

Protocolo automático para el tratamiento de imágenes Landsat

MANUAL

Diego García Díaz | Laboratorio de SIG y Teledetección. Estación Biológica de Doñana
(CSIC) | Versión 1 | 02 de noviembre de 2015

1. Prefacio y Requisitos

El protocolo que se detalla a continuación es la continuación del protocolo manual que se ha visto haciendo en el LAST desde 2004. Ese protocolo manual (que se encuentra perfectamente detallado en su propio manual), estaba basado en los softwares ENVI, Idrisi, Miramón y Access como base de datos.

La versión automática que nos ocupa ha prescindido de los *softwares* ENVI e Idrisi y ha sustituido Access por MongoDB como base de datos. En este caso la base de datos también ha cambiado en cuanto a su diseño, tema del que nos ocuparemos más adelante.

La filosofía que se ha seguido es la de tratar de prescindir al máximo de cualquier programa y cuando fuera necesario hacerlo con *software* libre. Miramón se ha seguido usando por la gestión de metadatos tan completa que realiza y porqué la corrección radiométrica de las imágenes se hace con su módulo **Corrad**. Respecto al resto de funciones necesarias que se realizaban con ENVI e Idrisi han sido sustituidas por distintas librerías de Python (fundamentalmente [GDAL](#), [Rasterio](#), [Numpy](#) y [Scipy](#)), que es el lenguaje de programación en el que está realizado el protocolo.

Un elemento nuevo incorporado a este protocolo es el uso de [Fmask](#)¹ como máscara de nubes. Esta máscara en principio será la empleada por el USGS como sustitución de la máscara de calidad que viene actualmente con las escenas Landsat 8, y que previsiblemente se añadirá a escenas antiguas, ya que este algoritmo también es aplicable para Landsat 4, 5 y 7.



¹ Zhu, Z. and Woodcock, C. E., Improvement and Expansion of the Fmask Algorithm: Cloud, Cloud Shadow, and Snow Detection for Landsats 4-7, 8, and Sentinel 2 Images, Remote Sensing of Environment 152 (2014) 217-234

1.1 Antes de comenzar

1.1.1 Anaconda e instalación de módulos en python

La versión de python sobre la que está instalada el protocolo es [Anaconda](#) (en su versión Miniconda). Esta versión es la que está configurada como la versión por defecto de Python en el sistema. Durante la instalación ya nos da la opción de definir esto como definir la variable de sistema PYTHONPATH. En la práctica esto significa que para instalar módulos de Python podemos hacer desde el cmd indicando “conda install + nombre del modulo”. La inmensa mayoría de los modulos están accesibles para conda, si algún modulo no lo estuviera podría hacerse directamente con las setuptools de Python vía “pip install + nombre del modulo” también desde el cmd.

La mayoría de los módulos no dan ningún problema durante su instalación, sin embargo, GDAL sí suele dar bastantes problemas durante su instalación por la compatibilidad con otras librerías de Python (en Windows, en Linux va todo como la seda!). Hay varios tutoriales sobre como instalar GDAL en Windows, pero lo más sencillo es usar el Wheel disponible en este [link](#)². En general lo más cómodo para evitar problemas de compatibilidad, será descargar tanto GDAL como rasterio de ese enlace e instalarlos mediante el comando “pip install + path + nombre del Wheel”.

En cualquier caso, en la máquina virtual desde la que correrá el Protocolo ya está todo instalado y configurado y solo hay que iniciar el Ipython notebook, lo cual haremos abriendo una ventana de comandos y escribiendo ipython notebook, lo que nos iniciará un servidor de Python vía localhost que se abrirá en nuestro navegador preferido en puerto 8888 (<http://localhost:8888/tree>).

1.1.2 MongoDB

El protocolo según va ejecutando procesos va guardando distintos parámetros en una base de datos MongoDB. Esta base de datos se va llamando y escribiendo durante los distintos procesos del Protocolo. Por tanto, antes de empezar a ejecutar el Protocolo debemos de activar el servidor de la base de datos escribiendo “mongod” en una ventana de comandos. Previamente hemos tenido que añadir al path del sistema la ruta a los archivos ejecutables de MongoDB, que por defecto será: C:\Program Files\MongoDB\Server\3.0\bin

1.1.3 La máquina virtual

El protocolo está instalado en la máquina virtual con IP: 198.162.44.220. Los datos de acceso son: Usuario: “Diego”; Contraseña: “LASTØ1”.

Se trata de una máquina con un procesador i7, 8 gigas de RAM y 180 gigas de disco duro. Actualmente se está en espera de que amplíen todos estos parámetros.

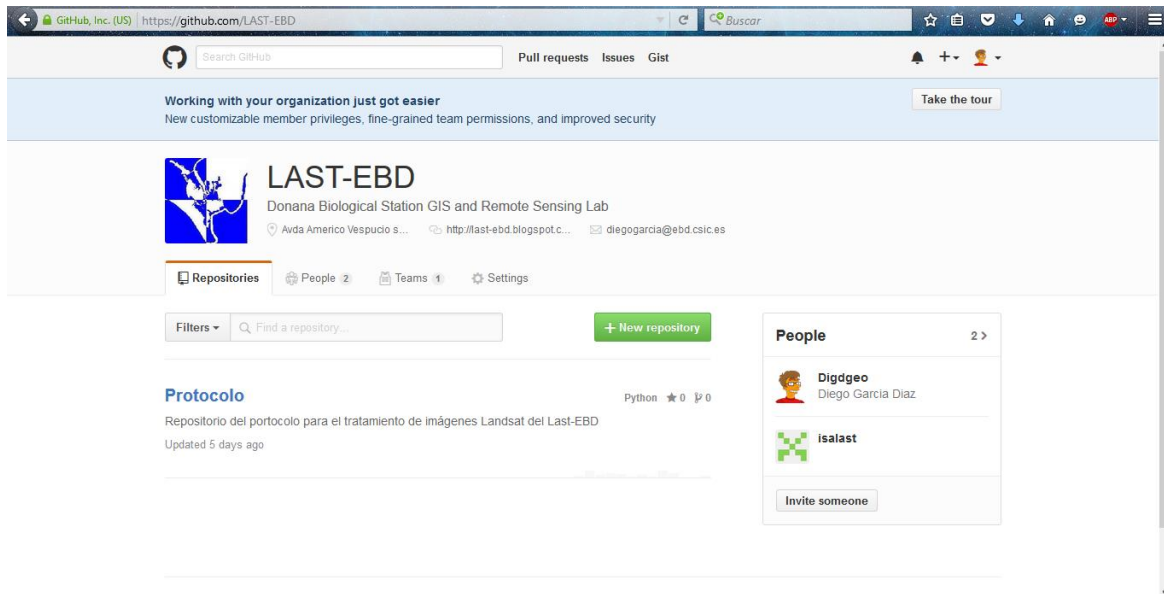
RESUMEN U-1:

El Protocolo automático está escrito en Python, concretamente en la versión Anaconda, utilizando las librerías GDAL y Rasterio para hacer algunos procesos y el software Miramon para otros. Esto hace que como pre-requisitos para la ejecución del Protocolo tengamos que iniciar un ipython notebook y el servidor de mongod. Y por supuesto, estar seguros de que tenemos Miramón y Fmask instalados en nuestro equipo (preferiblemente en las rutas: C:\MiraMon y C:\Fmask).

² <http://www.lfd.uci.edu/~gohlke/pythonlibs/#gdal>

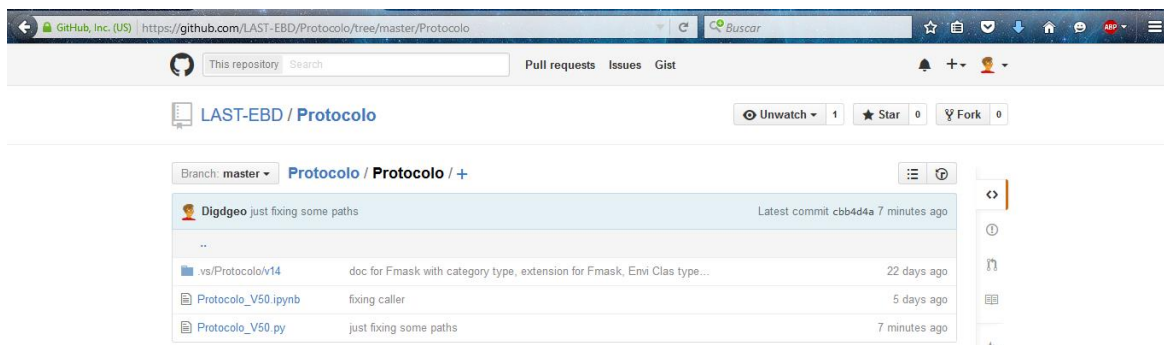
2. Repositorio del código en GitHub

Existe un repositorio del código subido a GitHub desde el cual se puede descargar el Protocolo completo: <https://github.com/LAST-EBD/Protocolo>. La responsable de la cuenta es Isabel Afan Asencio.

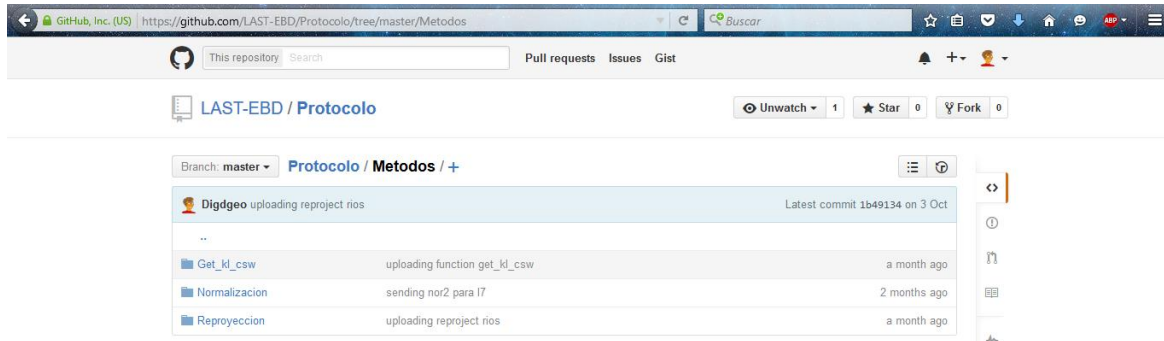


Dentro de la cuenta del LAST-EBD hay 3 carpetas \Protocolo, \Landsat_Scripts y \Métodos.

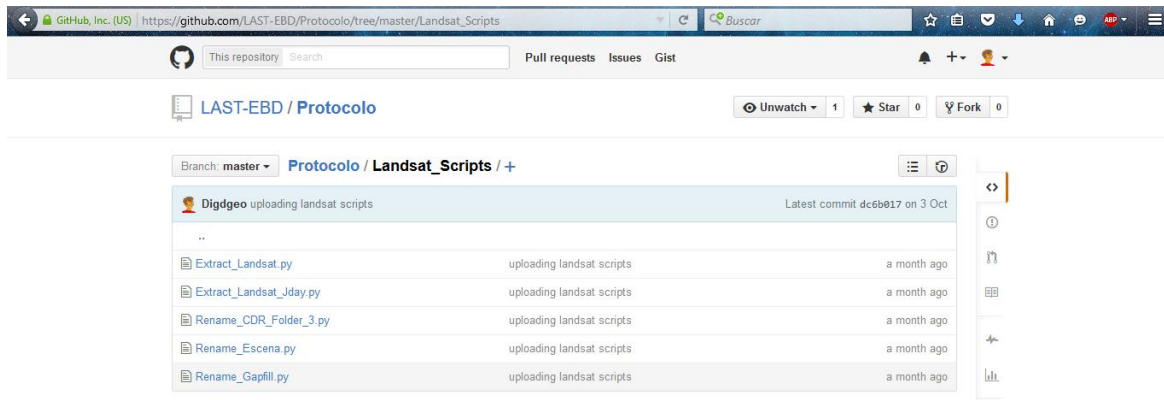
Dentro de la carpeta Protocolo tenemos 2 archivos llamados Protocolo (+v+el número de la versión con la que esté trabajando). Uno de ellos con extensión '.py' y otro con extensión '.ipynb' (extensión de los ipython notebooks).



Dentro de la carpeta Métodos, están guardados por separado los métodos de la clase como funciones, por si se quisieran ejecutar de ese modo por algún motivo.



Dentro de la carpeta Landsat_Scripts hay algunos de los scripts que se han ido realizando dentro de lo que es el trabajo con Landsat (para renombrar la escena del USGS_ID al Last_ID, para hacer el rename previo al Gapfill, para hacer el rename a las CDR, etc.)³



Este código ya está descargado en el equipo virtual, pero si se quisiera descargar desde otro equipo, lo primero que habría que hacer es instalar **Git** (hay una versión para windos de GitHub Desktop) y luego copiar la url que aparece en la pestaña lateral derecha de la página principal del repositorio.

³ Está la opción de crear otros repositorios del LAST-EBD para guardar otro tipo de trabajos que se desarrollen dentro de a labor de apoyo a personal de la EBD.



Luego, abrimos una ventana del cmd y escribimos “git clone + url”. Lo cual en cuestión de segundos, nos clonará el repositorio completo en nuestro equipo.

Si estamos editando el código desde el equipo en el que estemos para actualizar la versión alojada en GitHub debemos de abrir el **Git Shell** (ventana de comandos de Git) y escribir git add . -all (esto leerá los cambios que se han producido en el repositorio local, tanto dentro de los propios archivos (cambios en el código), como la creación o borrado de algunos archivos). Luego haremos un git commit -m ‘aquí iría un mensaje con la descripción que queremos que vaya asociada al cambio’ (por ejemplo git commit -m ‘fixing some paths’). Finalmente, para subir los cambios al repositorio web de GitHub teclearemos git push.

```
MINGW32/c/Users/Diego/Documents/GitHub/Protocolo
Welcome to Portable Git (version ghfw)

Run 'git help git' to display the help index.
Run 'git help <command>' to display help for specific commands.

Diego@DIEGO-PC ~/Documents/GitHub (master)
$ cd Protocolo

Diego@DIEGO-PC ~/Documents/GitHub/Protocolo (master)
$ git add . --all

Diego@DIEGO-PC ~/Documents/GitHub/Protocolo (master)
$ git commit -m 'no really changes'
On branch master
Your branch is up-to-date with 'origin/master'.

nothing to commit, working directory clean

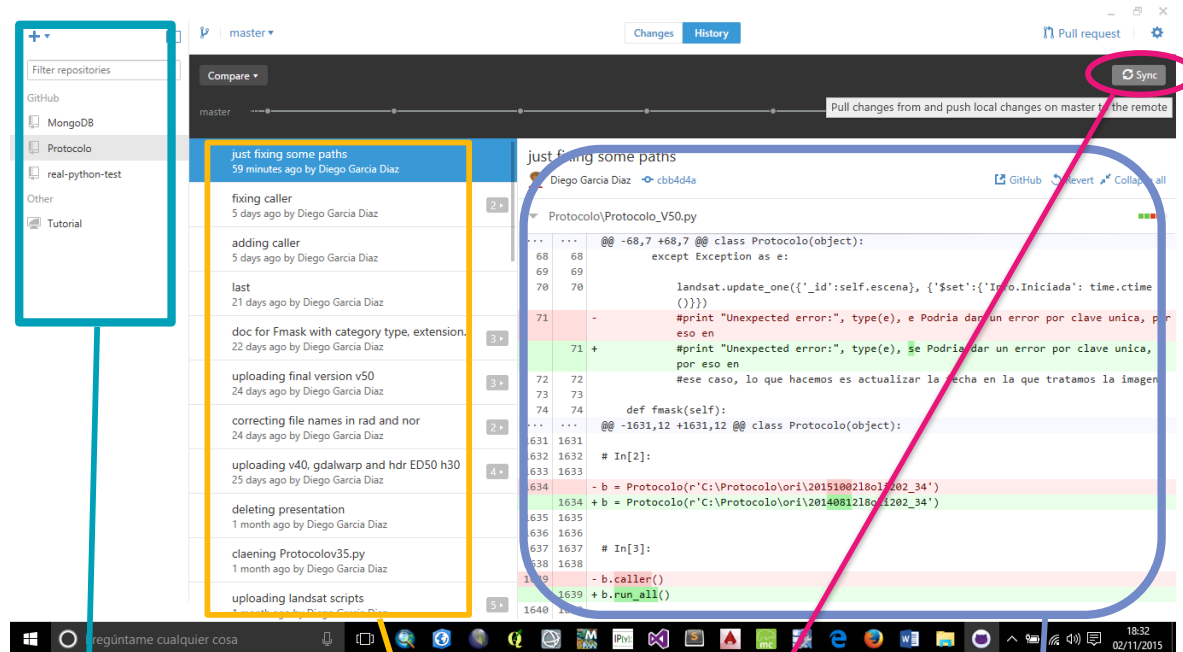
Diego@DIEGO-PC ~/Documents/GitHub/Protocolo (master)
$ git push
Everything up-to-date

Diego@DIEGO-PC ~/Documents/GitHub/Protocolo (master)
$
```

Si queremos sincronizar el repositorio local con el repositorio que haya en GitHub, que será el caso más corriente, lo que debemos de hacer es abrir el **Git Shell**, ir al directorio donde está el Protocolo y

escribir git pull. Con esto cualquier cambio que se hubiera realizado en el repositorio en red, se actualizaría también en nuestro equipo.

Existe también una versión de escritorio de GitHub para Windows que estará instalada en la máquina virtual y desde la que quizás resulte más amigable la interfaz para clonar un repositorio o mantenerlo sincronizado web/local.



Pestaña con los repositorios que tenemos sincronizados.

Desde el signo "+" podemos agregar nuevos repositorios.

Historial del archivo.
Cambios realizados, quien los realiza, fecha, etc.

En definitiva, la información provista en los commits realizados.

Botón para sincronizar, tanto desde local a web, como desde web a local.

Esta pestaña nos muestra los cambios realizados en el código en sus distintas versiones.

RESUMEN U-2:

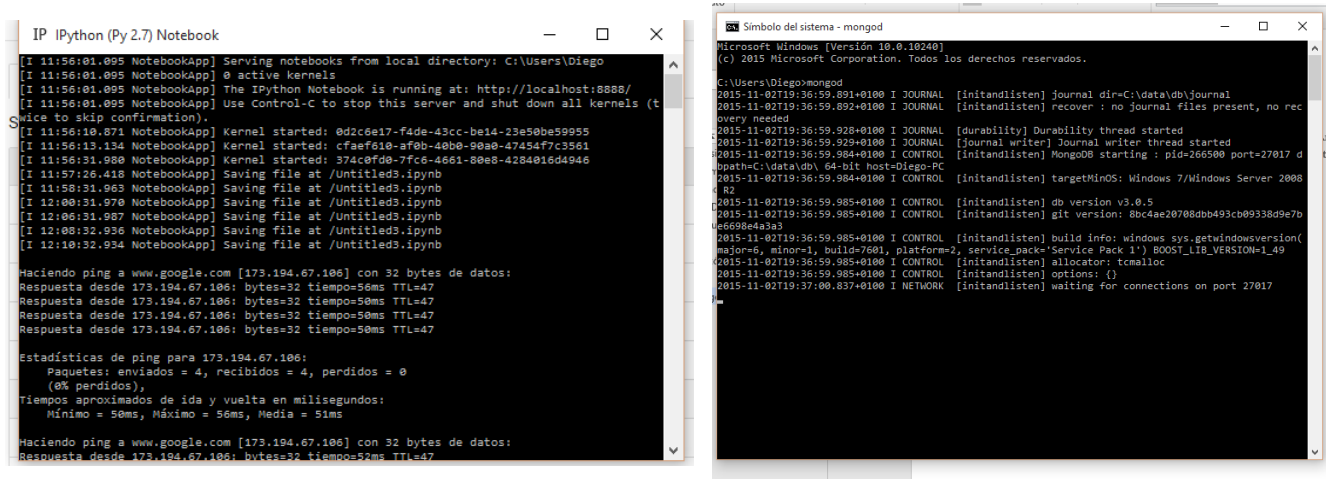
Tenemos disponible todo el código del Protocolo en GitHub, cualquier cambio que se haga por cualquier persona desde cualquier equipo se actualizará fácilmente. Para poder editar deben de tener permisos para ello, deben de estar incluidos en el repositorio como usuarios, pero el repositorio estará siempre en una cuenta de acceso público. En principio la licencia del Protocolo será del tipo *creative commons Attribution 4.0 International*.



3. Protocolo: Código y ejecución

3.1 Ejecución

Dando ya por supuesto que tenemos Fmask, Miramón, Anaconda y MongoDB instalados en el equipo, abrimos 2 cmds y en una iniciamos el ipython notebook y en la otra MongoDB.



```
[I 11:56:01.095 NotebookApp] Serving notebooks from local directory: C:\Users\Diego
[I 11:56:01.095 NotebookApp] 0 active kernels
[I 11:56:01.095 NotebookApp] The IPython Notebook is running at: http://localhost:8888/
[I 11:56:01.095 NotebookApp] Use Control-C to stop this server and shut down all kernels (t
Switch to skip confirmation).
[I 11:56:10.871 NotebookApp] Kernel started: 0d2c6e17-f4de-43cc-be14-23e50be59955
[I 11:56:13.134 NotebookApp] Kernel started: cfaef610-af0b-40b0-90a0-47454f7c3561
[I 11:56:31.980 NotebookApp] Kernel started: 374c0fd0-7fc6-4661-80e8-4284016d4946
[I 11:57:26.418 NotebookApp] Saving file at /Untitled3.ipynb
[I 11:58:31.963 NotebookApp] Saving file at /Untitled3.ipynb
[I 12:00:31.970 NotebookApp] Saving file at /Untitled3.ipynb
[I 12:00:31.987 NotebookApp] Saving file at /Untitled3.ipynb
[I 12:00:32.936 NotebookApp] Saving file at /Untitled3.ipynb
[I 12:10:32.934 NotebookApp] Saving file at /Untitled3.ipynb

Haciendo ping a www.google.com [173.194.67.106] con 32 bytes de datos:
Respuesta desde 173.194.67.106: bytes=32 tiempo=56ms TTL=47
Respuesta desde 173.194.67.106: bytes=32 tiempo=50ms TTL=47
Respuesta desde 173.194.67.106: bytes=32 tiempo=50ms TTL=47
Respuesta desde 173.194.67.106: bytes=32 tiempo=50ms TTL=47

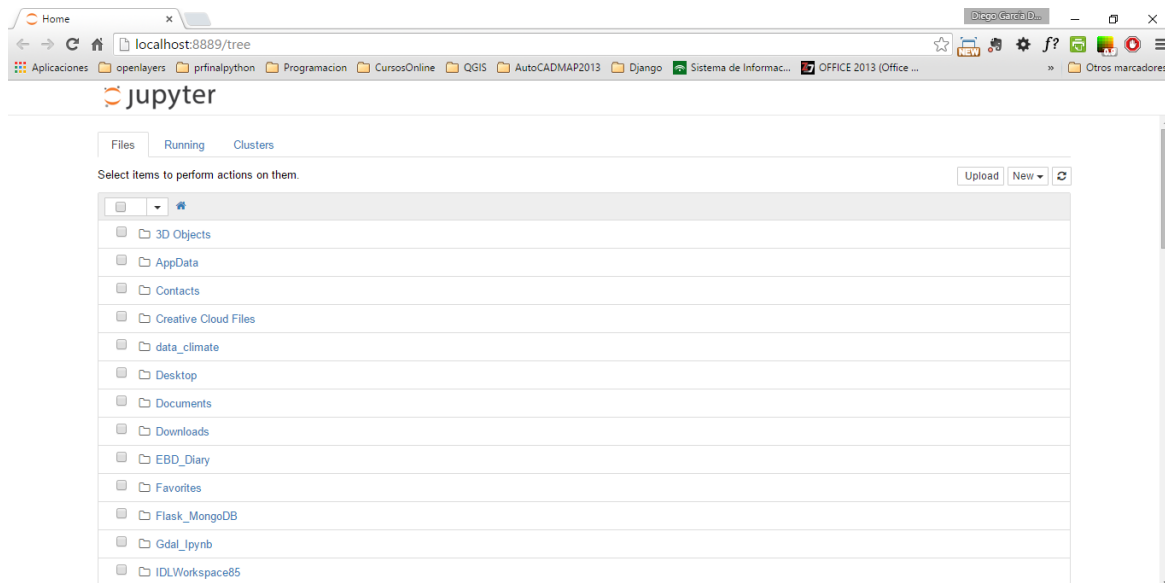
Estadísticas de ping para 173.194.67.106:
Paquetes: enviados = 4, recibidos = 4, perdidos = 0
(0% perdidos),
Tiempo aproximado de ida y vuelta en milisegundos:
Mínimo = 50ms, Máximo = 56ms, Media = 51ms

Haciendo ping a www.google.com [173.194.67.106] con 32 bytes de datos:
Respuesta desde 173.194.67.106: bytes=32 tiempo=52ms TTL=47

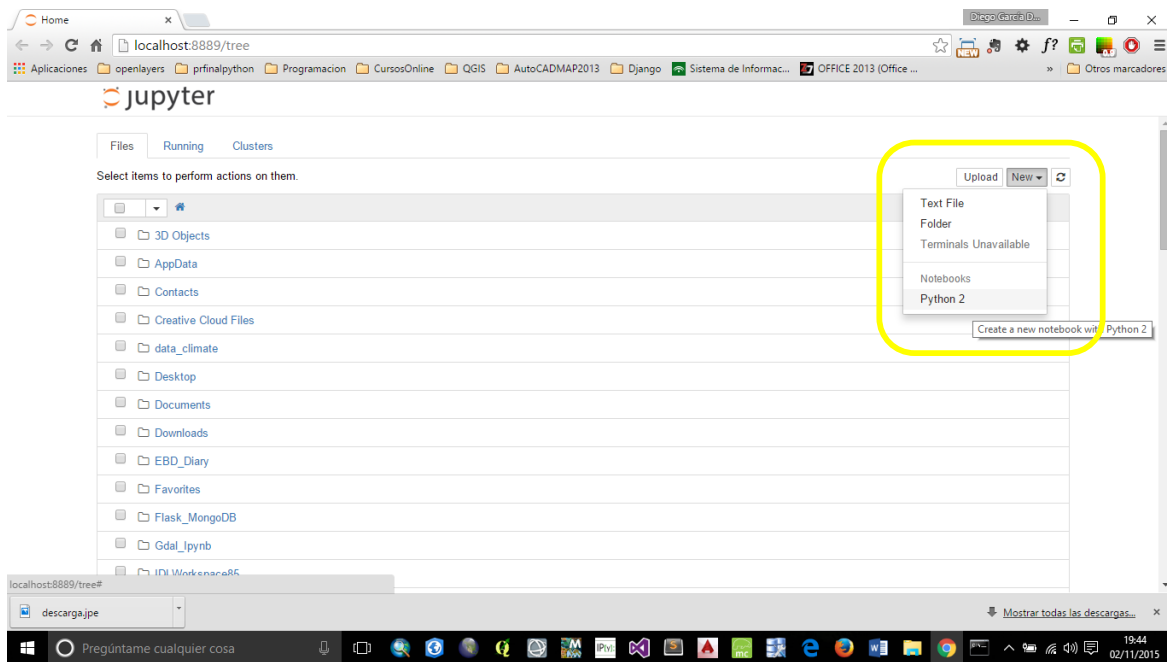
Microsoft Windows [versión 10.0.10240]
(c) 2015 Microsoft Corporation. Todos los derechos reservados.

C:\Users\Diego>mongod
2015-11-02T19:36:59.891+0100 I JOURNAL [initandlisten] journal dir=C:\data\db\journal
2015-11-02T19:36:59.892+0100 I JOURNAL [initandlisten] recover: no journal files present, no rec
covery needed
2015-11-02T19:36:59.928+0100 I JOURNAL [durability] Durability thread started
2015-11-02T19:36:59.929+0100 I JOURNAL [journal writer] Journal writer thread started
2015-11-02T19:36:59.984+0100 I CONTROL [initandlisten] MongoDB starting : pid=266580 port=27017 d
bpath=C:\data\db\ 64-bit host=Diego-PC
2015-11-02T19:36:59.984+0100 I CONTROL [initandlisten] targetMinOS: Windows 7/Windows Server 2008
R2
2015-11-02T19:36:59.985+0100 I CONTROL [initandlisten] db version v3.0.5
2015-11-02T19:36:59.985+0100 I CONTROL [initandlisten] git version: 8bc4ae20708dbb493cb09338d9e7b
e6698e4a3a3
2015-11-02T19:36:59.985+0100 I CONTROL [initandlisten] build info: windows sys.getWindowsVersion(
major=6, minor=1, build=7601, platform=2, service_pack='Service Pack 1') BOOST_LIB_VERSION=1_49
2015-11-02T19:36:59.985+0100 I CONTROL [initandlisten] allocator: tcmalloc
2015-11-02T19:36:59.985+0100 I CONTROL [initandlisten] options: {}
2015-11-02T19:37:00.837+0100 I NETWORK [initandlisten] waiting for connections on port 27017
```

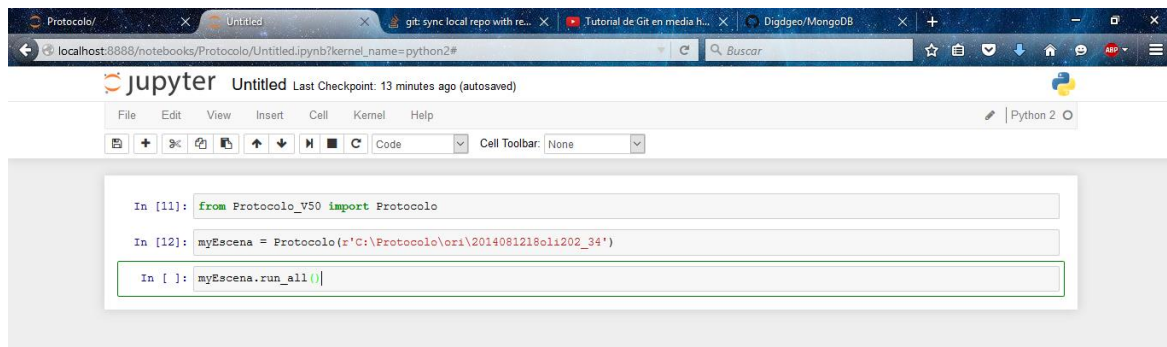
Se nos abrirá la ventana principal de ipython notebook (que en la última actualización pasó a denominarse Jupyter).



Desde esta ventana podemos acceder al código completo del Protocolo (alrededor de 1800 líneas de código), o simplemente, y seguramente más cómodo en nuestro caso, pulsar en “new” y crear un nuevo archivo Python 2.



Esto nos llevará a un nuevo notebook desde el que tan solo debemos importar el Protocolo del siguiente modo, iniciar la clase y ejecutar el Protocolo.



En estas 3 líneas de código ya estaríamos ejecutando el Protocolo. El tiempo medio por escena está en torno a los 10 minutos de procesado, dependiendo principalmente del tiempo que tarde Fmask en generar la máscara, que está en función de la complejidad nubosa de la escena.

3.2 El código



En este apartado, que presumiblemente será el más extenso, vamos a analizar paso a paso, como está estructurado el código y que es lo que hace exactamente.

El Protocolo se ha creado dentro del paradigma de la programación orientada a objetos. Así, el Protocolo consiste en una clase que cuenta con una serie de métodos (39), que son las que van a realizar los procesos necesarios para que la imagen se normalice del mismo modo que lo hacía en el Protocolo manual.

Los 4 métodos que agrupan a los otros 35, estos métodos son Importación, Reproyección, Corrad y Normalización.

Importación	Reproyección	Corrad	Normalización
fmask()	reproject()	copy_files_GR()	normalize()
fmask_legend()	copyDocG()	createR_bat()	nor1()
mascara_cloud_pn()	modifyDocG()	callR_bat()	nor2()
get_cloud_pn()	modifyRelG()	cleanR()	copyDocR()
createG_bat()	modify_hdr_rad_pro()	modify_hdr_rad()	modifyRelNor()
callG_bat()		correct_sup_inf()	fmask_legend()
get_kl_csw()		clean_correct_R()	fmask_binary()
remove_masks()		modify_hdr_rad_255()	modify_hdr_rad_pro()
		translate_bytes_gdal()	
		modifyRelRad()	
		re_clean_R()	
		modifyDocR()	
		modify_hdr_rad_pro()	

Finalmente hay un método llamado *run_all()* que llama a estos 4 métodos, de modo que para llevar a cabo todo el proceso de manera automática, este es el método al que hay que llamar.

3.2.1 Importación

3.2.1.1 `__init__()`

Hay que señalar que antes incluso de la importación, lo primero que hace el Protocolo al instanciar la clase es ejecutar el método `__init__()` (también llamado “constructor de la clase”). Este método hace varias cosas que nos serán útiles a lo largo de la ejecución del proceso.

Por un lado crea una serie de variables o parámetros de la clase, entre las que se encuentran los distintos path que vamos a utilizar⁴, algunos archivos necesarios para el Protocolo (máscaras para la normalización, shape del Parque Nacional para obtener la cobertura nubosa sobre él), y algunos archivos internos que usará para guardar datos antes de escribirlos en archivos de texto y en la base de datos.

Al instanciar la clase también busca en el archivo de metadatos original ‘escena.MTL’ de dónde saca la cobertura nubosa de la escena y el usgs_id de la misma, parámetros que se guardarán en la base de datos como veremos más adelante.

Por último, al instanciar la clase se crea el registro (documento) en la base de datos. La base de datos solo admite un documento por escena, de modo que no podemos tener una escena registrada 2 veces. Por tanto, para evitar un error de clave primaria duplicada, lo que se hace si se vuelve a correr una escena que ya está en la base de datos, lo que se hará es actualizar la fecha en la que se procesa la escena.

3.2.1.2 Importación

En este paso importaremos la escena con Miramón para generar los archivos .img, .doc y el archivo de metadatos .rel. En la importación se usan 8 métodos, en primer lugar se corre la máscara de nubes de Fmask. Este método `fmask()` lo que hace es llamar al ejecutable de la propia herramienta. En la línea 82 encontramos lo siguiente: `os.system('C:/Cloud_Mask/Fmask 1 1 0 50')`. Lo que estamos haciendo es llamar a la herramienta, para ello nos debemos de situar primero en la carpeta en la que está la escena que queremos procesar (cosa que hemos hecho 2 líneas antes en el código), y luego llamar a la herramienta indicando el path al ejecutable y 4 valores separados por espacios, que por defecto los estamos introduciendo como “1 1 0 50”.

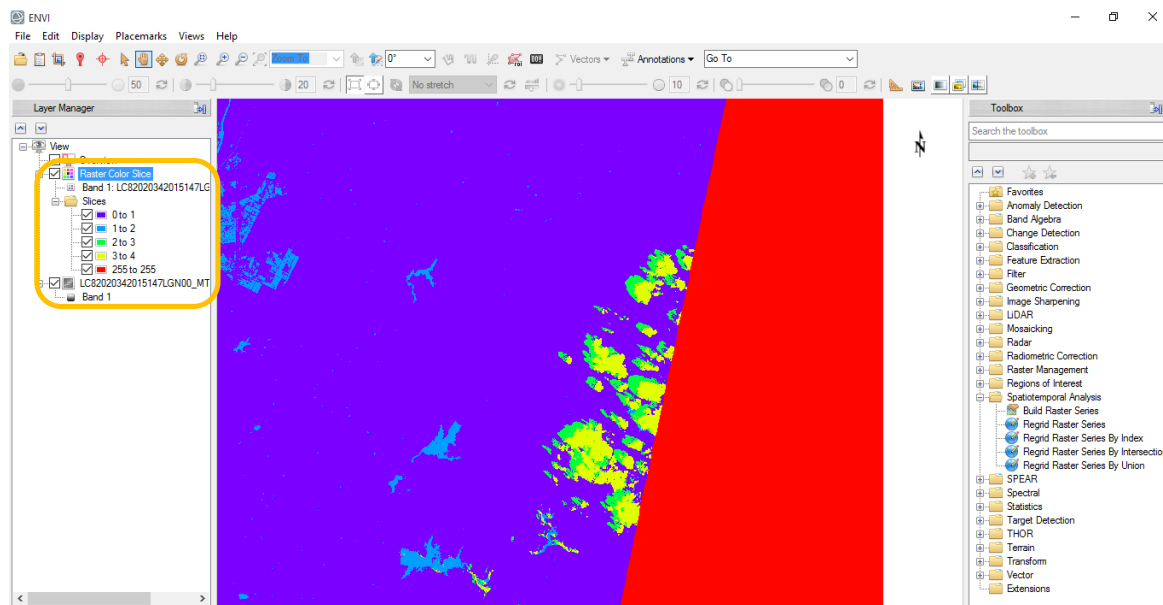
El valor más importante es el último que indica con que grado de confianza queremos trabajar, después de ver todas las escenas con nubes se decidió que lo más correcto es trabajar al 50% de confianza. Los otros 3 valores son el número de píxeles que queremos que se “expanda” nuestra

⁴ Importante!!: Las rutas son relativas (la mayoría), pero sí exigen que la imagen a procesar esté en una carpeta llamada /ori y que al mismo nivel de dicha carpeta se encuentren las carpetas /geo, /rad, /nor y /data. Por defecto el Protocolo se ejecuta sobre C:\Protocolo, aunque no es relevante, siempre y cuando sobre la carpeta raíz estén todas las carpetas que se han señalado antes.

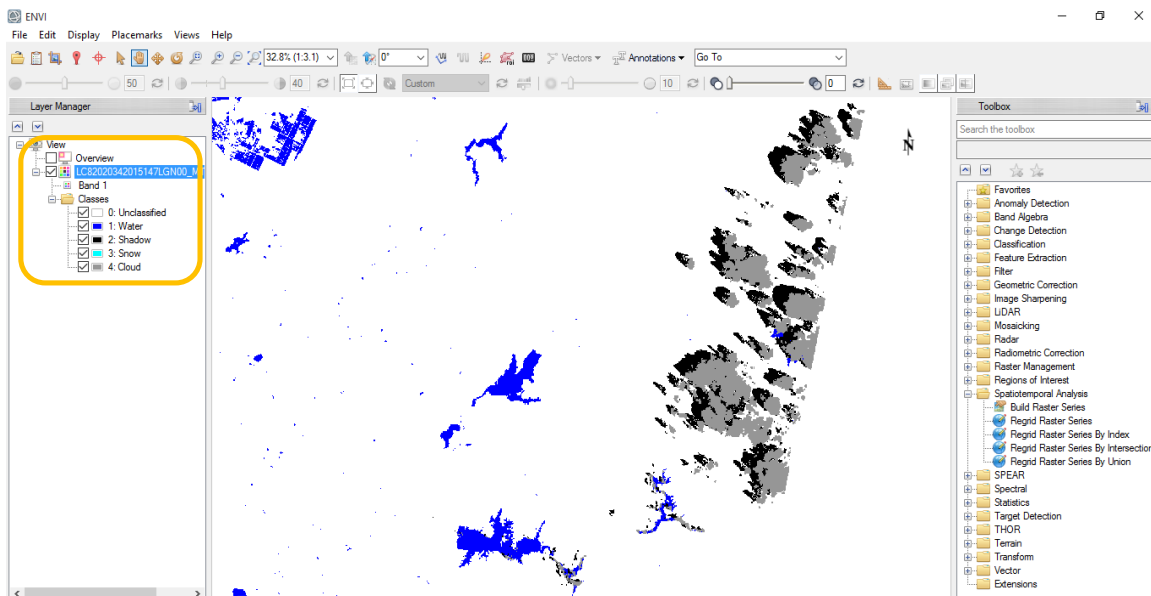
máscara. Es decir, si queremos que se amplíe a algunos píxeles vecinos. Los 2 primeros valores corresponden a valores de nubes y sombra de nubes y el tercero a la máscara de nieve⁵

Si corremos Fmask por defecto (sin incluir ningún valor, estaremos usando los valores “3 3 o 22.5”), pero al igual que pasó con el porcentaje de confianza se ha visto que funciona mejor con los valores “1 1 o 50”. En cualquier caso, siempre que se quiera cambiar alguno de estos valores, es tan simple como cambiarlo en esa línea 82 de código.

La salida de Fmask es una máscara en bytes con valores {255: NoData, 0: Ninguna clase, 1: Agua, 2: nubes, 3: nieve, 4: Sombra de nubes}.



En el siguiente método `fmask_legend()` se modifica el hdr para añadirle la leyenda, de modo que ya directamente la abriremos viendo que es cada valor y con su propio código de colores.



⁵ Toda la información al respecto puede consultarse en <https://code.google.com/p/fmask/>

Una de las cosas que hace el código en esta primera fase es calcular el porcentaje de nubes sobre el Parque Nacional (pensando que podría ser una buena opción de filtrado a posteriori). Para ello emplea el método `mascara_cloud_pn()` con el que realiza una serie de máscaras temporales de cada banda, con un shape del PN que se encuentra en /data. Una vez hechas las máscaras el método `get_cloud_pn()` que calcula la superficie que hay con nubes y/o sombra de nubes y la divide por la superficie total del PN.

Una vez obtenida esa superficie (y metida en la base de datos por supuesto, pero eso lo veremos más tarde), pasamos a lo que la importación verdadera, es decir, a ejecutar el *Miramón Import*. Este proceso se realizará mediante 2 métodos `createG_bat()` y `callG_bat()`. El primer método genera la macro para realizar la importación desde Miramón y el segundo simplemente la ejecuta. La escena importada se guardará en una carpeta llamada /miramon_import dentro de la carpeta de la escena en /ori.

El siguiente método es el más largo de la importación y es el método por el cual se consigue extraer los valores del objeto oscuro para cada banda. Esos valores se guardarán automáticamente, tanto en la base de datos como en un archivo de texto en la carpeta /rad llamado '*nombre_escena_kl.rad*'.

El método se denomina `get_kl_csw()` y hace varias cosas:

- Genera un dtm con la extensión de la escena en cuestión. Para ello, utiliza un dtm de 30 m de resolución ([Aster GDEM v2 Global Elevation Data](#)) con una extensión lo suficientemente grande como para abarcar cualquier escena. Obviamente realizando una máscara con *Gdalwarp*, el shape para hacer la máscara lo saca de la extensión de la propia escena con *Gdaltindex*.
- Realiza un mapa de sombras o hillshade con los parámetros concretos para cada escena de altura y azimut solar. Esos datos los saca del archivo de metadatos 'usgs_id.MTL' y los aplica mediante *Gdaldem*. El hillshade junto con el resto de archivos que se vayan generando en este paso se guardarán en /data/temp.
- Obtiene el valor mínimo para cada banda, esto lo hace enmascarando las matrices de valores de los rasters, buscando solo en las zonas en las que la máscara de nubes tiene valor 1 (agua) o en las zonas de sombra (es decir, aquellas con valor 0 en la máscara de nubes y percentil 20 del hillshade). De este modo nos aseguramos que el valor del kl sea en agua o sombras, excluyendo el resto de la escena (aquí se podría ver, aprovechando que ya se cuenta con 35 escenas, el tema de meter algún criterio de calidad).
- Genera los histogramas de los 1000 valores más bajos (en principio, éste es uno de los valores susceptibles de ser cambiados si se desea, se encuentra en la línea 381 del código). Estos histogramas se guardarán con el nombre adecuado para cada banda en la carpeta /rad.
- Genera el ya citado archivo '*escena_kl.rad*'
- Finalmente, aunque en realidad es lo primero que hace el método. Borra de la carpeta temp el hillshade y el shape con la extensión de la escena. Como decimos, esto es lo primero que se hace al llamar a este método. Se hace así con la idea de que, si se quiere comprobar algo al finalizar una escena, esos archivos estén disponibles para su consulta, y no serán borrados hasta que se corra una escena nueva.
- Cuando la imagen que se corra sea una Landsat 7, entonces se añadirá otra máscara que será la suma (la multiplicación realmente), de las *Gapmask* que vienen en la imagen original. De modo que se excluirán también como zona válida para el *Kl*, aquellas zonas en las que en cualquiera de las bandas, esos píxeles estén dentro del *Gapfill*.

Por último hay un método ***remove_masks()*** que borra la carpeta donde se guardaron las máscaras para obtener la cobertura de nubes del PN.

Hasta este paso habrán transcurrido entre 2 y 3 minutos desde que iniciamos el Protocolo.

3.2.2 Reproyección

Este es el segundo gran proceso de nuestro Protocolo automático, es también el más corto en cuanto a código y en principio el que más tiempo consume, aunque al final el proceso se ha acortado bastante, pasando de unos 60 segundos por cada banda a tan solo 10 segundos por banda. EL cambio ha sido llamar de utilizar las *Python bindings* con para Gdal a llamar directamente a *Gdalwarp* vía subprocess.

Lo primero que se hace es llamar al método ***reproject()*** que busca todos los .TIF dentro de la carpeta de la escena en /ori y los reprojecta mediante EXACTAMENTE LA MISMA REPROYECCIÓN DE 3 PARÁMETROS QUE SE USABA EN ENVI 4.6 al CRS de salida.

Durante la reproyección se hacen varias cosas, junto con la propia reproyección se realiza el translate a formato de ENVI (img + hdr), y se hace el rename del nombre de las bandas en /ori (el nombre de la USGS_id) al formato Last (“escena'+satélite+sensor+path+'_'+row+pasoproceso+banda+.img”).

Al igual que se reprojectan las .TIF originales se reprojecta también la máscara de nubes (que ya se encuentra en formato img + hdr) directamente a la carpeta de la escena en /nor.

Una vez realizado este método tendremos en la carpeta de la escena en /geo los img y hdr de cada banda, ahora debemos de copiar los .doc y el .rel de la carpeta miramon_import a esta carpeta /geo. Esto se realiza mediante el método ***copyDocG()***. Posteriormente esos .doc y .rel deben de modificarse para que tengan los valores correctos con el nuevo CRS de la escena, así como añadir los strings de los procesos realizados/técnico que los realiza, etc...

Estos cambios se realizan mediante los métodos ***modifyDocG()*** y ***modifyRelG()*** (imagino que no hace falta decir cual hace que tarea ☺).

Con esto ya tendríamos realizada la reproyección. Al finalizar este paso ya tendremos las bandas reprojectadas en /geo, más la máscara de nubes reprojectada en /nor, junto con el archivo de kls y los histogramas en /rad (esto último ya viene hecho desde la importación).

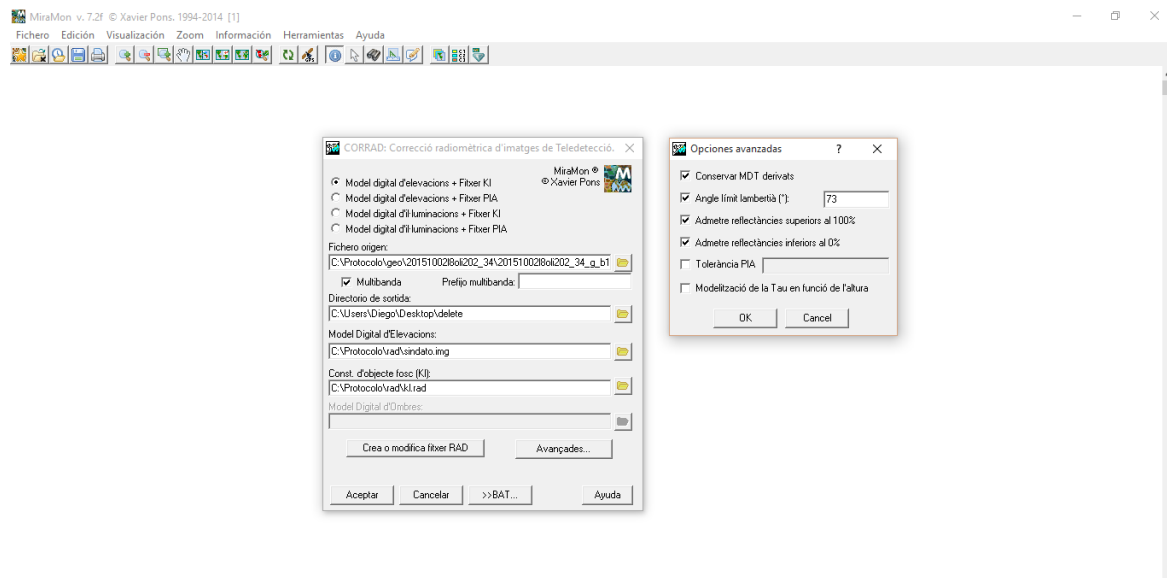
Hasta este paso habrán transcurrido aproximadamente entre 6 y 7 minutos desde que comenzamos el Protocolo.

3.2.3 Corrección Radiométrica (Corrad)

La corrección radiométrica comienza copiando los archivos (las bandas más sus archivos de metadatos hdr, doc y el rel) desde la carpeta de la escena en /geo a la carpeta de la escena en /rad. Esto se realiza mediante un método denominado **copy_files_GR()**.

A continuación, del mismo modo que hicimos para la importación, hay 2 métodos encargados de crear y ejecutar el .bat necesario para realizar el Corrad: **createR_bat()** y **callR_bat()**.

Dentro de los parámetros que se pueden seleccionar para ejecutar el Corrad, nosotros estamos seleccionando las opciones:



- 1) Modelo Digital de Elevaciones + fichero kl
- 2) Conservar MDT derivados
- 3) Límite de Ángulo Lambertiano 73°
- 4) **Admitir Reflectancias superiores a 100 e inferiores a 0**

Respecto al DTM sigue siendo el mismo que se usaba en el Protocolo Manual ('sindato.img'), ya no se guardan los modelos de sombra e iluminación⁶. Lo más importante aquí es señalar que al

⁶ A pesar de que sigue existiendo la opción de Conservar MDT Derivados en la interfaz, realmente para generar y conservar esos productos hay 2 nuevas herramientas en Miramón (Herramientas-→Radiometría de Imágenes-→Obtener el Modelo Digital de Iluminaciones; Obtener el Modelo Digital de Sombras. En principio, se ha decidido que no se iban a generar. No obstante, como ya hemos visto en la importación, Modelo de Sombras si generamos con GDAL para obtener el KI, aunque de momento como ya hemos visto también, solo se guarda en la carpeta /data/temp durante el período que transcurra entre la normalización de una escena y la siguiente (esto sería fácilmente subsanable, si realmente se quisiera corregir el modelo de sombras. También sería muy fácil generar un modelo de iluminaciones, tanto con Miramón como con GDAL).

contrario que se hacía antes, ahora se están seleccionando las opciones Admitir **Reflectancias** Inferiores a 0 y Superiores a 1. Los valores por debajo de 0 serán posteriormente reclasificados a 0 y los superiores a 1 pasarán a 254 (esto fue “avalado” por la gente de Miramón durante el curso celebrado con motivo del Congreso).

En la Corrección Radiométrica hay que decir, que como es sabido, la principal diferencia es que las imágenes, ya no entran en formato byte si no en *uint16* o enteros reales positivos, con valores que van desde 0 a 65535. La salida del *Corrad* también ha cambiado y ahora sale en formato *float32* (valores reales desde -32768 a 32768). Obviamente esto es incompatible con el proceso anterior donde se trabajaba todo en bytes. A la espera de ver si se normalizan las Landsat 8 con una nueva escena de referencia también Landsat 8, (en cuyo caso se haría todo en el formato de salida del *Corrad*), nos vemos obligados de momento a convertir la salida del *Corrad* a byte, de modo que puedan ser usadas las bandas en la normalización, ya que hasta ahora Landsat 8 se sigue normalizando con la escena de referencia Landsat 7 del 18 de julio del 2002.

Después de ejecutar el Bat, tendremos los ya mencionados archivos en *float32*, junto a ellos tendremos los archivos que copiamos de /geo, que debemos de eliminarlos. Para ello se emplea el método *cleanR()*, que solo dejará en /rad los archivos generados por el *Corrad* y los *hdr*. Además de borrar los archivos innecesarios, este método también hace el rename desde el nombre generado por el *Corrad*, a la nomenclatura empleada en el Last⁷.

Una vez tenemos solo los archivos que genera el *Corrad*, tenemos que trabajar con ellas con GDAL para corregir los valores de reflectividad superiores a 1 e inferiores a 0 y para hacer el translate de *float32* a *byte*, para ello lo primero que debemos de hacer es cambiarle el valor del “data type” en los *.hdr*, para que GDAL reconozca que es un *float32*. Esto se hace con el método *modify_hdr_rad()*. Otra cosa que se hace en ese método es definir cual es el valor NoData (-3.40282347e+38).

Posteriormente se aplica el método *correct_sup_inf()* que llevará los valores por debajo de 0 a 0 y los valores por encima de 100 a 100 (el NoData se lleva a 255). Esto se hace de nuevo leyendo el raster como array con **Rasterio**, modificando los valores y guardándolo en otros rasters con el prefijo ‘crt_’ (aún como *float32* en formato *.img* más *.hdr*).

A continuación se aplica el método *clean_correct_R()* que lo que hace es borrar los archivos *.img* originales que salieron del *Corrad*, y los *.hdr* que iban con esos *.img*. Este método también hace ya el *rename* de los archivos a la nomenclatura correcta.

Antes de realizar el *translate* a *bytes*, volvemos a modificar los *.hdr*, tan solo para definir ya el NoData en 255 (tal y como quedo al aplicar *correct_sup_inf()*). Para ello se llama al método *modify_hdr_rad_255()*. Una vez ejecutado ya podemos realizar el *translate* y pasar los rasters de *float32* a *byte*.

El paso de reales a *byte* se hace con GDAL, concretamente con *gdal_translate* y el método se llama *translate_bytes_gdal()*. La estructura de la línea de código que ejecuta la herramienta es “*gdal_translate -ot Byte -of ENVI -scale 0 100 0 254 -a_nodata 255*”, donde le estamos diciendo que

⁷ Otra cosa que realiza este método es la copia del original, es decir, aquel que se decidió que estuviera para consultar los valores mínimos y máximos que salen del *Corrad*. Esto se hizo en su momento para consultarlo con la gente de Miramón.

queremos que la salida sea en formato Byte de ENVI (.img + .hdr) y que escale los valores [0-100] a [0-254], siendo el NoData 255 (tal y como definimos antes).

En este punto tenemos los .img y .hdr resultado del *translate* junto con los .doc, el .rel (y los .png de los histogramas y el .rad). Pero solo los .hdr tienen los valores correctos, por ello debemos de correr 2 métodos para modificar el .rel y los .doc con los valores correctos. En ambos casos esto se realiza leyendo los archivos como texto y reemplazando los valores incorrectos por los valores correctos. Estos métodos se denominan *modifyRelRad()* y *modifyDocR()*.

Los cambios en las bandas salidas del Corrad se han ido guardando en una carpeta llamada /byte dentro de la carpeta de la escena en rad. Una vez que todo está correcto se copian a la carpeta de la escena en rad y se borra la carpeta byte (esto se hace con el método *re_clean_R()*). De modo que ya se concluye la Corrección Radiométrica, teniendo las bandas corregidas radiométricamente en formato .img + .doc + .hdr, junto con el .rel.

Finalizado este paso llevaremos alrededor de 8-9 minutos de proceso.

3.2.4 Normalización

3.2.4.1 Recordatorio Protocolo Manual

Este paso consta de 8 métodos. En él se realizarán los pasos equivalentes a las macros realizadas anteriormente en **Idrisi**.

Antes de comentar la normalización en el Protocolo Automático, vamos a recordar brevemente el proceso de normalización que se definió en el Protocolo Manual:

Para llevar a cabo la normalización entre la escena que estemos tratando con la escena de referencia (2002071817etm202_34), necesitamos hacer (para cada banda con la banda equivalente), una regresión entre los valores que se encuentran definidos dentro de una serie de **Áreas Pseudo-Invariantes (PIAs)**.

Esto se produce en 2 fases, en una primera se hace una regresión de todos los píxeles incluidos en las **PIAs**, y en un segundo paso se eliminan todos aquellos píxeles cuya **desviación estándar (std)** sea mayor a 11, y se vuelve a calcular la recta de regresión entre las bandas de la escena, con los píxeles resultantes.

La normalización *per se* la haremos aplicando la ecuación obtenida en la segunda regresión a cada banda:

$$B_iN = B_i * slope + intercept$$

- B_iN = Banda Normalizada
- B_i = Banda
- **slope** = Pendiente de la recta de regresión
- **intercept** = Intersección en el eje Y

Para que se considere que se puede hacer la normalización es necesario que se cumplan 2 condiciones:

1. EL coeficiente de correlación R debe ser > 0.85
2. El número mínimo de píxeles en cualquiera de las PIAs debe de ser ≥ 10 (ver tabla 1)

<i>Id</i>	Tipo Área	Nº Píxeles Mascara No Equilibrada	Nº Píxeles Máscara Equilibrada
1	Mar	49501	4284
2	Embalses	5025	1044
3	Pinar	1672	
4	Urbano-1	927	
5	Urbano-2	1101	
6	Aeropuertos	922	
7	Arenas	783	
8	Pastizal	298	
9	Minería	332	
Total		60561	11363

Tabla 1. Número de píxeles por cada PIA según máscara.

Para evitar ocasiones en las que no se cumplían los requisitos de normalización se establecieron otra serie de criterios admisibles, con un orden de prioridades definidos tal y como se puede ver en la tabla 2 (pag. siguiente). Hay 2 cuestiones fundamentales, que son la inclusión de una nueva máscara de PIAs “Equilibrada”, donde el peso del mar y los embalses es mucho menor (tabla 1), y la otra cuestión importante, es la ampliación de la tolerancia de los píxeles que se van a considerar válidos en función de la desviación estándar de la primera regresión.

De este modo se definen hasta 5 pasos más, en los cuales sería adecuado hacer la normalización de la banda. Esto se realizaba con una macro de Idrisi (nor 1 bis), que corría esos 5 pasos para que posteriormente, un técnico viera cual era el primer paso según el orden de prioridad que cumplía las condiciones, y tomar los valores de slope e intercept de dicho paso.

Finalmente, en la macro Nor 2 se llevaba a cabo la operación puramente aritmética de aplicar la ecuación de regresión a las bandas a normalizar.

3.2.4.2 Normalización en el Protocolo Automático

El primer método que se ejecuta en la normalización es *normalize()*. En este método se produce realmente todo el proceso de normalización, siendo el resto de métodos para modificar el .rel, .docs, etc.

El método *normalize()* lo que hace es llamar al método *nori()*, que a su vez llama al método *nor2()* en cuanto se cumplan los requisitos de $R > 0.85$ y nº mínimo de píxeles ≥ 10 .

Lo que se hace aquí es regular el flujo de criterios y prioridades establecidos en el Protocolo Manual, ordenados del modo que se muestra en la tabla 2.

<i>Tipo Área PIA</i>	<i>Desviación Típica</i>	<i>Prioridad</i>
<i>No Equilibrada</i>	11	1
<i>No Equilibrada</i>	22	2
<i>Equilibrada</i>	11	3
<i>Equilibrada</i>	22	4
<i>No Equilibrada</i>	33	5
<i>Equilibrada</i>	33	6

Tabla 2. Tabla de prioridades y criterios para obtener los parámetros de normalización.

Tanto *nori()* como *nor2()* son métodos que requieren de unos parámetros de entrada para ejecutarse. En el caso de *nori()* los parámetros que necesita son: el número de banda, la máscara a emplear y la std (que por defecto es 11):

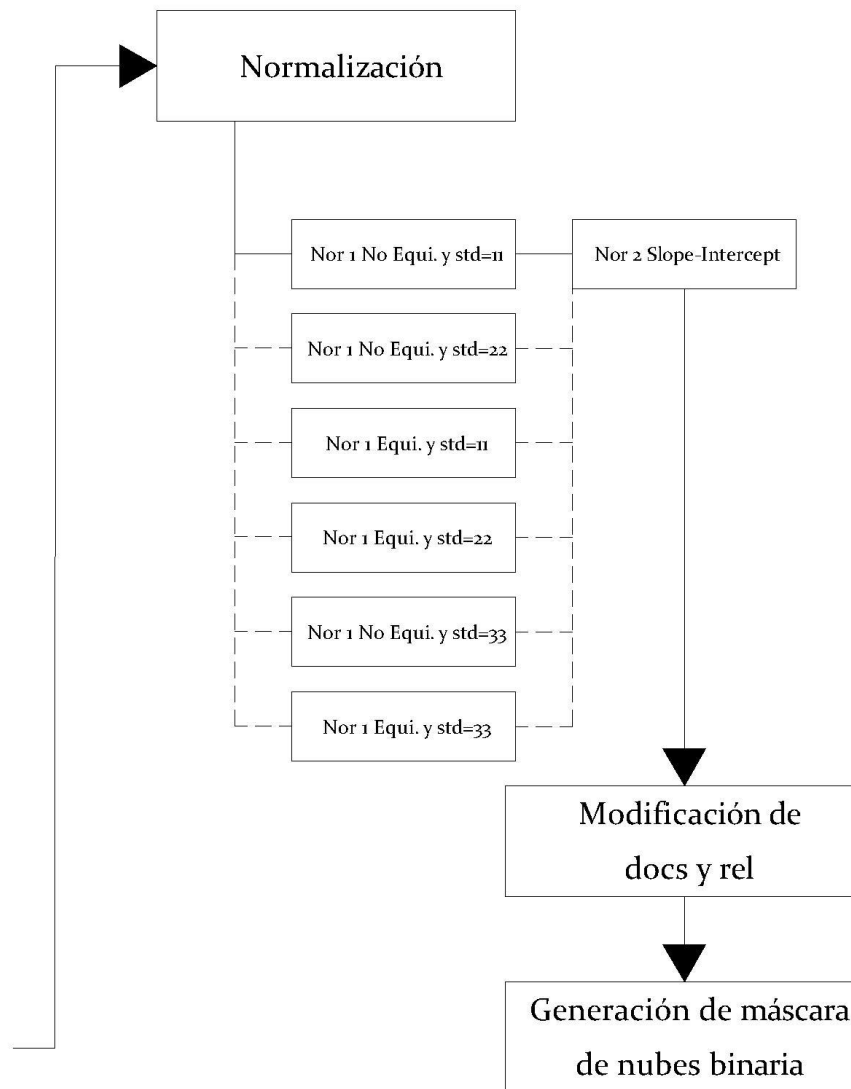
```
def nori(self, banda, mascara, std = 11):
```

De modo que lo que el método *normalize()* hace es ir llamando a *nori()* con los parámetros indicados en la tabla 2. En el momento en el que se cumplan las 2 condiciones ($R > 0.85$ y nºPíxeles-PIA ≥ 10), el método *nori()* llamará al método *nor2()*, que también es un método que necesita parámetros de entrada para ejecutarse, en este caso, esos parámetros son: el número de banda, el *slope* y el *intercept*:

```
def nor2(self, banda, slope, intercept):
```

Como vemos, todo se controla desde *normalize()* y ya no es necesario correr los 5 pasos alternativos si no se puede dar la normalización en el primer caso, sino que se irán llamando a los distintos pasos siempre y cuando no se hayan podido obtener valores válidos en el paso anterior.

El esquema del funcionamiento sería el siguiente:



En ambos métodos (***nor1()*** y ***nor2()***) se trabaja con arrays.

En ***nor1()*** se van aplicando las distintas máscaras a los arrays, hasta que se consiguen los valores de la regresión, que se pasarán a ***nor2()***. La regresión que se es una regresión de mínimos cuadrados (OLS), empleando para ello la librería de Python SciPy⁸.

En ***nor2()*** se utiliza **Rasterio**, para abrir los rasters como arrays, aplicar la ecuación de la recta de regresión y volver a guardar el array como raster, pasando primero los valores que superan el 255 a 254 (valor válido más reflectivo) y los valores que queden en negativo a 0 (valor válido menos reflectivo).

Una vez finalizado este paso, ya tendremos en la carpeta de nuestra escena en /nor las bandas en formato .img + .hdr, junto con la máscara de nubes. Ahora debemos de copiar y modificar de la carpeta de la escena en /rad, los .docs y el .rel. Esto se hace con los pasos ***copyDocR()*** y ***modifyRelNor()***. En el método ***copyDocR()*** también se genera un .doc para la máscara de nubes, de cara a poder abrirla con Miramón y que la reconozca como archivo de clases.

Ya por último se aplica (por petición popular) el método ***fmask_binary()*** que lo que hace es pasar la máscara de nubes a una máscara con solo valores 0 (no nubes ni sombra de nubes) y 1 (nubes y sombras de nubes) y que es la que se guardará con el nombre 'xxxx_CM.img', para ser comparable con las realizadas con el Protocolo Manual. La máscara de Fmask se guardará con el nombre 'xxxx_Fmask.img'.

Por desgracia la máscara de 0 y 1 no se puede guardar en binario, ya que el formato más reducido en que puede guardarse es *byte*.

Por último, se han añadido 2 métodos (***caller()*** y ***caller_js()***) que escriben otros tantos archivos, necesarios para que Ricardo Díaz-Delgado suba las escenas normalizadas a los servidores⁹. Estos métodos lo que hacen es modificar los archivos caller.bat y caller.js ubicados en la carpeta /data, añadiendo las escenas que se van normalizando a dichos archivos en función de su estructura.

4. Base de datos

4.1 Diseño y estructura de la base de datos

La base de datos elegida para el Protocolo Automático ha sido **MongoDB**. Se trata de una base de datos No Relacional o NoSQL, que tiene como características principales el presentar un esquema dinámico, ya que al no tener que estar sujeta a una estructura fija de tablas, cada documento (el equivalente a cada registro o cada fila en una base de datos relacional), puede tener una estructura propia (cosa que por ejemplo, resultará muy útil para almacenar los valores de la antigua base de datos dentro de ésta).

⁸ <http://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.stats.linregress.html>

⁹ En este momento (20/11/15) solo está implementado ***caller()***, pero espero que en cuestión de pocos días quede también implementado el otro.

En las bases de datos NoSQL los documentos están integrados en colecciones, que serían el equivalente a las tablas en el mundo relacional. Esos documentos llevan la información asociada a modo de pares “Clave”: “Valor”, de forma que las claves serían el equivalente a los campos en las bases de datos relacionales. En principio¹⁰ no se admiten los *joins*, lo que se solventa “embebiendo” la información que habría en otra tabla dentro del propio documento, lo cual es fácil de hacer ya que el esquema dinámico permite que los valores sean a su vez otros diccionarios con el mismo esquema Clave: Valor, listas o cualquier objeto soportado por la base de datos.

En el diseño de esta base de datos se ha decidido incluir algunos valores que no se incluían en la anterior, así como prescindir de algunos elementos que antes se guardaban. En nuestra base de datos la estructura es similar a la estructura del propio Protocolo:

En un primer nivel tenemos el **_id** de la escena¹¹, que en nuestro caso es el nombre de la escena según el formato del Last-EBD, en este primer nivel también tenemos el **USGS_id**, que es el nombre de la escena en el formato USGS. En el primer nivel de la colección Landsat también se encuentran las claves **Info** y **Clouds**, que a su vez se despliegan en más diccionarios con distinta información.

La clave **Clouds** se compone de 3 claves que son **Cloud_Scene** (% de nubes en la escena), **Cloud_PN** (% de nubes sobre el Parque Nacional de Doñana) y **Umbral** (% de confianza usado al llamar a Fmask, por defecto se ha decidido dejarlo en 50).

La clave **Info** es la que contiene la mayor parte de información, se compone de cuatro claves que son **Iniciada** (fecha en la que se instancia la escena), **Técnico** (técnico que aplica el Protocolo, aunque por defecto se va a dejar Protocolo Automático), **Finalizada** (fecha/hora en la que se termina la normalización) y **Pasos** (que es donde se guarda la mayor parte de la información relevante).

La clave Pasos se compone de 3 claves que son **Geo**, **Rad** y **Nor**. A su vez se vuelven a dividir en más diccionarios, la clave **Geo** es la que cuenta con menos información, ya que solo indica si está realizada la reproyección y la fecha/hora a la que se hizo. La clave **Rad** ya indica más información, además de indicar si se ha realizado la corrección radiométrica y la fecha/hora en la que se realizó, nos da los valores que ha tenido el *kl* en cada banda.

La clave **Nor** es la que más información almacena, junto con indicar si se hizo la normalización y la fecha/hora en que se hizo, hay una clave llamada **Nor-Values**, donde se guardan para cada banda que se haya podido normalizar, 2 tipos de valores distintos, guardados a su vez en 2 nuevas claves: **Parámetros** y **Tipo-Área**. En el primero se guardan los valores obtenidos durante la regresión: **Slope**, **Intercept**, **R**, **NºPíxeles** válidos en la máscara e **Iter** (paso según el orden de prioridades, en el que se obtienen los valores necesarios para hacer la regresión). En el segundo se guarda el número de píxeles válidos en cada tipo de área (Mar, Arena, etc.).

En las siguientes páginas se muestra a modo de ejemplo (que seguro es mucho más fácil de entender que mis explicaciones), un documento (escena L8 del 19 de abril de 2013) de la base de datos con toda la información incorporada.

¹⁰ Esto va a cambiar en breve con la salida de [MongoDB 3.2](#)

¹¹ El campo ‘_id’ en MongoDB está reservado a la clave primaria, y siempre está presente en cualquier colección. Si no lo especificamos nosotros MongoDB lo creará automáticamente, en lo que sería el equivalente a un id auto-numérico. La idea de definirlo y de que sea el nombre de la escena en el formato del Last, es que así en ningún caso, se podrá almacenar la misma escena más de 1 vez.

4.1.1 Ejemplo de documento (registro) de la base de datos

```
{
  "_id" : "2013041918oli202_34",
  "Info" : {
    "Iniciada" : "Wed Nov 11 14:07:12 2015",
    "Pasos" : {
      "rad" : {
        "Fecha" : "Mon Oct 12 17:52:49 2015",
        "K1-Values" : {
          "b4" : NumberInt(5923),
          "b5" : NumberInt(4760),
          "b6" : NumberInt(4775),
          "b7" : NumberInt(4899),
          "b1" : NumberInt(9025),
          "b2" : NumberInt(8035),
          "b3" : NumberInt(6739),
          "b9" : NumberInt(4989)
        },
        "Corrad" : "True"
      },
      "geo" : {
        "Georef" : "True",
        "Fecha" : "Mon Oct 12 17:50:42 2015"
      },
      "nor" : {
        "Normalize" : "True",
        "Fecha" : "Wed Nov 11 14:07:57 2015",
        "Nor-Values" : {
          "b4" : {
            "Parametros" : {
              "slope" : 1.2381384666085515,
              "r" : 0.9898070894919756,
              "intercept" : -9.270773940528631,
              "iter" : NumberInt(1),
              "N" : NumberInt(49101)
            },
            "Tipo_Area" : {
              "Arena" : NumberInt(681),
              "Embalses" : NumberInt(1176),
              "Mineria" : NumberInt(133),
              "Mar" : NumberInt(45106),
              "Urbano-1" : NumberInt(582),
              "Urbano-2" : NumberInt(801),
              "Aeropuertos" : NumberInt(311),
              "Pastizales" : NumberInt(42),
              "Pinar" : NumberInt(269)
            }
          },
          "b5" : {
            "Parametros" : {
              "slope" : 1.5264298067464988,
              "r" : 0.9930058412885155,
              "intercept" : -17.946949737252627,
```

```

        "iter" : NumberInt(1),
        "N" : NumberInt(48150)
    },
    "Tipo_Area" : {
        "Arena" : NumberInt(396),
        "Embalses" : NumberInt(1558),
        "Mineria" : NumberInt(120),
        "Mar" : NumberInt(43707),
        "Urbano-1" : NumberInt(354),
        "Urbano-2" : NumberInt(575),
        "Aeropuertos" : NumberInt(161),
        "Pastizales" : NumberInt(64),
        "Pinar" : NumberInt(1215)
    }
},
    "b6" : {
        "Parametros" : {
            "slope" : 1.4944998285383388,
            "r" : 0.9955498240152382,
            "intercept" : -7.009228186474065,
            "iter" : NumberInt(1),
            "N" : NumberInt(51133)
        },
        "Tipo_Area" : {
            "Arena" : NumberInt(700),
            "Embalses" : NumberInt(2020),
            "Mineria" : NumberInt(169),
            "Mar" : NumberInt(45782),
            "Urbano-1" : NumberInt(547),
            "Urbano-2" : NumberInt(645),
            "Aeropuertos" : NumberInt(295),
            "Pastizales" : NumberInt(81),
            "Pinar" : NumberInt(894)
        }
    },
    "b7" : {
        "Parametros" : {
            "slope" : 1.758614924561872,
            "r" : 0.9948433648300684,
            "intercept" : -2.988529494980268,
            "iter" : NumberInt(1),
            "N" : NumberInt(51594)
        },
        "Tipo_Area" : {
            "Arena" : NumberInt(695),
            "Embalses" : NumberInt(2062),
            "Mineria" : NumberInt(198),
            "Mar" : NumberInt(45782),
            "Urbano-1" : NumberInt(575),
            "Urbano-2" : NumberInt(666),
            "Aeropuertos" : NumberInt(340),
            "Pastizales" : NumberInt(49),
            "Pinar" : NumberInt(1227)
        }
    }
},

```

```

        "b2" : {
            "Parametros" : {
                "slope" : 1.9416526501826286,
                "r" : 0.9820650705987837,
                "intercept" : -12.389622945430116,
                "iter" : NumberInt(1),
                "N" : NumberInt(37642)
            },
            "Tipo_Area" : {
                "Arena" : NumberInt(688),
                "Embalses" : NumberInt(179),
                "Mineria" : NumberInt(104),
                "Mar" : NumberInt(34999),
                "Urbano-1" : NumberInt(273),
                "Urbano-2" : NumberInt(421),
                "Aeropuertos" : NumberInt(238),
                "Pastizales" : NumberInt(25),
                "Pinar" : NumberInt(715)
            }
        },
        "b3" : {
            "Parametros" : {
                "slope" : 1.6056184938934905,
                "r" : 0.9840512049025575,
                "intercept" : -14.96610394246239,
                "iter" : NumberInt(1),
                "N" : NumberInt(42273)
            },
            "Tipo_Area" : {
                "Arena" : NumberInt(456),
                "Embalses" : NumberInt(130),
                "Mineria" : NumberInt(115),
                "Mar" : NumberInt(39745),
                "Urbano-1" : NumberInt(418),
                "Urbano-2" : NumberInt(687),
                "Aeropuertos" : NumberInt(272),
                "Pastizales" : NumberInt(59),
                "Pinar" : NumberInt(391)
            }
        }
    },
    "Tecnico" : "LAST-EBD Auto",
    "Finalizada" : "Wed Nov 11 14:08:01 2015"
},
"Clouds" : {
    "cloud_scene" : 0.28,
    "umbral" : NumberInt(50),
    "cloud_PN" : 0.0
},
"usgs_id" : "LC82020342013109LGN01"
}

```

4.2 Interfaz gráfica de gestión de la base de datos (MongoChef)

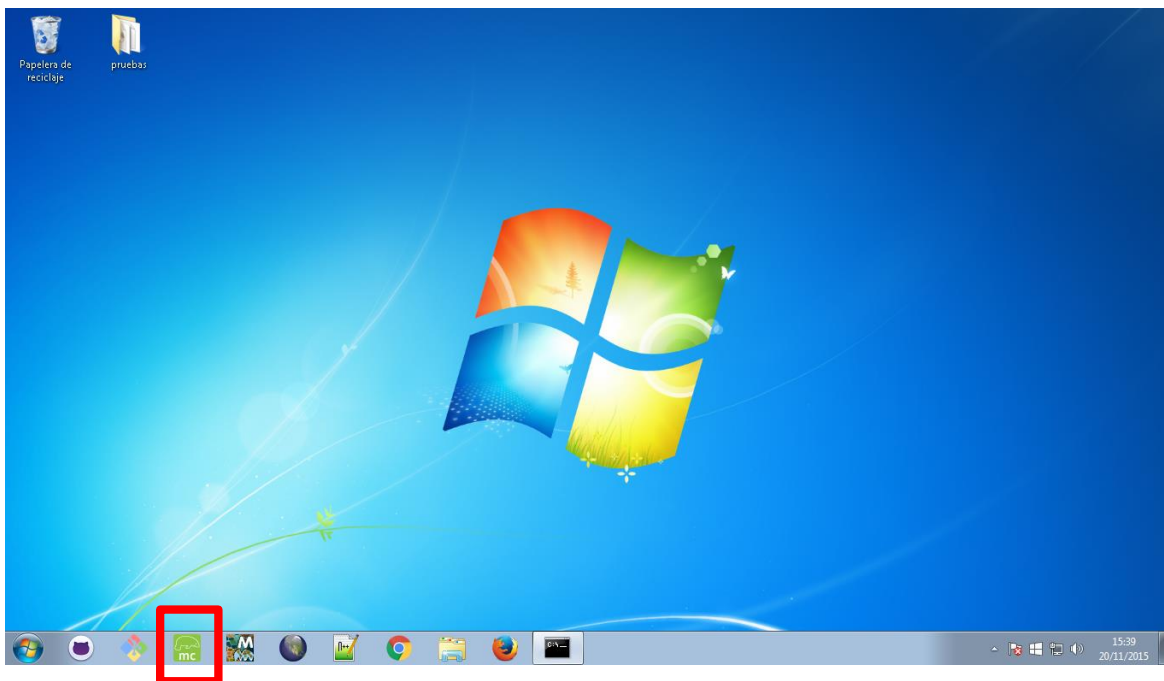
Por lo general MongoDB suele consultarse y/o actualizarse desde la línea de comandos o desde algún programa que se conecta con ella vía *API*, que es lo que se hace durante el Protocolo Automático, que va conectando mediante PyMongo (driver para conectar con MongoDB mediante Python). No obstante, existen varias soluciones para gestionar la base de datos desde un cliente con interfaz gráfica.

En nuestro caso hemos optado por [MongoChef](#), ya que no ha parecido el más intuitivo y completo de usar¹².

A continuación, mediante una serie de capturas de pantallas vamos a tratar de explicar brevemente como utilizar este programa.

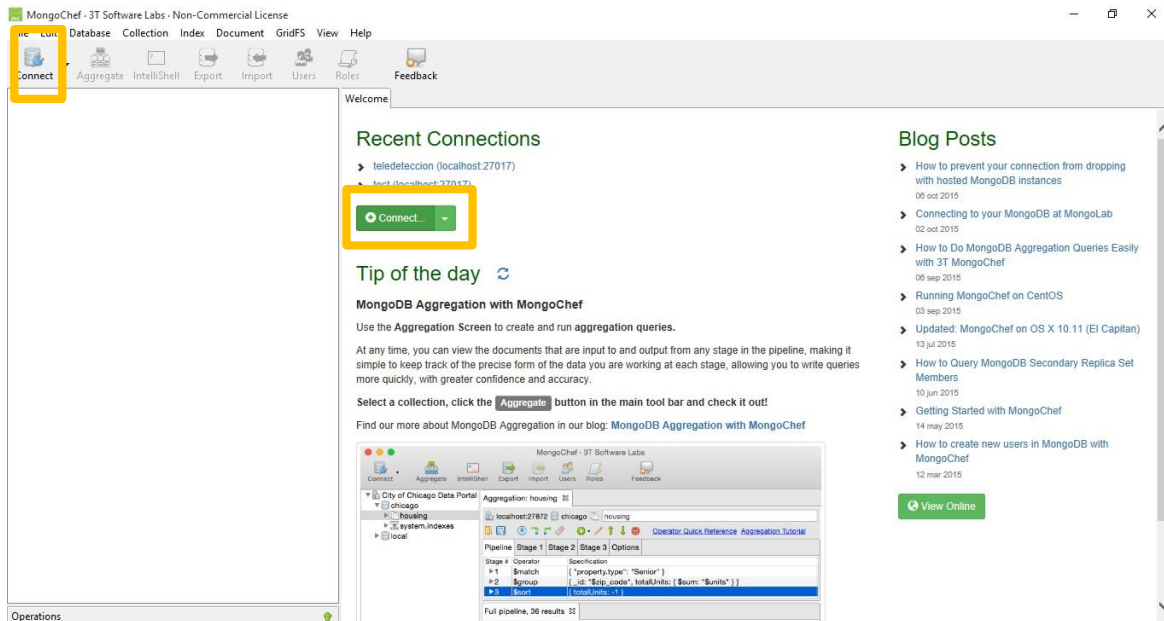
4.2.1 Usando MongoChef

Abrimos **MongoChef** desde el icono ubicado en la barra de herramientas:

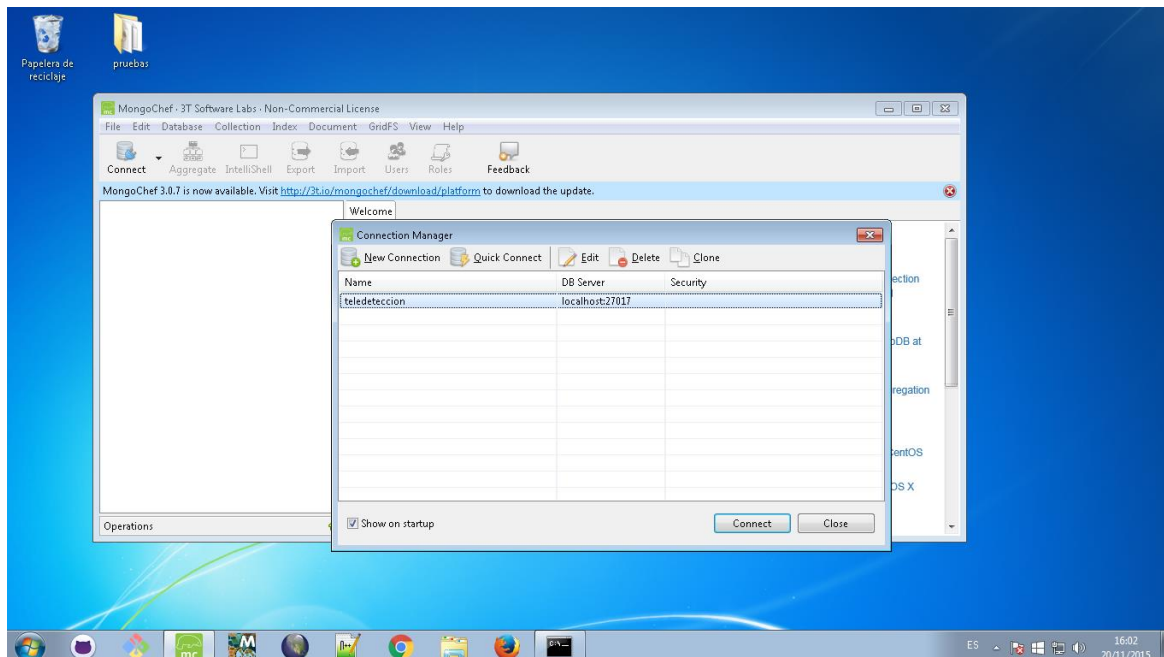


La pantalla principal es la siguiente:

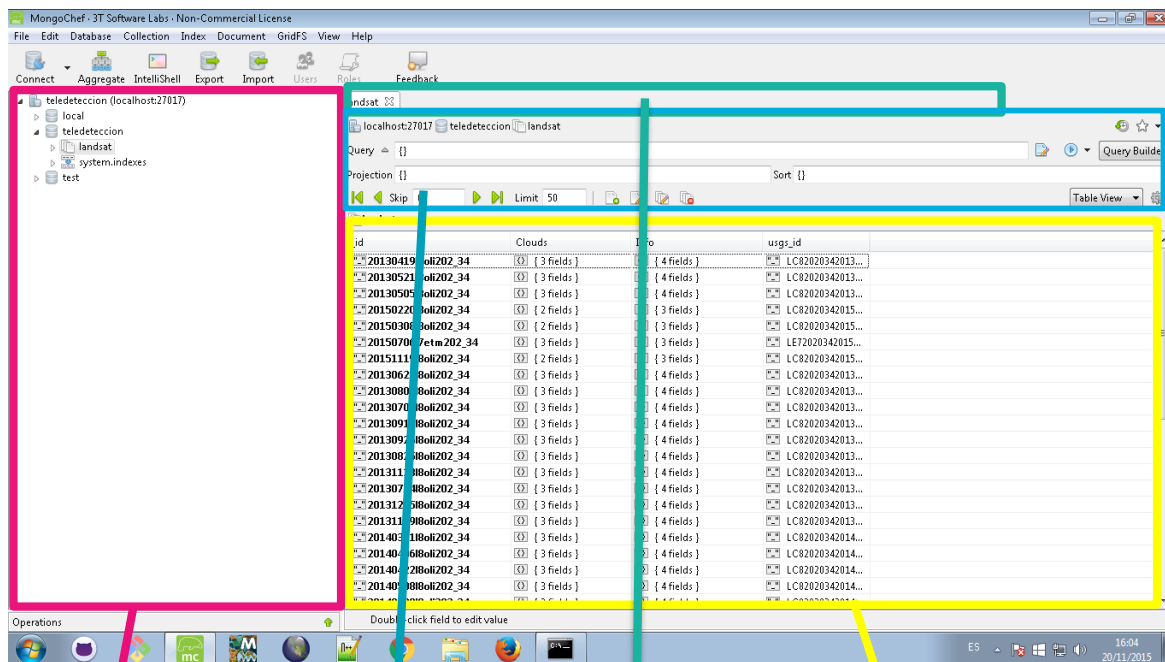
¹² En la próxima versión de MongoDB (3.2), vendrá de serie una interfaz gráfica [MongoDB Compass](#) muy potente, será cuestión de actualizar en su momento.



Una vez aquí debemos de pulsar a cualquier de los 2 iconos donde vemos “connect”, lo cual nos conectará el programa con las bases de datos disponibles. Es necesario que el servidor (**mongod**) este corriendo, ya que de lo contrario no podríamos conectar a la base de datos.



Esto nos abrirá el gestor de conexiones, donde ya por defecto se nos mostrará la conexión llamada “teledetección”, simplemente la marcamos y hacemos click en **connect**.



Conexiones
existentes

Área de
consultas

“Vista de tablas”.
Información
cargada lista para
consultarse

Vista de
documentos
(registros)

Una vez conectados al servidor de MongoDB veremos las bases de datos disponibles en el panel lateral izquierdo. En nuestro caso vemos 2 databases que se crean por defecto y nuestra base de datos que se llama **teledetección**. Si abrimos teledetección veremos las colecciones que hay dentro de ella, que en principio solo será 1 llamada **Landsat**.

Si hacemos doble click en Landsat desde el panel izquierdo se nos abrirá en la vista de tablas, la colección con todos sus elementos. Existen 3 posibilidades de visualización de los documentos: vista *json*, vista de árbol y vista de tabla.

La vista *json* es la que ya conocemos en la que se nos muestra el documento con la estructura real que tiene. Para ver el documento completo en esta vista, se trata simplemente de ir deslizándose con el ratón.

La vista en tabla es la más similar a la vista de una tabla en una base de datos relacional. En ella tenemos como columnas las claves principales del documento, donde se nos indica si se componen de más campos (y el número de ellos) o nos muestra el valor que corresponda. Si queremos profundizar al siguiente nivel solo debemos hacer doble click.

MongoChef Professional - 3T Software Labs - Non-Commercial License

File Edit Database Collection Index Document GridFS View Help

Connect Aggregate IntelliShell Export Import Users Roles Feedback

teledeteccion (localhost:27017)

- agg
- blog
- gridfs_example
- local
- m101
- reddit
- school
- teledeteccion
 - landsat
 - system.indexes
- test

landsat

localhost:27017 teledeteccion landsat

Query {}

Projection {}

Skip 0 Limit 50

Table View

id	Info	Clouds	usgs_id
20130419180i202_34	(4 fields)	(3 fields)	LC82020342013109LGN01
20130505180i202_34	(4 fields)	(3 fields)	LC82020342013125LGN01
20130521180i202_34	(4 fields)	(3 fields)	LC82020342013141LGN01
20150103180i202_34	(3 fields)	(2 fields)	LC82020342015003LGN00
20150119180i202_34	(3 fields)	(2 fields)	LC82020342015019LGN00
20150204180i202_34	(3 fields)	(2 fields)	LC82020342015035LGN00
2015070617etm202_34	(3 fields)	(3 fields)	LE720203420151875G100
20130622180i202_34	(4 fields)	(3 fields)	LC82020342013173LGN00
20130708180i202_34	(4 fields)	(3 fields)	LC82020342013169LGN00
20130724180i202_34	(4 fields)	(3 fields)	LC82020342013205LGN00
20130809180i202_34	(4 fields)	(3 fields)	LC82020342013221LGN00
20130825180i202_34	(4 fields)	(3 fields)	LC82020342013237LGN00
20130910180i202_34	(4 fields)	(3 fields)	LC82020342013253LGN00
20130926180i202_34	(4 fields)	(3 fields)	LC82020342013269LGN00
20131113180i202_34	(4 fields)	(3 fields)	LC82020342013317LGN00
20131129180i202_34	(4 fields)	(3 fields)	LC82020342013333LGN00
20131215180i202_34	(4 fields)	(3 fields)	LC82020342013349LGN00
20140406180i202_34	(4 fields)	(3 fields)	LC82020342014080LGN00
20140422180i202_34	(4 fields)	(3 fields)	LC82020342014096LGN00
20140406180i202_34	(4 fields)	(3 fields)	LC82020342014112LGN00
20140508180i202_34	(4 fields)	(3 fields)	LC82020342014128LGN00
20140609180i202_34	(4 fields)	(3 fields)	LC82020342014160LGN00

Operations 1 item selected 0.042s

MongoChef Professional - 3T Software Labs - Non-Commercial License

File Edit Database Collection Index Document GridFS View Help

Connect Aggregate IntelliShell Export Import Users Roles Feedback

teledeteccion (localhost:27017)

- agg
- blog
- gridfs_example
- local
- m101
- reddit
- school
- teledeteccion
 - landsat
 - system.indexes
- test

landsat

localhost:27017 teledeteccion landsat

Query {}

Projection {}

Skip 0 Limit 50

Table View

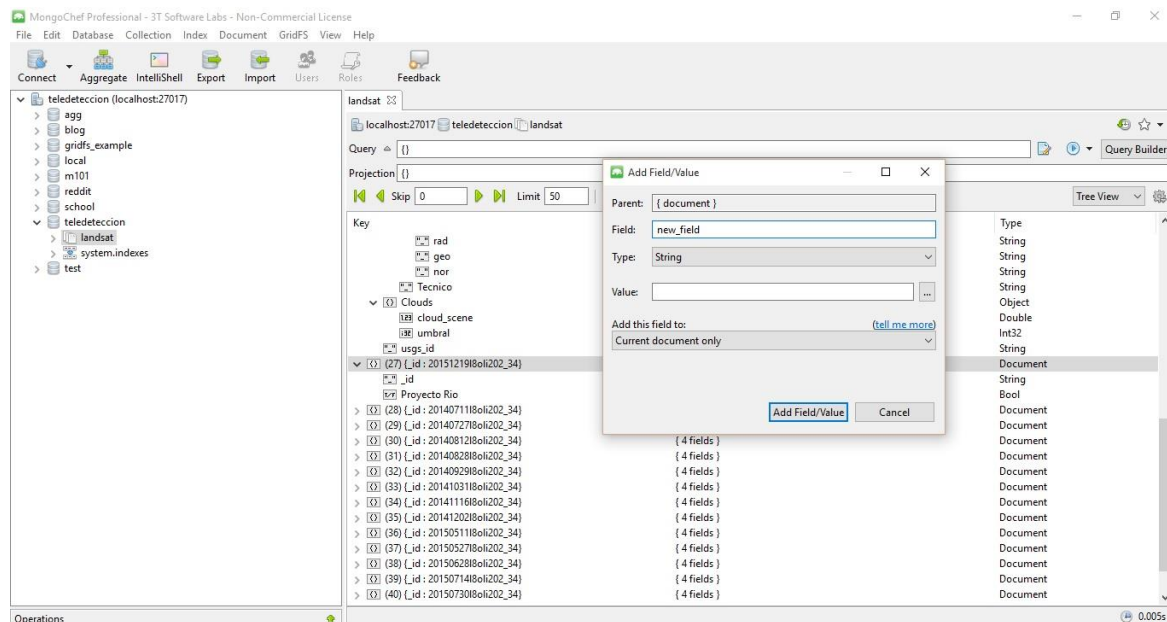
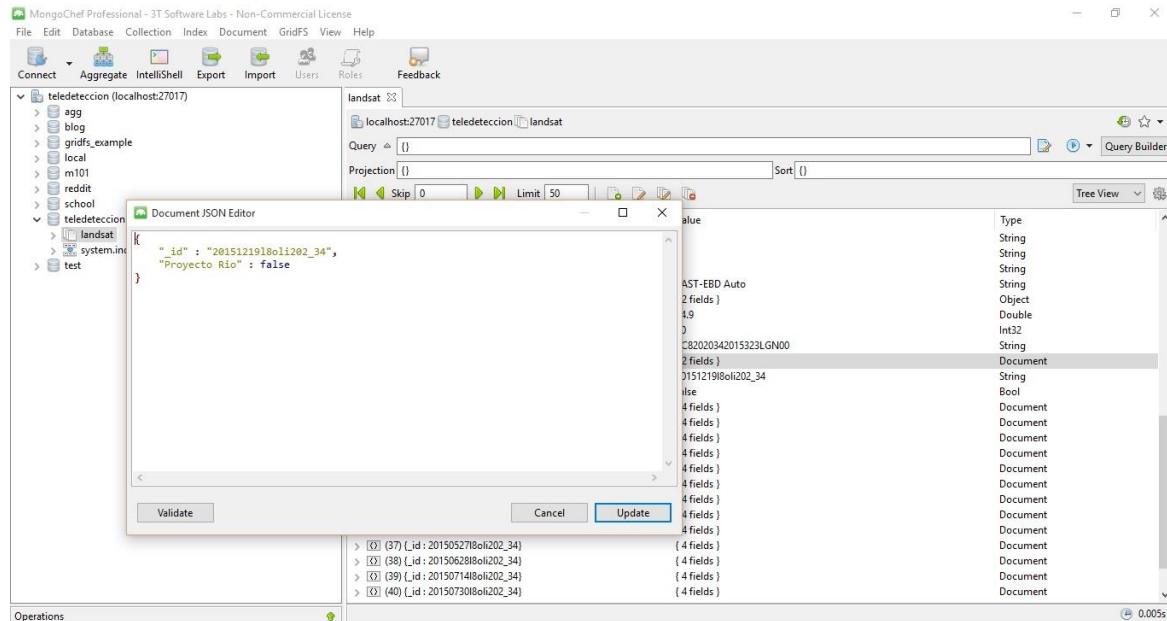
landsat> Info> Pasos> Nor-Values> b5> Parametros

Nivel en el que nos encontramos

(Document id)	slope	r	intercept	iter	N
20150714180i202_34	1.58150201766...	0.99254154898...	-12.8298272819...	1	44454
20151002180i202_34	1.56447507938...	0.99400822771...	-7.88393507952...	1	52001
20140727180i202_34	1.54139869956...	0.99588072358...	-5.04110168379...	1	33361
20130419180i202_34	1.52642980674...	0.99300584128...	-17.9469497372...	1	48150
20150527180i202_34	1.51277580506...	0.99617774895...	-8.18973927323...	1	52532
20150730180i202_34	1.50181931460...	0.99287838525...	-12.0002424317...	1	54233
20150628180i202_34	1.49212488607...	0.99371327235...	-6.10197687089...	1	55627
20150511180i202_34	1.48325379624...	0.99537009722...	-1.68864306771...	1	53576
20140609180i202_34	1.48160248846...	0.98837608686...	-21.2740220283...	1	50208
20140812180i202_34	1.47743669101...	0.99509824419...	-3.53667922349...	1	53699
20140711180i202_34	1.47351843939...	0.99448219254...	-8.00214649348...	1	54093
20140711180i202_34	1.47289025909...	0.98800505935...	-11.9772726688...	1	50034
20130724180i202_34	1.47096918528...	0.99488103760...	-6.87064946629...	1	54196
20130809180i202_34	1.46245412897...	0.99689431080...	-4.98337870690...	1	54994
20141116180i202_34	1.45736591111...	0.99501505380...	-9.48204661455...	6	3177
20130521180i202_34	1.45682684274...	0.99191924258...	-15.4840203917...	1	52159
20130505180i202_34	1.45561907744...	0.99680044079...	-8.86384417987...	1	53275
20130622180i202_34	1.45482488487...	0.99000230209...	-10.2583408211...	1	54631
20140929180i202_34	1.45240607504...	0.99437383058...	-3.05730397001...	1	52831
20130910180i202_34	1.44580110003...	0.99538036610...	-1.69243011539...	1	53335
20140508180i202_34	1.43968124457...	0.99301152582...	-13.920409555...	1	53591
20130708180i202_34	1.43830981826...	0.99757969626...	-6.18985369333...	1	55791

Operations 1 item selected 0.004s

Por supuesto desde **MongoChef** podemos editar fácilmente los documentos, lo cual podemos hacer seleccionando el documento y *clickando* con el botón derecho, por medio de las opciones “EDIT JSON” o “Add Field/Value”.



Cualquier cambio que hagamos desde aquí se guardará automáticamente en la base de datos.

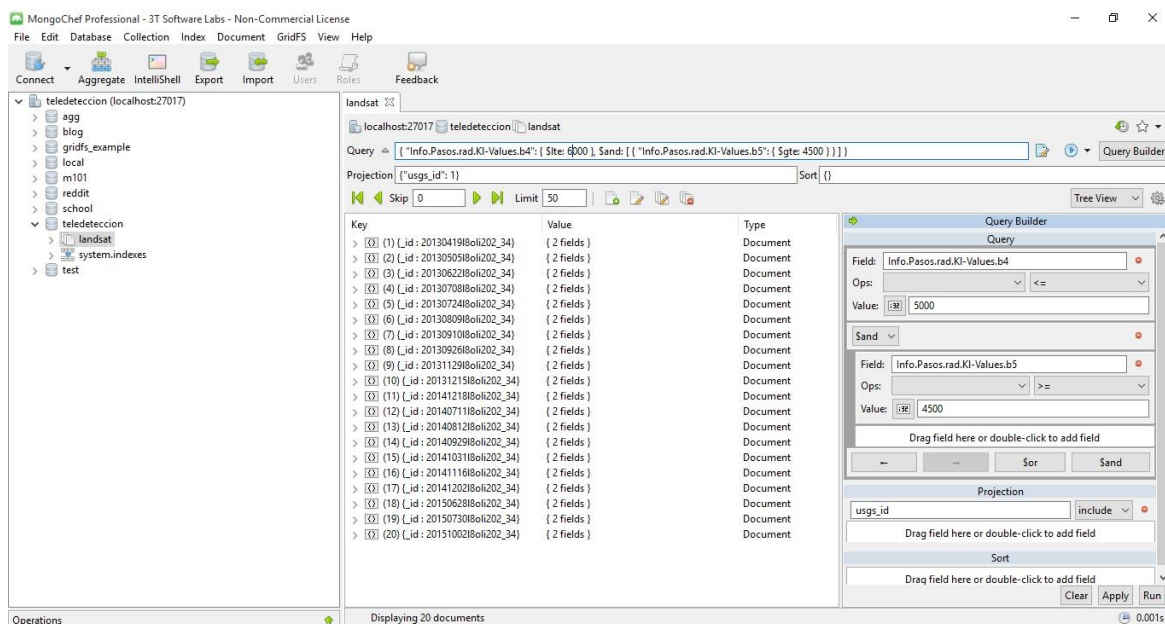
4.2.2 Consultas: Explicación

MongoChef permite la realización de consultas de forma muy sencilla e intuitiva. Dentro del panel de consultas distinguimos varias zonas:



1. Query: En esta línea es donde debemos escribir las consultas
2. Projection: Aquí especificamos los campos que queremos que se muestren de vuelta en la consulta. Por defecto la clave `_id` se mostrará siempre, si no queremos que aparezca en una consulta debemos especificarlo
3. Sort: El campo, los campos por el que queremos ordenar (puede ser de modo ascendente o descendente)
4. Skip: Si queremos saltar algunas líneas en la consulta
5. Limit: Si queremos limitar el número de documentos que la consulta puede retornar
6. 6. Herramientas de creación, edición actualización y borrado de documentos
7. Constructor de consultas

De todos los elementos, quizás el más interesante de aprender sea el constructor de consultas, ya que engloba de modo muy simple, y a través de una interfaz gráfica, al resto de los elementos.



Por medio de esta interfaz podemos ir añadiendo campos a la consulta, definir la proyección que queremos, ordenar los resultados por campos, etc. Al darle a *Apply* se nos cargará la consulta realizada por esta interfaz, en la línea de consulta. Por lo que nos puede servir para aclarar dudas a la hora de realizar el diseño de alguna consulta.

En cualquier caso, dado que las consultas no son equivalentes a las consultas en el mundo relacional (aquí no se utiliza SQL), merece la pena dedicar unas líneas a explicar cómo funcionan las consultas en NoSQL y cuál es la sintaxis que utilizan.

En **MongoDB** como ya sabemos trabajamos con archivos *json*¹³ (JavaScript Object Notation), y las consultas para recuperar documentos de esta base de datos, emplea la misma sintaxis *json*.

Si queremos recuperar un valor cuyo `_id` conocemos, para recuperarlo solo tenemos que escribir:

```
db.landsat.find({ _id : '20140812l8oli202_34' })
```

La sintaxis es **db (base de datos) + “.” + colección + “.” + find({})**. Esto lo haríamos desde una ventana del cmd, si realizamos la consulta desde MongoChef la parte **db.colección.find()** estaría incluida en nuestra consulta sin necesidad de escribirlo, con lo que solo tendríamos que escribir la parte que va entre las llaves, que es donde se especifica el par “clave : valor” que queremos buscar.

Si quisiéramos especificar la salida, tendríamos que añadir otro *json* con las claves que queremos que aparezcan en la salida de la consulta. Si no se especifica nada, saldrá toda la información por defecto, si por el contrario queremos definir una o varias claves para la salida debemos indicarlo añadiendo lo siguiente: “clave : 1” (True) o “clave: 0” (False).

```
db.landsat.find({ _id : '20140812l8oli202_34' }, { USGS_id : 1, _id : false })
```

En este caso estaríamos buscando la escena ‘20140812l8oli202_34’, pero solo queremos que nos dé como salida solo el USGS_id. Si no hubiéramos añadido “_id : 0” saldría por defecto el _id. Al igual que antes la parte `db.landsat.find({})` no haría falta ponerla desde **MongoChef**, tan solo lo que va entre las llaves.

Por supuesto las consultas se pueden complicar *ad infinitum*, mediante la combinación de subconsultas con los métodos “or”, “and”, etc. También dispone de operadores como `$lt` (less than), `$lte` (les sor equal tan), `$gt` (greater than), etc...

Dada la amplitud de esta temática, en lugar de tratar de explicar todas las consultas disponibles en MongoDB, vamos a poner a continuación un apartado con múltiples consultas, para que puedan servir de guía para la elaboración de consultas futuras.

¹³ Realmente en MongoDB se emplea *bson*, una implementación binaria del *json*.