Curso Nbdev – Módulo con Clusters

Hacer todo el desarrollo

# Ejemplo (Tema 3)

Un uso común en el mundo de los datos es el uso de algoritmos de agrupamiento o clusterización. A continuación vamos a tomar distintos algoritmos de clusterización y vamos a ponerlos todos dentro de un módulo. Este simple ejemplo aunque sencillo nos mostrará de forma más extensa el uso de nbdev.

Primero generamos nuestro repositorio como lo hicimos en el ejemplo [anterior](../Tema2_extendido/1_pasos.qmd#generar-un-repositorio).

Una vez con nuestro repositorio podemos iniciar nuestro repositorio de forma local y vamos a modificarlo para hacer el módulo de ejemplo.

## Entendiendo un poco más

En la [sección anterior](../Tema2_extendido/1_pasos.qmd#generar-un-modulo) vimos como construir un ejemplo básico, en este vamos a generar un módulo que de una biblioteca simple donde contenga ejemplos básicos de distintos tipos algoritmos de clusterizaciones.

Entendamos la estructura que utiliza nbdev para generar nuestra nueva biblioteca, en el directorio nbs se encuentran los notebooks que vamos a modificar en nuestro ejemplo. Al momento de creación del repositorio una vez ejecutado el comando nbdev\_new nos genera toda la estructura dentro de nuestro repositorio.

Si no se hizo de la misma forma que como se describio anteriormente, esto se puede hacer de forma manual, se ejecuta el mismo comando y se rellenan los campos solicitados

> nbdev\_new

|  |
| --- |
| Figure 1: Salida nbdev\_new |

En el archivo settings.ini podemos ver las configuraciones, como ejemplo podemos ver que el archivo recien creado en el repositorio muestra los valores que se insertaron.

|  |
| --- |
| Figure 2: contenido de settings.ini |

En este archivo se puede modificar cosas como la licencia, la url de la documentación, donde se aloja la documentación, el autor, el nombre de la biblioteca, la ruta en donde se localizan los notebooks, etc.

Como ya hemos visto para modificar algo en nuestra biblioteca es necesario hacerlo en los notebooks, con este proposito, nbdev generará los archivos correspondientes tanto de la documentación, como los archivos con el código que corresponden a los módulos de nuestra biblioteca.

También nbdev generará el archivo README.md el cual sirve como portada en nuestro repositorio. Para hace modificaciones a este, se deben hacer desde el archivo nbs/index.ipynb el cual nos permitirá una vez terminada nuestra biblioteca dar instrucciones simples para su instalación así como la posibilidad de añadir links a la documentación de nuestra biblioteca. Se puede generar únicamente el archivo README.md ejecutando el comando nbdev\_readme, una vez ejecutado se puede abrir el archivo y ver las modificaciones realizadas.

nbdev\_readme

Si se desea ver desde github sera necesario actualizar los cambios en el repositorio, para esto haremos un ejemplo desde la terminal sólo se muestra como ejemplo pues un curso completo de git queda fuera de los temas de este curso.

git add README.md  
git commit -c "Se modifica el README"  
git push

* git add README.md añade el archivo que se modifico y que se desea que actualizar, o bien añadir el archivo si no se tiene dentro de nuesto conjunto de archivos vigilados.
* git commit -c "Se modifica el README" se “comitea” es decir se añade a un registro de cambios de forma local los cambios hechos. -c es para añadir el comentario "Se modifica el README" a los cambios hechos, esto nos sirve como guía para ver que cambios se han hecho y cual fue la razon de nuestros cambios. Esto es muy útil cuando se busca errores en el desarrollo y poder regresar al estado anterior.
* git push Sube los cambios hechos a el servidor de git, en nuestro caso lo va a subir a github pues el es servicio que determinamos usar.

Podemos ver las modificaciones en nuestro repositorio.

### Otros comandos git

Otros comandos importantes de git

* git clone sirve poara clonar repositorios
* git pull sirve para traer los cambios desde el servidor de git. El servidor depende de donde se tiene el servicio (github, gitlab o similares).
* git config configura distintos valores de el repositorio.
* git config –global user.name “Tu nombre” Cambia el valor de nombre de usuario de forma global.
* git config –global user.email “tumail@example.com” Cambia el valor global del mail.
* git init Crea un nuevo repositorio.
* git commit -a “Comitea” todos los archivos añadidos anteriormente al repositorio.
* git status Enlista los archivos que se han cambiado dentro y aquellos que todavia se necesitan añadir o “comitear”.
* git checkout -b <nombre\_rama> crea una nueva rama en el repositorio.
* git checkout <nombre\_rama>cambia a la rama .
* git branch Enlista todas las ramas en el repositorio y en que rama te encuentras actualmente.
* git branch -d <nombre\_rama> Borra la rama .

# Regresar al ejemplo

Ya comprendimos un poco más como es la estructura que genera nbdev, ahora vamos hacer un ejemplo y para generar nuevos módulos y como se hace el manejo de las dependencias y otras herramientas que estan disponibles en nbdev.

Dentro del directorio nbs se encuentra el archivo 00\_core.ipynb cambiemos el nombre de dicho archivo a 00\_cluster.ipynb. Podemos abrir el archivo y como lo hicimos anteriormente este archivo nos servira para generar tanto el código como la documentación de la bibloteca.

En la primera linea de código podemos observar que se tiene una directriz default\_exp la cual nos sirve para determinar a que módulo se exportaran lo que se genere dentro del notebook. Lo modificamos para que el modulo a exportar se llame cluster\_ejemplo.

#| default\_exp cluster\_ejemplo

Podemos eliminar todas las demás celdas exepto la última. Esta última es la que indica celda es la que genera el módulo.

#| hide  
import nbdev; nbdev.nbdev\_export()

Vamos a modificar las cosas para hacer nuestra biblioteca. Añadimos el siguiente contenido a una nueva celda.

#| hide  
#| export  
from nbdev.showdoc import \*  
import matplotlib  
import sklearn  
import random   
import numpy as np  
import shapely  
from sklearn.cluster import DBSCAN, HDBSCAN, OPTICS, KMeans  
import matplotlib.pyplot as plt

La directriz #| hide esconde la celda al momento de generar un html, es especialmente útil para hacer la documentación mientras que la directriz #| export va a exportar a nuestro módulo lo que se encuentre en esta celda.

Añadimos un objeto para generar datos de forma aleatoria

#| export  
class data\_points:  
 """Una clase que contiene a los datos"""  
 def \_\_init\_\_(self, n, min\_x = 0, min\_y = 0, max\_x= 1, max\_y=1, seed= None):  
 if seed != None:  
 random.seed(seed)  
   
 self.Points = np.random.uniform(low= min\_x, high= max\_x, size=(n, 2));  
 self.Points = [shapely.Point(x[0],x[1]) for x in self.Points]  
  
   
 def get\_points(self):  
 """Una funcion que para obtener los puntos"""  
 return self.Points  
  
 def get\_Multypoint(self):  
 """Regresa un objeto MultiPoint con los puntos"""  
 return shapely.geometry.MultiPoint(self.Points)  
   
 def get\_X(self):  
 """Regresa las coordenadas X"""  
 return [x.x for x in self.Points]  
  
 def get\_Y(self):  
 """Regresa las coordenadas X"""  
 return [x.x for x in self.Points]  
  
 def centroid(self):  
 """Regresa el centroide de los puntos """  
 return shapely.centroid(self.get\_Multypoint())

Esta es una clase simple que nos sirve de ejemplo para ver algunas de las funcionalidades de nbdev. Cuando se contruye la documentación de forma automática nos genera una documentación simple como se muestra en la imagen [Figure 3](#fig-3-W5-nbdev_docu).

|  |
| --- |
| Figure 3: Ejemplo de documentacion |

Todavía no vamos a hacer la documentación, antes de hacerlo vamos a añadir ciertas cosas. Se añaden las siguientes celdas como se puede observar ninguna de las celdas define nada nuevo simplemente se inicializa el objeto y se hace uso de sus métodos internos.

#| hide  
datos\_simples = data\_points(40)

datos\_simples.get\_points()

datos\_simples.get\_points()

datos\_simples.get\_Multypoint()

datos\_simples.get\_X()

datos\_simples.get\_Y()

Todas estas celdas nos serviran para que hacer pruebas y comprobar las funcionalidades, en este caso estamos probando que los métodos funcionen de forma correcta.

Incluso se pueden hacer pruebas más complicadas, tomemos como ejemplo el código de la siguientes celda.

if len(datos\_simples.get\_X()) != len(datos\_simples.get\_Y()):  
 raise Exception("Si hay problema la evaluacion ")

Este código generará un error si el número de elementos fuesen distintos, es claro que la clase como se encuentra implementada siempre pasara esta prueba. Las pruebas tienen que ser pensadas para que la idea las pasen sin importar como se hizo la implementación, como es el caso de la prueba anterior. Dentro del desarrollo podriamos cambiar la forma en la cual se generan los puntos y almacenar estos como si fueran dos lista, donde cada lista almacena los datos de cada coordenada, esto dependería de cada desarrollador. Si se tiene un error en la generación de estas lista, como puede ser simplemente generar la lista pero no llenarla con datos, el tamaño seria distinto y no pasaria la prueba. Para ver si el código dentro de los notebooks que generan de nuestros módulos pasan las pruebas se ejecuta la linea de comandos nbdev\_test.

nbdev\_test

lo cual mostrará

|  |
| --- |
| Figure 4: Exito pruebas |

## Clase clusters

Vamos añadir una nueva clase a nuestro módulo, la cual llamaremos Clusters, esta clase nos sirve para generar clusters usando distintos algoritmos.

class data\_points:  
 """Una clase que contiene a los datos"""  
 def \_\_init\_\_(self, n, min\_x = 0, min\_y = 0, max\_x= 1, max\_y=1, seed= None):  
 if seed != None:  
 random.seed(seed)  
  
 Xpoint = np.random.uniform(low= min\_x, high= max\_x, size=n)  
 Ypoint = np.random.uniform(low= min\_y, high= max\_y, size=n)  
   
 #self.Points = np.random.uniform(low= min\_x, high= max\_x, size=(n, 2))  
 self.Points = [shapely.Point(x[0],x[1]) for x in zip(Xpoint, Ypoint)]  
   
 @classmethod  
 def from\_list(cls, list\_points):  
 """Se genera los datos usando una lista de puntos"""  
 toto = cls(0)  
 toto.Points = list\_points  
 return toto  
   
 def get\_points(self):  
 """Una funcion que para obtener los puntos"""  
 return self.Points  
  
 def get\_Multypoint(self):  
 """Regresa un objeto MultiPoint con los puntos"""  
 return shapely.geometry.MultiPoint(self.Points)  
   
 def get\_X(self):  
 """Regresa las coordenadas X"""  
 return [x.x for x in self.Points]  
  
 def get\_Y(self):  
 """Regresa las coordenadas X"""  
 return [x.y for x in self.Points]  
   
 def centroid(self):  
 """Regresa el centroide de los puntos """  
 return shapely.centroid(self.get\_Multypoint())  
   
 def as\_array(self):  
 """Regresa los puntos como un array"""  
 list\_arrays = np.array([ (geom.xy[0][0], geom.xy[1][0]) for geom in self.Points])  
 return list\_arrays  
  
 def add\_points(self, data\_points):  
 """Agrega puntos de otra """  
 self.Points = self.Points + data\_points.Points  
 return

Para exportar la clase al módulo es necesario añadir la directriz #| export al inicio de la celda. Adicionalmente añadimos pruebas simples usando los datos que ya se generaron.

cluster\_all = Clusters(datos\_simples)

cluster\_all.DBSCAN()

cluster\_all.KMeans()

cluster\_all.HDBSCAN()

cluster\_all.OPTICS()

Como estas celdas nos sirven de pruebas vamos a añadir la directriz #| hide para evitar que estas sean parte de la documentación. Ahora podemos añadir ejemplos del uso de la clase cluster e incluso añadir como se usan los métodos.

## Documentación

Para generar la documentación se hace usando el comando

nbdev\_docs

En este caso, en la terminal podemos observar lo que se muestra en la imagen [Figure 5](#fig-5-W5-nbdev_docs), como se observa en el directorio \_docs se encuentra un archivo html que puede abrirse usando un navegador y se puede ver la documentación y se puede ver que solo muestra el ‘docstring’ generado por las clase como en la figura [Figure 3](#fig-3-W5-nbdev_docu) y el respectivo para la clase Clusters.

|  |
| --- |
| Figure 5: Generación documentación |

Para mostrar un poco más lo que hace nuestra clase simple añadimos las siguientes celdas

show\_doc(Clusters.KMeans)

show\_doc(Clusters.DBSCAN)

show\_doc(Clusters.HDBSCAN)

show\_doc(Clusters.OPTICS)

Estas mostraran lo que se haya puesto dentro de los ‘docstrings’ de los métodos en la clase Cluster.

También podemos generar la documentación y hacer una previsualización haciendo uso del comando

nbdev\_preview

lo cual nos mostrara

|  |
| --- |
| Figure 6: Muestra la documentación |

llevandonos a la documentación de forma local.

|  |
| --- |
| Actividad |
| Muestre los métodos para la clase data\_points en la documentación |

## Mas documentación

Podemos hacer uso de las facilidades de los jupyter notebooks para generar ejemplos y tutoriales, para esto vamos hacer dentro del mismo notebook.

Creamos un conjunto de datos usando los distintos metodos en las clases que ya hicimos.

datos\_simples\_3 = data\_points(500, min\_x = 0.6, min\_y= 0.6, max\_x= 1.0, max\_y= 1.0 )  
datos\_simples\_2 = data\_points(500, min\_x = 0.0, min\_y= 0.6, max\_x= 0.4, max\_y= 1.0 )  
datos\_simples\_1 = data\_points(500, min\_x = 0.6, min\_y= 0.0, max\_x= 1.0, max\_y= 0.4 )  
datos\_simples\_0 = data\_points(500, min\_x = 0.0, min\_y= 0.0, max\_x= 0.4, max\_y= 0.4 )

Unimos todos los puntos con el metodo add\_points de la clase data\_points.

datos\_simples\_0.add\_points(datos\_simples\_1)  
datos\_simples\_0.add\_points(datos\_simples\_2)  
datos\_simples\_0.add\_points(datos\_simples\_3)

Veamos los puntos dentro de nuestra estructura.

datos\_simples\_0.get\_Multypoint()

Ahora inicialicemos nuestro objeto Clusters

cluster\_all = Clusters(datos\_simples\_0)

Y ahora podemos hacer uso de los algoritmos usando los metodos de las distintas las clusterizaciones.

cluster\_all.OPTICS()  
cluster\_all.HDBSCAN()  
cluster\_all.DBSCAN(eps=0.1, min\_samples=10)  
cluster\_all.KMeans()

Y visualizemos las clusterizaciones

fig, ax = plt.subplots(1,1, figsize=(6,6))  
plt.scatter(  
 cluster\_all.data\_structure.get\_X(),  
 cluster\_all.data\_structure.get\_Y(),  
 c = cluster\_all.data\_structure.classes\_Kmeans  
 )

fig, ax = plt.subplots(1,1, figsize=(6,6))  
plt.scatter(  
 cluster\_all.data\_structure.get\_X(),  
 cluster\_all.data\_structure.get\_Y(),  
 c = cluster\_all.data\_structure.classes\_DBSCAN  
 )

fig, ax = plt.subplots(1,1, figsize=(6,6))  
plt.scatter(  
 cluster\_all.data\_structure.get\_X(),  
 cluster\_all.data\_structure.get\_Y(),  
 c = cluster\_all.data\_structure.classes\_HDBSCAN  
 )

fig, ax = plt.subplots(1,1, figsize=(6,6))  
plt.scatter(  
 cluster\_all.data\_structure.get\_X(),  
 cluster\_all.data\_structure.get\_Y(),  
 c = cluster\_all.data\_structure.classes\_OPTICS  
 )

Si se ejecuta nbdev\_preview o nbdev\_docs en la documentación mostrara el ejemplo dentro de la misma.

Ahora para preparar verificar que todo esta bien ejecutamos

nbdev\_prepare

Lo cual mostrara  si no hay ningun problema. En el fondo lo que hace nbdev\_prepare preprocesar los archivos dentro de nbs y generar los

# Publicar todo

Vamos a entender un poco más que es lo que pasa cuando se hace la publicación de la documentación generada. Ya hemos dicho que por abajo nbdev usa *Quarto* para la generación de la documentación, lo que sucede es que nuestros notebooks son preprocesados generando notebooks que se usan como base para convertirlos a documentos html. Estos notebooks se encuentran en el directorio \_proc (se genera cuando se executa nbdev\_docs) los cuales tienen en sus celdas directrizes de *Quarto* para generar los resultados deseados. Podemos inspeccionar estos archivos comprobando observar que son similares a los notebook originales, si se inspecciona el directorio \_docs se vera que los documentos html son los que fueron generados a partir de los notebooks en \_prod.

De esta forma hemos generado todo para nuestra biblioteca, tiene documentación y haciendo uso de nbdev\_prepare se generan los módulos para poder instalar. Esto nos sirve si se desea publicar la documentación en un servicio distinto a *github*.

A continuación vamos a integrar todo en *Github* para que tanto las pruebas como la creación de la documentación realizadas usando [github-actions](https://docs.github.com/en/actions) como ya se hizo en los [primeros pasos de este curso](../Tema2_extendido/1_pasos.qmd#habilitar-github-pages-y-flujos-de-trabajo) con la creación de una biblioteca muy sencilla.

Como hemos añadido bibliotecas y hemos modificado las cosas si intentamos seguir los pasos descritos con anterioridad ni las prueba ni la documentación podran se realizadas de forma correcta. Por tal motivo vamos a dar una explicación más extensa de los pasos.

## Entendiendo github-actions.

Las acciones de Github acitions nos permiten realizar acciones automatizadas dentro de nuestro directorio de Github, en nuestro caso dichas acciones se encuentran dentro de los archivos que se encuentran en el directorio .github dentro del directorio workflows los cuales son deploy.yaml y test.yaml. Estos dos archivos controlan los dos flujos de trabajo que se realizan dentro de github. Vamos a modificarlos y explicad un poco que hacen cada uno.

El archivo deploy es el que utilizamos para generar la documentación, vamos a sustituir el contenido del archivo con el que se encuentra en la celda siguiente:

### Archivo deploy.yaml

name: Deploy to GitHub Pages  
on: [workflow\_dispatch, pull\_request, push]   
permissions:  
 contents: write  
jobs:  
 deploy:  
 runs-on: ubuntu-latest  
 steps:   
 - uses: actions/checkout@v3  
 - uses: actions/setup-python@v4  
 - name: Install Dependencies and create documentation   
 shell: bash  
 run: |  
 python -m pip install --upgrade pip  
 pip install -Uq git+https://github.com/fastai/ghapi.git # you need this for enabling pages  
 pip install -Uq git+https://github.com/fastai/fastcore.git  
 pip install -Uq git+https://github.com/fastai/execnb.git  
 pip install -U git+https://github.com/fastai/nbdev.git  
 wget -q $(curl https://latest.fast.ai/pre/quarto-dev/quarto-cli/linux-amd64.deb)  
 sudo dpkg -i quarto\*.deb  
 pip install -Uq matplotlib  
 pip install -Uq setuptools  
 pip install -Uq scikit-learn  
 pip install -Uq shapely  
 nbdev\_docs  
# pip install -e ".[dev]"  
   
 - name: Deploy to GitHub Pages  
 uses: peaceiris/actions-gh-pages@v4  
 with:  
 github\_token: ${{ secrets.GITHUB\_TOKEN }}  
 force\_orphan: true  
 publish\_dir: ./\_docs  
 # The following lines assign commit authorship to the official GH-Actions bot for deploys to `gh-pages` branch.  
 # You can swap them out with your own user credentials.  
 user\_name: github-actions[bot]  
 user\_email: 41898282+github-actions[bot]@users.noreply.github.com

Expliquemos a grandes rasgos lo que esta haciendo este archivo. Las primeras lineas le dan nombre a la acción y cuando se realiza esta. En este caso llamamos a la acción ‘Deploy to GitHub Pages’ y la acción será realizada cuando se solicite de forma manual (‘workflow\_dispatch’) cuando se haga una petición pull (‘pull\_request’) o bien quando se realize un push.

name: Deploy to GitHub Pages  
on: [workflow\_dispatch, pull\_request, push]

El trabajo que va a realizar se denomina ‘deploy’ el cual utilizará una máquina virtual con la versión más reciente de ubuntu (‘ubuntu-latest’).

En las siguientes lineas estamos dando permiso a la maquina virtual de poder escribir dentro de nuestro repositorio.

permissions:  
 contents: write

Despues el trabajo consiste en los siguientes pasos:

- uses: actions/checkout@v3  
- uses: actions/setup-python@v4

Utiliza las acciones actions/checkout@v3 y actions/setup-python@v4 que son acciones que tenemos estandarizadas dentro de github actions. Las cuales hace un check out del repositorio dentro de la maquina virtual para realizar el trabajo. actions/setup-python@v4 instala python dentro de la maquina virtual.

El siguiente paso lo llamamo ‘Install Dependencies and create documentation’ aquí podemos ver que hacemos un conjunto de acciones que hicimos con anterioridad cuando instalamos las cosas necesarias para hacer uso de nbdev, git, quarto, etc.

- name: Install Dependencies and create documentation   
 shell: bash  
 run: |  
 python -m pip install --upgrade pip  
 pip install -Uq git+https://github.com/fastai/ghapi.git # you need this for enabling pages  
 pip install -Uq git+https://github.com/fastai/fastcore.git  
 pip install -Uq git+https://github.com/fastai/execnb.git  
 pip install -U git+https://github.com/fastai/nbdev.git  
 wget -q $(curl https://latest.fast.ai/pre/quarto-dev/quarto-cli/linux-amd64.deb)  
 sudo dpkg -i quarto\*.deb  
 pip install -Uq matplotlib  
 pip install -Uq setuptools  
 pip install -Uq scikit-learn  
 pip install -Uq shapely  
 nbdev\_docs

Pongamos especial atención en las últimas lineas, aquí instalamos las dependencias que se utilizaron en hacer nuestra biblioteca y son necesarias para que los notebooks que hicimos puedan correr. Una vez instaladas las dependencias podemos generar la documentación con ‘nbdev\_docs’.

Como en la maquina virtual ya generamos nuestra documentación ahora vamos a ponerla dentro de un nuevo ‘branch’, por eso le tenemos que dar permiso para escribir, dentro del repositorio y usar ese ‘branch’ para publicar la documentación usando el siguiente paso:

- name: Deploy to GitHub Pages  
 uses: peaceiris/actions-gh-pages@v4  
 with:  
 github\_token: ${{ secrets.GITHUB\_TOKEN }}  
 force\_orphan: true  
 publish\_dir: ./\_docs  
 # The following lines assign commit authorship to the official GH-Actions bot for deploys to `gh-pages` branch.  
 # You can swap them out with your own user credentials.  
 user\_name: github-actions[bot]  
 user\_email: 41898282+github-actions[bot]@users.noreply.github.com

Esto usa las acciones descritas en peaceiris/actions-gh-pages@v4 las cuales ya estan automatizadas para generar un nuevo branch con nombre ‘gh-pages’ y añadir el contenido de la documentación dentro de el branch. Lo que se añade dentro del branch es lo que se encuentra dentro del directorio ’./\_docs’. ‘github\_token’ nos permite autenticar el trabajo realizado por el flujo de trabajo. Los parámetros de ‘user\_name’ y ‘user\_email’ permiten avisar que acciones fueron realizadas por el script ‘peaceiris/actions-gh-pages@v4’ y tendran como las cuales apareceran que fueron realizadas por github-pages bot. Estas acciones nos enviaran un mail con el resultado del trabajo realizado, es decir si este fue exitoso o no.

### Archivo test.yaml

En el caso de las pruebas estas se hacen en las acciones en el archivo ‘test.yaml’ y vamos a remplazar su contenido por el código siguiente:

name: Test CI  
on: [workflow\_dispatch, pull\_request, push]  
jobs:  
 test:  
 runs-on: ubuntu-latest  
 steps:  
 - uses: actions/checkout@v3  
 - uses: actions/setup-python@v4  
 - name: Install Nbdev Dependencies  
 shell: bash  
 run: |  
 python -m pip install --upgrade pip  
 pip install -Uq git+https://github.com/fastai/ghapi.git # you need this for enabling pages  
 pip install -Uq git+https://github.com/fastai/fastcore.git  
 pip install -Uq git+https://github.com/fastai/execnb.git  
 pip install -U git+https://github.com/fastai/nbdev.git  
 - name: Install package dependencies  
 shell: bash  
 run: |  
 python -m pip install -Uq matplotlib  
 python -m pip install -Uq setuptools  
 python -m pip install -Uq scikit-learn  
 python -m pip install -Uq shapely  
  
 - name: Test the package  
 shell: bash  
 run: |  
 echo "Check we are starting with clean git checkout"  
 if [[ `git status --porcelain -uno` ]]; then  
 git diff  
 echo "git status is not clean"  
 false  
 fi  
 echo "Trying to strip out notebooks"  
 nbdev\_clean  
 echo "Check that strip out was unnecessary"  
 git status -s # display the status to see which nbs need cleaning up  
 if [[ `git status --porcelain -uno` ]]; then  
 git status -uno  
 echo -e "!!! Detected unstripped out notebooks\n!!!Remember to run nbdev\_install\_hooks"  
 echo -e "This error can also happen if you are using an older version of nbdev relative to what is in CI. Please try to upgrade nbdev with the command `pip install -U nbdev`"  
 false  
 fi  
 nbdev\_export  
 if [[ `git status --porcelain -uno` ]]; then  
 echo "::error::Notebooks and library are not in sync. Please run nbdev\_export."  
 git status -uno  
 git diff  
 exit 1;  
 fi  
 if [ ! $SKIP\_TEST ]; then  
 nbdev\_test --flags "$FLAGS"  
 fi

De forma similar el nombre del trabajo es ‘Test CI’ los trabajos se realizaran cuando cuando se hace una petición de forma manual o se hace un pull o un push.

Se corre en una maquina virtual con ubuntu ‘runs-on: ubuntu-latest’ y hace un checkout, se instala python y nbdev junto con sus dependencias.

En el siguiente paso se instalan las dependencias de la biblioteca en la maquina virtual.

- name: Install package dependencies  
 shell: bash  
 run: |  
 python -m pip install -Uq matplotlib  
 python -m pip install -Uq setuptools  
 python -m pip install -Uq scikit-learn  
 python -m pip install -Uq shapely

Y en lo que resta del código verifica si estas cosas y hace las pruebas usando nbdev\_test dentro de la maquina virtual

* git se verifica el contenido de la versión del repositorio que se encuentra en la máquina virtual
* Verifica que los notebooks no tengan cosas inecesarias, si no es así marca error
* Construye la biblioteca dentro de la máquina virtual usando ‘nbdev\_export’
* Hace las pruebas usando ‘nbdev\_test’

name: Test the package  
 shell: bash  
 run: |  
 echo "Check we are starting with clean git checkout"  
 if [[ `git status --porcelain -uno` ]]; then  
 git diff  
 echo "git status is not clean"  
 false  
 fi  
 echo "Trying to strip out notebooks"  
 nbdev\_clean  
 echo "Check that strip out was unnecessary"  
 git status -s # display the status to see which nbs need cleaning up  
 if [[ `git status --porcelain -uno` ]]; then  
 git status -uno  
 echo -e "!!! Detected unstripped out notebooks\n!!!Remember to run nbdev\_install\_hooks"  
 echo -e "This error can also happen if you are using an older version of nbdev relative to what is in CI. Please try to upgrade nbdev with the command `pip install -U nbdev`"  
 false  
 fi  
 nbdev\_export  
 if [[ `git status --porcelain -uno` ]]; then  
 echo "::error::Notebooks and library are not in sync. Please run nbdev\_export."  
 git status -uno  
 git diff  
 exit 1;  
 fi  
 if [ ! $SKIP\_TEST ]; then  
 nbdev\_test --flags "$FLAGS"  
 fi

Como se observa, los pasos anteriores son los que ya hemos desde la instalación y la construcción de la biblioteca.

Con los archivos que hemos modificado hemos dado las instrucciónes para que estas acciones se desarrollen en máquinas virtuales en servidores de github. Si todo se encuentra en orden podemos salvar los archivos, hacer un commit, y hacer un push. Como las acciones se ejecuntan cuando se hace un push podemos ver en las acciones de github e ir a la página de github para verificar que el flujo de trabajo funciona de forma adecuada. En el menu Actions del repositrio.

|  |
| --- |
| Figure 7: Menu github actions |

En este menu veremos las aciones que se llevan del lado izquierdoo acabo en nuestro caso:

* Deploy to Github Pages
* Test CI
* pages-build-deployment (ejecutado al hacer el branch ‘gh-pages’)

En el centro una lista con los commits realizado y el resultado de las acciones que se llevaron acabo y si estas tuvieron exito (✅) o no (❌).

|  |
| --- |
| Figure 8: Woorkflows Actions |

Para habilitar nuestra pagina de documentación se tienen que seguir los mismos pasoes que vimos [anteriormente](../Tema2_extendido/1_pasos.qmd#ver-la-documentación) y se selecciona el branch de ‘gh-pages’ para habilitar la documentación.

# Comentarios finales

Como hemos visto nbdev tiene muchas funcionalidades, esto nos permite tener una integración continua para generar tanto nuestros módulos como la documentación de los mismo sin hacer mucho esfuerzo extra.