SY09 Printemps 2019
TP 7 — Introduction à l'apprentissage supervisé, méthode des K plus proches voisins

On souhaite utiliser l'algorithme des K plus proches voisins sur différents jeux de données, à des fins de discrimination. On complétera tout d'abord les fonctions fournies, puis on les testera sur des données synthétiques (générées selon une distribution prédéfinie) puis réelles.

1 Méthode des K plus proches voisins

On rappelle que la méthode des K plus proches voisins ne nécessite pas de phase d'apprentissage à proprement parler. On considérera les deux fonctions `kppv.val`, qui permet de calculer les classes $f(\boldsymbol{x}_1), f(\boldsymbol{x}_2), \ldots, f(\boldsymbol{x}_m)$ prédites pour chacun des m individus de test $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_m$; et `kppv.tune`, à compléter, qui doit permettre de trouver la valeur de K donnant les meilleurs résultats sur un ensemble de données.

1.1 Implémentation

Fonction `kppv.val`

Analyser la fonction `kppv.val` : comment fonctionne-t-elle ? Comment la sortie est-elle déterminée ? Quelle information pourrait-on retourner en plus du classement des individus de test ?

La fonction calcule les distances entre chacun des individus de test et chacun des individus d'apprentissage. Les distances sont triées par ordre croissant, et pour chaque individu de test la distance seuil est définie comme la k$^{\text{e}}$ plus petite distance. Les voisins de l'individu de test sont identifiés comme les points d'apprentissage éloignés d'une distance inférieure ou égale à cette distance seuil.

Une fois les K plus proches voisins identifiés, on calcule le score de chaque classe dans cet ensemble de voisins, et on retourne la classe de score maximal. Il serait donc possible de retourner les proportions des classes en plus de la classe choisie (estimation des probabilités *a posteriori* des classes dans un voisinage de chaque point de test).

Fonction `kppv.tune`

Compléter la fonction `kppv.tune`, qui doit déterminer le nombre « optimal » de voisins K_{opt} (choisi parmi un vecteur `nppv` de valeurs possibles), c'est-à-dire donnant les meilleurs performances sur un ensemble de validation étiqueté (tableau individus-variables X_{val} de dimensions $n_{\text{val}} \times p$; et vecteur z_{val} de sorties associées, de longueur n_{val}).

La fonction prend donc en entrée :
- les données utilisées pour faire le classement : tableau individus-variables `Xapp` et vecteur `zapp` des étiquettes associées ;
- le tableau individus-variables `Xval` et le vecteur `zval` à utiliser pour la validation ;
- un ensemble de valeurs `nppv` correspondant aux différents nombres de voisins à tester.

Elle retourne la valeur `Kopt` choisie dans l'ensemble `nppv` et donnant les meilleurs résultats sur `Xval`.

La fonction est très simple à programmer :

```
> kppv.tune <- function(Xapp, zapp, Xval, zval, nppv)
> {
>     taux <- rep(0,length(nppv))
>
>     for (k in 1:length(nppv))
>     {
>         deci <- kppv.val(Xapp, zapp, nppv[k], Xval)
>         taux[k] <- mean(deci!=zval)
>     }
>
>     Kopt <- nppv[which.min(taux)]
> }
```

Pour visualiser les frontières de décision obtenues à l'aide de la fonction `kppv.val`, on pourra utiliser la fonction `front.kppv` fournie :

```
> Kopt <- kppv.tune(Xapp, zapp, Xval, zval, seq(from=1,to=11,by=2))
> front.kppv(Xapp, zapp, Kopt, 1000)
```

1.2 Sélection de modèle et évaluation des performances

Pour un jeu de données, on séparera aléatoirement l'ensemble des données disponibles, de manière à former un ensemble d'apprentissage et un ensemble de test (et éventuellement un ensemble de validation si nécessaire). L'ensemble d'apprentissage est réservé à l'apprentissage du modèle uniquement (s'il y a lieu, on optimisera les hyper-paramètres sur l'ensemble de validation ou via une procédure spécifique) ; et l'ensemble de test n'est utilisé que pour l'estimation des performances.

Remarque 1 *Dans certains cas, pour obtenir une estimation plus robuste des performances du modèle, on pourra répéter les étapes de séparation des données disponibles, apprentissage du modèle et estimation des performances (taux d'erreur de prédiction) par un taux d'erreur moyen ou médian.*

Il est parfois de coutume de calculer en plus un intervalle de confiance ou un diagramme en boîte à partir de ces données. Or, si la moyenne des taux d'erreur empiriques est un estimateur sans biais *du taux d'erreur espéré du classifieur, ce n'est pas le cas de sa variance empirique (ou de la variance empirique corrigée) : les différents ensembles d'apprentissage (ou de test) obtenus par séparation des données ne sont pas distincts, un même individu pouvant être présent dans plusieurs ensembles.*

De ce fait, les taux d'erreur moyens calculés au cours de ces expériences répétées ne sont pas indépendants. S'il est rigoureux d'utiliser leur moyenne pour estimer le risque, utiliser des intervalles de confiance ou des diagrammes en boîte pour comparer les performances de plusieurs modèles peut au contraire mener à des conclusions erronées, du fait du biais des estimations utilisées.

1.3 Questions

Jeux de données synthétiques

On dispose de cinq jeux de données (téléchargeables sur le site de l'UV) : `Synth1-40`, `Synth1-100`, `Synth1-500`, et `Synth1-1000`.Pour chacun de ces jeux de données, les classes ont été générées suivant des lois normales bivariées, identiques et de mêmes proportions pour tous les jeux de données : `Synth1-40`, `Synth1-100`, `Synth1-500` et `Synth1-1000` diffèrent ainsi essentiellement par le nombre d'observations.

1. Pour chacun des jeux de données, estimer les paramètres μ_k et Σ_k des distributions conditionnelles, ainsi que les proportions π_k des classes.

```
> for (k in 1:nlevels(z))
> {
>     print(apply(Xapp[which(zapp==k),], 2, mean))
>     print(cov.wt(Xapp[which(zapp==k),], method='ML')$cov)
> }
```

2. Effectuer une séparation aléatoire de l'ensemble de données en un ensemble d'apprentissage et un ensemble de test (on pourra utiliser la fonction `separ1`). Déterminer le nombre optimal de voisins à l'aide de la fonction `kppv.tune`, *en utilisant l'ensemble d'apprentissage comme ensemble de validation*. Quel est le nombre optimal de voisins déterminé ? Pourquoi ?

Cela se fait facilement en utilisant les fonctions `separ1` et `kppv.tune` :

```
> DSep <- separ1(X, z)
> Kopt <- kppv.tune(DSep$Xapp, DSep$zapp, DSep$Xapp, DSep$zapp,
  ↪  nppv=seq(from=1,to=11,by=2))
```

On obtient évidemment $K_{\text{opt}} = 1$ comme nombre optimal de voisins, ce qui met en évidence le caractère biaisé (optimiste) du taux d'erreur d'apprentissage comme estimateur du taux d'erreur d'un classifieur.

3. Écrire un script qui effectue $N = 20$ séparations aléatoires du jeu de données `Synth1-1000` en ensembles d'apprentissage, de validation, et de test (on pourra utiliser la fonction `separ2`) ; et qui, pour chacune :
   — d'une part, détermine (et stocke) le nombre optimal de voisins K_{opt} ;
   — et d'autre part, calcule (et stocke) les taux d'erreur d'apprentissage, de validation et de test pour différentes valeurs de K (les mêmes que celles testées pour déterminer K_{opt}).

On peut utiliser le code suivant :

```
> nppv <- seq(from=1,to=21,by=2)
>
> taux.app <- matrix(0,nrow=20,ncol=length(nppv))
> taux.val <- matrix(0,nrow=20,ncol=length(nppv))
> taux.tst <- matrix(0,nrow=20,ncol=length(nppv))
> Kopt <- rep(0,20)
> for (t in 1:20)
> {
>     DSep <- separ2(X, z)
>
>     Kopt[t] <- kppv.tune(DSep$Xapp, DSep$zapp, DSep$Xval, DSep$zval,
      ↪  nppv)
>
>     for (k in 1:length(nppv))
>     {
>         deci.app <- kppv.val(DSep$Xapp, DSep$zapp, nppv[k], DSep$Xapp)
>         deci.val <- kppv.val(DSep$Xapp, DSep$zapp, nppv[k], DSep$Xval)
>         deci.tst <- kppv.val(DSep$Xapp, DSep$zapp, nppv[k], DSep$Xtst)
>
>         taux.app[t,k] <- mean(deci.app!=DSep$zapp)
>         taux.val[t,k] <- mean(deci.val!=DSep$zval)
>         taux.tst[t,k] <- mean(deci.tst!=DSep$ztst)
>     }
> }
>
> plot(nppv, apply(taux.app, 2, mean), type='l')
> plot(nppv, apply(taux.val, 2, mean), type='l')
> plot(nppv, apply(taux.tst, 2, mean), type='l')
```

Représenter les taux d'erreur d'apprentissage, de validation et de test. L'estimation du nombre optimal de voisins semble-t-elle stable ? Pourquoi ?

L'estimation du nombre optimal de voisins peut changer — et beaucoup — d'une répétition à l'autre, même pour une taille conséquente de l'ensemble de validation (ici, $n_{\text{val}} = 250$). Le calcul d'un unique taux d'erreur de validation pour choisir K_{opt} semble être une stratégie trop aléatoire. Cela justifie le recours à des méthodes de sélection de modèle plus avancées, telle que la validation croisée.

Jeux de données réelles

On considère maintenant les jeux de données `Pima` et `Breastcancer`. Traiter ces jeux de données suivant le protocole utilisés sur les données synthétiques. Calculer les estimations de ε sur l'ensemble d'apprentissage et sur l'ensemble de test. Commenter et interpréter les résultats obtenus.

2 Méthode des « K plus proches prototypes »

La méthode des K plus proches voisins présente des propriétés intéressantes, mais cette stratégie reste coûteuse : elle nécessite, en phase de test, de calculer la distance entre chaque individu de test et tous les individus d'apprentissage. On souhaite ici en tester une variante, dans laquelle l'ensemble d'apprentissage sera résumé par un ensemble de points caractéristiques que nous appellerons *prototypes*.

Le bénéfice attendu d'une telle opération est évidemment calculatoire ; notons qu'elle a également une influence sur le plan des performances, en fonction du nombre de prototypes choisi pour résumer une classe et de la manière dont ces prototypes sont déterminés.

2.1 Apprentissage des prototypes

Cette variante de la méthode des K plus proches voisins comporte à présent une phase d'apprentissage : le calcul des prototypes qui résument les individus d'apprentissage dans chaque classe.

Pour réaliser cet apprentissage, on utilisera l'algorithme des « C_k-means » [1] : pour *chaque classe* ω_k, on déterminera ainsi C_k centres qui résumeront la classe. L'ensemble de ces centres (étiquetés) sera ensuite utilisé à la place de l'ensemble d'apprentissage pour classer les individus de test.

Les paramètres C_k, qui fixent pour chaque classe ω_k le nombre de prototypes qui la résument, doivent bien être différenciés du paramètre K, qui détermine le nombre de plus proches prototypes utilisés en phase de test pour classer les individus.

2.2 Questions

Pour les jeux de données synthétiques, on pourra utiliser la fonction `front.kppp` pour afficher l'ensemble d'apprentissage, les prototypes obtenus et les frontières de décision associées.

1. Supposons que l'on fixe $C_k = 1$ pour tout $k = 1, \ldots, g$, et $K = 1$: à quel classifieur correspond alors la méthode des K plus proches prototypes ?

   Il s'agit alors du classifieur euclidien.

2. Si l'on fixe à présent $C_k = n_k = \sum_{i=1}^{n} z_{ik}$, quel classifieur retrouve-t-on ?

   On retrouve évidemment la méthode des K plus proches voisins.

1. Il se peut que l'on veuille utiliser un indicateur de tendance centrale plus robuste aux points atypiques que la moyenne ; cela revient à remplacer l'algorithme des C_k-means par une autre méthode de partitionnement, comme par exemple la stratégie des C_k-*médoïdes* (dans laquelle on substitue la médiane à la moyenne).

3. Programmer une fonction `kppp.app` qui permettra de déterminer les C_k prototypes de chaque classe : elle prendra en arguments d'entrée l'ensemble d'apprentissage étiqueté (matrice d'observations `Xapp` et vecteur d'étiquettes `zapp`) et le nombre de prototypes par classe (vecteur `Ck`), et fournira en sortie les prototypes `Xpro` et les étiquettes associées `zpro`.

```
> kppp.app <- function(Xapp, zapp, Ck=rep(1,nlevels(zapp)), nstart=1,
  ↪  part=1)
> {
>   Xapp <- as.matrix(Xapp)
>
>   Xpro <- NULL
>   zpro <- NULL
>
>   for (k in 1:length(Ck))
>   {
>     Xpro <- rbind(Xpro, kmeans(Xapp[which(zapp==k),], centers=Ck[k],
      ↪  nstart)$centers)
>     zpro <- c(zpro, rep(k, Ck[k]))
>   }
>
>   out <- NULL
>   out$Xpro <- Xpro
>   out$zpro <- as.factor(zpro)
>   out
> }
```

4. Tester la méthode des K plus proches prototypes sur les jeux de données synthétiques du paragraphe 1.3, en choisissant $C_k \in \{1, 2, 3, 4, 5\}$ [2] et en faisant varier le nombre K de prototypes utilisés en phase de test. Commenter (on pourra comparer aux résultats obtenus dans le cas de la méthode des K plus proches voisins).

   Voir la figure 4 pour quelques exemples avec $C_k \in \{3, 5\}$ et $K \in \{1, 3\}$.

5. Toujours pour $C_k \in \{1, 2, 3, 4, 5\}$, déterminer le nombre optimal de prototypes K_{opt}, tout d'abord en utilisant la fonction `kppv.tune` avec un ensemble d'apprentissage pour la validation, puis un ensemble de validation distinct. Qu'observez-vous ?

   Utiliser l'ensemble d'apprentissage comme ensemble de validation ne conduit plus systématiquement à choisir $K_{\text{opt}} = 1$ (sauf, évidemment, si l'on choisit $C_k = n_k$ pour chaque classe ω_k).

   Le paramètre qui semble à présent influer le plus sur la détermination de K_{opt} est la taille de l'ensemble utilisé pour la validation : si l'on modifie la fonction `separ2` pour fixer la taille des trois ensembles (apprentissage, validation et test) à $n/3$, on obtient généralement des valeurs similaires de K_{opt}, que l'on utilise `Xapp` ou `Xval`.

   Il faudra quand même insister sur la nécessité de distinguer ensembles d'apprentissage et de validation dans une démarche rigoureuse d'apprentissage machine.

6. Traiter à présent les données `Pima` et `Breastcancer`. Commenter en comparant aux résultats obtenus au paragraphe 1.3.

2. On fixera la même valeur de C_k pour toutes les classes.

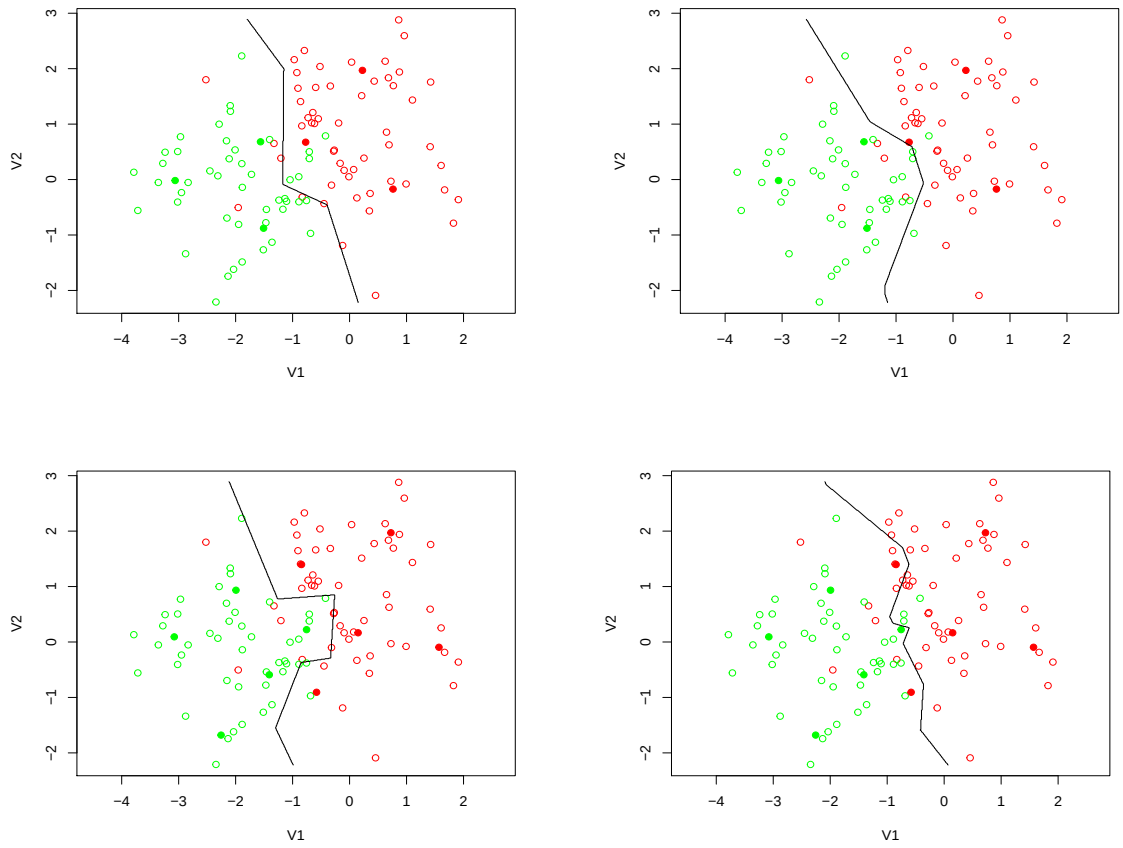

FIGURE 1 – Résultats obtenus avec $C_k = 3$ et $K = 1$ (haut-gauche), $C_k = 3$ et $K = 3$ (haut-droite), $C_k = 5$ et $K = 1$ (bas-gauche) et $C_k = 5$ et $K = 3$ (bas-droite).