Présentation PDF

Marc XU

29 mars 2019

Réseaux de neurones

Avec le logiciel Rmarkdown (ou RStudio), on peut utiliser deux librairies qui permettent toutes les deux de faire des calculs d'apprentissage sur les réseaux neurones.
Les deux librairies sont `nnet` et `neuralnet`.

Libraries `nnet`

La librarie `nnet` est la plus ancienne des deux librairies. Cette librairie est utiliser pour les réseaux neuronaux en aval avec une seule couche cachée (et pour les modèles log-linéaires multinomiaux).

Il existe une fonction `nnet` qui permet d'installer un réseau neuronal à couche unique caché. Pour le calcul de probabilités, nous pouvons utiliser la fonction `predict`.

Librairie `neuralnet`

La librarie `neuralnet` permet des réglages flexibles par le biais d'un choix personnalisé d'erreurs et d'une fonction d'activation.

Avec la fonction `neuralnet`, nous pouvons installer un réseau neuronal à un ou plusieurs couches cachés, nous sommes donc plus limités dans le nombre de couches cachés. Cependant, pour calculer les probabilités, nous devrons alors utiliser la fonction `compute`.

Utilisation des deux méthodes sur les données d'IRIS

Création de la matrice des données

Pour les deux méthodes, nous avons crée trois nouveaux vecteurs qui servent à montrer si l'individu est-il de l'espèce d'IRIS ou pas. Par exemple pour l'espèce SETOSA, on crée un vecteur qui contient deux valeurs. Soit "1" soit "0", qui représente le fait d'être SETOSA et le fait de ne pas être SETOSA respectivement.

Réseaux de neurones avec `nnet`

Les trois vecteurs crées serviront pour calculer AUC (aire sous la courbe)

Réseaux de neurones avec `neuralnet`

Les trois vecteurs crées remplaceront la variable "Species" dans le jeu de données. Car la fonction `neuralnet` nécessite que la variable soit un facteur avec des valeurs binaires ("0" ou "1").

Prédiction

Nous avons utilisé la fonction `sample` qui permet de sélectionner des individus aléatoires dans le jeu de données. Nous créons deux data.frame qui correspond aux échantillons d'apprentissage et aux échantillons de test. Dans l'échantillon d'apprentissage, nous avons pris 2/3 des individus du jeu de données de départ. Alors que dans l'échantillon de test, nous avons les individus restants (1/3).

Cela permet d'avoir un modèle ayant des performances non biaisées.

Réseaux de neurones avec `nnet`

La fonction `nnet` retourne un objet de la classe `nnet` ou `nnet.formula`. Principalement structure interne, mais avec des composants :

- `wts` : le meilleur ensemble de poids trouvé.
- `value` : valeur du critère d'ajustement plus le terme de décroissance du poids.
- `fitted.values` : les valeurs ajustées pour les données d'entraînement.
- `residuals` : les résidus pour les données de formation.
- `convergence` : 1 si le nombre maximum d'itérations a été atteint, sinon 0.

Ensuite , on utilise la fonction `predict` qui prédit si l'individu est de quelle espèce (parmi les trois espèces). Puis, nous pouvons créer une matrice de fonction avec la fonction `table`. Dans la matrice, la diagonale représente toutes les individus qui sont correctement prédits, en dehors de la diagonale, la prédiction n'a pas été correct depuis les autres variables (qui aident à prédire l'individu est de quelle espèce).

```
predict.ind = predict(iris.nn, newdata = iris.app, type = "class")
matrice = table(predict.ind, iris.app$Species, dnn = c("predit", "observe"))
```

Réseaux de neurones avec `neuralnet`

La fonction `neuralnet` retourne un objet de classe nn. Un objet de la classe nn est une liste contenant au maximum les composants suivants :

- `call` : ce qu'on a saisit dans la fonction `neuralnet`
- `response` : extrait de l'argument data
- `covariable` : les variables extraites de l'argument data
- `données` : l'argument des données.
- `résultat net` : une liste contenant le résultat global du réseau de neurones pour chaque répétition
  - probabilité pour chaque individu d'être ou pas l'espèce)
- `generalized.weights` : une liste contenant les poids généralisés du réseau de neurones pour chaque répétition
- `result.matrix` : une matrice contenant le seuil atteint, les étapes nécessaires, l'erreur, AIC et BIC (si calculé) et les poids pour chaque répétition. Chaque colonne représente une répétition

Avec cette fonction, nous pouvons choisir le nombre de couches cachés, ainsi nous pouvons utiliser la fonction `neuralnet` pour différents nombres de couches et voir les résultats s'ils sont meilleurs ou pas.

Ensuite, on utilise la fonction `compute` qui permet de calculer les probabilités et nous pouvons visualiser la matrice des probabilités avec `compute.prob$net.result`.

```
compute.prob = compute(neuralnet(...), data, rep = 1)
```

Si les probabilités sont corrects, nous pouvons obtenir un tableau de confusion avec pour variable "observé" et "prédiction" dont chaque variable possède deux valeurs ("0" et "1").

Les critères

1. TBP (Taux de bien prédit) : les individus bien prédits sur le total des individus
2. Taux d'erreur : les individus mal prédits sur le total des individus ou `1-TBP`
3. Sensibilité : les individus prédits positifs sur le nombre total des individus positifs
4. Spécificité : les individus prédits négatifs sur le nombre total des individus négatifs
5. F1 score
6. Coefficient de corrélation de Matthiews

Courbe ROC

La courbe ROC est un outil qui catégorise des éléments en deux groupes distincts sur la base d'une ou plusieurs caractéristiques de chacun des éléments.

Nous créons un vecteur "seuil" qui contient les différentes valeurs du seuil. Ce vecteur montrera si l'individu est-il ou pas de telle espèce par rapport ce seuil.
Il y a également deux vecteurs qu'on crée qui contient chaque valeur de la sensibilité et de la spécificité.

Nous allons employer une boucle `for` qui fera trois choses :

- un vecteur sera utilisé pour contenir tous les individus dans l'échantillon d'apprentissage avec une valeur `TRUE` ou `FALSE`. La valeur `TRUE` indique que l'individu à une probabilité supérieure que le seuil d'être l'espèce indiquée. Et la valeur `FALSE` indique que l'individu n'a pas de probabilité supérieure par rapport au seuil
- vecteur_sensibilité : qui calculera la sensibilité de la matrice des individus prédits et des individus observés
- vecteur_spécificité : qui calculera la spécificité de la matrice des individus prédits et des individus observés

Recherche des paramètres optimaux

On utilise la fonction `tune.nnet`
La valeur de `best performance` donne le meilleur rendement atteint