Série apprentissage supervisé
Mme Hamdad Leila
BDM. 2CSSIT; 2CSSIL G1
Amriou/Chikouche

Réseau de neurones et comparaison: Script utilisant "Glass"

I. Réseau de neurones

- La première étape consiste toujours à importer la dataset "Glass" et la partitionner ensuite, en training set et test set; (comme vu dans les TP précédents).
- La prochaine étape consiste à préparer l'architecture de notre réseau de neurones en précisant quelques paramètres ainsi que hyperparamètres (decay, size, maxit…) qui vont guider l'apprentissage du modèle. On peut consulter les détails à l'aide de la commande:

  ```
  > help(nnet)
  ```

  On pourra, donc, lancer le training du neural network comme suit:

  ```
  RN<- nnet(Type~., Appren, size = 9, decay = )
  summary(RN)
  ```

  Voici les résultats obtenu:

  ```
  > RN<- nnet(Type~., Appren, size = 9, decay = )
  # weights:  150
  initial  value 281.390542
  final  value 227.503432
  converged
  > summary(RN)
  a 9-9-6 network with 150 weights
  options were - softmax modelling
   b->h1 i1->h1 i2->h1 i3->h1 i4->h1 i5->h1 i6->h1 i7->h1 i8->h1 i9->h1
  ```

  Interprétation: Notre réseau de neurones contient 150 poids ou "nombre de paramètres ajustables" avec une valeur initiale d'entropie de 281.4 et une valeur finale de 227.5. Notre neural network a 9 nœuds d'entrée, une couche cachée avec 9 nœuds et 6 nœuds de sortie.
  Le summary indique que la couche de sortie du réseau utilise la fonction d'activation softmax (Classification multiclasse).
  Enfin, on affiche les tous les paramètres du modèle (poids) et leurs valeurs.
- L'étape suivante est la prédiction sur la Testset afin de pouvoir évaluer notre modèle (table de confusion, error, accuracy… etc) :

  ```
  pred=predict(RN,Test, type="class")
  ```
- Pour améliorer les performances du modèle, on peut faire appel à la fonction du tuning pour effectuer une Grid search qui permettra a titre d'exemple de trouver le decay et size optimaux avec la ligne:

  ```
  results<-tune.nnet(Type~.,data=Glass,size = 4:9)
  best_size <- results$best.parameters$size # avoir le best size
  ```
- Enfin, on effectue un nouveau training en permanent le size optimal pour voir les résultats

Remarque: Glass est une Dataset déséquilibrée et compliquée. Donc, la performance du modèle sera, presque, considérée moyenne et il n y aura pas un overfitting.
Dans le cas ou un réseau de neurones soufre d'un surapprentissage. Une "Feature selection" ou une "Réduction de dimensionnalité" sera nécessaire.

II. Comparaison

On a décidé de comparer le réseau de neurones avec les algorithmes qu'on avait vu durant les séances précédentes, on reste toujours dans la même Dataset et on garde la même partition (train/test) dans tous les algorithmes. Voici les résultats obtenus:

```
> cat("Accuracy Nnet: ", accuracy, "\n")
Accuracy Nnet:  0.6153846
> cat("Exec Time Nnet: ", Te, "\n")
Exec Time Nnet:  0.03825378
>
```

```
> cat("Accuracy Decision tree: ", accuracy2, "\n")
Accuracy Decision tree:  0.6615385
> cat("Exec time Decision Tree: ", Te2, "\n")
Exec time Decision Tree:  0.007698774
```

```
> cat("Accuracy Random Forest: ", accuracy3, "\n")
Accuracy Random Forest:  0.7538462
> cat("Exec time Random Forest: ", Te3, "\n")
Exec time Random Forest:  0.06019545
>
```

```
> cat("Accuracy SVM: ", svm_acc, "\n")
Accuracy SVM:  0.6769231
> cat("Exec Time SVM: ", Te4, "\n")
Exec Time SVM:  0.006704569
>
```

Conclusion: On voit que le meilleur algorithme en termes de "Accuracy" est Random Forest. En termes de temps d'exécution, SVM est le plus rapide. Ce dernier, n'est pas pénalisé par la dimensionnalité.
Le tableau ci-dessous résume les résultats obtenus:

Algorithme	Accuracy	Temps d'exécution
Decision Tree	0.6615385	0.007698774
Random Forest	0.7538462	0.06019545
SVM	0.6769231	0.006704569
Réseau de neurones	0.6153846	0.03825378

Temps d'exécution: SVM < Decision Tree < Nnet < Random Forest
Accuracy: Rnadom Forest > SVM > Decision Tree > Nnet