UTILISATION DES ENVIRONNEMENTS CONDA : ILLUSTRATION VIA L'INSTALLATION DE SCIKIT-LEARN-INTELEX

Dans ce document nous travaillerons avec la version optimisée de scikit-learn proposée par Intel. Si vous désirez employer la version standard, il vous suffit d'importer sklearn et évidemment ne pas faire référence à la bibliothèque optimisée sklearnex.

Par ailleurs nous supposons que vous utilisez Visual Studio Code comme éditeur de code source python ainsi que l'environnement Anaconda[i]. On suppose naturellement également que vous avez au minimum installé l'extension python de Microsoft dans VS Code[ii]. Avant toute chose, comme Conda et PowerShell ne font pas toujours bon ménage, je vous conseille si nécessaire de désélectionner 'PowerShell' en sélectionnant 'Command Prompt' par défaut via le menu 'Launch Profile' accessible dans la fenêtre du terminal ouvert par VS Code.

Cette installation se fera dans un environnement virtuel, qui sera nommé ici '*envsk'* différent de l'environnement '*base'* créé par Anaconda, mais vous pouvez évidemment le nommer autrement.

1- Création d'un environnement virtuel conda et installation des packages

Pour cette installation 3 lignes de codes suffisent. La seule condition étant qu'elles doivent être entrées dans la fenêtre de commande Anaconda (éventuellement Anaconda Prompt (miniconda 3) si vous avez suivi nos conseils). Dans celle-ci vous devriez voir une invite de commande précédée de (base) qui est l'environnement activé par défaut et de l'indication précisant la directory dans laquelle se situe cet environnement `base` et vous indiquant que les commandes vont s'exécuter dans cet environnement. Vous entrez alors les commandes ::

```
1. conda config --add channels conda-forge
2. conda config --set channel_priority strict
3. conda create -n envsk python scikit-learn-intelex
```

Les deux premières assurent que la recherche des packages via `conda install` s'effectuera en priorité sur le dépôt `conda-forge` sur lequel se trouve généralement les versions les plus récentes des packages que nous rechercherons. La troisième créé l'environnement

virtuel de nom `envsk` et y installe notamment les versions les plus récentes de `python`, `sklearn` et `sklearnex`[iii] compatibles entre elles. Pour rappel, ce ne sont pas obligatoirement les plus récentes de chacun de ces packages.

Pour vérifier que tout s'est bien passé vous pouvez entrer l'instruction :

```
conda env list
```

en réponse elle devrait vous donner la liste des environnements virtuels connus par Conda, ainsi que leur emplacement sur votre disque. Si vous avez installé Miniconda normalement vous devriez y retrouver les environnements `base` et `envsk` associé à des chemins du type :

- c/\users\username\miniconda3\base
- c/\users\username\miniconda3\envs\envsk

Notez que par défaut les environnements virtuels, à l'exception de `base`, sont stockés dans la sous-directory envs de la directory miniconda3.

A ce stade cet environnement envsk contient notamment la version standard `scikit-lean`, la version avec les algorithmes optimisés, `sckit-leanr-intelex` ainsi que `numpy` et évidemment Python.

Il est alors possible de travailler dans ce nouvel espace. Pour cela toujours dans la fenêtre Anaconda Prompt, on va activer l'environnement envsk avec :

```
conda activate envsk
```

Comme à tout moment on ne peut avoir qu'un seul environnement actif, cette commande va également désactiver `base.`

Les invites de commandes doivent être maintenant précédée de (envsk) rappelant que les commandes suivantes s'exécuteront dans cet espace. Par exemple on peut y installer Matplotlib pour les graphiques ainsi que pandas pour la gestion des DataFrames en effectuant simplement :

```
(envsk) conda install matplotlib pandas
```

2- Association de VS Code à un dossier de travail et à l'environnement virtuel

L'environnement virtuel python envsk dans lequel nous voulons travailler est maintenant complet. Il reste à créer une directory de travail dans laquelle seront stockés nos fichier .py ou .ipynb, et à lancer VS Code en lui faisant comprendre quelle est cette directory et dans quel environnement il doit exécuter les programmes qui s'y trouvent.

Pour la création de la directory, vous utilisez la démarche usuelle dans Windows: Nouveau Dossier à l'endroit que vous désirez et auquel vous donnez le nom de votre choix.

Pour l'association de VS Code à ce dossier, plusieurs façons de procéder existent.

2.1- une première solution :

L'une des plus simples consiste à se placer dans le dossier en question en envoyant des commandes dans la fenêtre Anaconda Prompt. Pour cela vous disposez des commandes habituelles cd, D :, cd.., etc. Cela fait, vous devriez alors voir dans cette fenêtre une invite de commande comme :

```
(envsk) C:\Users\<UserName>\...\nom du dossier>
```

Vous appelez alors VS Code en envoyant la commande code . (n'oubliez pas le point !).

Par exemple si avez créé un dossier de nom `projet` sous `D:`,il faut que dans la fenêtre Anaconda prompt s'affiche :

```
(envsk) D:\projet>code .
```

L'exécution de cette commande va lancer une session VS Code dans laquelle votre dossier de travail sera projet, i.e. les opérations de création, de sauvegarde et de lecture de fichiers s'effectueront dans cette directory, et dont les exécutions se dérouleront dans l'environnement Conda de nom envsk en disposeront de toutes les ressources que vous y avez installées.

Pour vous assurer que tout fonctionne, créez et lancez un programme simple du genre print('hello'). Dans le terminal de VS Code vous devriez voir passer une commande

```
conda activate envsk
```

qui sera automatiquement exécutée, de sorte que les invites suivantes seront précédées de (envsk), ce que vous pouvez aussi vérifier visuellement en bas à droite de la fenêtre VS Code où vous devez voir ('envsk' :conda), sinon vous l'activez par une commande conda activate envsk . L'interpréteur Python utilisé sera naturellement celui qui est présent dans cet environnement[iv]. On récapitule les étapes nécessaires :

```
1. Création du dossier de travail
2. Se placer dans ce dossier via la fenêtre Anaconda Prompt
3. Activer l'environnement Conda que l'on veut utiliser
4. Lancer code .
```

2.2- Une seconde solution :

Dans le menu File de VS Code vous choisissez `Open Folder` et sélectionnez votre dossier qui va dès lors devenir votre dossier de travail.

Il reste à sélectionner l'environnement Conda que vous désirez utiliser, pour cela :

- Vous pouvez passer par le menu `View` et choisissez `Command Palette` dans la zone d'invite qui s'ouvre vous tapez `Python Select Interpreter` et sélectionnez dans les environnements Conda qui s'affichent celui dans lequel vous voulez vous situer. Normalement dans le `Terminal` de VS Code vous devriez voir passer une commande d'activation et à droite dans la dernière ligne de l'éditeur une indication rappelant la version de Python et le nom de l'environnement, par exemple 3.11.7(envsk), et des invites de commande dans le `Terminal` qui commencent par (envsk).
- Sachez que le raccourci Ctrl+Shift+P permet un accès direct à l'invite de la Command Palette.
- Sachez aussi que si un environnement est déjà actif, par exemple 3.11.7(envsk) est visible en bas à droite, en double cliquant sur cette zone vous ferez immédiatement apparaître la liste des environnements connus et pourrez faire votre choix dans cette liste.

En résumé :

```
1.Créer le dossier de travail et le sélectionner dans VS Code
2.Chercher 'Python Select Interpreter' dans la Command Palette
3.Sélectionner l'environnement désiré dans la liste
```

Les associations entre l'éditeur de code, un dossier de travail et un environnement Anaconda étant réalisées, on passe maintenant au dernier objectif de cette illustration : faire en sorte que les commandes `scikit-learn` fassent appel à la version optimisée et non pas à la version standard. Pour cela il faut activer le processus de substitution des algorithmes optimisés aux algorithmes de cette version standard via la technique du `patching`.

3- Utilisations des algorithmes optimisés : mise en œuvre du patching

Deux choses à savoir :

- Les algorithmes optimisés portent le même nom que ceux de la version standard. L'avantage est que pour passer d'une version à l'autre aucun changement de code n'est nécessaire du côté de l'utilisateur : tous les programmes écrits pour `sklearn` s'exécuteront sur `sklearnex`,
- Tous les algorithmes de `scikit-learn` ne sont pas optimisés[v] mais le `patching` fait en sorte de rendre ceci totalement transparent : si un algorithme optimisé est disponible alors il est utilisé, sinon la version standard est employée. **Évidemment pour que cela soit possible il faut avoir importé `scikit-learn` et chose importante, cette importation doit être effectuée après que les commandes ou lignes de codes activant le `patching` aient été exécutées**.

L'activation du `patching` pour la session en cours peut se faire soit par un bout de code dans le programme python concerné, soit dans la commande qui lance le programme dans le terminal de VS Code, soit enfin via un `global patching` qui appellera les algorithmes optimisés pour la session courante et toutes les sessions ultérieures se déroulant dans le même environnement. Cette dernière méthode qui est réversible est naturellement la plus pratique.

1. Activation du `patching` dans le programme :
   on ajoutera dans le script python les instructions[vi]

   ```
   from sklearnex import patch_sklearn
   patch_sklearn()
   import sklearn
   ```

2. Activation via la commande de lancement du programme concerné :
   dans le terminal de VS Code on exécutera :

   ```
   python -m sklearnex nom_du_programme.py
   ```

   Dans ce programme on doit évidemment trouver l'instruction `import sklearn`.

3. Activation du `global patching` :

3.1- soit avec la ligne de commande suivante dans le terminal VS Code :

```
python -m sklearnex.glob patch_sklearn
```

3.2- soit via un script python :

```
from sklearnex import patch_sklearn
patch_sklearn(global=True)
import sklearn
```

Sachez aussi qu'il est possible de faire du `global patching` sur des algorithmes spécifiques, par exemple avec :

```
python -m sklearnex.glob patch_sklearn -a SVC NearestNeighbors
```

seuls les deux estimateurs `SVC` et `NearestNeighbors` seront remplacés par leurs versions optimisées pour la session courante et les sessions ultérieures.
Les arguments `-a` et `--algorithm` pouvant être indifféremment employés.

Enfin, pour mettre fin au `global patching`, deux solutions peuvent indifféremment être utilisées :

3.4.1- via le terminal en exécutant la commande :

```
Python – m sklearnex.glob unpatch_sklearn
```

3.4.2- via un code python : les deux premières lignes ci-dessous réalisent le `unpatching` de `sklearn`. Elles doivent être suivies de la l'importation des modules standards que l'on veut utiliser dans la session courante

```
from sklearnex import unpatch_sklearn
unpatch_sklearn(global_patch=True)
from sklearn.ensemble import RandomForestClassifier
```

Pour conclure : vous pouvez trouver des notebooks d'exemples d'utilisation de `sklearnex` sur différents algorithmes ainsi que des évaluations des gains de performances à l'adresse https://www.kaggle.com/code/lordozvlad/introduction-to-scikit-learn-intelex.

---

[i] On vous conseille alors d'installer Miniconda et ensuite d'épingler 'Anaconda Prompt' à la barre des tâches pour accéder rapidement à la fenêtre de commande Anaconda (qu'il ne faut pas confondre avec l'invite de commande de Windows).

[ii] Ce qui vous permet évidemment d'exécuter des programmes écrits en python possédant généralement l'extension '.py'. Non nécessaire ici mais utile, pensez aussi à l'extension Jupyter de Microsoft qui rend possible d'exécuter des notebooks Jupyter, ayant l'extension '.ipynb', à l'intérieur de VS Code. L'extension Intellicode, également offerte par Microsoft, peut s'avérer utile pour vous proposer gratuitement des complétions de code plus riches que celles données par Intellisense qui est installée par l'extension python. Pour info, les extensions Python Indent et Python Rainbow sont sûrement moins utiles mais néanmoins sympatiques.

[iii] Scikit-learn sera en effet installé car c'est une dépendance de scikit-learn-intelex. Il en va de même pour Numpy.

[iv] Si vous tentez d'exécuter un programme .ipynb créé sous un autre environnement il vous sera demandé de préciser un kernel python. Dans la liste des environnements conda reconnus vous sélectionnez celui que vous désirez, par exemple ici 'envsk'. Il est possible que VS Code vous demande d'installer IPython, interpréteur interactif utilisé par Jupyter, et il vous donne alors la commande à exécuter, ce qui peut être fait dans le terminal de VS Code. Naturellement vous l'installez et cela fait votre programme devrait pouvoir être exécuté.

[v] La liste des algorithmes optimisés est visible à https://intel.github.io/scikit-learn-intelex/latest/algorithms.html

[vi] Ces lignes autorisent l'appel de n'importe quel algorithme optimisé. Il est possible de limiter l'appel à un algorithme spécifique par exemple avec : from sklearnex.neighbors import NearestNeighbors