Data Mining 2: Modélisation statistique et apprentissage **G. Cohen**

TD 05 mars 2018

ETheme 5 Réseaux de neurones et Support Vecteurs Machines

1 Réseaux de neurones (Source P. Besse)

Il s'agit d'estimer un modèle de type perceptron multicouche avec en entrée les variables qualitatives construites précédemment et en sortie la variable binaire à prévoir. Des fonctions R pour l'apprentissage d'un perceptron élémentaire ont été réalisées par différents auteurs et sont accessibles sur le réseau. La librairie nnet de (Ripley, 1999), est limitée aux perceptrons à une couche. La variable à expliquer étant binomiale, la fonction de transfert du neurone de sortie est choisie sigmoïdale comme toutes celles de la couche cachée. Le paramètre important à déterminer est le nombre de neurones sur la couche cachée parallèlement aux conditions d'apprentissage (temps ou nombre de boucles). Une alternative à la déterminatio du nombre de neurones est celle du decay qui est un paramètre de régularisation analogue à celui utilisé en régression rdge. Il pénalyse la norme du vecteurs des paramètres et contraint ainsi la flexibilité du modèle. Très approximativement il est d'usage de considérer, qu'en moyenne, il faut une taille d'échantillon d'apprentissage 10 fois supérieure au nombre de poids c'est-à-dire au nombre de paramètres à estimer. On remarque qu'ici la taille de l'échantillon (#800) est modeste pour une application raisonnable du perceptron. Seuls des nombres restreints de neurones peuvent être considérés et sur une seule couche cachée.
La base de données utilisée retraçe l'historique mensuel bancaire et les caractéristiques de tous les clients. L'objectif est de classer les possesseurs de cartes visa premier (CARVP).

1.1 Estimation

```
library(MASS)
library(nnet)
visappt<-read.table("D:\\visappt.dat", header = T, sep = " ") # a remplacer par votre
var=names(visappt)
varquant=var[1:30]
varqual=var[31:54]
```

```
visapptq=visappt[,c("CARVP", varqual)]
Avec bien entendu l'endroit où se trouve votre fichier à la place de : D:\\visappt.da
visapptq=visappt[,c("CARVP", varqual)] # sélection des prédicteurs qualitatifs
vistestq=vistest[,c("CARVP", varqual)] # pour l'échantillon test
vis.nnet<-nnet(CARVP~.,data=visapptq,size=3) # apprentissage
summary(vis.nnet)
```

Le détail des poids de chaque entrée de chaque neurone ne constituent pas des résultats très explicite ! Une approche de type "validation croisée" est nécessaire afinde tenter d'optimiser les choix en présence : nombre max d'itérations, "decay" et nombre de neurones.

2 Nombre de neurones

Il n'y a pas de fonction dans la librairie nnet permettant d'optimiser le nombre de neurones de la couche cachée par validation croisée. Voici le texte d'une fonction de k-validation croisée adaptée aux réseaux de neurones.

1. Ouvrir l'éditeur (xemacs) pour la création de la fonction : `fix(CVnn)`
2. Rentrer le texte de la fonction. Les espaces ne sont pas importants, la mise en page est automatique.

```
function(formula, data, size, niter = 1, nplis = 10, decay = 0, maxit = 100)
{
n = nrow(data)
tmc <- 0
un <- rep(1, n)
ri <- sample(nplis, n, replace = T)
cat(" k= ")
for(i in sort(unique(ri))) {
cat(" ", i, sep = "")
for(rep in 1:niter) {
learn <- nnet(formula, data[ri != i, ], size = size,
trace = F, decay = decay, maxit = maxit)
tmc = tmc + sum(un[(data$CARVP[ri == i] == "Coui") !=
(predict(learn, data[ri == i, ]) > 0.5)])
}
}
```

```
cat("\n", "Taux de mal classes")
tmc/(niter * length(unique(ri)) * n)
}
```

3. Sauver le texte, quitter l'éditeur

4. Si des messages d'erreur apparaissent, `CVnn=edit()` permet de relancer l'éditeur pour les corriger. Le paramètre niter permet de répliquer l'apprentissage du réseau et de moyenner les résultats afin de réduire la variance de l'estimation de l'erreur. Il est par défaut de 1 mais peut-être augmentée à condition de se montrer plus patient. R, langage interprété comme matlab, n'est pas particulièrement véloce lorsque plusieurs boucles sont imbriquées.
   Tester la fonction ainsi obtenue et l'exécuter pour différentes valeurs de size (2, 3, 4) et decay (0, 1, 2).

Attention, l'initialisation de l'apprentissage d'un réseau de neurone comme celle de l'estimation de l'erreur par validation croisée sont aléatoires. Chaque exécution donne donc des résultats différents. À ce niveau, il serait intéressant de construire un plan d'expérience à deux facteurs (ici, les paramètres de taille et decay) de chacun trois niveaux. Plusieurs réalisations pour chaque combinaison des niveaux suivies d'un test classique d'anova permettraient de se faire une idée plus juste de l'influence de ces facteurs sur l'erreur.

```
CVnn(CARVP~.,data=visapptq,size=3, decay=1)
...
```

Noter la taille et le "decay" optimaux. Il faudrait aussi faire varier le nombre total d'itérations. Cela risque de prendre un peu de temps !

2.1 Test

Réestimer le réseau avec les paramètres optimaux puis estimer le taux d'erreur sur l'échantillon test. Noter le taux d'erreur.

```
vis.nnet<-nnet(CARVP~.,data=visapptq,size=3,decay=1)
pred.vistest<-predict(vis.nnet,vistestq)>0.5 # code ‘‘true’’ la prévision de ‘‘Coui’’
table(pred.vistest,vistest$CARVP=="Coui")
```

Noter le taux d'erreur.

3 Machine à vecteurs supports

1. On utilise ici un jeu de données disponible dans R. Tapez les commandes suivantes pour charger les données dans le système:

   ```
   library(MASS)
   data(cats)
   ```

   On charge le jeu de données cats, qui contient trois colonnes avec les poids du coeur et du corps des chats mâles et femelles. Pour connaitre ces données tapez `help(cats)`.

2. Quelques statistiques avec `summary(cats)`.

3. Rendre le nom des colonnes disponible `attach(cats)`

4. Faire quelques histogrammes conditionnels des variables conditionnées par la classe (sex) en exécutant `library(lattice)` puis `histogram(~Bwt | Sex)`. Faire la même chose avec la variable poids du coeur.

5. Maintenant, inspecter la corrélation des deux variables poids: `cor(Bwt,Hwt)`. Qu'en pensez-vous ?

6. Visualiser ces données dans un diagramme de dispersion (scatterplot)

   ```
   col<-c("RED","BLUE")
   plot(Bwt,Hwt,col=col[Sex])
   ```

   Pouvez-vous voir les deux classes? Pouvez-vous trouver une bonne frontière de décision les séparant ?

7. Maintenant que vous avez visualisé les données, on va procéder à de la discrimination. En premier lieu, on découpe le jeu de données en ensemble d'apprentissage et de test. Exécutez:

   ```
   index<-1:nrow(cats)
   testindex<-sample(index, trunc(length(index)/3))
   testset<-cats[testindex,]
   trainset<-cats[-testindex,]
   ```

   Donc, on a 1/3 des données pour le test et 2/3 pour l'apprentissage.

8. Ensuite, entrainer une SVM sur ces données, mais d'abord vous devez installer le package qui contient les svms. Exécutez:

```
install.packages('e1071')
library('e1071')
```

9. Maintenant, vous êtes prêt pour entrainer votre classifieur (classeur) SVM . Regarder la description de la fonction svm: `help(svm)`. Puis exécutez:

```
svm.linear<-svm(Sex~., data = trainset, probability=TRUE, cross=10, kernel="line
```

Cette commande crée une SVM à l'aide des données d'apprentissage, elle utilise un noyau linéaire et une validation croisée à 10 partitions. Regarder les résultats du classement:

```
plot(svm.linear, trainset).
```

10. Tester le modèle en calculant ses prédictions sur l'ensemble de test. Exécutez:

```
svmlinear.preds<-predict(svm.linear, testset[,-1], probability=TRUE)
```

Inspectez le contenu de ces nouvelles variables. Qu'est ce que vous voyez ? Est-ce que le modèle est performant (taux de bien classés)?

11. Ensuite, on construit une matrice de confusion. Tapez

```
confmat.linear <- table(pred = svmlinear.preds,
true = testset[,1])
```

et inspectez le contenu de cette nouvelle variable. Quelle est la performance de ce modèle?

12. Essayer de changer les paramètres du classeur SVM :

```
svm.pol<-svm(Sex~., data = trainset, probability=TRUE, cross=10, kernel="polynom
plot(svm.pol, trainset)
svmpol.preds<-predict(svm.pol, testset[,-1], probability=TRUE)
confmat.pol <- table(pred = svmpol.preds, true = testset[,1])
```

et regardez la matrice de confusion

13. Maintenant que vous êtes dans une étape d'évaluation, vous tracerez une courbe ROC pour vos résultats. Rappelez-vous qu'une courbe ROC montre la relation entre le taux de vrais positifs et le taux de faux positifs pour un classeur binaire comme votre SVM. Les courbes ROC sont tracés dans un graphe 2D dans lequel un taux de vrais positifs est représenté dans l'axe y, alors que le taux de faux posisitifs est représenté dans l'axe x. Il y a quelque points importants à noter dans ce graphique:

    - Le point $(0, 0)$ représente un classement non-positif, ce qui signifie que la sortie du classeur était négative tout le temps.
    - Le point $(1, 1)$ d'un autre coté, représente tout les classements positifs tout le temps. Ce qui signifie que , la sortie du classeur est toujours positive. Ainsi que vous pouvez l'imaginer, dans un scénario de la vraie vie les deux situations sont incorrectes.
    - Le point $(0, 1)$ représente un classement parfait, alors que le point $(1, 0)$ représente un parfait "mauvais" classement.
      Si vous désirez plus d'informations sur les courbes ROC, vous pouvez regarder cette introduciton par Tom Fawcett: *http://www.sciencedirect.com/science/*
      Avant de tracer les courbes ROC dans R vous devez installer le package correspondant: `install.packages('ROCR')` et suivez les directives de l'installation. Puis, taper `library("ROCR")` pour charger la librarie.

14. Dans votre cas, la courbe ROC sera tracée en utilisant les probabilités obtenues à partir du classement de la SVM. Exécutez:

```
svmlinear.rocr<-prediction(attr(svmlinear.preds,"probabilities")[,2],
testset[,1] == "M")
svmlinear.perf<-performance(svmlinear.rocr, measure = "tpr", x.measure = "fpr")
svmpol.rocr<-prediction(attr(svmpol.preds,"probabilities")[,2],
testset[,1] == "M")
svmpol.perf<-performance(svmpol.rocr, measure = "tpr", x.measure = "fpr")
plot(svmlinear.perf,col="BLUE")
plot(svmpol.perf,add=TRUE,col="RED")
```

15. Ensuite, on va calculer l'aire sous la courbe ROC (AUC). Une courbe ROC est une représentation visuelle de la performance du classeur. Cependant, pour comparer des classeurs on aimerait avoir une simple valeur scalaire: c'est précisément l'AUC, dont la valeur se trouve dans

l'intervalle $[0, 1]$. Parce qu'un classeur qui déciderait de façon aléatoire produirait une ligne entre $(0, 0)$ et $(1, 1)$, qui donne un AUC de $0, 5$, aucun classeur réaliste ne peut avoir un AUC inférieur à $0, 5$. Pour calculer l'AUC des courbes des classeurs exécutez:

```
svmlinear.auc<-as.numeric(performance(svmlinear.rocr, measure = "auc", x.measure
= "cutoff")@ y.values)
```

puis

```
svmpol.auc<-as.numeric(performance(svmpol.rocr, measure = "auc", x.measure
= "cutoff")@ y.values)
```

Comparez ces valeurs.

16. Basé sur ce que vous venez de voir, selon-vous quel est le meilleur classeur? Comment pouvez vous les comparer?

17. Pour réaliser un test de McNemar pour vérifier si ces classeurs diffèrent significativement. Ce test aide à répondre a la question suivante: soit deux classeurs lequel des deux sera plus performant sur des nouveaux exemples test. Dans ce test, l'hypothèse nulle établit que les deux classeurs doivent avoir le même taux d'erreur. Pour approfondir sa connaissance sur les tests utilisés pour les algorithmes de datamining vous pouvez lire ce papier de Thomas Dietterich: *http://www.mitpressjournals.org/doi/abs/10.116* On va maintenant construire une table qui doit contenir le nombre d'exemples mals classé par les deux algorithmes($n00$), le nombre d'exemples mals classés par le premier algorithme mais pas par le second ($n01$), le nombre d'exemples dans la situation opposée ($n10$), et finalement le nombre d'exemples mals classés par aucuns des deux ($n11$). La somme de ces quatre quantités est le nombre total d'exemples dans l'ensemble de test. Exécutez les commandes suivantes pour construire une telle table:

```
dummy1<-as.data.frame(predict(svm.linear, testset[,-1]),optional=TRUE)
dummy2<-as.data.frame(predict(svm.pol, testset[,-1]),optional=TRUE)
```

18. Ensuite, prenez les exemples qui ont été classés correctement par la SVM avec un noyau linéaire en exécutant

```
correct.linear<-(dummy1[,] == testset[,1]).
```

Faites la même chose avec un noyau polynomial . (Appelez le correct.pol)

19. Vous allez construire la matrice de contingence par l'exécution des commandes suivantes:

```
n00<-sum(!correct.linear & !correct.pol)
n10<-sum(!correct.linear & correct.pol)
n01<-sum(correct.linear & !correct.pol)
n11<-sum(correct.linear & correct.pol)
```

20. Par l'exécution de `sum(n00,n10,n01,n11)` on peut vérifier que cela compte pour la totalité de l'ensemble de test.

21. Pour réaliser un test de McNemar tapez:

```
mcnemar.test(matrix(c(n00,n10,n01,n11),nrow=2))
```

Regarder la p-value. Qu'est-ce que vous notez? que pouvez-vous dire de la comparaison des deux classeurs et de l'hypothèse nulle?