Intelligence artificielle & Machine learning
TP discrimination multi-classes : Reconnaissance de caractères manuscrits par l'algorithme des k plus proches voisins

F. Lauer

Pré-requis

Les algorithmes de machine learning sont implémentés en python dans la bibliothèque scikit-learn (`sklearn`). En règle générale, seule la partie utilisée de la bibliothèque est importée. Par exemple

```
from sklearn import neighbors
```

importe les outils liés aux plus proches voisins.

L'algorithme des K-plus proches voisins (Kppv) (*KNN pour K-nearest neighbors* en anglais) pour la classification s'utilise ensuite ainsi :

```
kppv = neighbors.KNeighborsClassifier(K)  # création du modèle
kppv.fit(X, Y)                            # apprentissage sur X,Y
```

où

$$\texttt{X} = \begin{bmatrix} \boldsymbol{x}_1^T \\ \boldsymbol{x}_2^T \\ \vdots \\ \boldsymbol{x}_m^T \end{bmatrix}, \quad \texttt{Y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}$$

contiennent la base d'apprentissage.

Pour calculer un vecteur de prédictions `Ypred` sur les points stockés en lignes dans la matrice `Xtest`, on utilise ensuite :

```
Ypred = kppv.predict(Xtest)
```

Les fonctions `.fit()` et `.predict()` sont les méthodes standards pour l'apprentissage et la prédiction pour tous les modèles de classifieur ou de régression dans scikit-learn.

1 Test de l'algorithme dans le plan

1.1 Classification binaire linéairement séparable

Dans le programme `kppv2D.py` :

1. Générez deux groupes de 100 points en 2D avec la souris et approximativement selon la distribution correspondant à $Y \in \{-1, +1\}$, $P(Y = 1) = 0.5$ et

$$p(x|Y=1) = \begin{bmatrix} 3 \\ 3 \end{bmatrix} + \sigma_1 V, \qquad p(x|Y=-1) = \begin{bmatrix} -3 \\ -3 \end{bmatrix} + \sigma_2 V, \qquad V \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$$

2. Pour visualiser la classification du plan obtenue par les Kppv à partir de ces données, nous allons tracer une grille de points sur la totalité du graphique dont la couleur sera prédite par le classifieur Kppv.

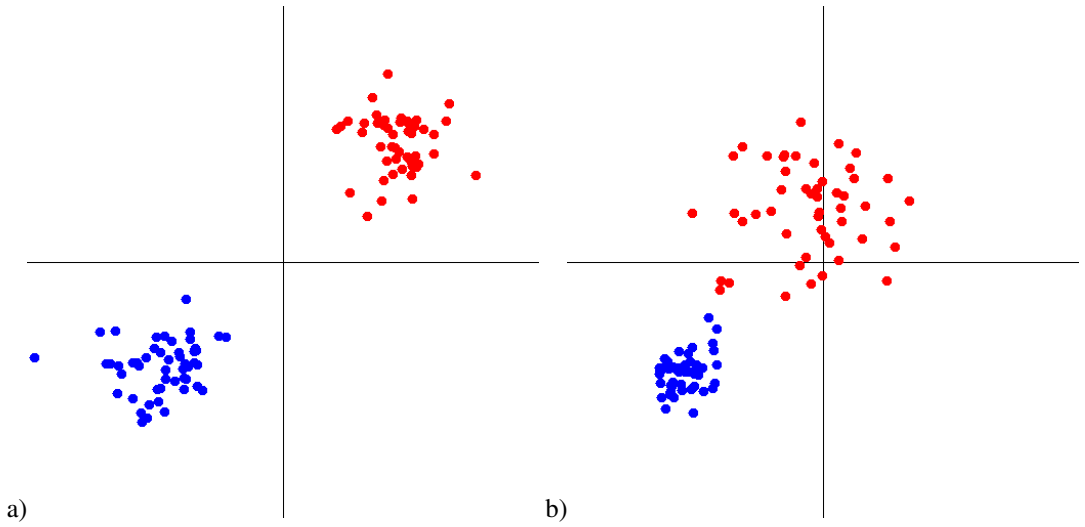

FIGURE 1 – Exemples de bases d'apprentissages linéairement séparables.

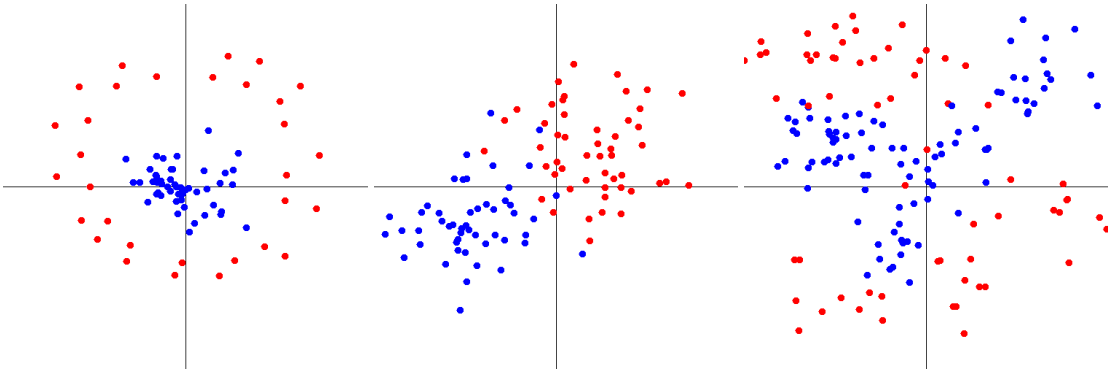

FIGURE 2 – Exemples de bases d'apprentissages non linéairement séparables.

Tracez cette figure pour la base d'apprentissage décrite ci-dessus avec $\sigma_1 = \sigma_2 = 1$ et similaire à la Fig. 1-a). De quel type de surface de séparation se rapproche la frontière donnée par l'algorithme Kppv ici ?

3. Idem mais en utilisant des variances très différentes (voir l'exemple Fig. 1-b). *De quelle forme se rapproche maintenant la frontière de séparation ?*

1.2 Cas non séparable

Testez maintenant l'algorithme sur des données non linéairement séparables (vous pouvez vous inspirer des exemples de la Fig. 2).
Quelle est l'influence du nombre de voisins sur la surface de séparation ?

1.3 Problèmes multi-classes

Ajoutez une catégorie supplémentaire et visualiser le résutlat obtenu. *Observe-t-on les mêmes phénomènes ?*

2 Application à la reconnaissance de caractères

Dans cette partie du TP, nous allons créer un système de reconnaissance de chiffres manuscrits à partir de la base d'images MNIST [1] utilisée par un grand nombre de chercheurs dans le monde entier. Cette base contient 70000 exemples de chiffres manuscrits habituellement découpés en 60000 exemples d'apprentissage et 10000 exemples de test. Chaque exemple est représenté par le niveau de gris en chaque pixel d'une image de 28×28 pixels.

2.1 Base de données MNIST

Récupérez le fichier `kppvmnist.py` permettant de charger `m` exemples d'apprentissage et `mtest` exemples de test de la base MNIST avec

```
Xtrain, Xtest, ytrain, ytest = load_mnist(m, mtest)
```

Chaque ligne de `Xtrain` (ou `Xtest`) représente une image sous la forme d'un vecteur de 784 niveaux de gris entre 0 et 255. Les étiquettes correspondantes sont dans les vecteurs `ytrain` et `ytest`.

Vous pouvez afficher la $i$ème image de la base d'apprentissage avec

```
showimage(Xtrain[i])
```

(attention : cela bloque le programme en attendant la fermeture de la fenêtre). Pour se faire une idée de la qualité des images, le programme `showmnist.py` (à télécharger sur ARCHE) permet de faire défiler les images de la base en fermant à chaque fois la fenêtre (ctrl+C pour quitter).

2.2 Classification par les K plus proches voisins

Classer toute la base de teste en sa basant sur les 60000 exemples de la base d'apprentissage peut prendre du temps. Nous considérerons donc dans un premier temps qu'une base d'apprentissage de 10000 exemples et une base de test de 1000 exemples.

1. Commencez par créer un classifieur Kppv à partir de 10000 exemples. Quelle est la taille approximative du modèle en octets ?
2. Utilisez le modèle pour prédire les étiquettes des 1000 exemples de test et estimez le risque du classifieur.
3. Si les calculs sont trop longs, pour aider à patienter, réalisez une boucle pour calculer les prédictions sur 100 exemples à la fois et afficher un petit message. Par exemple :
   ```
   Exemples 1-100 : 22 erreurs
   Exemples 101-200 : 5 erreurs
   ...
   ```
4. Testez avec différentes valeurs de K et déterminez la meilleure valeur entre 1 et 10 pour cet hyperparamètre.
5. Analysez les erreurs de classification en regardant de plus près les exemples mal classés par l'algorithme (y a-t-il des chiffres plus difficiles à reconnaître que d'autres ? Quels sont les couples de chiffres souvent confondus ? Pouvez-vous en tant qu'humain reconnaître tous les chiffres sans ambiguïté ? ...).
6. La valeur optimale de K dépend fortement des données et notamment des données de test. Régler K en se basant sur l'erreur de test biaise celle-ci et conduit à une estimation trop optimiste de l'erreur de généralisation. Proposez une méthode pour régler K sans ce défaut et appliquez-la.
7. Modifiez le programme pour réaliser l'apprentissage avec la base MNIST complète (60000 exemples) uniquement pour la meilleure valeur de K déterminée précédemment.
8. Quelle est la mémoire nécessaire au système de reconnaissance basé sur la base complète ? Calculez l'erreur de test obtenue par ce système sur les 1000 exemples de la base de test. Comparez l'erreur de test et le temps de calcul avec les résultats précédents.
9. Implémentez vous-même l'algorithme Kppv dans une fonction `kppvpredict(Xtest, Xapp, Yapp, K)` et essayez de voir comment rendre les calculs plus rapides.

1. La base de chiffres manuscrits MNIST (`http://yann.lecun.com/exdb/mnist/`) est une version modifiée de la base rendue disponible par le NIST (*National Institute of Standards and Technology*).