TP6

Exercice 1

ANN (Artificial Neural Network) : réseaux de neurones sous Weka

1. Ouvrez WEKA et charger le fichier « iris.arff »
2. Lancer la classification en utilisant les réseaux de neurones avec les paramètres par défaut et en testant sur les données d'apprentissage (pour rendre les tests plus rapide).
   Pour utiliser les réseaux de neurones sous Weka, il faut d'abord être sous l'onglet « classify » puis choisir dans « classifier » le classifieur « **MultiLayerPerceptron** » dans le répertoire des classifieur « functions ».
3. Observez et analysez les résultats de classification
4. Visualisez les erreurs de classification
5. Observez le modèle créer et le nombre de neurones utilisé
6. Etudier l'effet de la variation du nombre de neurones sur les résultats de la classification
7. A-t-on vraiment besoin de 3 neurones pour avoir les mêmes résultats
8. Que signifie la valeur de l'option *trainingTime*
9. Changer le nombre d'*Epochs* (training time) de 500 à 50 et relancez la classification :
   - Que remarquez-vous ?
   - Observez de nouveau les erreurs de classification
10. Combien faut-il d'Epochs pour bien classer toutes les instances ?
11. Que se passe-t-il si on ne normalise pas les attributs ?
12. Il est possible de créer manuellement le réseau. Pour le faire, il faut Les principales options disponibles dans Weka pour MultiLayerPerceptron sont les suivantes :

Options

GUI : permet l'utilisation d'une interface graphique (pour plus de détails, lire la doc avec le bouton more dans la fenêtre de choix des options).

autobuild : Ajoute et connecte les couches cachées.

decay : si vrai, permet de minimiser le temps et le taux d'apprentissage: les poids sont moins modifiés au fur et à mesure de l'apprentissage.

hiddenLayers : permet de décrire le nombre et la taille des couches cachées. La description est :

- Soit une suite d'entiers (le nombre de neurones par couche) séparés par des virgules.
- Soit les valeurs spéciales déterminant une seule couche cachée :
  - a : (nombre d'attributs+nombre de classes)/2
  - i : nombre d'attributs
  - o : nombre de classes
  - t : nombre d'attributs+nombre de classes

learning rate : la valeurs de mise à jour des poids

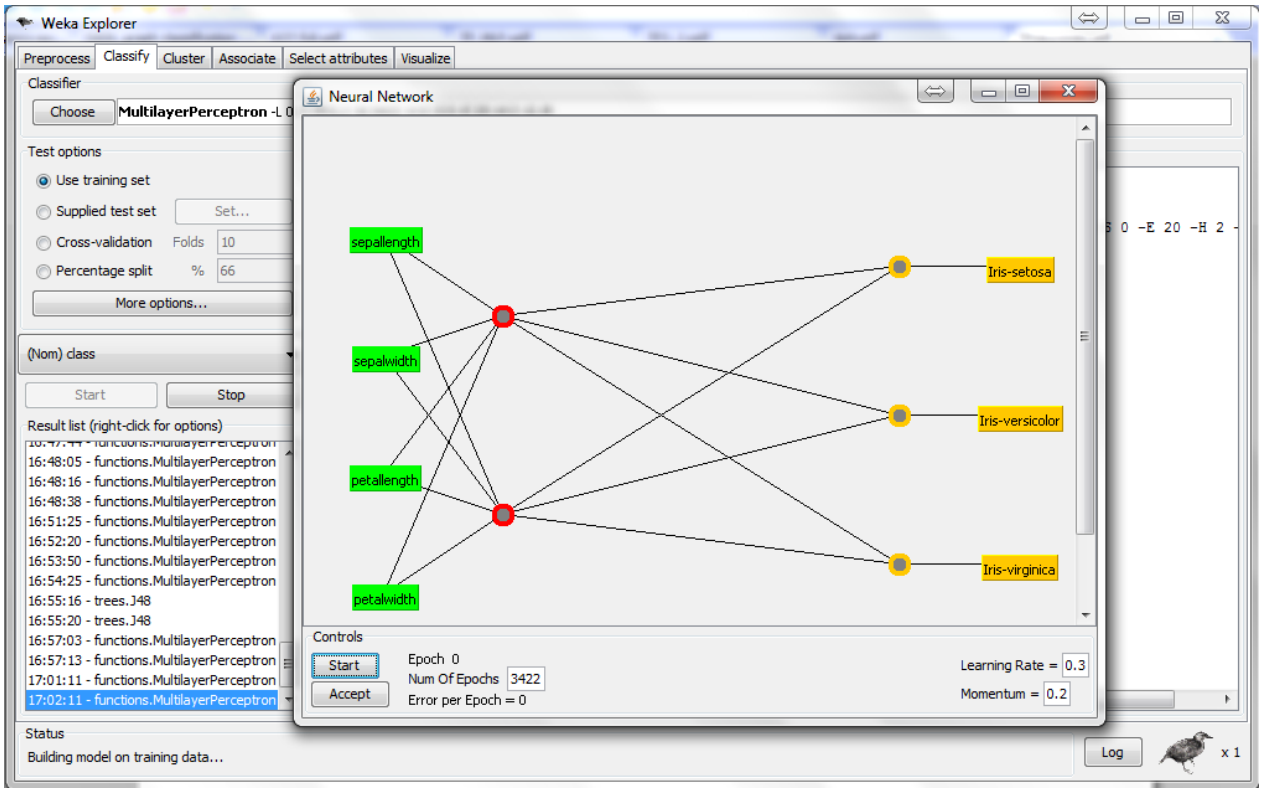
momentum : momentum appliqué sur les poids durant la mise à jour

nominalToBinaryFilter : transforme les attributs nominaux en attributs binaires : un attribut pouvant prendre k valeurs différentes sera transformé en k attributs binaires. Ça permet d'améliorer les performances.

normalizeAttributes : les valeurs des attributs (y compris les attributs nominaux qui seront passées dans le filtre nominalToBinaryFilter) seront toutes ramenées entre -1 et 1.

normalizeNumericClass : permet de normaliser la classe entre -1 et 1, si elle est numérique (permet d'améliorer les performances du réseaux). La normalisation est faite en interne et les valeurs du résultats seront remis aux valeurs originales de la classe.

trainingTime : le nombre de fois d'apprentissage (où l'on fera passer l'ensemble d'apprentissage à travers le réseau).



Exercice 2

ANN sous R

1. Ouvrez R et installez le package *« nnet»* avec
   ```
   > install.packages("nnet")
   ```
2. Chargez le jeu de données
   ```
   > iris <- read.csv("iris.arff.csv")
   > summary(iris)
   ```
3. Chargez la bibliothèque avec
   ```
   > library(nnet)
   ```
4. Découpez notre jeu de données en 2 parties (apprentissage et validation) de manière aléatoire
   ```
   > set.seed(1509)
   > index = sample(1:nrow(iris), round(2*nrow(iris)/3), replace=FALSE)
   > app = iris[index,]
   > val = index[-index,]
   ```
5. Construire le modèle de classification sur un les données d'apprentissage *app*
   ```
   > model <- nnet(Class ~ ., data = app, size = 2, decay = 0, maxit=100, linout=T)
   > print(model)
   > summary(model)
   ```
6. Vérifiez la prédiction du modèle construit sur les données de test
   ```
   > pred = predict(model, newdata = val[, -10], type = "class")
   > mat = table(pred, val[, 10])
   > mat
   > taux = sum(diag(mat))/sum(mat)
   > taux
   ```
7. Optimisation du modèle : Trouvez les paramètres optimales avec
   ```
   > library(e1071)
   > set.seed(06072012)
   > tune.model = tune.nnet(Class ~ ., data = app, size = c(1, 3, 5), decay = c(0.1, 0.001,
   0.000001))
   > tune.model
   ```
8. Refaire les questions 5 et 6 en utilisant les nouveaux paramètres