TD : `sklearn` & sélection de modèles

Romain Tavenard

Dans cette séance, nous nous focaliserons sur la sélection de modèle pour la classification supervisée avec `sklearn`.

1 Préparation des données

Nous allons travailler, pour ce TD, sur un jeu de données ultra classique en *machine learning* : le jeu de données "Iris". Ce jeu de données est intégré dans `sklearn` pour être utilisé facilement.

1. Chargez ce jeu de données à l'aide de la fonction `load_iris` du module `sklearn.datasets`. Faites en sorte de stocker les prédicteurs dans une matrice `X` et les classes à prédire dans un vecteur `y`. Quelles sont les dimensions de `X` ?
2. Découpez ce jeu de données en un jeu d'apprentissage et un jeu de test de mêmes tailles et faites en sorte que chacune de vos variables soient centrées-réduites.

2 Le modèle `SVC` (*Support Vector Classifier*)

Lorsque l'on souhaite faire de la classification supervisée avec `sklearn`, le fonctionnement sera toujours le même :

```
mon_classifieur = NomDeLaClasse(a=12, b=3, c=None)
mon_classifieur.fit(X_train, y_train)
mon_classifieur.predict(X_test)
```

où `a`, `b` et `c` sont les hyper-paramètres du modèle.

À vous donc de dénicher la classe `sklearn` correspondant au modèle de votre choix et d'éplucher sa documentation pour comprendre le fonctionnement des différents hyper-paramètres associés au modèle.

3. Apprenez un modèle SVM linéaire (classe `SVC` dans `sklearn`) pour votre problème.

4. Évaluez ses performances sur votre jeu de test à l'aide de la fonction `accuracy_score` du module `sklearn.metrics`.
5. Faites de même avec un modèle SVM à noyau gaussien. Faites varier la valeur de l'hyperparamètre lié à ce noyau et observez l'évolution du taux de bonnes classifications.

3 Validation croisée

Il existe dans `sklearn` de nombreux itérateurs permettant de faire de la validation croisée, qui sont listés sur cette page.

6. Définissez un objet `cv` de la classe `KFold`. Exécutez le code suivant :

```
for train, valid in cv.split(X_train, y_train):
print(train, valid)
```

   Qu'est-ce qui est affiché ?
7. Faites de même avec des objets des classes `StratifiedKFold` et `LeaveOneOut` et vérifiez que, même en ne mélangeant pas les données (c'est-à-dire sans spécifier `shuffle=True`), les découpages obtenus sont différents.

4 Sélection de modèle

En pratique, vous n'utiliserez pas vous même ces appels aux méthodes `split()` des itérateurs de validation croisée, car il existe dans `sklearn` un outil très pratique pour vous aider lors de votre étape de sélection de modèle.

Cet outil s'appelle `GridSearchCV`. Là encore, il s'agit d'une classe `sklearn`, et vous l'utiliserez quasiment de la même manière qu'un classifieur, à la nuance près des paramètres que vous passerez lors de la construction d'une nouvelle instance. Nous verrons dans ce TD trois de ces paramètres :

— `estimator` est un classifieur (créé mais pas encore appris sur des données) ;
— `param_grid` est une grille d'hyper-paramètres à tester pour ce classifieur ;
— `cv` est un itérateur de validation croisée, tel que l'un de ceux définis à la section précédente.

Le paramètre `param_grid` est un dictionnaire (ou une liste de dictionnaire, comme on le verra plus tard) dont les clés sont les noms des hyper-paramètres à faire varier et les valeurs associées sont des listes de valeurs à tester [1].

1. Pour générer ces listes, on pourra avoir recours aux fonctions `numpy.linspace` et `numpy.logspace`.

8. Reprenez le cas d'un classifieur SVM linéaire et faites varier l'hyper-paramètre `C` entre 1 et 10 (en prenant 5 valeurs espacées régulièrement).
9. Affichez les paramètres du modèle choisi par la validation croisée. Évaluez les performances de ce modèle sur votre jeu de test.
10. Parfois, certains hyper-paramètres sont liés entre eux. Dans le cas du SVM par exemple, le choix d'un noyau implique de régler certains hyper-paramètres spécifiques (*ex.* : le paramètre `gamma` du noyau Gaussien). Dans ce cas, on peut définir `param_grid` comme une liste de dictionnaires, chaque dictionnaire correspondant à un cas de figure. Utilisez cette possibilité pour choisir le meilleur modèle pour votre problème entre un SVM linéaire et un SVM à noyau Gaussien (pour les deux, on fera varier `C` entre 1 et 10, et pour le noyau Gaussien, on fera de plus varier `gamma` entre 10^{-2} et 100 sur une échelle logarithmique).
11. Étendez cette approche à un autre classifieur supervisé de votre choix et comparez ses performances à celles du meilleur modèle SVM trouvé jusqu'alors.

5 La notion de *Pipeline*

Bien souvent, pour mettre en oeuvre une solution de *machine learning*, vous allez passer par plusieurs étapes successives de transformation de vos données avant de les fournir à votre classifieur. Il est possible que ces pré-traitements aient, eux aussi, des hyper-paramètres à régler. Pour pouvoir sereinement prendre en compte toutes les configurations possibles, `sklearn` définit la notion de `Pipeline`.

12. Modifiez vos données d'origine pour mettre des `numpy.nan` à toutes les valeurs plus grandes que 2 (en valeur absolue).
13. Créez un *pipeline* qui soit constitué des 3 étapes suivantes :

    a. une imputation des valeurs manquantes ;
    b. une standardisation des données ;
    c. une classification supervisée par un classifieur de votre choix.

14. Mettez en place une validation croisée qui permette de choisir si l'imputation doit se faire par valeurs médianes ou moyennes et qui détermine également un ou plusieurs hyper-paramètres du classifieur que vous avez choisi.
15. Chargez maintenant de nouvelles données à l'aide de la fonction `load_digits` du module `sklearn.datasets`. Imaginez un *pipeline* qui consiste à effectuer tout d'abord une ACP sur ces données puis une régression logistique dans l'espace de l'ACP. Mettez en place une validation croisée pour choisir de manière optimale les paramètres de ces deux étapes de votre *pipeline*. Comparez votre solution à celle disponible là.