Université de Caen
Lab of Mathematics Nicolas Oresme

Statistics & Data Science
UMR CNRS 6139

Classification supervisée : Les K-plus proches voisins

par Faïcel Chamroukhi

Table des matières

1 Objectifs

L'objectif de ce projet est de développer une bibliothèque implémentant un algorithme de classification supervisée, dite aussi discrimination de données brutes.
Le package développé s'appellera `Knn` (pour K-nearest neighbors).

2 Contexte

2.1 Qu'est ce que la classification supervisée ?

En quelques mots, la classification supervisée, dite aussi discrimination est la tâche qui consiste à discriminer des données, de façon supervisée (ç-à-d avec l'aide préalable d'un expert), un ensemble d'objets ou plus largement de données, de telle manière que les objets d'un même groupe (appelé classes) sont plus proches (au sens d'un critère de (dis)similarité choisi) les unes au autres que celles des autres groupes. Généralement, on passe par une première étape dite d'apprentissage où il s'agit d'apprendre une règle de classification partir de données annotées (étiquetées) par l'expert et donc pour les quelles les classes sont connues, pour ıprédire les classes de nouvelles données, pour lesquelles (on suppose que) les données sons inconnues. La prédiction est une tâche principale utilisée dans de nombreux domaines, y compris l'apprentissage automatique, la reconnaissance de formes, le traitement de signal et d'images, la recherche d'information, etc.

2.2 Quelles données ?

Les données traitées en classification peuvent être des images, signaux, textes, autres types de mesures, etc. Dans le cadre de ce projet les données seront des données multidimensionnelles, par exemple une image (couleur). Chaque donnée est donc composée de plusieurs variables (descripteurs). Pour le cas de données multidimensionnelles standard, chaque donnée étant de dimension d (donc un point dans l'espace $\mathbb{R}^d$) et éventuellement étiquetée (dans le cas où l'on connaitrait sa classe d'appartenance (le "label")) et peut donc être modélisée, par exemple, par un e structure contenant au moins les coordonnées du point et le champ "label". Les n données peuvent donc être modélisées comme étant un tableau de n éléments, chaque élément du tableau étant une structure "donnée" comme décrite précédemment.
Dans le cas d'une image (couleur), l'image contenant n lignes et m colonnes et donc $n \times m$ pixels couleurs, chaque pixel est composé de trois composantes RVB, peut être modélisée par un tableaux de $n \times m$ structures. Chaque structure "pixel" est composée au moins des champs, couleur et "label".
Pour cette partie `Knn` du projet, on commencera par traiter des données simulées, des données iris et ensuite des images.

2.3 Qu'est ce qu'une classe ?

En gros, une classe (ou groupe) est un ensemble de données formée par des données homogènes (qui "se ressemblent" au sens d'un critère de similarité (distance, densité de probabilité, etc)). Par exemple, une classe peut être une région dans une image couleur, un évènement particulier dans un signal sonore, la classe spam et classes non spam dans le cas détection de spams dans un mail, etc.

2.4 Combien de classes ?

Le nombre de groupes (qu'on notera K) en prédiction est supposé fixe (donné par l'utilisateur). C'est le cas par exemple si l'on s'intéresse à classer des images de chiffres manuscrits (nombre de classes = 10 : $0, ..., 9$) ou de lettres manuscrites (nombre de classes = nombres de caractères de l'alphabet), etc.

2.5 Exemple Introductif

Considérons par exemple une image couleur, chaque image contient n pixels $(\mathbf{x}_1, \ldots, \mathbf{x}_n)$, chaque pixel $\mathbf{x}_i$ contient $d = 3$ valeurs (RGB). On peut donc représenter donc le $i$ème pixel $(i = 1, \ldots, n)$ par un vecteur $\mathbf{x}_i$ de dimension $d = 3 : \mathbf{x}_i = (x_{i1}, x_{i2}, x_{i3})^T \in \{0, 1, ..., 255\}^3$. Si l'on connaît les classes de certains pixels, on pourra prédire les classes des autres pixels on choisissant une mesure de (dis)similarité, par exemple une simple distance, ou une mesure de probabilité, etc. Chaque pixel à classer aura donc la classe de celui qui lui est le plus proche au sens de la mesure de (dis)similarité choisie. Ceci peut être utilisé par exemple en segmentation d'image. De manière générale, on peut représenter les données comme un ensemble de vecteurs $(\mathbf{x}_1, \ldots, \mathbf{x}_n)$, chaque $\mathbf{x}_i$ est composée de d composantes réelles : $\mathbf{x}_i = (x_{i1}, ..., x_{ij}, ..., x_{id})^T \in \mathbb{R}^d$.

3 Algorithme des K-ppv

C'est une approche très simple et directe. Elle ne nécessite pas d'apprentissage mais simplement le stockage des données d'apprentissage. Son principe est le suivant. Une donnée de classe inconnue est comparée à toutes les données stockées. On choisit pour la nouvelle donnée la classe majoritaire parmi ses K plus proches voisins (Elle peut donc être lourde pour des grandes bases de données) au sens d'une distance choisie.

Quelle distance

Afin de trouver les K plus proches d'une donnée à classer, on peut choisir la distance euclidienne. Soient deux données représentées par deux vecteurs $\mathbf{x}_i$ et $\mathbf{x}_j$, la distance entre ces deux données est donnée par

$$d(\mathbf{x}_i, \mathbf{x}_j) = \sqrt{\sum_{k=1}^{d} (x_{ik} - x_{jk})^2} \tag{1}$$

4 Travail à réaliser

Il s'agit d'implémenter (en `Matlab`, `R`, `Pyhton` ou `Octave`) l'algorithme des K-plus proches voisins pour prédire les classes de nouvelles données à partir de données étiquetées (données d'apprentissage). On commencera par le 1-ppv. L'algorithme `1nn` est donné par le pseudo-code 1 ci-après.

Algorithm 1: Algorithme du plus proches voisin (1-ppv)

Input: Données d'apprentissage ; $\mathbf{X}^{\text{train}} = (\mathbf{x}_1^{\text{train}}, \ldots, \mathbf{x}_n^{\text{train}})$; classes des données d'apprentissage $\mathbf{z}^{\text{train}} = (z_1^{\text{train}}, \ldots, z_n^{\text{train}})$; $\mathbf{X}^{\text{test}} = (\mathbf{x}_1^{\text{test}}, \ldots, \mathbf{x}_m^{\text{test}})$

Algorithme `Knn` :

for $i \leftarrow 1$ ***to*** m **do**
  for $j \leftarrow 1$ ***to*** n **do**
    Calculer la distance euclidienne entre $\mathbf{x}_i^{\text{test}}$ et $\mathbf{x}_j^{\text{train}}$ en utilisant l'équation (1)
    $\boldsymbol{d}_j \leftarrow d(\mathbf{x}_i^{\text{test}}, \mathbf{x}_j^{\text{train}})$
  end
  Calculer la classe z_i^{test} du $i$ème exemple qui vaut la classe de son pvv :

  `trouver l'indice du ppv de` $\mathbf{x}_i^{\text{test}}$:

  $ind\_ppv_i \leftarrow \arg\min_{j=1}^{n} \boldsymbol{d}_j$

  `trouver la classe du ppv de` $\mathbf{x}_i^{\text{test}}$ (qui est $\mathbf{x}_{ind\_ppv_i}^{\text{train}}$) :

  $z_i^{\text{test}} = z_{ind\_ppv_i}^{\text{train}}$
end

Result: classes des données de test $\mathbf{z}^{\text{test}} = (z_1^{\text{test}}, ..., z_n^{\text{test}})$

Ensuite, étendez votre code pour le cas des K-pvv pour une valeur de $K \geq 1$.
L'algorithme `Knn` est donné par le pseudo-code 2 ci-après.

Algorithm 2: Algorithme des K-plus proches voisins

Input: Données d'apprentissage ; $\mathbf{X}^{\text{train}} = (\mathbf{x}_1^{\text{train}}, ..., \mathbf{x}_n^{\text{train}})$; classes des données d'apprentissage $\mathbf{z}^{\text{train}} = (z_1^{\text{train}}, ..., z_n^{\text{train}})$; $\mathbf{X}^{\text{test}} = (\mathbf{x}_1^{\text{test}}, ..., \mathbf{x}_m^{\text{test}})$; nombre des ppv K

Algorithme `Knn` :

for $i \leftarrow 1$ ***to*** m **do**
  for $j \leftarrow 1$ ***to*** n **do**
    Calculer la distance euclidienne d_{ij} entre $\mathbf{x}_i^{\text{test}}$ et $\mathbf{x}_j^{\text{train}}$ en utilisant l'équation (1)
    $\boldsymbol{d}_j \leftarrow d_{ij}$
  end
  Calculer la classe z_i^{test} du $i$ème exemple qui vaut la classe de son pvv :

  `/* trouver les K-ppv de` $\mathbf{x}_i^{\text{test}}$ `*/` :

  Trier les distances $\boldsymbol{d}_j$ selon un ordre croissant pour $j = 1, \ldots, n$
  Récupérer en même temps les indices $IndVoisins$ avant le tri des $\boldsymbol{d}_j$
  Récupérer les classes des K premiers ppv à partir des indices $IndVoisins$ et en trouver la classe majoritaire :

  $C_k \leftarrow 0 \quad (k = 1, \ldots, K)$
  for $k \leftarrow 1$ ***to*** K **do**
    $ind\_voisin_k \leftarrow \text{IndVoisins}_k$
    $h \leftarrow z^{\text{train}}_{ind\_voisin_k}$
    $C_h = z_h + 1$
  end

  `/* trouver la classe du ppv de` $\mathbf{x}_i^{\text{test}}$:
   `(la classe majoritaire de celles de ses` K-ppv`) */` :

  $z_i^{\text{test}} = \arg\max_{k=1}^{K} C_k$
end

Result: classes des données de test $\mathbf{z}^{\text{test}} = (z_1^{\text{test}}, ..., z_n^{\text{test}})$

5 Données et présentation des résultats

Pour les données, vous pouvez par exemple utiliser celles-ci : Xtrain, klastrain, Xtest. Vous pouvez également tester votre programme sur les données iris téléchargeable ici (Iris, un jeu de données très utilisé en classification automatique).
Pour la présentation des résultats, pour commencer, les résultats trouvés peuvent être affichés directement à l'écran ou écrites dans un fichier.
Pour aller plus loin, la présentation des résultats pourra se faire par des graphiques en affichant par exemple les données dans l'espace, chaque ensemble de données appartenant à une même

classe est colorée par une couleur différente, etc. Vous pouvez vous inspirez de l'exemple suivant de la figure 1. Les données pour cet exemple sont : Xtrain, klastrain, Xtest.

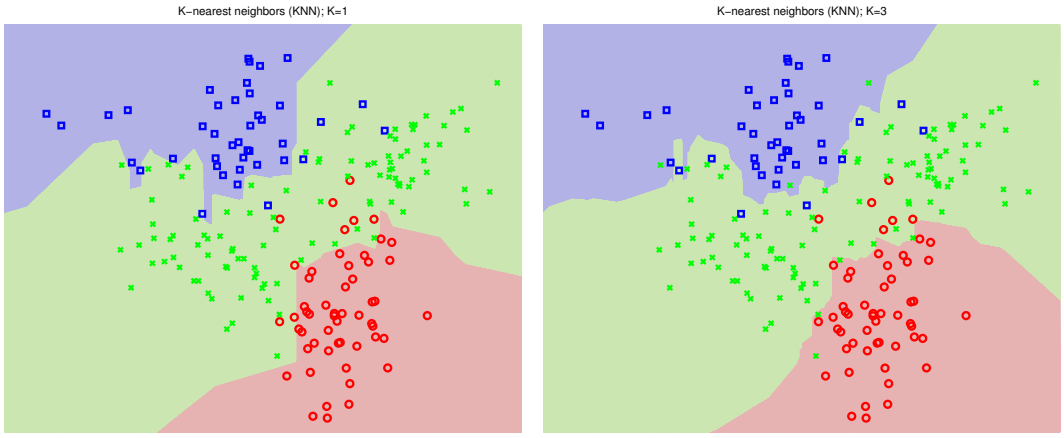

FIGURE 1 – Un exemple où l'on a trois classes. Les données d'apprentissage sont représentées par □, ×, ∘. Le reste représente les données de test et les couleurs correspondent aux classes prédites par l'algorithme K-ppv pour deux valeurs différentes de K (1 et 3).

Si vous manipulez des données pour lesquelles vos connaissez les classes des données de test (le cas des iris par exemple), vous donnerez également le taux d'erreur de classification en comparant les classes fournies par l'algorithme avec les vraies classes (par exemple en créant une fonction `classif_error`). Essayer de dresser une matrice de confusion, etc