De la classification de chiffres manuscrits aux courbes d'apprentissage

Les fichiers à télécharger pour ce TP sont `tp_digits_learning_curve.py` et `learning_curve.py`.

- Découverte de Python -

Consulter les pages suivantes pour démarrer ou bien trouver quelques rappels

⋆⋆⋆ http://scipy-lectures.github.io

⋆⋆⋆ http://perso.telecom-paristech.fr/~gramfort/cours_python/1-Intro-Python.html

⋆⋆⋆ http://perso.telecom-paristech.fr/~gramfort/cours_python/2-Numpy.html

⋆⋆⋆ http://perso.telecom-paristech.fr/~gramfort/cours_python/3-Scipy.html

⋆⋆⋆ http://scikit-learn.org/stable/index.html

⋆⋆ http://www.loria.fr/~rougier/teaching/matplotlib/matplotlib.html

⋆⋆ http://jrjohansson.github.io/

- Classification de chiffres manuscrits -

On se propose dans ce TP de classifier des images de chiffres manuscrits, "digits" en Anglais.

Définitions et notations

On rappelle que dans la cadre de la classification supervisée on utilise les notations :

- $\mathcal{Y}$ l'ensemble des étiquettes (ou *labels* en anglais), communément $\mathcal{Y} = \{0, K-1\}$ dans le cas multi-classes,
- $\mathbf{x} = (x_1, \ldots, x_p) \in \mathcal{X} \subset \mathbb{R}^p$ est une observation, une donnée (ou un *sample* en anglais),
- p désigne le nombre de variables, d'attributs (ou *features* en anglais),
- $\mathcal{D}_n = \{(\mathbf{x}_i, y_i), i = 1, \ldots n\}$ un ensemble d'apprentissage contenant n exemples et leurs étiquettes,
- Il existe un modèle probabiliste qui gouverne la génération de nos observations selon des variables aléatoires X et Y : $\forall i \in \{1, \ldots, n\}, (\mathbf{x}_i, y_i) \overset{i.i.d}{\sim} (X, Y)$.
- On cherche à construire à partir de l'ensemble d'apprentissage $\mathcal{D}_n$ une fonction $\hat{f} : \mathcal{X} \mapsto \{0, K-1\}$ qui pour un point inconnu (*i.e.,* qui n'est pas présent dans l'ensemble d'apprentissage) prédit son étiquette : $\hat{f}(\mathbf{x})$.

Prise en main de la base de données

La base de données que l'on utilisera est *digits* fournie par la librairie Scikit-Learn. Le code pour accéder aux données est le suivant :

```
from sklearn.datasets import load_digits
digits = load_digits()
X, y = digits.data, digits.target
```

1. Quel est le type de `X` et `y` et quelles sont les dimensions des données ? Comment voit-on qu'il s'agit d'un problème de classification ? Que valent les variables n et p dans notre cas ?

2. Le tableau `X` a 64 colonnes. Utiliser le code suivant pour explorer le contenu de la base.

```
import numpy as np
import matplotlib.pyplot as plt
plt.imshow(np.reshape(X[0], (8, 8)), cmap=plt.cm.gray, interpolation='nearest')
plt.show()
```

3. Combien y-a-t-il de classes dans la base de données ? Utiliser la fonction `np.unique` de Numpy.
4. Pour chaque classe k, calculer la moyenne et la variance de la variable numéro i (ici pixel). On notera ces quantités $\mu_{i,k}$ et $\sigma_{i,k}$. Visualiser à l'aide du code précédent ces valeurs sous forme d'image. Vous utiliserez les fonctions `np.mean` et `np.var` de Numpy pour calculer moyennes et variances.
5. L'objet `StandardScaler` de Scikit-Learn permet de standardiser les variables. Pourquoi fait-on cela en pratique ?

Mise en oeuvre

1. Mettre en oeuvre un algorithme de classification de votre choix sur ces données
2. Comment évaluer la performance de la méthode ?

Validation croisée

Il s'agit ici d'évaluer la performance du modèle par validation croisée et d'observer l'impact de la taille de l'ensemble d'apprentissage sur les résultats. On pourra utiliser la fonction `sklearn.cross_validation.cross_val_score` pour estimer les erreurs de prédiction en validation croisée.

1. Avant tout calcul quelle doit être la forme de la courbe de validation croisée en fonction de la taille de l'ensemble d'apprentissage ?
2. Vérifier cette prédiction numériquement pour différent types de classifieurs.

Courbe d'apprentissage

La validation croisée reporte la performance du classifieur sur des nouvelles données. On parle de performance en généralisation. En pratique, il est aussi pertinent d'évaluer la performance sur les données d'apprentissage car cela permet d'évaluer si le classifieur est assez *complexe* et si il sur-apprend (*overfit* en anglais). Par exemple un classifieur linéaire est moins complexe qu'un classifieur non-linéaire qui apprend sur une plus grande classe de fonctions.

1. Avant tout calcul quelle doit être la forme de la courbe de performance sur l'ensemble d'apprentissage en fonction de sa taille ?
2. Vérifier cette prédiction numériquement en utilisant la fonction `plot_learning_curve` fournie dans le fichier `learning_curve.py`. On comparera la courbe obtenue avec celle d'un SVM (Support Vector Machine) accessible dans Scikit-Learn accessible avec le code suivant :

```
from sklearn.svm import SVC
clf = SVC(gamma=0.001)
```

   Que peut-on en conclure ? En quoi de telles courbes peuvent-elles permettre de gagner une compétition de machine learning ?