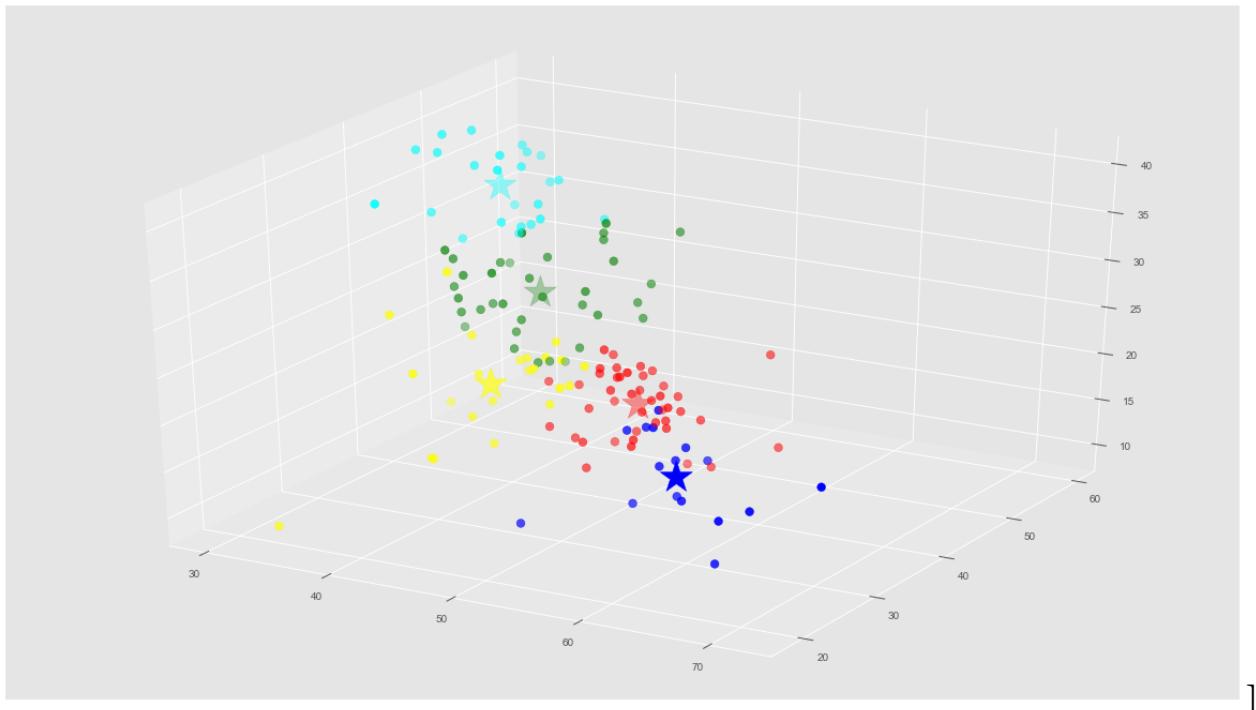
Ejecutamos el algoritmo para 5 clusters y obtenemos las etiquetas y los centroids.

```
1 kmeans = KMeans(n_clusters=5).fit(X)
2 centroids = kmeans.cluster_centers_
3 print(centroids)

[[ 49.80086386  40.8972579   17.48224326]
 [ 39.63830586  44.75784737  25.86962057]
 [ 58.58657531  31.02839375  15.6120435 ]
 [ 34.5303535   48.01261321  35.01749504]
 [ 42.302263    33.65449587  20.812626 ]]
```

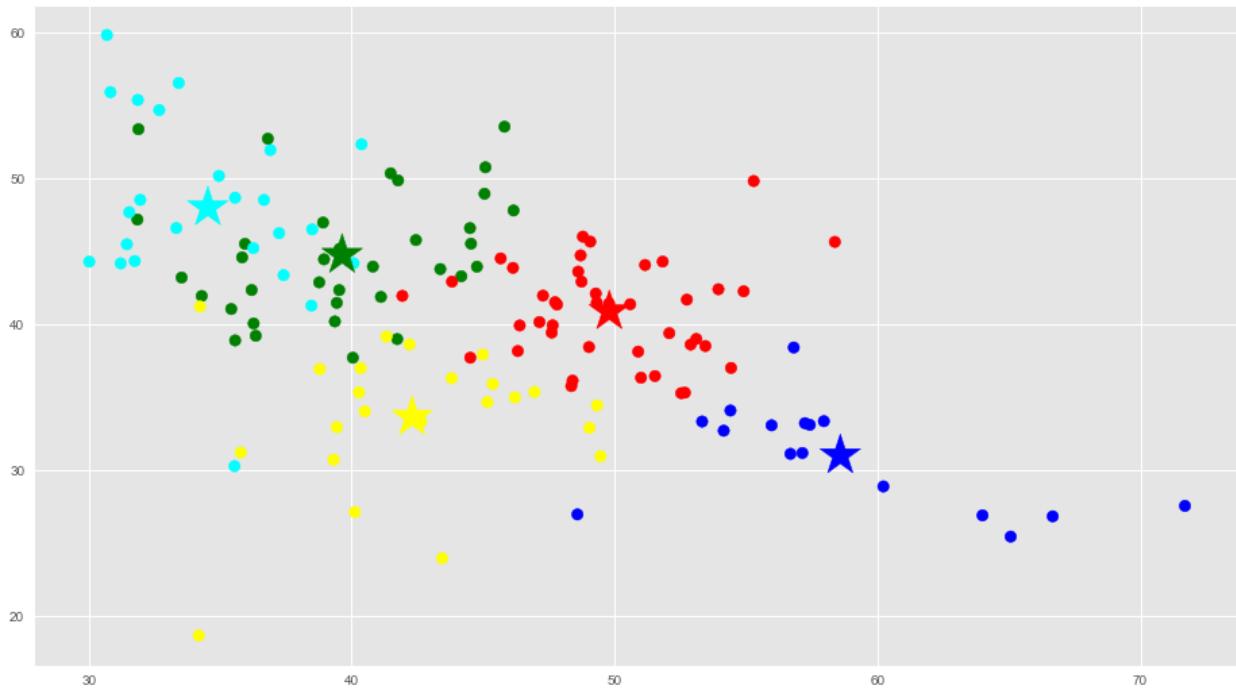
Ahora veremos esto en una gráfica 3D con colores para los grupos y veremos si se diferencian: (las estrellas marcan el centro de cada cluster)

```
1 # Predicting the clusters
2 labels = kmeans.predict(X)
3 # Getting the cluster centers
4 C = kmeans.cluster_centers_
5 colores=['red','green','blue','cyan','yellow']
6 asignar=[]
7 for row in labels:
8     asignar.append(colores[row])
9
10 fig = plt.figure()
11 ax = Axes3D(fig)
12 ax.scatter(X[:, 0], X[:, 1], X[:, 2], c=asignar,s=60)
13 ax.scatter(C[:, 0], C[:, 1], C[:, 2], marker='*', c=colores, s=1000)
```

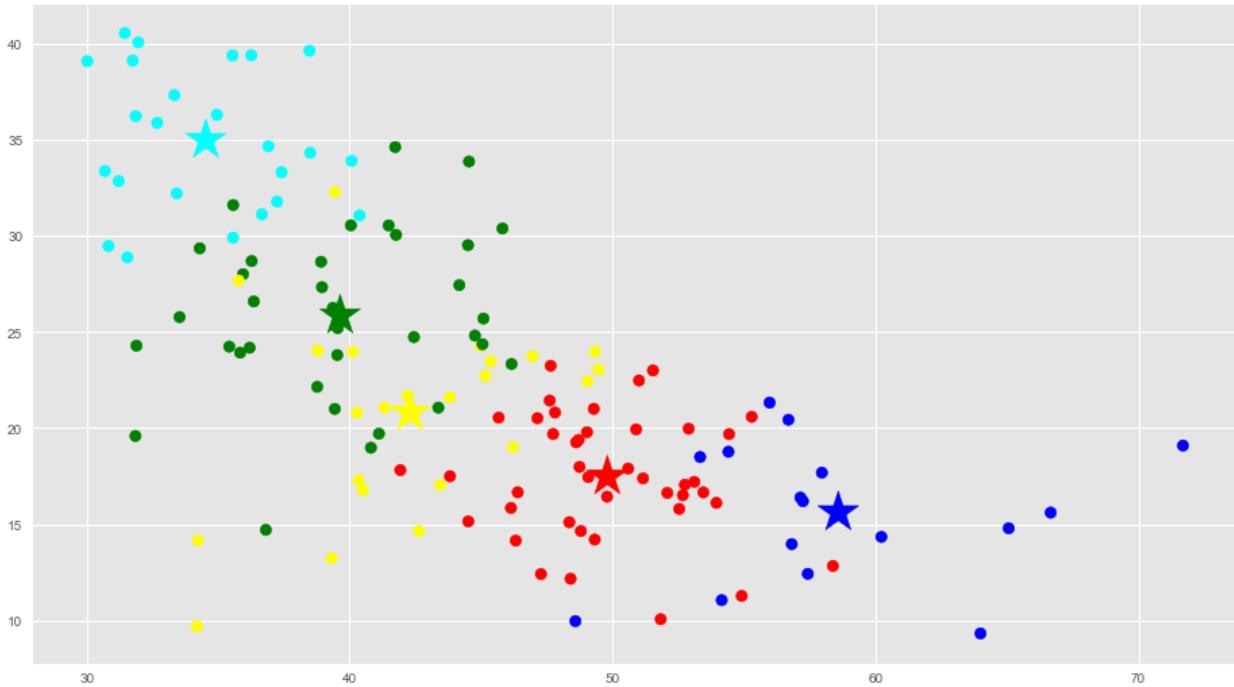


Aquí podemos ver que el Algoritmo de K-Means con $K=5$ ha agrupado a los 140 usuarios Twitter por su personalidad, teniendo en cuenta las 3 dimensiones que utilizamos: Openess, Extraversión y Agreeableness. Pareciera que no hay necesariamente una relación en los grupos con sus actividades de Celebrity. Haremos 3 gráficas en 2 dimensiones con las proyecciones a partir de nuestra gráfica 3D para que nos ayude a visualizar los grupos y su clasificación:

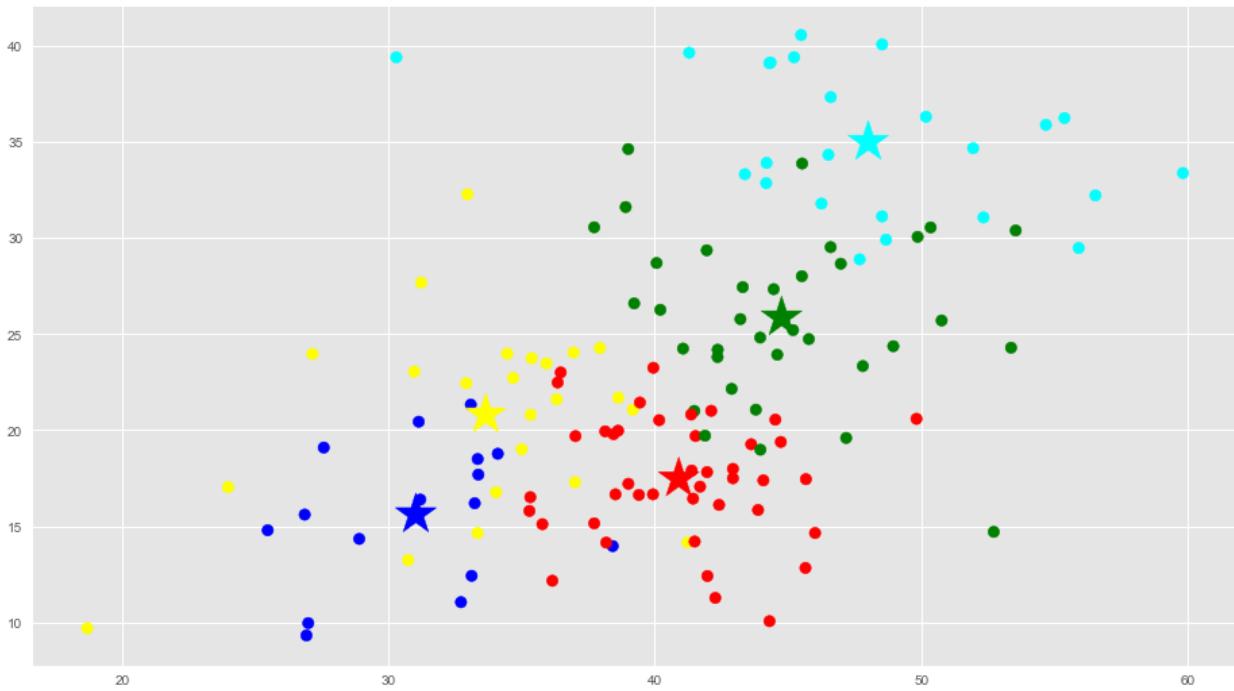
```
1 # Getting the values and plotting it
2 f1 = dataframe['op'].values
3 f2 = dataframe['ex'].values
4
5 plt.scatter(f1, f2, c=asignar, s=70)
6 plt.scatter(C[:, 0], C[:, 1], marker='*', c=colores, s=1000)
7 plt.show()
```



```
1 # Getting the values and plotting it
2 f1 = dataframe['op'].values
3 f2 = dataframe['ag'].values
4
5 plt.scatter(f1, f2, c=asignar, s=70)
6 plt.scatter(C[:, 0], C[:, 2], marker='*', c=colores, s=1000)
7 plt.show()
```



```
1 f1 = dataframe['ex'].values
2 f2 = dataframe['ag'].values
3
4 plt.scatter(f1, f2, c=asignar, s=70)
5 plt.scatter(C[:, 1], C[:, 2], marker='*', c=colores, s=1000)
6 plt.show()
```



En estas gráficas vemos que están bastante bien diferenciados los grupos. Podemos ver cada uno de los clusters cuantos usuarios tiene:

```

1 copy = pd.DataFrame()
2 copy['usuario']=dataframe['usuario'].values
3 copy['categoria']=dataframe['categoria'].values
4 copy['label'] = labels;
5 cantidadGrupo = pd.DataFrame()
6 cantidadGrupo['color']=colores
7 cantidadGrupo['cantidad']=copy.groupby('label').size()
8 cantidadGrupo

```

	color	cantidad
0	red	42
1	green	33
2	blue	16
3	cyan	27
4	yellow	22

Y podemos ver la diversidad en rubros laborales de cada uno. Por ejemplo en el grupo 0 (rojo), vemos que hay de todas las actividades laborales aunque predominan de actividad 1 y 2 correspondiente a Actores y Cantantes con 11 y 15 famosos.

```

1 group_referrer_index = copy['label'] ==0
2 group_referrals = copy[group_referrer_index]
3
4 diversidadGrupo = pd.DataFrame()
5 diversidadGrupo['categoria']=[0,1,2,3,4,5,6,7,8,9]
6 diversidadGrupo['cantidad']=group_referrals.groupby('categoria').size()
7 diversidadGrupo

```

	categoria	cantidad
0	0	NaN
1	1	11.0
2	2	15.0
3	3	6.0
4	4	3.0
5	5	1.0
6	6	2.0
7	7	2.0
8	8	1.0
9	9	1.0

De categoría 3 “modelos” hay 6 sobre un total de 9. Buscaremos los usuarios que están más cerca a los centroids de cada grupo que podríamos decir que tienen los rasgos de personalidad característicos que representan a cada cluster:

```

1 #vemos el representante del grupo, el usuario cercano a su centroid
2 closest, _ = pairwise_distances_argmin_min(kmeans.cluster_centers_, X)
3 closest

1 array([21, 107, 82, 80, 91]) #posicion en el array de usuarios

```

```

1 users=dataframe[ 'usuario' ].values
2 for row in closest:
3     print(users[row] )

1 carmenelectra Pablo_Iglesias_ JudgeJudy JPVarsky kobe Bryant

```

En los centros vemos que tenemos una modelo, un político, presentadora de Tv, locutor de Radio y un deportista.

Clasificar nuevas muestras

Y finalmente podemos agrupar y etiquetar nuevos usuarios twitter con sus características y clasificarlos. Vemos el ejemplo con el usuario de David Guetta y nos devuelve que pertenece al grupo 1 (verde).

```

1 X_new = np.array([[45.92,57.74,15.66]]) #davidguetta
2
3 new_labels = kmeans.predict(X_new)
4 print(new_labels)

1 [1]

```

Resumen

El algoritmo de K-means nos ayudará a crear clusters cuando tengamos grandes grupos de datos sin etiquetar, cuando queramos intentar descubrir nuevas relaciones entre features o para probar o declinar hipótesis que tengamos de nuestro negocio. Atención: Puede haber casos en los que **no existan grupos naturales**, o clusters que contengan una verdadera razón de ser. Si bien K-means siempre nos brindará “k clusters”, quedará en nuestro criterio reconocer la utilidad de los mismos o bien revisar nuestras features y descartar las que no sirven o conseguir nuevas. También tener en cuenta que en este ejemplo estamos utilizando como medida de similitud entre features la **distancia Euclídea¹⁷²** pero podemos utilizar otras diversas funciones que podrían arrojar mejores resultados (como **Manhattan¹⁷³**, **Lavenshtein¹⁷⁴**, **Mahalanobis¹⁷⁵**, etc). Hemos visto una descripción del algoritmo, aplicaciones y un ejemplo python paso a paso, que podrán descargar también desde los siguientes enlaces:

¹⁷²https://es.wikipedia.org/wiki/Distancia_euclidiana

¹⁷³https://es.wikipedia.org/wiki/Geometria_Del_taxista

¹⁷⁴https://en.wikipedia.org/wiki/Levenshtein_distance

¹⁷⁵https://en.wikipedia.org/wiki/Mahalanobis_distance

- Notebook Jupiter Online¹⁷⁶
- Descargar archivo csv¹⁷⁷ y notebook ejercicio K-means¹⁷⁸
- Visualizar¹⁷⁹ y descargar desde jbagnato Github¹⁸⁰

¹⁷⁶http://nbviewer.jupyter.org/github/jbagnato/machine-learning/blob/master/Ejercicio_K_Means.ipynb

¹⁷⁷<http://www.aprendemachinelearning.com/wp-content/uploads/2018/03/analisis.csv>

¹⁷⁸http://www.aprendemachinelearning.com/wp-content/uploads/2018/03/Ejercicio_K_Means.ipynb

¹⁷⁹https://github.com/jbagnato/machine-learning/blob/master/Ejercicio_K_Means.ipynb

¹⁸⁰<https://github.com/jbagnato/machine-learning>

K-Nearest-Neighbor

K-Nearest-Neighbor es un algoritmo [basado en instancia¹⁸¹](#) de tipo [supervisado¹⁸²](#) de Machine Learning. Puede usarse para clasificar nuevas muestras (valores discretos) o para predecir (regresión, valores continuos). Al ser un método sencillo, es ideal para introducirse en el mundo del Aprendizaje Automático. Sirve esencialmente para clasificar valores buscando los puntos de datos “más similares” (por cercanía) aprendidos en la etapa de entrenamiento ([ver 7 pasos para crear tu ML¹⁸³](#)) y haciendo conjeturas de nuevos puntos basado en esa clasificación.

A diferencia de [K-means¹⁸⁴](#), que es un algoritmo [no supervisado¹⁸⁵](#) y donde la “K” significa la cantidad de “grupos” ([clusters¹⁸⁶](#)) que deseamos clasificar, en K-Nearest Neighbor la “K” significa la cantidad de “puntos vecinos” que tenemos en cuenta en las cercanías para clasificar los “n” grupos -que ya se conocen de antemano, pues es un algoritmo supervisado-.

¿Qué es el algoritmo k-Nearest Neighbor ?

Es un método que simplemente busca en las observaciones más cercanas a la que se está tratando de predecir y clasifica el punto de interés basado en la mayoría de datos que le rodean. Como dijimos antes, es un algoritmo:

- **Supervisado:** esto -brevemente- quiere decir que tenemos etiquetado nuestro conjunto de datos de entrenamiento, con la clase o resultado esperado dada “una fila” de datos.
- **Basado en Instancia:** Esto quiere decir que nuestro algoritmo no aprende explícitamente un modelo (como por ejemplo en Regresión Logística o árboles de decisión). En cambio memoriza las instancias de entrenamiento que son usadas como “base de conocimiento” para la fase de predicción.

¿Dónde se aplica k-Nearest Neighbor?

Aunque sencillo, se utiliza en la resolución de multitud de problemas, como en **sistemas de recomendación, búsqueda semántica y detección de anomalías**.

¹⁸¹<http://www.aprendemachinelearning.com/principales-algoritmos-usados-en-machine-learning/#instancia>

¹⁸²<http://www.aprendemachinelearning.com/aplicaciones-del-machine-learning/#supervisado>

¹⁸³<http://www.aprendemachinelearning.com/7-pasos-machine-learning-construir-maquina/>

¹⁸⁴<http://www.aprendemachinelearning.com/k-means-en-python-paso-a-paso/>

¹⁸⁵http://www.aprendemachinelearning.com/aplicaciones-del-machine-learning/#no_supervisado

¹⁸⁶<http://www.aprendemachinelearning.com/principales-algoritmos-usados-en-machine-learning/#clustering>

Pros y contras

Como **pros** tiene sobre todo que es sencillo de aprender e implementar. Tiene como **contras** que *utiliza todo el dataset* para entrenar “cada punto” y por eso requiere de uso de mucha memoria y recursos de procesamiento (CPU). Por estas razones kNN tiende a funcionar mejor en datasets pequeños y sin una cantidad enorme de features (las columnas).

Para reducir la cantidad de dimensiones (features) podemos aplicar PCA¹⁸⁷

¿Cómo funciona kNN?

1. Calcular la distancia entre el item a clasificar y el resto de items del dataset de entrenamiento.
2. Seleccionar los “k” elementos más cercanos (con menor distancia, según la función que se use)
3. Realizar una “votación de mayoría” entre los k puntos: los de una clase/etiqueta que <> decidirán su clasificación final.

Teniendo en cuenta el punto 3, veremos que para decidir la clase de un punto **es muy importante el valor de k**, pues este terminará casi por definir a qué grupo pertenecerán los puntos, sobre todo en las “fronteras” entre grupos. Por ejemplo -y a priori- yo elegiría valores impares de k para desempatar (si las features que utilizamos son pares). No será lo mismo tomar para decidir 3 valores que 13. Esto no quiere decir que necesariamente tomar más puntos implique mejorar la precisión. Lo que es seguro es que *cuantos más “puntos k”, más tardará nuestro algoritmo en procesar* y darnos respuesta ;) Las formas más populares de “medir la cercanía” entre puntos son la **distancia Euclíadiana** (la “de siempre”) o la **Cosine Similarity **(mide el ángulo de los vectores, cuanto menores, serán similares). Recordemos que este algoritmo -y prácticamente todos en ML- funcionan mejor con varias características de las que tomemos datos (las columnas de nuestro dataset). Lo que entendemos como “distancia” en la vida real, quedará abstracto a muchas dimensiones que no podemos “visualizar” fácilmente (como por ejemplo en un mapa).

Un ejemplo k-Nearest Neighbor en Python

Exploraremos el algoritmo con Scikit learn

Realizaremos un ejercicio usando Python y su librería scikit-learn que ya tiene implementado el algoritmo para simplificar las cosas. Veamos cómo se hace.

¹⁸⁷<http://www.aprendemachinelearning.com/comprende-principal-component-analysis/>

Requerimientos

Para realizar este ejercicio, crearemos una [Jupyter notebook¹⁸⁸](#) con código Python y la librería SkLearn muy utilizada en Data Science. Recomendamos utilizar la suite para python de [Anaconda¹⁸⁹](#). Puedes [leer este artículo¹⁹⁰](#) donde muestro paso a paso como [instalar el ambiente de desarrollo¹⁹¹](#). Podrás descargar los archivos de entrada csv o visualizar la notebook online (al final de este artículo los enlaces).

El Ejercicio: App Reviews

Para nuestro ejercicio tomaremos 257 registros con Opiniones de usuarios sobre una app (Reviews). Utilizaremos 2 columnas de datos como fuente de alimento del algoritmo. Recuerden que sólo tomaré 2 features para poder graficar en 2 dimensiones, PERO para un problema “en la vida real” conviene tomar más características de lo que sea que queramos resolver. Esto es únicamente con fines de enseñanza. Las columnas que utilizaremos serán: **wordcount** con la cantidad de palabras utilizadas y **sentimentValue** con un valor entre -4 y 4 que indica si el comentario fue valorado como positivo o negativo. Nuestras etiquetas, serán las estrellas que dieron los usuarios a la app, que son valores discretos del 1 al 5. Podemos pensar que si el usuario puntúa con más estrellas, tendrá un sentimiento positivo, pero no necesariamente siempre es así.

Comencemos con el código!

Primero hacemos imports de librerías que utilizaremos para manejo de datos, gráficas y nuestro algoritmo.

```

1 import pandas as pd
2 import numpy as np
3 import matplotlib.pyplot as plt
4 from matplotlib.colors import ListedColormap
5 import matplotlib.patches as mpatches
6 import seaborn as sb
7
8 %matplotlib inline
9 plt.rcParams['figure.figsize'] = (16, 9)
10 plt.style.use('ggplot')
11
12 from sklearn.model_selection import train_test_split
13 from sklearn.preprocessing import MinMaxScaler

```

¹⁸⁸<http://data-speaks.luca-d3.com/2018/03/python-para-todos-2-jupyter-notebook.html>

¹⁸⁹<https://www.anaconda.com/download/>

¹⁹⁰<http://www.aprendemachinelearning.com/installar-ambiente-de-desarrollo-python-anaconda-para-aprendizaje-automatico/>

¹⁹¹<http://www.aprendemachinelearning.com/installar-ambiente-de-desarrollo-python-anaconda-para-aprendizaje-automatico/>

```

14 from sklearn.neighbors import KNeighborsClassifier
15 from sklearn.metrics import classification_report
16 from sklearn.metrics import confusion_matrix

```

Cargamos el archivo entrada csv con pandas, usando separador de punto y coma, pues en las reviews hay textos que usan coma. Con head(10) vemos los 10 primeros registros.

```

1 dataframe = pd.read_csv(r"reviews_sentiment.csv",sep='; ')
2 dataframe.head(10)

```

	Review Title	Review Text	wordcount	titleSentiment	textSentiment	Star Rating	sentimentValue
0	Sin conexión	Hola desde hace algo más de un mes me pone sin...	23	negative	negative	1	-0.486389
1	faltan cosas	Han mejorado la apariencia pero no	20	negative	negative	1	-0.586187
2	Es muy buena lo recomiendo	Andres e puta amoooo	4	NaN	negative	1	-0.602240
3	Version antigua	Me gustana mas la version anterior esta es mas...	17	NaN	negative	1	-0.616271
4	Esta bien	Sin ser la biblia.... Esta bien	6	negative	negative	1	-0.651784
5	Buena	Nada del otro mundo pero han mejorado mucho	8	positive	negative	1	-0.720443
6	De gran ayuda	Lo malo q necesitas depero la app es muy buena	23	positive	negative	1	-0.726825
7	Muy buena	Estaba más acostumbrado al otro diseño, pero e...	16	positive	negative	1	-0.736769
8	Ta to guapa.	Va de escándalo	21	positive	negative	1	-0.765284
9	Se han corregido	Han corregido muchos fallos pero el diseño es ...	13	negative	negative	1	-0.797961

Aprovechamos a ver un resumen estadístico de los datos:

```
1 dataframe.describe()
```

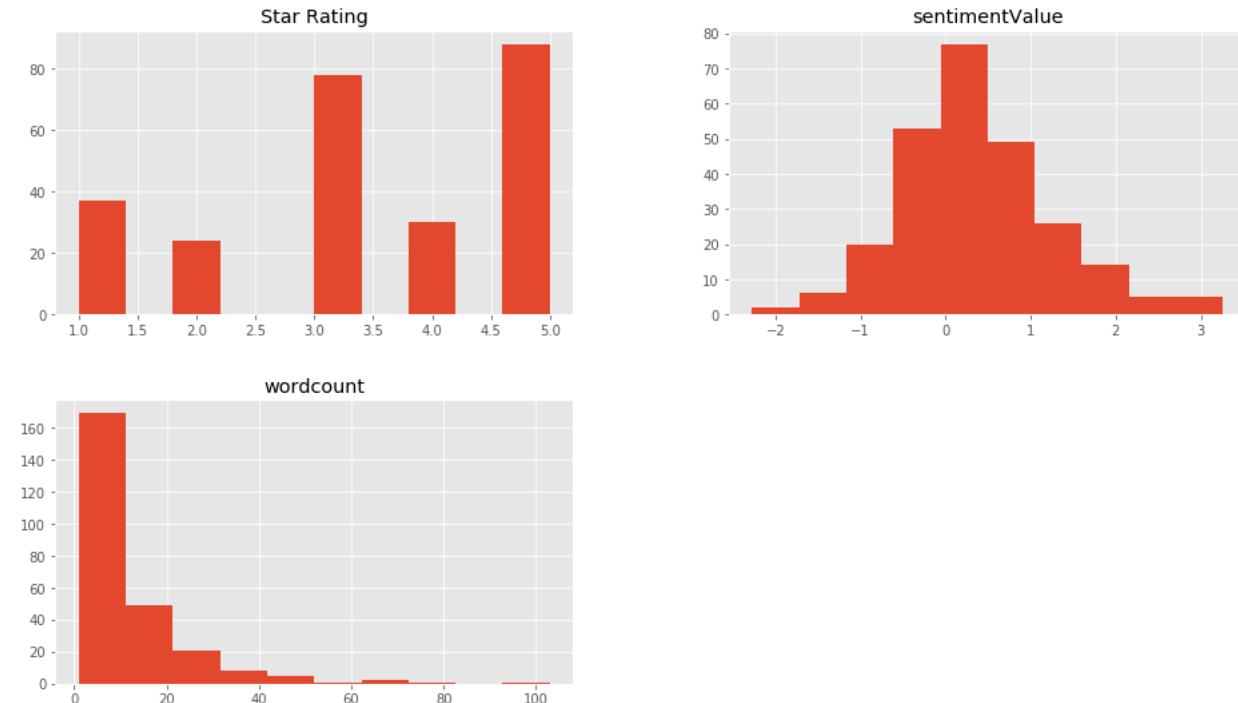
	wordcount	Star Rating	sentimentValue
count	257.000000	257.000000	257.000000
mean	11.501946	3.420233	0.383849
std	13.159812	1.409531	0.897987
min	1.000000	1.000000	-2.276469
25%	3.000000	3.000000	-0.108144
50%	7.000000	3.000000	0.264091
75%	16.000000	5.000000	0.808384
max	103.000000	5.000000	3.264579

Son 257 registros. Las estrellas lógicamente vemos que van del 1 al 5. La cantidad de palabras van de 1 sola hasta 103. y las valoraciones de sentimiento están entre -2.27 y 3.26 con una media de 0,38 y a partir del desvío estándar podemos ver que la mayoría están entre 0,38-0,89 y 0,38+0,89.

Un poco de Visualización

Veamos unas gráficas simples y qué información nos aportan:

```
1 dataframe.hist()
2 plt.show()
```



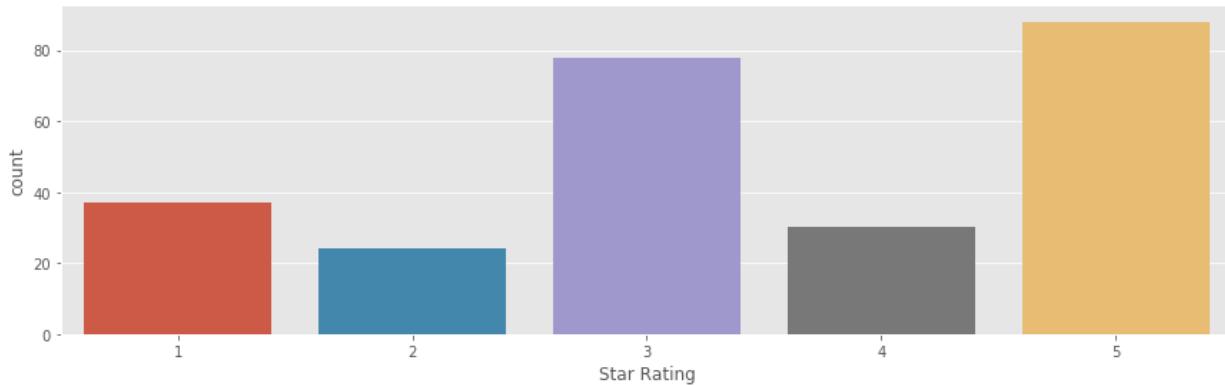
Vemos que la distribución de “estrellas” no está balanceada... esto no es bueno. Convendría tener las mismas cantidades en las salidas, para no tener resultados “tendenciosos”. Para este ejercicio lo dejaremos así, pero en la vida real, debemos equilibrarlos. La gráfica de Valores de Sentimientos parece bastante una campana movida levemente hacia la derecha del cero y la cantidad de palabras se centra sobre todo de 0 a 10. Veamos realmente cuantas Valoraciones de Estrellas tenemos:

```
1 print(dataframe.groupby('Star Rating').size())
```

```
Star Rating
1    37
2    24
3    78
4    30
5    88
dtype: int64
```

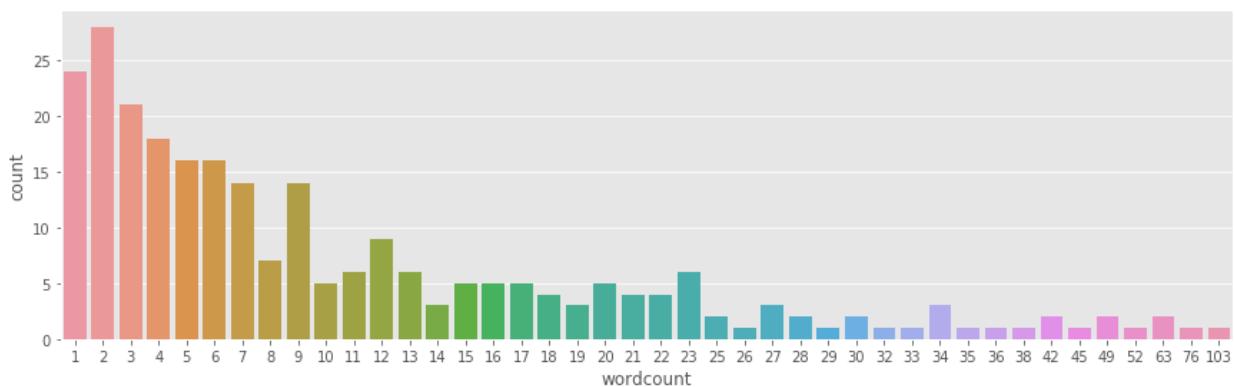
Con eso confirmamos que hay sobre todo de 3 y 5 estrellas. Y aqui una gráfica más bonita:

```
1 sb.factorplot('Star Rating', data=dataframe, kind="count", aspect=3)
```



Graficamos mejor la cantidad de palabras y confirmamos que la mayoría están entre 1 y 10 palabras.

```
1 sb.factorplot('wordcount', data=dataframe, kind="count", aspect=3)
```



Preparamos las entradas

Creamos nuestro X e y de entrada y los sets de entrenamiento y test.

```

1 X = dataframe[['wordcount', 'sentimentValue']].values
2 y = dataframe['Star Rating'].values
3
4 X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
5 scaler = MinMaxScaler()
6 X_train = scaler.fit_transform(X_train)
7 X_test = scaler.transform(X_test)
```

Usemos k-Nearest Neighbor con Scikit Learn

Definimos el valor de k en 7 (esto realmente lo sabemos más adelante, ya veréis) y creamos nuestro clasificador.

```

1 n_neighbors = 7
2
3 knn = KNeighborsClassifier(n_neighbors)
4 knn.fit(X_train, y_train)
5 print('Accuracy of K-NN classifier on training set: {:.2f}'
6      .format(knn.score(X_train, y_train)))
7 print('Accuracy of K-NN classifier on test set: {:.2f}'
8      .format(knn.score(X_test, y_test)))
```

1 Accuracy of K-NN classifier on training set: 0.90
2 Accuracy of K-NN classifier on test set: 0.86

Vemos que la precisión que nos da es de 90% en el set de entrenamiento y del 86% para el de test.

NOTA: como verán utilizamos la clase [KNeighborsClassifier¹⁹²](#) de SciKit Learn puesto que nuestras etiquetas son valores discretos (estrellas del 1 al 5). Pero deben saber que también existe la clase [KneighborsRegressor¹⁹³](#) para etiquetas con valores continuos.

Precisión del modelo

Confirmemos la precisión viendo la [Confusión Matrix y el Reporte¹⁹⁴](#) sobre el conjunto de test, que nos detalla los aciertos y fallos:

¹⁹²<http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html#sklearn.neighbors.KNeighborsClassifier>

¹⁹³<http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsRegressor.html#sklearn.neighbors.KNeighborsRegressor>

¹⁹⁴<http://www.aprendemachinelearning.com/clasificacion-con-datos-desbalanceados/>

```

1 pred = knn.predict(X_test)
2 print(confusion_matrix(y_test, pred))
3 print(classification_report(y_test, pred))

[[ 9  0  1  0  0]
 [ 0  1  0  0  0]
 [ 0  1  17  0  1]
 [ 0  0  2  8  0]
 [ 0  0  4  0  21]]
      precision    recall   f1-score   support
      1         1.00     0.90     0.95      10
      2         0.50     1.00     0.67       1
      3         0.71     0.89     0.79      19
      4         1.00     0.80     0.89      10
      5         0.95     0.84     0.89      25
avg / total     0.89     0.86     0.87      65

```

Cómo se ve la puntuación F1 es del 87%, bastante buena. NOTA: recuerden que este es sólo un ejercicio para aprender y tenemos MUY pocos registros totales y en nuestro conjunto de test. Por ejemplo de 2 estrellas sólo tiene 1 valoración y esto es evidentemente insuficiente.

Y ahora, la gráfica que queríamos ver!

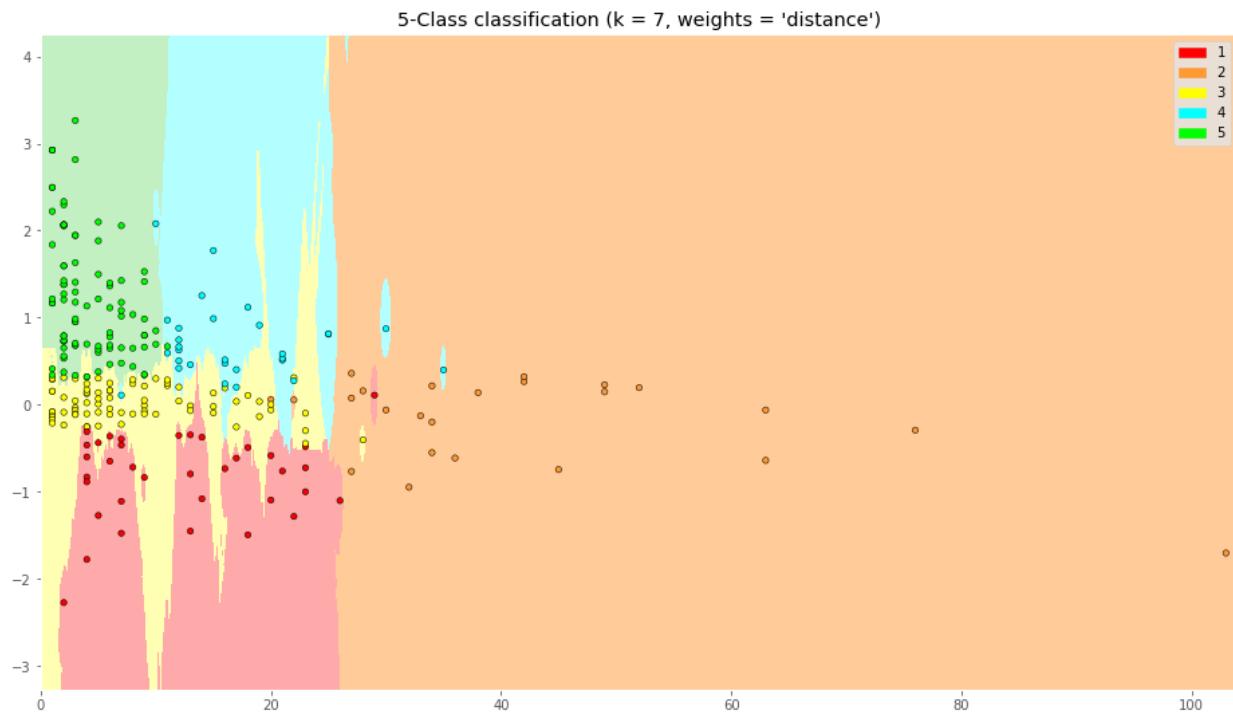
Ahora realizaremos la grafica con la clasificación obtenida, la que nos ayuda a ver fácilmente en donde caerán las predicciones. NOTA: al ser 2 features, podemos hacer la gráfica 2D y si fueran 3 podría ser en 3D. Pero para usos reales, podríamos tener más de 3 dimensiones y no importaría poder visualizarlo sino el resultado del algoritmo.

```

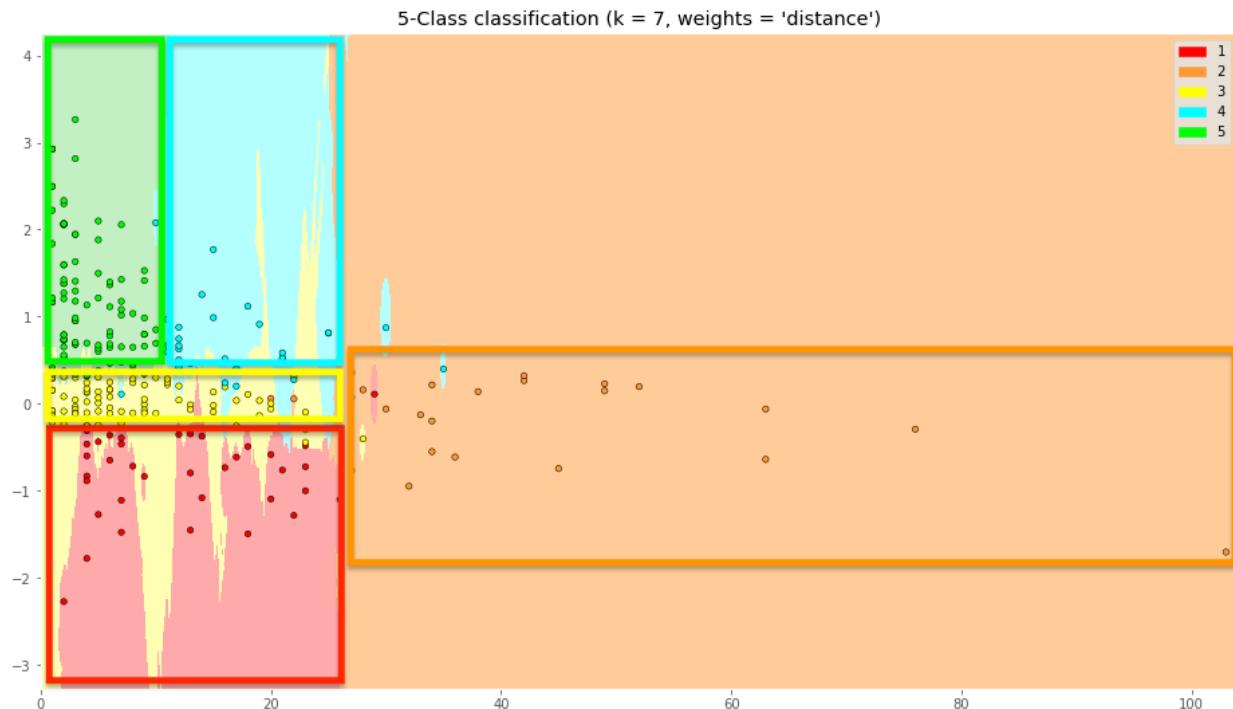
1 h = .02 # step size in the mesh
2
3 # Create color maps
4 cmap_light = ListedColormap(['#FFAAAA', '#ffcc99', '#fffffb3', '#b3ffff', '#c2f0c2'])
5 cmap_bold = ListedColormap(['#FF0000', '#ff9933', '#FFFF00', '#00ffff', '#00FF00'])
6
7 # we create an instance of Neighbours Classifier and fit the data.
8 clf = KNeighborsClassifier(n_neighbors, weights='distance')
9 clf.fit(X, y)
10
11 # Plot the decision boundary. For that, we will assign a color to each
12 # point in the mesh [x_min, x_max]x[y_min, y_max].

```

```
13 x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
14 y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
15 xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
16                      np.arange(y_min, y_max, h))
17 Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
18
19 # Put the result into a color plot
20 Z = Z.reshape(xx.shape)
21 plt.figure()
22 plt.pcolormesh(xx, yy, Z, cmap=cmap_light)
23
24 # Plot also the training points
25 plt.scatter(X[:, 0], X[:, 1], c=y, cmap=cmap_bold,
26              edgecolor='k', s=20)
27 plt.xlim(xx.min(), xx.max())
28 plt.ylim(yy.min(), yy.max())
29
30 patch0 = mpatches.Patch(color='#FF0000', label='1')
31 patch1 = mpatches.Patch(color='#ff9933', label='2')
32 patch2 = mpatches.Patch(color='#FFFF00', label='3')
33 patch3 = mpatches.Patch(color='#00ffff', label='4')
34 patch4 = mpatches.Patch(color='#00FF00', label='5')
35 plt.legend(handles=[patch0, patch1, patch2, patch3, patch4])
36
37 plt.title("5-Class classification (k = %i, weights = '%s')"
38            % (n_neighbors, weights))
39
40 plt.show()
```



Vemos las 5 zonas en las que se relacionan cantidad de palabras con el valor de sentimiento de la Review que deja el usuario. Se distinguen 5 regiones que podríamos dividir así:



Es decir que “a ojo” una review de 20 palabras y Sentimiento 1, nos daría una valoración de 4 (zona

celeste). Con estas zonas podemos intuir ciertas características de los usuarios que usan y valoran la app:

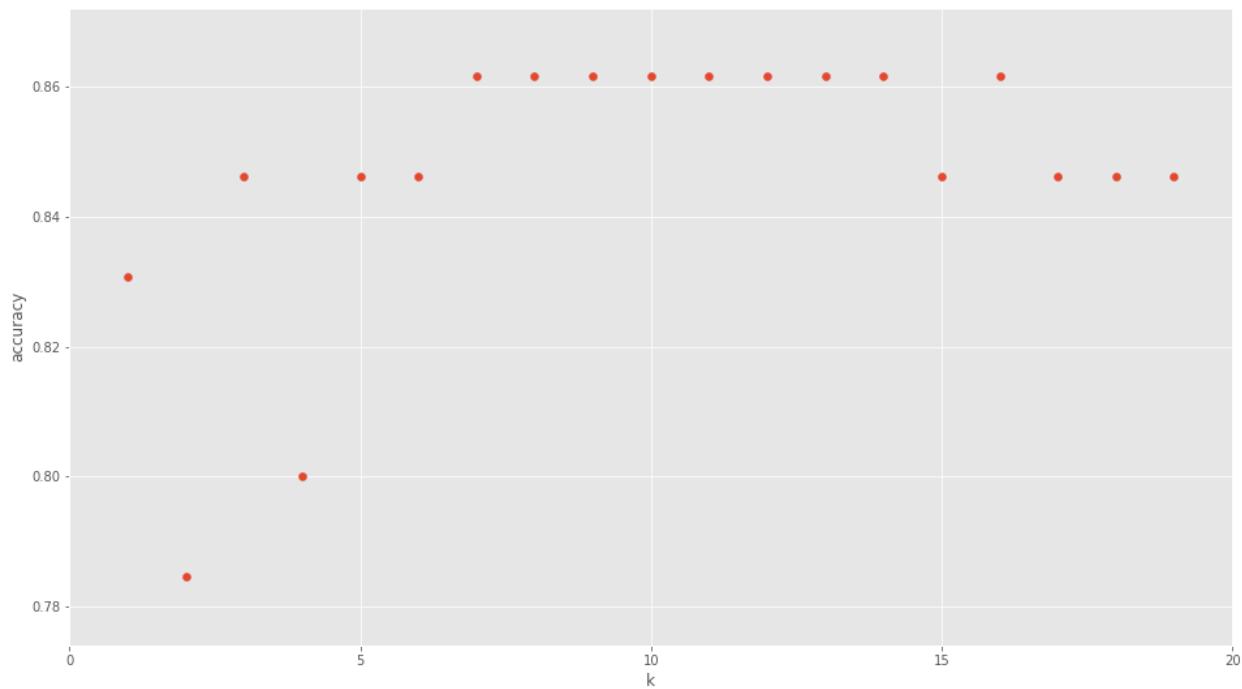
- Los usuarios que ponen 1 estrella tienen sentimiento negativo y hasta 25 palabras.
- Los usuarios que ponen 2 estrellas dan muchas explicaciones (hasta 100 palabras) y su sentimiento puede variar entre negativo y algo positivo.
- Los usuarios que ponen 3 estrellas son bastante neutrales en sentimientos, puesto que están en torno al cero y hasta unas 25 palabras.
- Los usuarios que dan 5 estrellas son bastante positivos (de 0,5 en adelante, aproximadamente) y ponen pocas palabras (hasta 10).

Elegir el mejor valor de k

(sobre todo importante para desempatar o elegir los puntos frontera!)

Antes vimos que asignamos el valor `n_neighbors=7` como valor de “k” y obtuvimos buenos resultados. ¿Pero de donde salió ese valor? Pues realmente tuve que ejecutar este código que viene a continuación, donde vemos distintos valores k y la precisión obtenida.

```
1 k_range = range(1, 20)
2 scores = []
3 for k in k_range:
4     knn = KNeighborsClassifier(n_neighbors = k)
5     knn.fit(X_train, y_train)
6     scores.append(knn.score(X_test, y_test))
7 plt.figure()
8 plt.xlabel('k')
9 plt.ylabel('accuracy')
10 plt.scatter(k_range, scores)
11 plt.xticks([0,5,10,15,20])
```



En la gráfica vemos que con valores $k=7$ a $k=14$ es donde mayor precisión se logra.

Clasificar ó Predecir nuevas muestras

Ya tenemos nuestro modelo y nuestro valor de k . Ahora, lo lógico será usarlo! Pues supongamos que nos llegan nuevas reviews! veamos como predecir sus estrellas de 2 maneras. La primera:

```
1 print(clf.predict([[5, 1.0]]))
```

```
1 [5]
```

Este resultado nos indica que para 5 palabras y sentimiento 1, nos valorarán la app con 5 estrellas. Pero también podríamos obtener las probabilidades que de nos den 1, 2, 3, 4 o 5 estrellas con `predict_proba()`:

```
1 print(clf.predict_proba([[20, 0.0]]))
```

```
1 [[0.00381998 0.02520212 0.97097789 0. 0. ]]
```

Aquí vemos que para las coordenadas 20, 0.0 hay 97% probabilidades que nos den 3 estrellas. Puedes comprobar en el gráfico anterior, que encajan en las zonas que delimitamos anteriormente.

Resumen

En este ejercicio creamos un modelo con Python para procesar y clasificar puntos de un conjunto de entrada con el algoritmo k-Nearest Neighbor. Como su nombre en inglés lo dice, se evalúan los “k vecinos más cercanos” para poder clasificar nuevos puntos. Al ser un algoritmo supervisado debemos contar con suficientes muestras etiquetadas para poder entrenar el modelo con buenos resultados. Este algoritmo es bastante simple y -como vimos antes- necesitamos muchos recursos de memoria y cpu para mantener el dataset “vivo” y evaluar nuevos puntos. Esto no lo hace recomendable para conjuntos de datos muy grandes. En el ejemplo, sólo utilizamos 2 dimensiones de entrada para poder graficar y ver en dos dimensiones cómo se obtienen y delimitan los grupos. Finalmente pudimos hacer nuevas predicciones y a raíz de los resultados, comprender mejor la problemática planteada.

Recursos y enlaces

- Descarga la [Jupyter Notebook¹⁹⁵](#) y el [archivo de entrada csv¹⁹⁶](#)
- ó puedes [visualizar online¹⁹⁷](#)
- o ver y descargar desde mi cuenta [github¹⁹⁸](#)

Más artículos de Interés sobre k-Nearest Neighbor (en Inglés)

- [Implementing kNN in scikit learn¹⁹⁹](#)
- [Introduction to kNN²⁰⁰](#)
- [Complete guide to kNN²⁰¹](#)
- [Solving a simple classification problem with Python²⁰²](#)

¹⁹⁵http://www.aprendemachinelearning.com/wp-content/uploads/2018/07/Ejercicio_k-NearestNeighbor.ipynb

¹⁹⁶http://www.aprendemachinelearning.com/wp-content/uploads/2018/07/reviews_sentiment.csv

¹⁹⁷http://nbviewer.jupyter.org/github/jbagnato/machine-learning/blob/master/Ejercicio_k-NearestNeighbor.ipynb

¹⁹⁸<https://github.com/jbagnato/machine-learning>

¹⁹⁹<https://towardsdatascience.com/implementing-k-nearest-neighbors-with-scikit-learn-9e4858e231ea>

²⁰⁰<https://towardsdatascience.com/introduction-to-k-nearest-neighbors-3b534bb11d26>

²⁰¹<https://kevinzakka.github.io/2016/07/13/k-nearest-neighbor/>

²⁰²<https://towardsdatascience.com/solving-a-simple-classification-problem-with-python-fruits-lovers-edition-d20ab6b071d2>

Clasificación de Imágenes en Python

Crearemos una [Convolutional Neural Network²⁰³](#) con Keras y Tensorflow en Python para reconocimiento de Imágenes.

En este capítulo iremos directo al grano: veremos el código que crea la red neuronal para visión artificial. En el [próximo capítulo explicaré bien los conceptos utilizados²⁰⁴](#), pero esta vez haremos un aprendizaje [Top-down²⁰⁵](#) ;)

Ejercicio: Clasificar imágenes de deportes

Para el ejercicio se me ocurrió crear “mi propio dataset [MNIST²⁰⁶](#)” con imágenes de deportes. Para ello, **seleccioné los 10 deportes más populares del mundo** -según la sabiduría de internet- : Fútbol, Basket, Golf, Fútbol Americano, Tenis, Fórmula 1, Ciclismo, Boxeo, Beisball y Natación (enumerados sin orden particular entre ellos). Obtuve entre 5000 y 9000 imágenes **de cada deporte**, a partir de videos de Youtube (usando [FFMpeg²⁰⁷](#)). Las imágenes están en tamaño <>diminuto>> de 21x28 pixeles **en color** y son un total de **77.000**. Si bien el tamaño en pixeles es pequeño ES SUFICIENTE para que nuestra red neuronal pueda distinguirlas!!! (¿increíble, no?).

El objetivo es que nuestra máquina “[*red neuronal convolucional²⁰⁸*](#)” aprenda a clasificar -por sí sola-, dada una nueva imagen, de qué deporte se trata.



²⁰³<http://www.aprendemachinelearning.com/como-funcionan-las-convolutional-neural-networks-vision-por-ordenador/>

²⁰⁴<http://www.aprendemachinelearning.com/como-funcionan-las-convolutional-neural-networks-vision-por-ordenador/>

²⁰⁵<https://classroom.synonym.com/difference-between-topdown-teaching-bottomup-teaching-12059397.html>

²⁰⁶<https://deeplearninglaptop.com/blog/2017/11/24/mnist/>

²⁰⁷<https://ffmpeg.org>

²⁰⁸<http://www.aprendemachinelearning.com/como-funcionan-las-convolutional-neural-networks-vision-por-ordenador/>

Ejemplo de imágenes de los deportes más populares del mundo



Dividiremos el set de datos en 80-20 para entrenamiento y para test. A su vez, **el conjunto de entrenamiento también lo subdividiremos en otro 80-20** para Entrenamiento y Validación en cada iteración (**EPOCH²⁰⁹**) de aprendizaje.

²⁰⁹<http://sgsai.blogspot.com/2015/02/epoch-batch-and-iteration.html>



Una muestra de las imágenes del Dataset que he titulado sportsMNIST. Contiene más de 70.000 imágenes de los 10 deportes más populares del mundo.

Requerimientos Técnicos

Necesitaremos tener Python 3.6 y como lo haremos en una [Notebook Jupyter](#)²¹⁰ recomiendo tener instalada [una suite como Anaconda](#)²¹¹, que nos facilitará las tareas. Además instalar Keras y Tensorflow como backend.

Necesitarás descargar el archivo zip con las imágenes (están comprimidas) y decomprimirlas **en el mismo directorio** en donde ejecutarás la Notebook con el código. Al descomprimir, se crearán 10 subdirectorios con las imágenes: uno por cada deporte.

- Descarga las imágenes MNIST-Deportes 63MB²¹² (no olvides descomprimir el .zip)
- Descarga la Jupyter Notebook con el código Python²¹³

Vamos al código Python

Por más que no entiendas del todo el código sigue adelante, intentaré explicar brevemente qué hacemos **paso a paso** y en el [próximo capítulo se explicará cada parte de las CNN](#)²¹⁴ (Convolutional Neural Networks). También encontrarás al final varios enlaces con información adicional que te ayudará. Esto es lo que haremos hoy:

1. Importar librerías
2. Cargar las 70.000 imágenes (en memoria!)
3. Crear dinámicamente las etiquetas de resultado.
4. Dividir en sets de Entrenamiento, Validación y Test, preprocesamiento de datos
5. Crear el modelo de la CNN
6. Ejecutar nuestra máquina de aprendizaje (Entrenar la red)
7. Revisar los resultados obtenidos

Empecemos a programar!:

1- Importar librerías

Cargaremos las libs que utilizaremos para el ejercicio.

²¹⁰<http://data-speaks.luca-d3.com/2018/03/python-para-todos-2-jupyter-notebook.html>

²¹¹<https://www.anaconda.com/download/>

²¹²<https://github.com/jbagnato/machine-learning/raw/master/sportimages.zip>

²¹³https://github.com/jbagnato/machine-learning/blob/master/Ejercicio_CNN.ipynb

²¹⁴<http://www.aprendemachinelearning.com/como-funcionan-las-convolutional-neural-networks-vision-por-ordenador/>

```
1 import numpy as np
2 import os
3 import re
4 import matplotlib.pyplot as plt
5 %matplotlib inline
6 from sklearn.model_selection import train_test_split
7 from sklearn.metrics import classification_report
8 import keras
9 from keras.utils import to_categorical
10 from keras.models import Sequential, Input, Model
11 from keras.layers import Dense, Dropout, Flatten
12 from keras.layers import Conv2D, MaxPooling2D
13 from keras.layers.normalization import BatchNormalization
14 from keras.layers.advanced_activations import LeakyReLU
```

2-Cargar las imágenes

Recuerda tener DESCOMPRIMIDAS las imágenes y ejecutar el código en el MISMO directorio donde descomprimiste el directorio llamado “*sportimages*” (contiene 10 subdirectorios: uno por cada deporte). Este proceso plt.imread(filepath) cargará a memoria en un array las 77mil imágenes, por lo que **puede tomar varios minutos** y consumirá algo de memoria RAM de tu ordenador.

```
1 dirname = os.path.join(os.getcwd(), 'sportimages')
2 imgpath = dirname + os.sep
3
4 images = []
5 directories = []
6 dircount = []
7 prevRoot=''
8 cant=0
9
10 print("leyendo imagenes de ",imgpath)
11
12 for root, dirnames, filenames in os.walk(imgpath):
13     for filename in filenames:
14         if re.search("\.(jpg|jpeg|png|bmp|tiff)$", filename):
15             cant=cant+1
16             filepath = os.path.join(root, filename)
17             image = plt.imread(filepath)
18             images.append(image)
19             b = "Leyendo..." + str(cant)
20             print (b, end="\r")
```

```

21         if prevRoot !=root:
22             print(root, cant)
23             prevRoot=root
24             directories.append(root)
25             dircount.append(cant)
26             cant=0
27             dircount.append(cant)
28
29     dircount = dircount[1:]
30     dircount[0]=dircount[0]+1
31     print('Directorios leidos:',len(directories))
32     print("Imagenes en cada directorio", dircount)
33     print('suma Total de imagenes en subdirs:',sum(dircount))

1 leyendo imagenes de /Users/xxx/proyecto_python/sportimages/ Directorios leidos: 10
2 Imagenes en cada directorio [9769, 8823, 8937, 5172, 7533, 7752, 7617, 9348, 5053, 7\
3 124]
4 suma Total de imagenes en subdirs: 77128

```

3- Crear etiquetas y clases

Crearemos las etiquetas en labels , es decir, le daremos valores de 0 al 9 a cada deporte. Esto lo hacemos para poder usar el [algoritmo supervisado](#)²¹⁵ e indicar que cuando cargamos una imagen de futbol en la red, ya sabemos que corresponde con la “etiqueta 6”. Y con esa información, entrada y salida esperada, la red al entrenar, ajustará los pesos de las neuronas. Luego convertimos las etiquetas y las imágenes en numpy array con np.array()

```

1 labels=[]
2 indice=0
3 for cantidad in dircount:
4     for i in range(cantidad):
5         labels.append(indice)
6         indice=indice+1
7     print("Cantidad etiquetas creadas: ",len(labels))
8
9 deportes=[]
10 indice=0
11 for directorio in directories:
12     name = directorio.split(os.sep)
13     print(indice , name[len(name)-1])

```

²¹⁵<http://www.aprendemachinelearning.com/aplicaciones-del-machine-learning/>

```

14     deportes.append(name[len(name)-1])
15     indice=indice+1
16
17     y = np.array(labels)
18     X = np.array(images, dtype=np.uint8) #convierto de lista a numpy
19
20     # Find the unique numbers from the train labels
21     classes = np.unique(y)
22     nClasses = len(classes)
23     print('Total number of outputs : ', nClasses)
24     print('Output classes : ', classes)

1 Cantidad etiquetas creadas: 77128
2 0 golf 1 basket 2 tenis 3 natacion 4 ciclismo 5 beisball 6 futbol 7 americano 8 f1 9\
3 boxeo
4 Total number of outputs : 10
5 Output classes : [0 1 2 3 4 5 6 7 8 9]

```

4-Creamos sets de Entrenamiento y Test, Validación y Preprocesar

Nótese la “forma” (shape) de los arrays: veremos que son de** 21×28 y por 3** pues el 3 se refiere a los 3 canales de colores que tiene cada imagen: RGB (red, green, blue) que tiene valores de 0 a 255. **Preprocesamos el valor de los pixeles** y lo normalizamos para que tengan un valor entre 0 y 1, por eso dividimos en 255. Ademas haremos el “[One-Hot encoding²¹⁶](#)” con `to_categorical()` que se refiere a convertir las etiquetas (nuestras clases) por ejemplo de fútbol un 6 a una salida de tipo (0 0 0 0 0 1 0 0 0) Esto es porque así funcionan mejor las redes neuronales para clasificar y se corresponde con una capa de salida de la red neuronal de 10 neuronas. NOTA: por si no lo entendiste, se pone un 1 en la “sexta posición” del array y el resto en ceros, PERO no te olvides que empieza a contar incluyendo el cero!!! por eso la “etiqueta 6” queda **realmente en la séptima posición**. Por último en este bloque, subdividimos los datos en 80-20 para test y entrenamiento con `train_test_split()` y nuevamente en 80-20 el de training para obtener un subconjunto de validación²¹⁷.

²¹⁶<https://machinelearningmastery.com/why-one-hot-encode-data-in-machine-learning/>

²¹⁷<https://medium.com/simple-ai/how-good-is-your-model-intro-to-machine-learning-4-ec7289bb7dca>

```

1  #Mezclar todo y crear los grupos de entrenamiento y testing
2  train_X,test_X,train_Y,test_Y = train_test_split(X,y,test_size=0.2)
3  print('Training data shape : ', train_X.shape, train_Y.shape)
4  print('Testing data shape : ', test_X.shape, test_Y.shape)
5
6  train_X = train_X.astype('float32')
7  test_X = test_X.astype('float32')
8  train_X = train_X / 255.
9  test_X = test_X / 255.
10
11 # Change the labels from categorical to one-hot encoding
12 train_Y_one_hot = to_categorical(train_Y)
13 test_Y_one_hot = to_categorical(test_Y)
14
15 # Display the change for category label using one-hot encoding
16 print('Original label:', train_Y[0])
17 print('After conversion to one-hot:', train_Y_one_hot[0])
18
19 train_X,valid_X,train_label,valid_label = train_test_split(train_X, train_Y_one_\
20 hot, test_size=0.2, random_state=13)
21
22 print(train_X.shape,valid_X.shape,train_label.shape,valid_label.shape)

```

```

1 Training data shape : (61702, 21, 28, 3) (61702,)
2 Testing data shape : (15426, 21, 28, 3) (15426,)
3 Original label: 0
4 After conversion to one-hot: [1. 0. 0. 0. 0. 0. 0. 0. 0.]
5 (49361, 21, 28, 3) (12341, 21, 28, 3) (49361, 10) (12341, 10)

```

5 - Creamos la red (Aquí la Magia)

Ahora sí que nos apoyamos en Keras para crear la Convolutional Neural Network. En un futuro artículo²¹⁸ explicaré mejor lo que se está haciendo. *Por ahora “confíen” en mí:*

- Declaramos 3 “constantes”:
 - El valor inicial del learning rate INIT_LR
 - cantidad de epochs y
 - tamaño batch de imágenes a procesar batch_size (cargan en memoria).
- Crearemos una primer capa de neuronas “Convolucional de 2 Dimensiones” Conv2D() , donde entrarán nuestras imágenes de 21x28x3.

²¹⁸<http://www.aprendemachinelearning.com/como-funcionan-las-convolutional-neural-networks-vision-por-ordenador/>

- Aplicaremos 32 filtros (kernel) de tamaño 3x3 (no te preocupes si aún no entiendes esto!) que detectan ciertas características de la imagen (ejemplo: líneas verticales).
- Utilizaremos la función `LeakyReLU`²¹⁹ como activación de las neuronas.
- Haremos un **MaxPooling (de 2x2)** que reduce la imagen que entra de 21x28 a la mitad,(11x14) manteniendo las características “únicas” que detectó cada kernel.
- Para evitar el **overfitting**, añadimos una técnica llamada `Dropout`²²⁰
- “Aplanamos” `Flatten()` los 32 filtros y creamos una capa de 32 neuronas “tradicionales” `Dense()`
- Y finalizamos la capa de salida con 10 neuronas con activación Softmax, para que se corresponda con el “hot encoding” que hicimos antes.
- Luego compilamos nuestra red `sport_model.compile()` y le asignamos un optimizador (en este caso de llama Adagrad).

```
1     INIT_LR = 1e-3
2     epochs = 6
3     batch_size = 64
4
5     sport_model = Sequential()
6     sport_model.add(Conv2D(32, kernel_size=(3, 3), activation='linear', padding='same' \
7 ,input_shape=(21,28,3)))
8     sport_model.add(LeakyReLU(alpha=0.1))
9     sport_model.add(MaxPooling2D((2, 2),padding='same'))
10    sport_model.add(Dropout(0.5))
11
12    sport_model.add(Flatten())
13    sport_model.add(Dense(32, activation='linear'))
14    sport_model.add(LeakyReLU(alpha=0.1))
15    sport_model.add(Dropout(0.5))
16    sport_model.add(Dense(nClasses, activation='softmax'))
17
18    sport_model.summary()
19
20    sport_model.compile(loss=keras.losses.categorical_crossentropy, optimizer=keras.\ \
21 optimizers.Adagrad(lr=INIT_LR, decay=INIT_LR / 100),metrics=[ 'accuracy' ])
```

²¹⁹[https://en.wikipedia.org/wiki/Rectifier_\(neural_networks\)](https://en.wikipedia.org/wiki/Rectifier_(neural_networks))

²²⁰https://en.wikipedia.org/wiki/Convolutional_neural_network#Dropout

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 21, 28, 32)	896
leaky_re_lu_1 (LeakyReLU)	(None, 21, 28, 32)	0
max_pooling2d_1 (MaxPooling2D)	(None, 11, 14, 32)	0
dropout_1 (Dropout)	(None, 11, 14, 32)	0
flatten_1 (Flatten)	(None, 4928)	0
dense_1 (Dense)	(None, 32)	157728
leaky_re_lu_2 (LeakyReLU)	(None, 32)	0
dropout_2 (Dropout)	(None, 32)	0
dense_2 (Dense)	(None, 10)	330

Total params: 158,954
Trainable params: 158,954
Non-trainable params: 0

6-Entrenamos la CNN

Llegó el momento!!! con esta linea `sport_model.fit()` iniciaremos el entrenamiento y validación de nuestra máquina. Pensemos que introduciremos miles de imágenes, pixeles, arrays, colores... filtros y la red se irá regulando sola, “aprendiendo” los mejores pesos para las más de 150.000 interconexiones para distinguir los 10 deportes. Esto tomará tiempo en un ordenador como mi Macbook Pro (del 2016) unos 4 minutos... puede parecer mucho o muy poco... según se lo mire.

NOTA: podemos ejecutar este mismo código pero utilizando GPU (en tu ordenador o en la [nube²²¹](#)) y los mismos cálculos tomaría apenas 40 segundos.

Por último guardamos la red YA ENTRENADA `sport_model.save()` en un formato de archivo `h5py` que nos permitirá poder utilizarla en el futuro **SIN necesidad de volver a entrenar **(y ahorrarnos los 4 minutos de impaciencia!).

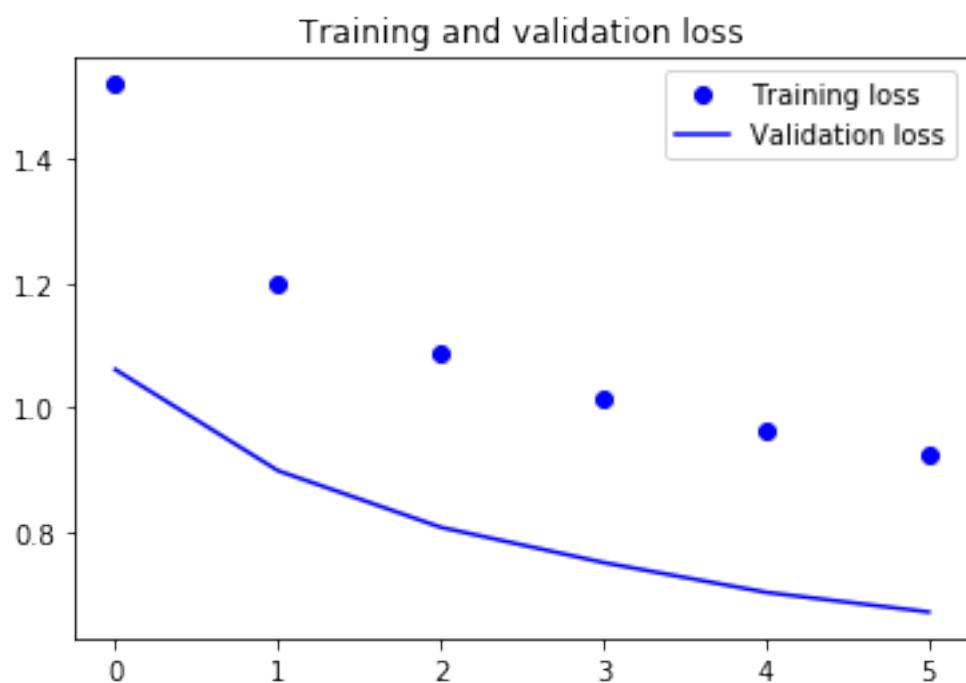
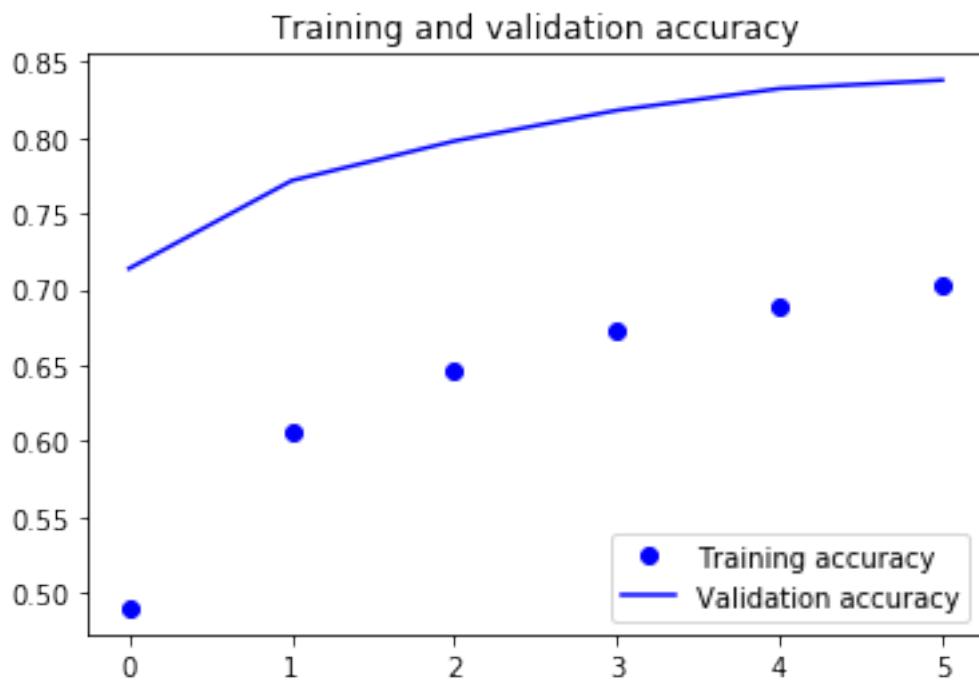
²²¹<https://colab.research.google.com/>

```
1     sport_train_dropout = sport_model.fit(train_X, train_label, batch_size=batch_size\
2 , epochs=epochs, verbose=1, validation_data=(valid_X, valid_label))
3
4     # guardamos la red, para reutilizarla en el futuro, sin tener que volver a entrenar
5
6     sport_model.save("sports_mnist.h5py")
```



```
1 Train on 49361 samples, validate on 12341 samples
2 Epoch 1/6 49361/49361 [=====] - 40s 814us/step - loss: 1.51\98 - acc: 0.4897 - val_loss: 1.0611 - val_acc: 0.7136
3 Epoch 2/6 49361/49361 [=====] - 38s 775us/step - loss: 1.20\02 - acc: 0.6063 - val_loss: 0.8987 - val_acc: 0.7717
4 Epoch 3/6 49361/49361 [=====] - 43s 864us/step - loss: 1.08\86 - acc: 0.6469 - val_loss: 0.8078 - val_acc: 0.7977
5 Epoch 4/6 49361/49361 [=====] - 41s 832us/step - loss: 1.01\66 - acc: 0.6720 - val_loss: 0.7512 - val_acc: 0.8180
6 Epoch 5/6 49361/49361 [=====] - 36s 725us/step - loss: 0.96\47 - acc: 0.6894 - val_loss: 0.7033 - val_acc: 0.8323
7 Epoch 6/6 49361/49361 [=====] - 40s 802us/step - loss: 0.92\58 - acc: 0.7032 - val_loss: 0.6717 - val_acc: 0.8379
```

Vemos que tras 6 iteraciones completas al set de entrenamiento, logramos un valor de precisión del 70% y en el set de validación alcanza un 83%. ¿Será esto suficiente para distinguir las imágenes deportivas?



7-Resultados de la clasificación

Ya con nuestra red entrenada, es la hora de la verdad: ponerla a prueba con el set de imágenes para Test que separamos al principio y que son muestras que nunca fueron “vistas” por la máquina.

```
1 test_eval = sport_model.evaluate(test_X, test_Y_one_hot, verbose=1)
2
3 print('Test loss:', test_eval[0])
4 print('Test accuracy:', test_eval[1])
```



```
1 15426/15426 [=====] - 5s 310us/step
2 Test loss: 0.6687967825782881
3 Test accuracy: 0.8409179307662388
```

En el conjunto de Testing vemos que alcanza una precisión del 84% reconociendo las imágenes de deportes. Ahora podríamos hacer un análisis más profundo, para mejorar la red, revisando los fallos que tuvimos... pero lo dejaremos para otra ocasión (**BONUS:** en la Jupyter Notebook verás más información con esto!) **Spoiler Alert:** La clase que peor detecta, son las de Fórmula 1.



Puedes probar con esta imagen de Basketball y de Fútbol a clasificarlas. En mi caso, fueron clasificadas con éxito.

²²²http://www.aprendemachinelearning.com/wp-content/uploads/2018/11/test_basket.png



En mis pruebas, a veces confundía esta imagen de Fútbol con Golf... ¿Será por el verde del campo?

Resumen

Creamos una red neuronal “novedosa”: una red convolucional, que aplica filtros a las imágenes y es capaz de distinguir distintos deportes con un tamaño de entrada de 21x28 pixels a color en tan sólo 4 minutos de entrenamiento. Esta vez fuimos a la inversa que en otras ocasiones y **antes de conocer la teoría** de las redes específicas para reconocimiento de imágenes (las CNN) les **he propuesto que hagamos un ejercicio práctico**. Aunque pueda parecer contra-intuitivo, muchas veces este método de aprendizaje (en humanos!) funciona mejor, pues **vuelve algo más dinámica la teoría**. Espero que les hayan quedado algunos de los conceptos y **los terminaremos de asentar en un próximo capítulo**²²⁴

Los recursos

- Descarga el conjunto de 70.000 imágenes para entrenar la red en archivo comprimido²²⁵ (ocupa 63MB)
- Descarga el código Python Completo, en Jupyter Notebook²²⁶
- Ver mi cuenta Github, con todos los ejercicios de Machine Learning²²⁷

Más enlaces con información sobre las Convolutional Neural Networks:

- De la universidad de Stanford, una referencia indudable: [CS231N CNN for Visual Recognition](#)²²⁸
- [Introducing Convolutional Neural Networks](#)²²⁹

²²³http://www.aprendemachinelearning.com/wp-content/uploads/2018/11/test_futbol.png

²²⁴<http://www.aprendemachinelearning.com/como-funcionan-las-convolutional-neural-networks-vision-por-ordenador/>

²²⁵<https://github.com/jbagnato/machine-learning/raw/master/sportimages.zip>

²²⁶https://github.com/jbagnato/machine-learning/blob/master/Ejercicio_CNN.ipynb

²²⁷<https://github.com/jbagnato/machine-learning>

²²⁸<http://cs231n.github.io/convolutional-networks/>

²²⁹http://neuralnetworksanddeeplearning.com/chap6.html#introducing_convolutional_networks

- Intuitively Understanding Convolutional Networks²³⁰
- Convolutional Neural Networks in Python with Keras²³¹

²³⁰<https://towardsdatascience.com/intuitively-understanding-convolutions-for-deep-learning-1f6f42faee1>

²³¹<https://www.datacamp.com/community/tutorials/convolutional-neural-networks-python#cnn>

¿Cómo funcionan las Convolutional Neural Networks?

En este capítulo intentaré explicar la teoría relativa a las Redes Neuronales Convolucionales ([en inglés CNN²³²](#)) que son el algoritmo utilizado en [Aprendizaje Automático²³³](#) para dar la capacidad de “ver” al ordenador. Gracias a esto, desde [apenas 1998²³⁴](#), podemos clasificar imágenes, detectar diversos tipos de tumores automáticamente, enseñar a conducir a los coches autónomos y un sinfín de [otras aplicaciones²³⁵](#).

El tema es bastante complejo/complicado e intentaré explicarlo lo más claro posible. En este artículo doy por sentado que tienes conocimientos básicos de cómo funciona una red neuronal artificial multicapa feedforward (fully connected).

La CNN es un tipo de Red Neuronal Artificial con aprendizaje supervisado que procesa sus capas imitando al cortex visual del ojo humano para identificar distintas características en las entradas que en definitiva hacen que pueda identificar objetos y “ver”. Para ello, la CNN contiene varias capas ocultas especializadas y con una jerarquía: esto quiere decir que **las primeras capas pueden detectar líneas, curvas y se van especializando** hasta llegar a capas más profundas que reconocen formas complejas como un rostro o la silueta de un animal.

Muchas imágenes

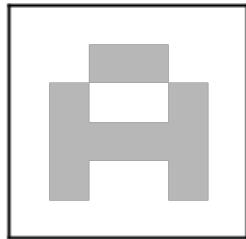
Recodemos que la red neuronal deberá aprender por sí sola a reconocer una diversidad de objetos dentro de imágenes y para ello necesitaremos una gran cantidad de imágenes -lease más de 10.000 imágenes de gatos, otras 10.000 de perros,...- para que la red pueda captar sus características únicas -de cada objeto- y a su vez, poder generalizarlo -esto es que pueda reconocer como gato tanto a un felino negro, uno blanco, un gato de frente, un gato de perfil, gato saltando, etc.-

²³²https://en.wikipedia.org/wiki/Convolutional_neural_network

²³³<http://www.aprendemachinelearning.com/que-es-machine-learning/>

²³⁴<http://www.aprendemachinelearning.com/breve-historia-de-las-redes-neuronales-artificiales/>

²³⁵<http://www.aprendemachinelearning.com/aplicaciones-del-machine-learning/>



Una imagen...

	0.6	0.6		
0.6			0.6	
0.6	0.6	0.6	0.6	
0.6			0.6	

...es una matriz de pixeles.

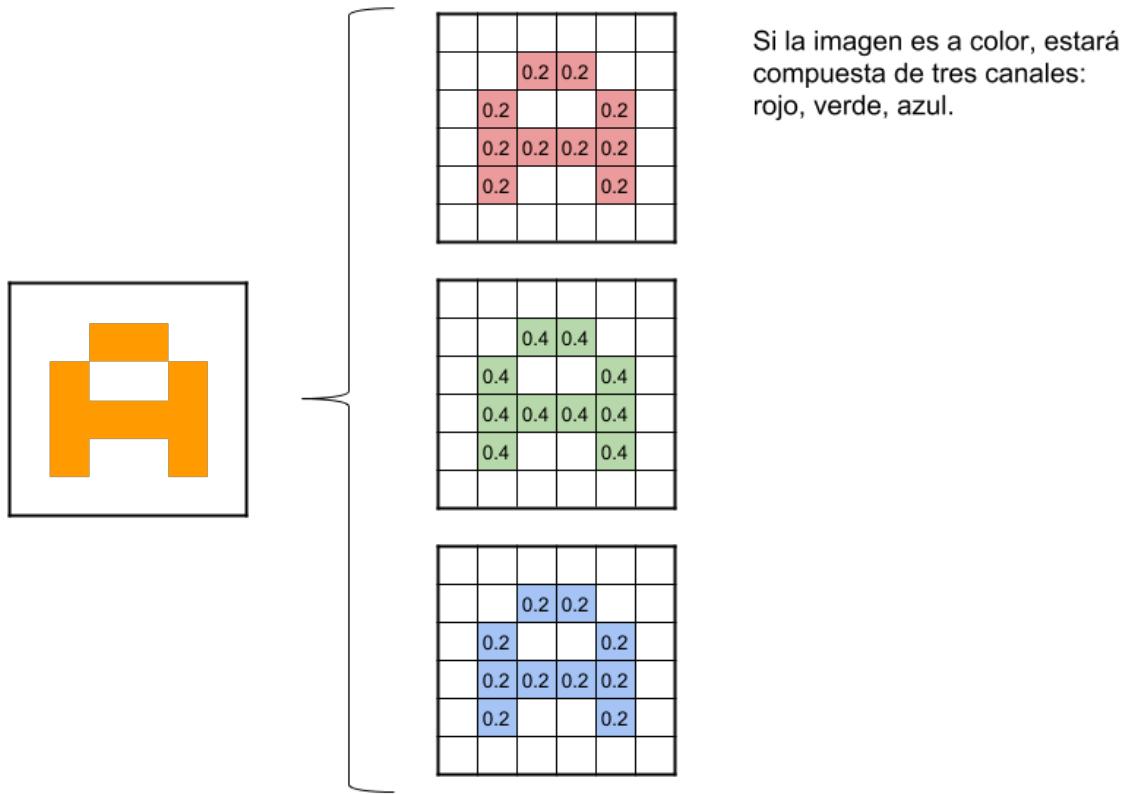
El valor de los pixeles va de 0 a 255 pero se normaliza para la red neuronal de 0 a 1

Pixeles y neuronas

Para comenzar, la red toma como entrada los pixeles de una imagen. Si tenemos una imagen con apenas 28x28 pixeles de alto y ancho, eso equivale a 784 neuronas. Y eso es si sólo tenemos 1 color (escala de grises). Si tuviéramos una imagen a color, necesitaríamos 3 canales: red, green, blue y entonces usaríamos $28 \times 28 \times 3 = 2352$ neuronas de entrada. Esa es nuestra capa de entrada. Para continuar con el ejemplo, supondremos que utilizamos la imagen con 1 sólo color.

No Olvides: Pre-procesamiento

Antes de alimentar la red, recuerda que como entrada nos conviene normalizar los valores. Los colores de los pixeles tienen valores que van de 0 a 255, haremos una transformación de cada pixel: "valor/255" y nos quedará siempre un valor entre 0 y 1.



Convoluciones

Ahora comienza el “procesado distintivo” de las CNN. Es decir, haremos las llamadas “convoluciones”: Estas consisten en tomar “grupos de pixeles cercanos” de la imagen de entrada e ir operando matemáticamente (producto escalar) contra una pequeña matriz que se llama *kernel*. Ese kernel supongamos de tamaño 3x3 pixels “recorre” todas las neuronas de entrada (de izquierda-derecha, de arriba-abajo) y genera una nueva matriz de salida, que en definitiva será nuestra nueva capa de neuronas ocultas.

NOTA: si la imagen fuera a color, el kernel realmente sería de 3x3x3: un filtro con 3 kernels de 3x3; luego esos 3 filtros se suman (y se le suma una *unidad bias*) y conformarán 1 salida (cómo si fuera 1 solo canal).

		0.6	0.6		
0.6				0.6	
0.6	0.6	0.6	0.6		
0.6				0.6	

Imagen de
entrada

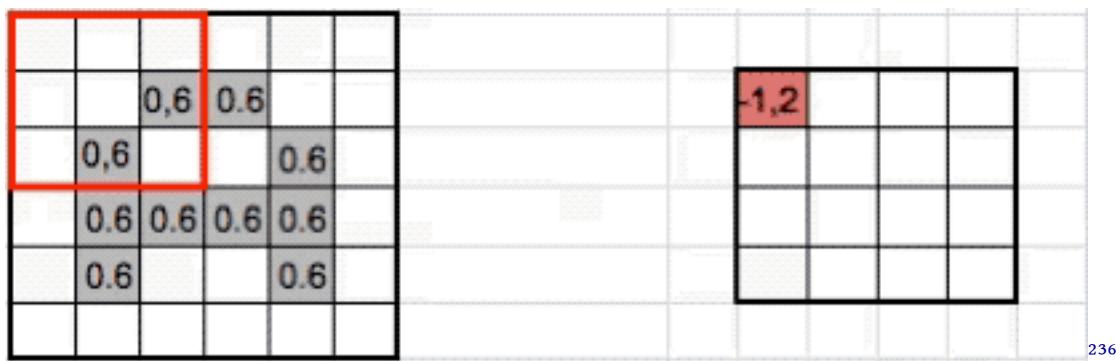
1	0	-1
2	0	-2
1	0	-1

kernel

El kernel tomará inicialmente valores aleatorios(1) y se irán ajustando mediante backpropagation.
 (1)Una mejora es hacer que siga una distribución normal siguiendo simetrías, pero sus valores son aleatorios.

Filtro: conjunto de kernels

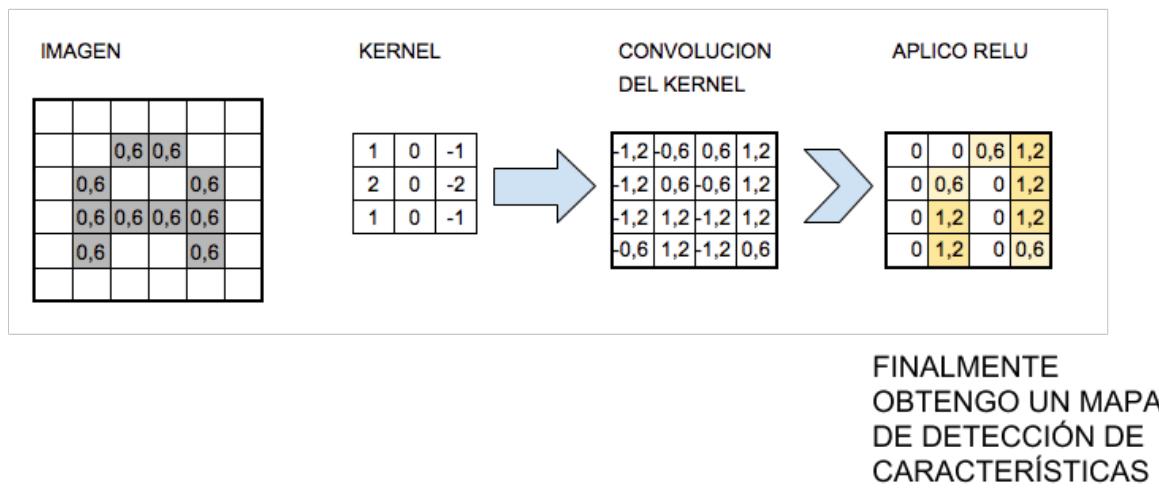
UN DETALLE: en realidad, no aplicaremos 1 sólo kernel, si no que tendremos muchos kernel (su conjunto se llama filtros). Por ejemplo en esta primer convolución podríamos tener 32 filtros, con lo cual realmente obtendremos 32 matrices de salida (este conjunto se conoce como “feature mapping”), cada una de 28x28x1 dando un total del 25.088 neuronas para nuestra PRIMER CAPA OCULTA de neuronas. ¿No les parecen muchas para una imagen cuadrada de apenas 28 pixeles? Imaginen cuántas más serían si tomáramos una imagen de entrada de 224x224x3 (que aún es considerado un tamaño pequeño)...



236

Aquí vemos al kernel realizando el producto matricial con la imagen de entrada y desplazando de a 1 pixel de izquierda a derecha y de arriba-abajo y va generando una nueva matriz que compone al mapa de features.

A medida que vamos desplazando el kernel y vamos obteniendo una “nueva imagen” filtrada por el kernel. En esta primer convolución y siguiendo con el ejemplo anterior, es como si obtuviéramos 32 “imágenes filtradas nuevas”. Estas imágenes nuevas lo que están “dibujando” son ciertas características de la imagen original. Esto ayudará en el futuro a poder distinguir un objeto de otro (por ej. gato ó perro).



237

La imagen realiza una convolución con un kernel y aplica la función de activación, en este caso ReLu[/caption]

²³⁶http://www.aprendemachinelearning.com/wp-content/uploads/2018/11/cnn_kernel.gif

²³⁷<http://www.aprendemachinelearning.com/wp-content/uploads/2018/11/CNN-04.png>

La función de Activación

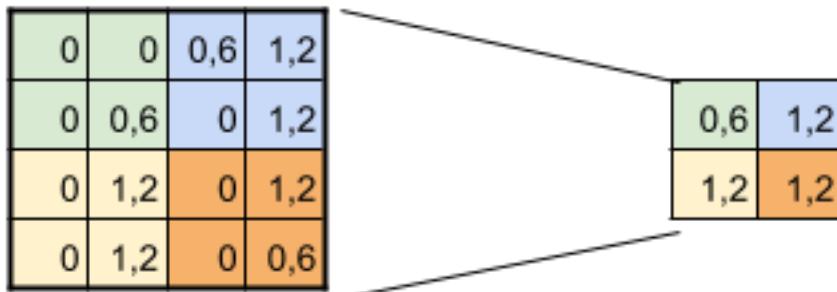
La función de activación más utilizada para este tipo de redes neuronales es la llamada **ReLU²³⁸** por Rectifier Linear Unit y consiste en $f(x)=\max(0,x)$.

Subsampling

Ahora viene un paso en el que reduciremos la cantidad de neuronas antes de hacer una nueva convolución. *¿Por qué?* Como vimos, a partir de nuestra imagen blanco y negro de 28x28px tenemos una primer capa de entrada de 784 neuronas y luego de la primer convolución obtenemos una capa oculta de 25.088 neuronas -que realmente son nuestros 32 mapas de características de 28x28-. Si hiciéramos una nueva convolución a partir de esta capa, el número de neuronas de la próxima capa se iría por las nubes (y ello implica mayor procesamiento)! Para reducir el tamaño de la próxima capa de neuronas haremos un proceso de subsampling en el que reduciremos el tamaño de nuestras imágenes filtradas pero en donde **deberán prevalecer las características más importantes que detectó cada filtro**. Hay diversos tipos de subsampling, yo comentaré el “más usado”: Max-Pooling

²³⁸[https://en.wikipedia.org/wiki/Rectifier_\(neural_networks\)](https://en.wikipedia.org/wiki/Rectifier_(neural_networks))

Subsampling con Max-Pooling



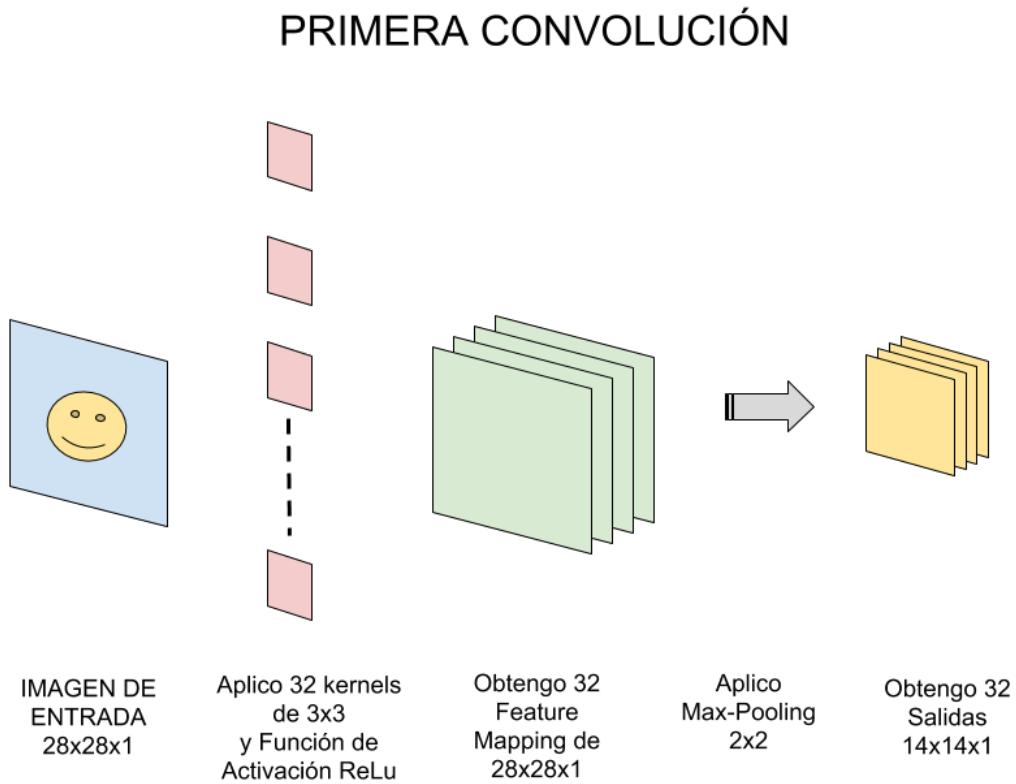
SUBSAMPLING:
Aplico Max-Pooling de 2x2
y reduzco mi salida a la mitad

239

Vamos a intentar explicarlo con un ejemplo: supongamos que haremos Max-pooling de tamaño 2x2. Esto quiere decir que recorreremos cada una de nuestras 32 imágenes de características obtenidas anteriormente de 28x28px de izquierda-derecha, arriba-abajo PERO en vez de tomar de a 1 pixel, tomaremos de “2x2” (2 de alto por 2 de ancho = 4 pixeles) e iremos preservando el valor “más alto” de entre esos 4 pixeles (por eso lo de “Max”). En este caso, usando 2x2, la imagen resultante es reducida “a la mitad” y quedará de 14x14 pixeles. Luego de este proceso de subsampling nos quedarán 32 imágenes de 14x14, pasando de haber tenido 25.088 neuronas a 6272, son bastantes menos y -en teoría- siguen almacenando la información más importante para detectar características deseadas.

²³⁹<http://www.aprendemachinelearning.com/wp-content/uploads/2018/11/cnn-05.png>

¿Ya terminamos? NO: ahora más convoluciones!!



240

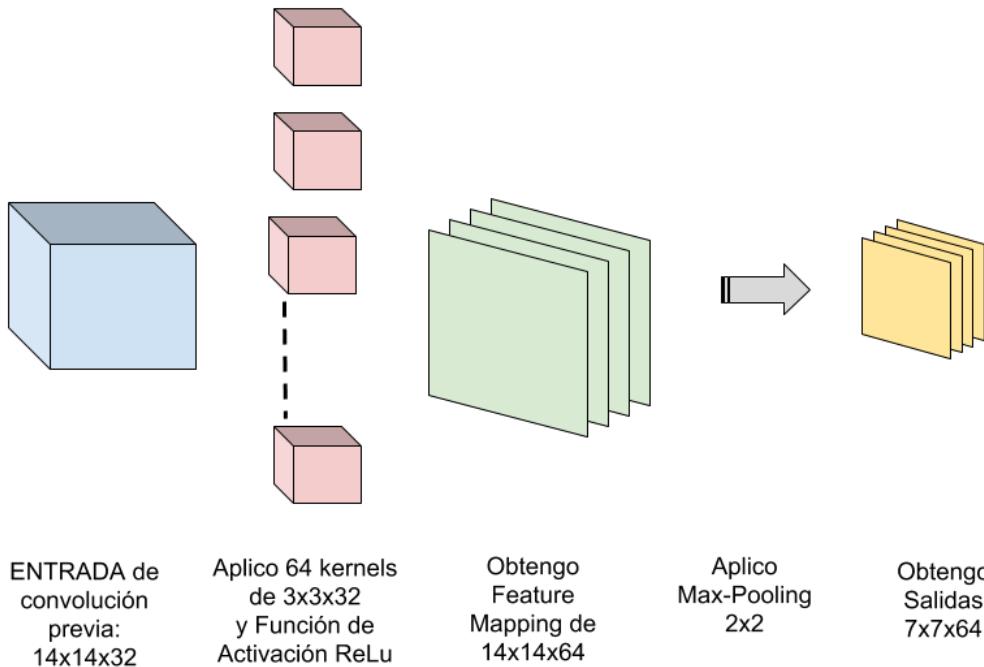
Muy bien, pues esa ha sido una primer convolución: consiste de una entrada, un conjunto de filtros, generamos un mapa de características y hacemos un subsampling. Con lo cual, en el ejemplo de imágenes de 1 sólo color tendremos:

- | | |
|--------------------------------------|-------------------|
| 1)Entrada: Imagen | 28x28x1 |
| 2)Aplico Kernel | 32 filtros de 3x3 |
| 3)Obtengo Feature Mapping | 28x28x32 |
| 4)Aplico Max-Pooling | de 2x2 |
| 5)Obtengo “Salida” de la Convolución | 14x14x32 |

La primer convolución es capaz de detectar características primitivas como líneas ó curvas. A medida que hagamos más capas con las convoluciones, los mapas de características serán capaces de reconocer formas más complejas, y el conjunto total de capas de convoluciones podrá “ver”. Pues ahora deberemos hacer una Segunda convolución que será:

²⁴⁰<http://www.aprendemachinelearning.com/wp-content/uploads/2018/11/cnn-06.png>

SEGUNDA CONVOLUCIÓN (y sucesivas)



241

- | | |
|--------------------------------------|----------------------------|
| 1)Entrada: Imagen | $14 \times 14 \times 32$ |
| 2)Aplico Kernel | 64 filtros de 3×3 |
| 3)Obtengo Feature Mapping | $14 \times 14 \times 64$ |
| 4)Aplico Max-Pooling | de 2×2 |
| 5)Obtengo "Salida" de la Convolución | $7 \times 7 \times 64$ |

La 3er convolución comenzará en tamaño 7×7 pixels y luego del max-pooling quedará en 3×3 con lo cual podríamos hacer sólo 1 convolución más. En este ejemplo empezamos con una imagen de 28×28 px e hicimos 3 convoluciones. Si la imagen inicial hubiese sido mayor (de 224×224 px) aún hubiéramos podido seguir haciendo convoluciones.

- | | |
|--------------------------------------|-----------------------------|
| 1)Entrada: Imagen | $7 \times 7 \times 64$ |
| 2)Aplico Kernel | 128 filtros de 3×3 |
| 3)Obtengo Feature Mapping | $7 \times 7 \times 128$ |
| 4)Aplico Max-Pooling | de 2×2 |
| 5)Obtengo "Salida" de la Convolución | $3 \times 3 \times 128$ |

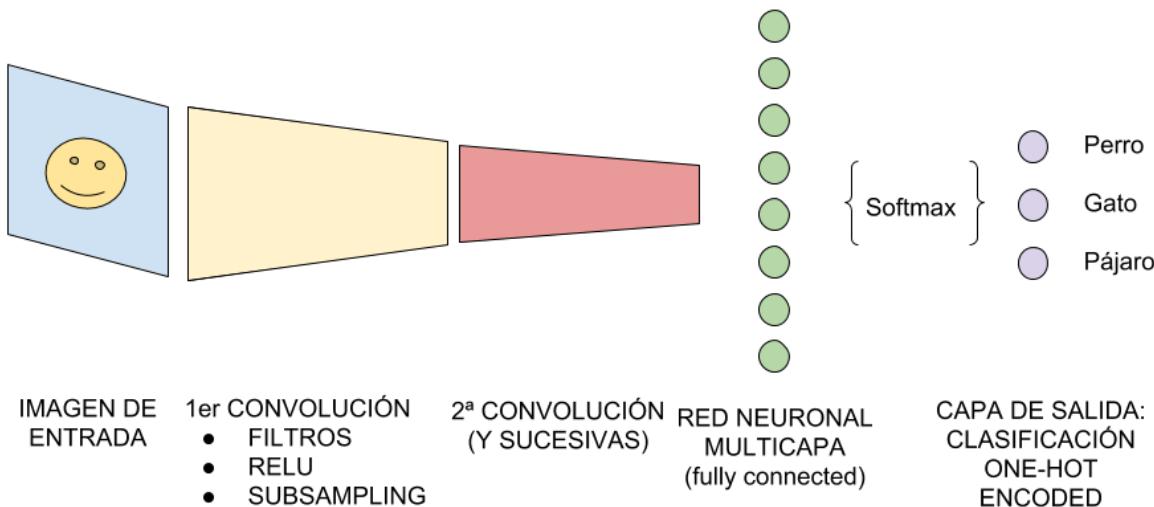
Llegamos a la última convolución y nos queda el desenlace...

²⁴¹<http://www.aprendemachinelearning.com/wp-content/uploads/2018/11/cnn-07.png>

Coneectar con una red neuronal “tradicional”.

Para terminar, tomaremos la última capa oculta a la que hicimos subsampling, que se dice que es “tridimensional” por tomar la forma -en nuestro ejemplo- $3 \times 3 \times 128$ (alto, ancho, mapas) y la “aplanamos”, esto es que deja de ser tridimensional, y pasa a ser una capa de neuronas “tradicionales”, de las que ya conocíamos. Por ejemplo, podríamos aplanar (y conectar) a una nueva capa oculta de 100 neuronas feedforward.

ARQUITECTURA DE UNA CNN



242

Entonces, a esta nueva capa oculta “tradicional”, le aplicamos una función llamada [Softmax²⁴³](#) que conecta contra la capa de salida final que tendrá la cantidad de neuronas correspondientes con las clases que estamos clasificando. Si clasificamos perros y gatos, serán 2 neuronas. Si es el dataset Mnist numérico serán 10 neuronas de salida. Si clasificamos coches, aviones ó barcos serán 3, etc. Las salidas al momento del entrenamiento tendrán el formato conocido como “[one-hot-encoding²⁴⁴](#)” en el que para perros y gatos sera: [1,0] y [0,1], para coches, aviones ó barcos será [1,0,0]; [0,1,0];[0,0,1]. Y la función de Softmax se encarga de pasar a probabilidad (entre 0 y 1) a las neuronas de salida. Por ejemplo una salida [0,2 0,8] nos indica 20% probabilidades de que sea perro y 80% de que sea gato.

²⁴²<http://www.aprendemachinelearning.com/wp-content/uploads/2018/11/CNN-08.png>

²⁴³https://en.wikipedia.org/wiki/Softmax_function

²⁴⁴<https://en.wikipedia.org/wiki/One-hot>

¿Y cómo aprendió la CNN a “ver”? : Backpropagation

El proceso es similar al de las redes tradicionales en las que tenemos una entrada y una salida esperada (por eso [aprendizaje supervisado²⁴⁵](#)) y mediante el backpropagation mejoramos el valor de los pesos de las interconexiones entre capas de neuronas y a medida que iteramos esos pesos se ajustan hasta ser óptimos. PERO... En el caso de la CNN, deberemos **ajustar el valor de los pesos de los distintos kernels**. Esto es una gran ventaja al momento del aprendizaje pues como vimos cada kernel es de un tamaño reducido, en nuestro ejemplo en la primer convolución es de tamaño de 3x3, eso son sólo 9 parámetros que debemos ajustar en 32 filtros dan un total de 288 parámetros. En comparación con los pesos entre dos capas de neuronas “tradicionales”: una de 748 y otra de 6272 en donde están TODAS interconectarlas con TODAS y **eso equivaldría a tener que entrenar y ajustar más de 4,5 millones de pesos** (repito: sólo para 1 capa).

Comparativa entre una red neuronal “tradicional” y una CNN

Dejaré un cuadro resumen para intentar aclarar más las diferencias entre las redes Fully connected y las Convolutional Neural Networks.

Característica	Red “tradicional” Feedforward multicapa	Red Neuronal Convolucional CNN
Datos de entrada en la Capa Inicial	Las características que analizamos. Por ejemplo: ancho, alto, grosor, etc. elegimos una cantidad de neuronas para las capas ocultas.	Pixeles de una imagen. Si es color, serán 3 capas para rojo, verde, azul
Capas ocultas		Tenemos de tipo: * Convolución (con un tamaño de kernel y una cantidad de filtros) * Subsampling
Capa de Salida	La cantidad de neuronas que queremos clasificar. Para “comprar” ó “alquilar” serán 2 neuronas.	Debemos “aplanar” la última convolución con una (ó más) capas de neuronas ocultas “tradicionales” y hacer una salida mediante SoftMax a la capa de salida que clasifica “perro” y “gato” serán 2 neuronas.
Aprendizaje	Supervisado	Supervisado

²⁴⁵<http://www.aprendemachinelearning.com/aplicaciones-del-machine-learning/#supervisado>

Característica	Red “tradicional” Feedforward multicapa	Red Neuronal Convolucional CNN
Interconexiones	Entre capas, todas las neuronas de una capa con la siguiente.	Son muchas menos conexiones necesarias, pues realmente los pesos que ajustamos serán los de los filtros/kernels que usamos.
Significado de la cantidad de capas ocultas	Realmente es algo desconocido y no representa algo en sí mismo.	Las capas ocultas son mapas de detección de características de la imagen y tienen jerarquía: primeras capas detectan líneas, luego curvas y formas cada vez más elaboradas.
Backpropagation	Se utiliza para ajustar los pesos de todas las interconexiones de las capas	Se utiliza para ajustar los pesos de los kernels.

Arquitectura básica

Resumiendo: podemos decir que los elementos que usamos para crear CNNs son:

- **Entrada:** Serán los pixeles de la imagen. Serán alto, ancho y profundidad será 1 sólo color o 3 para Red,Green,Blue.
- **Capa De Convolución:** procesará la salida de neuronas que están conectadas en “regiones locales” de entrada (es decir pixeles cercanos), calculando el producto escalar entre sus pesos (valor de pixel) y una pequeña región a la que están conectados en el volumen de entrada. Aquí usaremos por ejemplo 32 filtros o la cantidad que decidamos y ese será el volumen de salida.
- “CAPA RELU” aplicará la función de activación en los elementos de la matriz.
- **POOL ó SUBSAMPLING:** Hará una reducción en las dimensiones alto y ancho, pero se mantiene la profundidad.
- **CAPA “TRADICIONAL”** red de neuronas feedforward que conectarán con la última capa de subsampling y finalizará con la cantidad de neuronas que queremos clasificar.

No incluido

Quedan muchas cosas más que explicar. Temas y definiciones como padding, stride, evitar overfitting, image-aumentation, dropout... o por nombrar algunas redes famosas ResNet, AlexNet, GoogLeNet and DenseNet, al mismísimo Yann LeCun... todo eso.. se queda fuera de este texto. Este artículo pretende ser un punto inicial para seguir investigando y aprendiendo sobre las CNN. Al final dejo enlace a varios artículos para ampliar información sobre CNN.

Resumen

Hemos visto cómo este algoritmo utiliza variantes de una red neuronal tradicional y las combina con el comportamiento biológico del ojo humano, para lograr aprender a ver.

- Understanding Convolutional Neural Networks²⁴⁶
- CS231n Convolutional Neural Networks for Visual Recognition²⁴⁷
- Introducing convolutional networks²⁴⁸
- Intuitively Understanding Convolutions for Deep Learning²⁴⁹
- Feature Visualization—How neural networks build up their understanding of images²⁵⁰
- Back Propagation in Convolutional Neural Networks²⁵¹
- [<https://medium.com/technologymadeeasy/>]

the-best-explanation-of-convolutional-neural-networks-on-the-internet-fbb8b1ad5df8)

²⁴⁶<https://towardsml.com/2018/10/16/deep-learning-series-p2-understanding-convolutional-neural-networks/>

²⁴⁷<http://cs231n.github.io/>

²⁴⁸http://neuralnetworksanddeeplearning.com/chap6.html#introducing_convolutional_networks

²⁴⁹<https://towardsdatascience.com/intuitively-understanding-convolutions-for-deep-learning-1f6f42faee1>

²⁵⁰<https://distill.pub/2017/feature-visualization/>

²⁵¹<https://becominghuman.ai/back-propagation-in-convolutional-neural-networks-intuition-and-code-714ef1c38199>

Detección de Objetos con Python

Veremos de manera práctica cómo crear tu propio detector de objetos que podrás utilizar con imágenes estáticas, video o cámara. Avanzaremos paso a paso en una Jupyter Notebook con el código completo usando Keras sobre Tensorflow.

Agenda

Tenemos mucho por delante!

- ¿En qué consiste la Detección Yolo?
 - Algunos parámetros de la red
 - El proyecto propuesto
- Lo que tienes que instalar (y todo el material)
- Crear un dataset: Imágenes y Anotaciones
 - Recomendaciones para la imágenes
 - Anotarlo todo
 - El lego dataset
- El código Python
 - Leer el dataset
 - Train y Validación
 - Data Augmentation
 - Crear la red YOLO
 - Crear la red de Detección
 - Generar las Anclas
 - Entrenar
 - Revisar los Resultados
 - Probar la red!
- Conclusiones
- Material Adicional

¿En qué consiste la detección YOLO?

Vamos a hacer un detector de objetos en imágenes utilizando YOLO, un tipo de técnica muy novedosa (2016), acrónimo de “You Only Look Once” y que es la más rápida del momento, permitiendo su uso en video en tiempo real.

Esta técnica utiliza un tipo de red Neuronal Convolutiva llamada Darknet para la clasificación de imágenes y le añade la parte de la detección, es decir un “cuadradito” con las posiciones x e y, alto y ancho del objeto encontrado.

La dificultad de esta tarea es enorme!, poder localizar las áreas de las imágenes, que para una red neuronal es tan sólo una matriz de pixeles de colores, posicionar múltiples objetos y clasificarlos. YOLO lo hace todo “de una sola pasada” a su red convolucional. En resultados sobre el COCO Dataset detecta 80 clases de objetos distintos y etiquetar y posicionar 1000 objetos (en 1 imagen!!!)

NOTA : Este código se basa en varios trozos de código de diversos repositorios de Github y usa una arquitectura de YOLOv2 aunque YA Sé que es mejor la versión 3 y de hecho está por salir Yolo v4. Pero con fines didácticos la v2 nos alzanza!

Aunque ahondaré en la Teoría en un próximo capítulo, aquí comentaré varios parámetros que manearemos con esta red y que debemos configurar.

(Algunos) Parámetros de la red

- Tamaño de imagen que procesa la red: este será fijo, pues encaja con el resto de la red y es de 416 pixeles. Todas las imágenes que le pasemos serán redimensionadas antes de entrar en la red.
- Cantidad de cajas por imagen: Estás serán la cantidad de objetos máximos que queremos detectar.
- etiquetas: estas serán las de los objetos que queramos detectar. En este ejemplo sólo detectaremos 1 tipo de objeto, pero podrían ser múltiples.
- epochs: la cantidad de iteraciones sobre TODO el dataset que realizará la red neuronal para entrenar. (Recuerda, que a muchas épocas tardará más tiempo y también aumenta el riesgo de overfitting)
- train_times: este valor se refiera a la cantidad de veces de entrenar una MISMA imagen. Esto sirve sobre todo en datasets pequeños pues haremos data augmentation sobre las imágenes cada vez.
- saved_weights_name: una vez entrenada la red, guardaremos sus pesos en este archivo y lo usaremos para hacer las predicciones.

El proyecto Propuesto: Detectar personajes de Lego

Será porque soy padre, ó será porque soy Ingeniero... al momento de pensar en un objeto para detectar se me ocurrió esto: Legos! ¿Quién no tiene legos en su casa?... Por supuesto que puedes

crear tu propio dataset de imagenes y anotaciones xml para detectar el ó los objetos que tu quieras!

Lo que tienes que instalar

Primero que nada te recomiendo que crees un nuevo Environment de Python 3.6.+ e instalas estas versiones de librerías que usaremos.

En consola escribe:

```
1 $ python -m venv detectaEnv
```

Y luego lo ACTIVAS para usarlo en windows con:

```
1 $ detectaEnv\Scripts\activate.bat
```

ó en Linux / Mac con:

```
1 $ source detectaEnv/bin/activate
```

y luego instala los paquetes:

```
1 pip install tensorflow==1.13.2
2 pip install keras==2.0.8
3 pip install imgaug==0.2.5
4 pip install opencv-python
5 pip install h5py
6 pip install tqdm
7 pip install imutils
```

Aclaraciones: usamos una versión antigua de Tensorflow. Si tienes GPU en tu máquina, puedes usar la versión apropiada de Tensorflow.

Si vas crear tu propio dataset como se explica a continuación, deberás instalar LabelImg, que requiere:

```
1 pip install PyQt5
2 pip install lxml
3 pip install labelImg
```

Si no, puedes usar el dataset de legos que provee el blog y saltarte la parte de crear el dataset.

Además otros archivos que deberás descargar:

- * Archivo con Pesos iniciales de la red Darknet de Yolov2²⁵² (192MB)
- * Código Python detección de imágenes²⁵³ - Jupyter Notebook
- * OPCIONAL: Dataset de lego creado por mi²⁵⁴
- * OPCIONAL crea y usa tu propio dataset de imágenes y anotaciones

Crea un dataset: Imágenes y Anotaciones

Es hora de crear un repositorio de miles de imágenes para alimentar tu red de detección.

En principio te recomendaría que tengas al menos unas 1000 imágenes de cada clase que quieras detectar. Y de cada imagen deberás tener un archivo xml con un formato que en breve comentaré con la clase y la posición de cada objeto. Pero recuerda que al detectar imágenes podemos tener más de un objeto, entonces puedes tener imágenes que tienen a más de una clase.

Recomendaciones para las imágenes:

Algunas recomendaciones para la captura de imágenes: si vas a utilizar la cámara de tu móvil, puede que convenga que las hagas con “pocos megapixeles”, pues si haces una imagen de 5 MB, luego la red la reducirá a 416 pixeles de ancho, por lo que tendrá un coste ese preprocesado en tiempo, memoria y CPU.

Intenta tener fotos del/los objetos con distintas condiciones de luz, es decir, no tengas imágenes de gatitos siempre al sol. Mejor imágenes de interior, al aire libre, con poca luz, etc.

Intenta tener imágenes “torcidas”(rotadas), parciales y de distintos tamaños del objeto. Si sólo tienes imágenes en donde tu objeto supongamos que “mide 100pixeles” mal-acostumbrarás la red y sólo detectará en imágenes cuando sea de esas dimensiones.

Variaciones del mismo objeto: Si tu objeto es un gato, intenta clasificar gatos de distintos colores, razas y en distintas posiciones, para que la red pueda generalizar el conocimiento.

Anotarlo Todo

Muy bien, ya tienes tus imágenes hechas y guardadas en un directorio.

Ahora deberás crear un archivo XML donde anotarás cada objeto, sus posiciones x,y su alto y ancho.

El xml será de este tipo:

²⁵²https://drive.google.com/file/d/1Q9WhhRlqQbA4jgBkCDrynvgquRXZA_f8/view?usp=sharing

²⁵³https://github.com/jbagnato/machine-learning/blob/master/Ejercicio_Object_Detection.ipynb

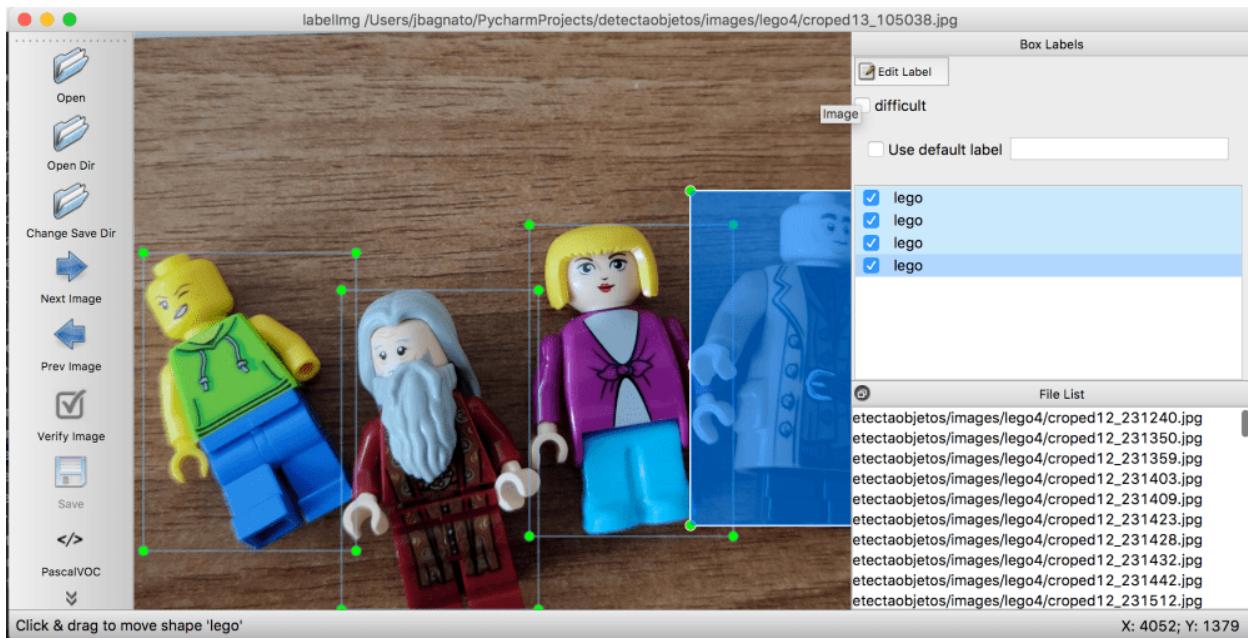
²⁵⁴<https://leapub.com/aprendem1>

```

1 <annotation>
2   <folder>lego</folder>
3   <filename>cropped12_231359.jpg</filename>
4   <path>/detectaobjetos/images/lego3/cropped12_231359.jpg</path>
5   <size>
6     <width>4512</width>
7     <height>4512</height>
8     <depth>3</depth>
9   </size>
10  <object>
11    <name>lego</name>
12    <bndbox>
13      <xmin>909</xmin>
14      <ymin>879</ymin>
15      <xmax>3200</xmax>
16      <ymax>4512</ymax>
17    </bndbox>
18  </object>
19 </annotation>

```

Y lo puedes hacer a mano... ó puedes usar un editor como labelImg.



Si lo instalaste con Pip, puedes ejecutarlo simplemente poniendo en línea de comandos labelImg. Se abrirá el editor y podrás

- seleccionar un directorio como fuente de imágenes

- otro directorio donde guardará los xml.

En el editor deberás crear una caja sobre cada objeto que quieras detectar en la imagen y escribir su nombre. Cuando terminas le das a Guardar y Siguiente.

El lego dataset

Si quieres puedes utilizar un dataset de imágenes que creé para este ejercicio y consta de 300 imágenes. Son fotos tomadas con móvil de diversos personajes lego. Realmente son 100 fotos y 200 variaciones en zoom y recortes. Y sus correspondientes 300 archivos de anotaciones xml

Dicho esto, recuerda que siempre es mejor más y más imágenes para entrenar.



cropped12_231928.jpg



cropped12_232100.jpg



cropped12_232104.jpg



cropped12_232106.jpg



cropped12_232110.jpg



cropped12_232112.jpg



cropped12_232117.jpg



cropped12_232121.jpg



cropped12_232124.jpg



cropped12_232140.jpg



cropped12_232153.jpg



cropped13_104942.jpg



cropped13_104954.jpg



cropped13_105004.jpg



cropped13_105025.jpg



cropped13_105029.jpg



cropped13_105038.jpg



cropped13_105127.jpg



El código Python

Usaremos Keras sobre Tensorflow para crear la red!, manos a la obra.

En el artículo copiaré los trozos de código más importante, pero recuerda [descargar la notebook Jupyter con el código completo desde Github²⁵⁵](#).

Leer el Dataset

Primer paso, será el de leer las anotaciones xml que tenemos creadas en un directorio e ir iterando los objetos para contabilizar las etiquetas.

```
1  xml_dir = "annotation/lego/"
2  img_dir = "images/lego/"
3  labels = ["lego"]
4  tamano = 416
5  mejores_pesos = "red_lego.h5"
6
7  def leer_annotations(ann_dir, img_dir, labels=[]):
8      all_imgs = []
9      seen_labels = {}
10
11     for ann in sorted(os.listdir(ann_dir)):
12         img = {'object': []}
13
14         tree = ET.parse(ann_dir + ann)
15
16         for elem in tree.iter():
17             if 'filename' in elem.tag:
18                 img['filename'] = img_dir + elem.text
19             if 'width' in elem.tag:
20                 img['width'] = int(elem.text)
21             if 'height' in elem.tag:
22                 img['height'] = int(elem.text)
23             if 'object' in elem.tag or 'part' in elem.tag:
24                 obj = {}
25
26                 for attr in list(elem):
27                     if 'name' in attr.tag:
28                         obj['name'] = attr.text
29
30                         if obj['name'] in seen_labels:
31                             seen_labels[obj['name']] += 1
32                         else:
```

²⁵⁵https://github.com/jbagnato/machine-learning/blob/master/Ejercicio_Object_Detection.ipynb

```

33             seen_labels[obj['name']] = 1
34
35         if len(labels) > 0 and obj['name'] not in labels:
36             break
37         else:
38             img['object'] += [obj]
39
40         if 'bndbox' in attr.tag:
41             for dim in list(attr):
42                 if 'xmin' in dim.tag:
43                     obj['xmin'] = int(round(float(dim.text)))
44                 if 'ymin' in dim.tag:
45                     obj['ymin'] = int(round(float(dim.text)))
46                 if 'xmax' in dim.tag:
47                     obj['xmax'] = int(round(float(dim.text)))
48                 if 'ymax' in dim.tag:
49                     obj['ymax'] = int(round(float(dim.text)))
50
51         if len(img['object']) > 0:
52             all_imgs += [img]
53
54     return all_imgs, seen_labels
55
56 train_imgs, train_labels = leer_annotations(xml_dir, img_dir, labels)
57 print('imagenes', len(train_imgs), 'labels', len(train_labels))

```

Train y Validación

Separaremos un 20% de las imágenes y anotaciones para testear el modelo. En este caso se utilizará el set de Validación al final de cada época para evaluar métricas, pero nunca se usará para entrenar.

```

1 train_valid_split = int(0.8*len(train_imgs))
2 np.random.shuffle(train_imgs)
3 valid_imgs = train_imgs[train_valid_split:]
4 train_imgs = train_imgs[:train_valid_split]
5 print('train:', len(train_imgs), 'validate:', len(valid_imgs))

```

Data Augmentation

El Data Augmentation sirve para agregar pequeñas alteraciones ó cambios a las imágenes de entradas aumentando nuestro dataset de imágenes y mejorando la capacidad de la red para detectar objetos.

Para hacerlo nos apoyamos sobre una librería imgaug que nos brinda muchas funcionalidades como agregar desenfoque, agregar brillo, ó ruido aleatoriamente a las imágenes. Además podemos usar OpenCV para voltear la imagen horizontalmente y luego recolocar la “bounding box”.

```

1  ### FRAGMENTO del código
2
3  iaa.OneOf([
4      iaa.GaussianBlur((0, 3.0)), # blur images
5      iaa.AverageBlur(k=(2, 7)), # blur image using local means with kernel
6      iaa.MedianBlur(k=(3, 11)), # blur image using local medians with kernel
7  ]),
8      iaa.Sharpen(alpha=(0, 1.0), lightness=(0.75, 1.5)), # sharpen images
9      iaa.AdditiveGaussianNoise(loc=0, scale=(0.0, 0.05*255), per_channel=0.5), # add \
10 gaussian noise to images
11     iaa.OneOf([
12         iaa.Dropout((0.01, 0.1), per_channel=0.5), # randomly remove up to 10% of th\
13 e pixels
14     ]),
15     iaa.Add((-10, 10), per_channel=0.5), # change brightness of images
16     iaa.Multiply((0.5, 1.5), per_channel=0.5), # change brightness of images
17     iaa.ContrastNormalization((0.5, 2.0), per_channel=0.5), # improve or worsen the \
18 contrast

```

Crear la Red de Clasificación

La red es conocida como Darknet y está compuesta por 22 capas convolucionales que básicamente aplican BatchNormalizarion, MaxPooling y activación por LeakyRelu para la extracción de características, es decir, los patrones que encontrará en las imágenes (en sus pixeles) para poder diferenciar entre los objetos que queremos clasificar.

Va alternando entre aumentar y disminuir la cantidad de filtros y kernel de 3x3 y 1x1 de la red convolucional.

```

1  #### FRAGMENTO de código, solo algunas capas de ejemplo
2
3  # Layer 1
4  x = Conv2D(32, (3,3), strides=(1,1), padding='same', name='conv_1', use_bias=False)(\
5  input_image)
6  x = BatchNormalization(name='norm_1')(x)
7  x = LeakyReLU(alpha=0.1)(x)
8  x = MaxPooling2D(pool_size=(2, 2))(x)
9

```

```
10 # Layer 2
11 x = Conv2D(64, (3,3), strides=(1,1), padding='same', name='conv_2', use_bias=False)(\
12 x)
13 x = BatchNormalization(name='norm_2')(x)
14 x = LeakyReLU(alpha=0.1)(x)
15 x = MaxPooling2D(pool_size=(2, 2))(x)
16
17 # Layer 3
18 x = Conv2D(128, (3,3), strides=(1,1), padding='same', name='conv_3', use_bias=False)\ \
19 (x)
20 x = BatchNormalization(name='norm_3')(x)
21 x = LeakyReLU(alpha=0.1)(x)
```

No olvides descargar y copiar en el mismo directorio donde ejecutes la notebook los pesos de la red Darknet²⁵⁶, pues en este paso se cargarán para inicializar la red.

Crear la Red de Detección

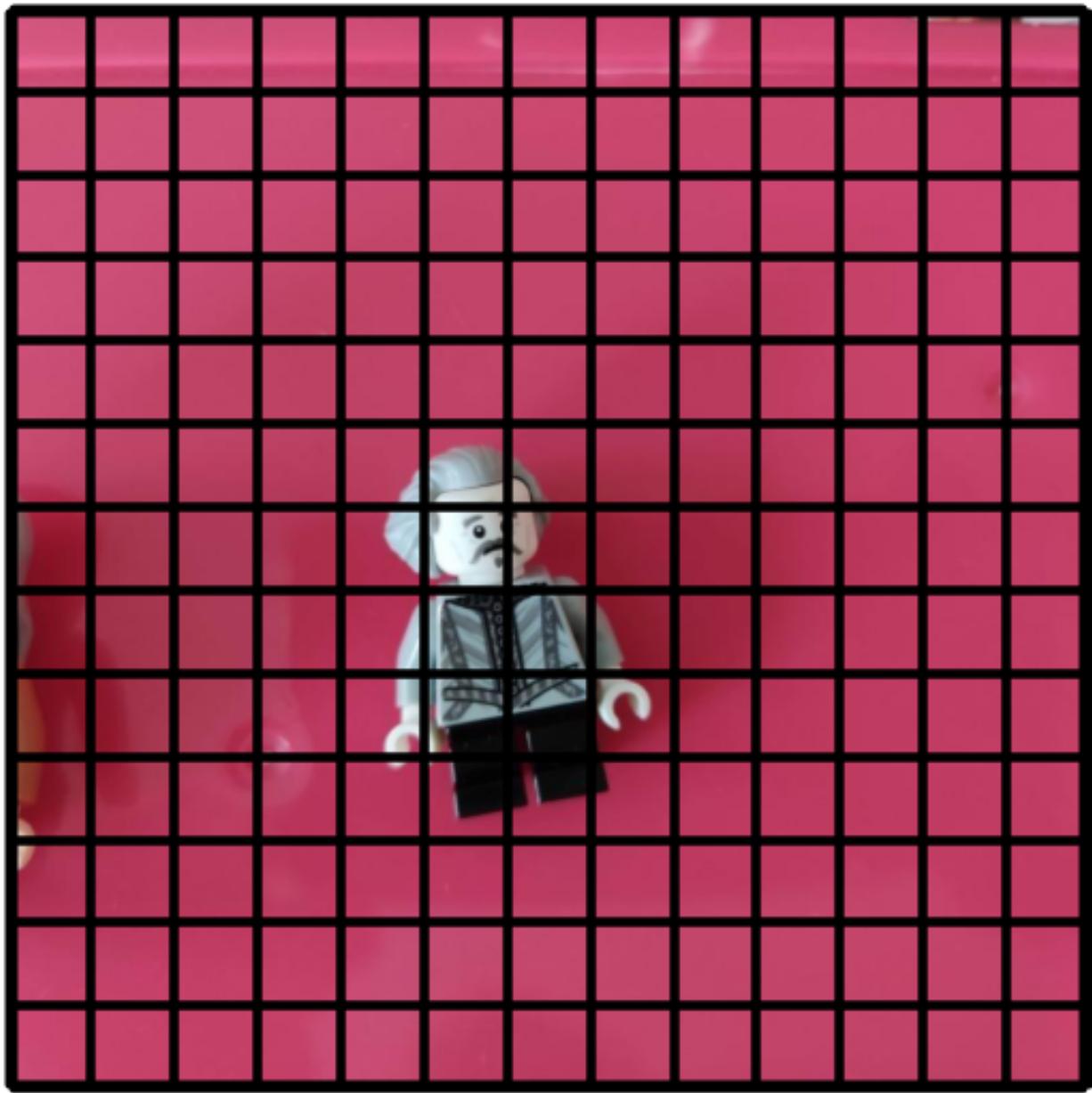
Esta red, utilizará la anterior (claseficación) y utilizará las features obtenidas en sus capas convolucionales de salida para hacer la detección de los objetos, es decir las posiciones x e y, alto y ancho. Para ello se valdrá de unas Anclas, en nuestro caso serán 5. Las Anclas son unas “ventanas”, o unas bounding boxes de distintos tamaños, pequeños, mediano grande, rectangulares o cuadrados que servirán para hacer “propuestas de detección”.

```
1  ### Fragmento de código
2
3      input_image      = Input(shape=(self.input_size, self.input_size, 3))
4      self.true_boxes = Input(shape=(1, 1, 1, max_box_per_image , 4))
5
6      self.feature_extractor = FullYoloFeature(self.input_size)
7
8      print(self.feature_extractor.get_output_shape())
9      self.grid_h, self.grid_w = self.feature_extractor.get_output_shape()
10     features = self.feature_extractor.extract(input_image)
11
12     # make the object detection layer
13     output = Conv2D(self.nb_box * (4 + 1 + self.nb_class),
14                     (1,1), strides=(1,1),
15                     padding='same',
16                     name='DetectionLayer',
```

²⁵⁶https://drive.google.com/file/d/1Q9WhhRlqQbA4jgBkCDrynvgquRXZA_f8/view?usp=sharing

```
17             kernel_initializer='lecun_normal')(features)
18     output = Reshape((self.grid_h, self.grid_w, self.nb_box, 4 + 1 + self.nb_cla\
19 ss))(output)
20     output = Lambda(lambda args: args[0])([output, self.true_boxes])
21
22     self.model = Model([input_image, self.true_boxes], output)
```

En total, la red crea una grilla de 13x13 y en cada una realizará 5 predicciones, lo que da un total de 845 posibles detecciones para cada clase que queremos detectar. Si tenemos 10 clases esto serían 8450 predicciones, cada una con la clase y sus posiciones x,y ancho y alto. Lo más impresionante de esta red YOLO es que lo hace Todo de 1 sólo pasada! increíble!



Para refinar el modelo y que detecte los objetos que hay, utilizará dos funciones con las cuales descartará áreas vacías y se quedará sólo con las mejores propuestas. Las funciones son:

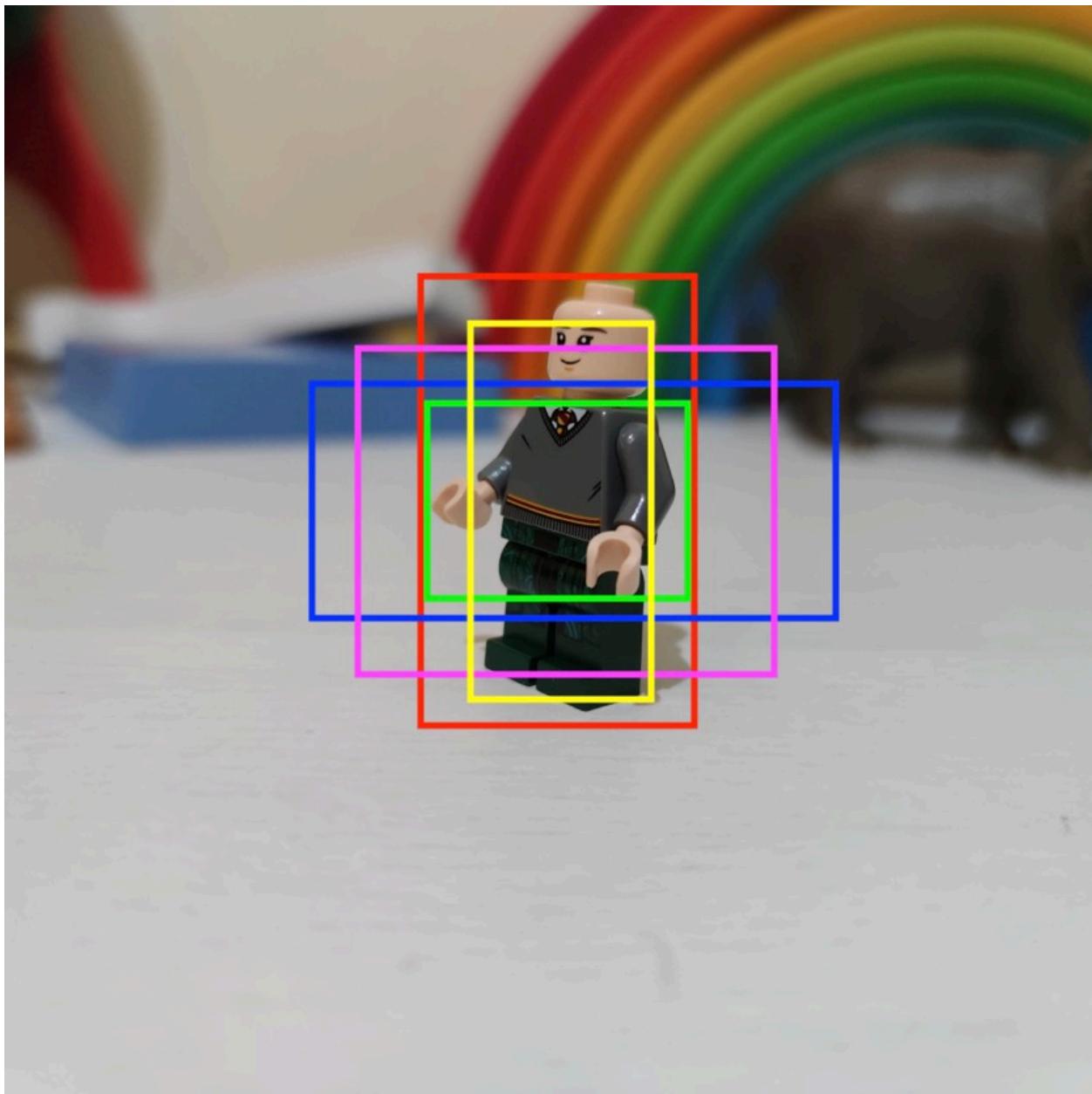
- IOU: Intersection Over Union, que nos da un porcentaje de acierto del área de predicción contra la “cajita” real que queremos predecir
- Non Maximum suppression: nos permite quedarnos de entre nuestras 5 anclas, con la que mejor se ajusta al resultado. Esto es porque podemos tener muchas áreas diferentes propuestas que se superponen. De entre todas, nos quedamos con la mejor y eliminamos al resto.

Entonces, pensemos que si en nuestra red de detección de 1 sólo clase detectamos 1 objeto, esto quiere decir que la red descarto a las 844 restantes.

NOTA: por más que haya separado en 2 redes: red YOLO y red de detección, realmente es 1 sola red convolucional, pues están conectadas y al momento de entrenar, los pesos se ajustan “como siempre” con el backpropagation.

Generar las Anclas

Como antes mencioné, la red utiliza 5 anclas para cada una de las celdas de 13x13 para realizar las propuestas de predicción. Pero... ¿qué tamaño tienen que tener esas anclas? Podríamos pensar en 5 tamaños distintos, algunos pequeños, otros más grandes y que se adapten a las clases que queremos detectar. Por ejemplo, el ancla para detectar siluetas de personas serán seguramente rectangulares en vertical.



Pues según lo que quieras detectar conviene ajustar esos tamaños. Ejecutaremos un pequeño script que utiliza k-means y determina los mejores 5 clusters (de dimensiones) que se adapten a tu dataset.

Entrenar la Red!

A entrenar la red neuronal. Como dato informativo, en mi ordenador macbook de 4 núcleos y 8GB de RAM, tardó 7 horas en entrenar las 300 imágenes del dataset de lego con 7 épocas (y 5 veces cada imagen).

```

1 yolo = YOLO(input_size          = tamano,
2               labels           = labels,
3               max_box_per_image = 5,
4               anchors          = anchors)

```

Al finalizar verás que se ha creado un archivo nuevo llamado “red_lego.h5” que contiene los pesos de la red creada.

Revisar los Resultados

Los resultados vienen dados por una métrica llamada *mAP* y que viene a ser un equivalente a un F1-Score pero para imágenes, teniendo en cuenta los falsos positivos y negativos. Comentaremos más sobre esto en el próximo capítulo, pero ten en cuenta que si bien la ventaja de YOLO es la detección en tiempo real, su contra es que es “un poco” peor en accuracy que otras redes que son lentas, lo podemos notar al ver que las “cajitas” no se ajustan del todo con el objeto detectado ó puede llegar a confundir a veces la clase que clasifica. Con el Lego Dataset he logrado al rededor de 63 de mAP... no está mal. Recordemos que este valor de mAP se obtiene al final de la última Epoch sobre el dataset de Validación (que no se usa para entrenar) y en mi caso eran 65 imágenes.

Probar la Red

Para finalizar, podemos probar la red con imágenes nuevas, distintas que no ha visto nunca, veamos cómo se comporta la red!

Crearemos unas funciones de ayuda para dibujar el rectángulo sobre la imagen original y guardar la imagen nueva:

```

1 def draw_boxes(image, boxes, labels):
2     image_h, image_w, _ = image.shape
3
4     for box in boxes:
5         xmin = int(box.xmin*image_w)
6         ymin = int(box.ymin*image_h)
7         xmax = int(box.xmax*image_w)
8         ymax = int(box.ymax*image_h)
9
10        cv2.rectangle(image, (xmin,ymin), (xmax,ymax), (0,255,0), 3)
11        cv2.putText(image,
12                    labels[box.get_label()] + ' ' + str(box.get_score()),
13                    (xmin, ymin - 13),
14                    cv2.FONT_HERSHEY_SIMPLEX,

```

```
15           1e-3 * image_h,
16           (0,255,0), 2)
17
18     return image
```

Recuerda que utilizaremos el archivo de pesos creado al entrenar, para recrear la red (esto nos permite poder hacer predicciones sin necesidad de reentrenar cada vez).

```
1 mejores_pesos = "red_lego.h5"
2 image_path = "images/test/lego_girl.png"
3
4 mi_yolo = YOLO(input_size          = tamano,
5                  labels            = labels,
6                  max_box_per_image = 5,
7                  anchors           = anchors)
8
9 mi_yolo.load_weights(mejores_pesos)
10
11 image = cv2.imread(image_path)
12 boxes = mi_yolo.predict(image)
13 image = draw_boxes(image, boxes, labels)
14
15 print('Detectados', len(boxes))
16
17 cv2.imwrite(image_path[:-4] + '_detected' + image_path[-4:], image)
```

Como salida tendremos una nueva imagen llamada “lego_girl_detected.png” con la detección realizada.



Esta imagen me fue prestada por [@Shundeez_official²⁵⁷](#), muchas gracias! Les recomiendo ver su cuenta de Instagram que es genial!

²⁵⁷https://www.instagram.com/Shundeez_official/



Imágenes pero también Video y Cámara!

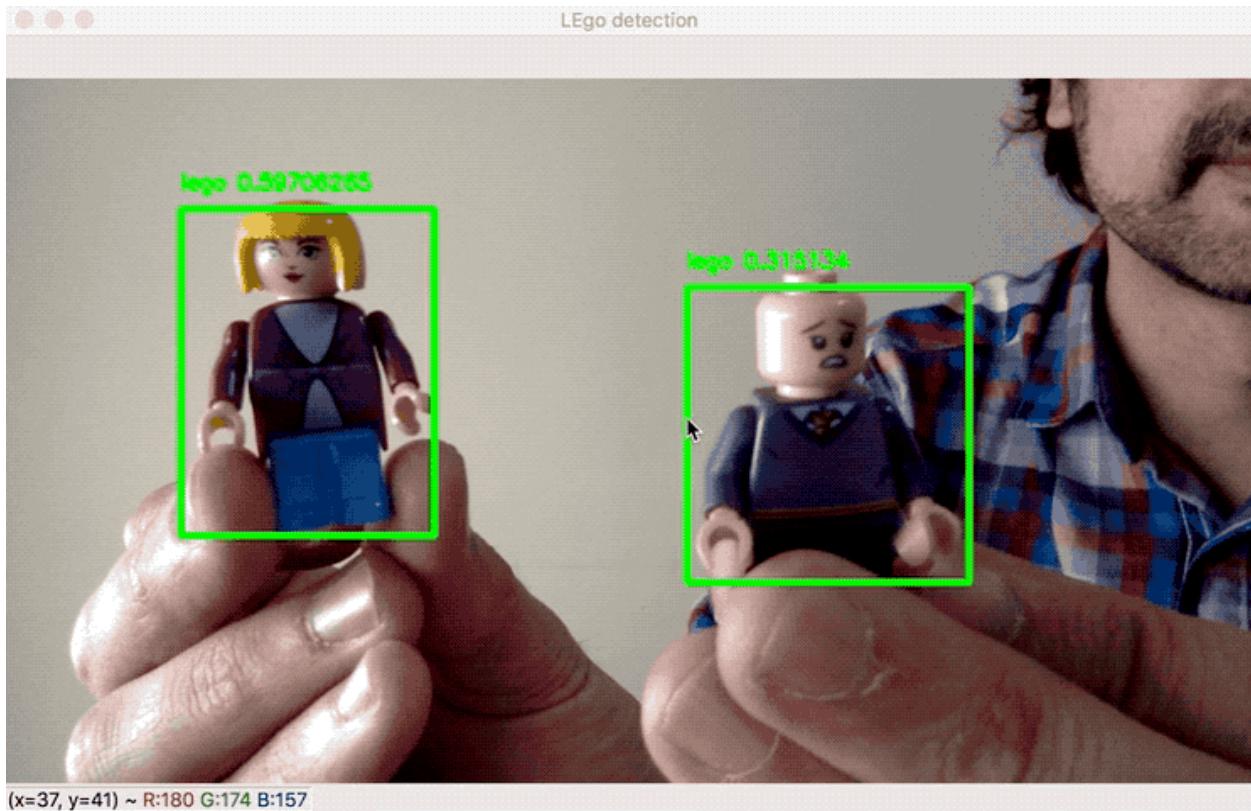
Puedes modificar un poco la manera de realizar predicciones para utilizar un video mp4 ó tu cámara web.

Para aplicarlo a un video:

```
1 from tqdm import *
2
3 video_path = 'lego_movie.mp4'
4 video_out = video_path[:-4] + '_detected' + video_path[-4:]
5 video_reader = cv2.VideoCapture(video_path)
6
7 nb_frames = int(video_reader.get(cv2.CAP_PROP_FRAME_COUNT))
8 frame_h = int(video_reader.get(cv2.CAP_PROP_FRAME_HEIGHT))
9 frame_w = int(video_reader.get(cv2.CAP_PROP_FRAME_WIDTH))
10
11 video_writer = cv2.VideoWriter(video_out,
12                                 cv2.VideoWriter_fourcc(*'MPEG'),
13                                 50.0,
14                                 (frame_w, frame_h))
15
16 for i in tqdm(range(nb_frames)):
17     _, image = video_reader.read()
18
19     boxes = yolo.predict(image)
20     image = draw_boxes(image, boxes, labels)
21
22     video_writer.write(np.uint8(image))
23
24 video_reader.release()
25 video_writer.release()
```

Luego de procesar el video, nos dejará una versión nueva del archivo mp4 con la detección que realizó cuadro a cuadro.

Y para usar tu cámara: (presiona ‘q’ para salir)



```
1 win_name = 'Lego detection'
2 cv2.namedWindow(win_name)
3
4 video_reader = cv2.VideoCapture(0)
5
6 while True:
7     _, image = video_reader.read()
8
9     boxes = yolo.predict(image)
10    image = draw_boxes(image, boxes, labels)
11
12    cv2.imshow(win_name, image)
13
14    key = cv2.waitKey(1) & 0xFF
15    if key == ord('q'):
16        break
17
18 cv2.destroyAllWindows()
19 video_reader.release()
```

Resumen

Esta fue la parte práctica de una de las tareas más interesantes dentro de la Visión Artificial, que es la de lograr hacer detección de objetos. Piensen todo el abanico de posibilidades que ofrece poder hacer esto! Podríamos con una cámara contabilizar la cantidad de coches y saber si hay una congestión de tráfico, podemos contabilizar cuantas personas entran en un comercio, si alguien toma un producto de una estantería y mil cosas más! Ni hablar en robótica, donde podemos hacer que el robot vea y pueda coger objetos, ó incluso los coches de Tesla con Autopilot... Tiene un gran potencial!

Además en este capítulo te ofrece la posibilidad de entrenar tus propios detectores, para los casos de negocio que a ti te importan.

En un próximo artículo espero escribir sobre la Teoría que hoy pusimos en práctica sobre Detección de Objetos.

Recursos

Recuerda todo lo que tienes que descargar:

- * Código Python completo en la Jupyter Notebook²⁵⁸
- * Los pesos iniciales de la red²⁵⁹ YOLOv2
- * Set de imágenes y anotaciones Lego²⁶⁰ (adquiriendo el libro de pago ó gratis)

Y enlaces a otros artículos de interés:

- * Object Detection²⁶¹
- * A Brief History of CNN in Image Segmentation²⁶²
- * Beginners Guito to implementing Yolov3 in Tensorflow²⁶³
- * Practical Guide to Object Detection with Yolo²⁶⁴
- * A very shallow overview of Yolo and Darknet²⁶⁵
- * Yolo v2 object detection²⁶⁶

²⁵⁸https://github.com/jbagnato/machine-learning/blob/master/Ejercicio_Object_Detection.ipynb

²⁵⁹https://drive.google.com/file/d/1Q9WhhRlqQbA4jgBkCDrynvgrXZA_f8/view?usp=sharing

²⁶⁰<https://leapub.com/aprendem>

²⁶¹<https://www.saagie.com/blog/object-detection-part1/>

²⁶²<https://blog.athelas.com/a-brief-history-of-cnns-in-image-segmentation-from-r-cnn-to-mask-r-cnn-34ea83205de4>

²⁶³<https://machinelearningspace.com/yolov3-tensorflow-2-part-1/>

²⁶⁴<https://www.analyticsvidhya.com/blog/2018/12/practical-guide-object-detection-yolo-framework-python/>

²⁶⁵<https://martinapugliese.github.io/recognise-objects-yolo/>

²⁶⁶<https://www.geeksforgeeks.org/yolo-v2-object-detection/>

Conjuntos de Train, Test y Validación

Trabajo en progreso...

Motor de Recomendación

Trabajo en progreso...

Naive Bayes

Trabajo en progreso...

Overfitting y la Generalización del conocimiento

Trabajo en progreso...

Redes Neuronales

Trabajo en progreso...

Deep Learning

Trabajo en progreso...

Tensorflow y Keras

Trabajo en progreso...

Coche Robot que conduce solo

Trabajo en progreso...

Series Temporales

Trabajo en progreso...

Crea tu propio servicio de Machine Learning

Trabajo en progreso...