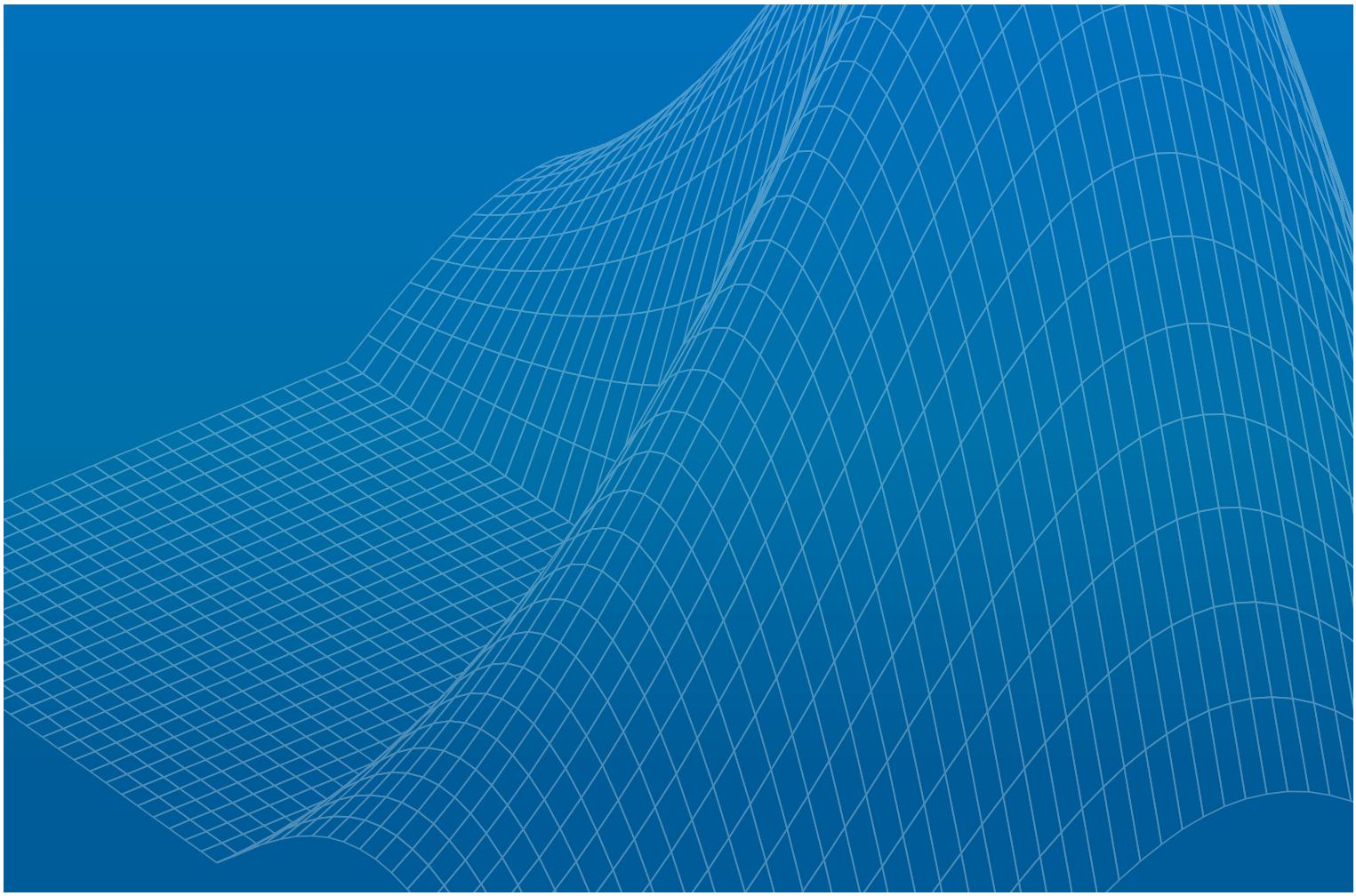


Les défis du Machine Learning : Comment choisir le meilleur modèle et éviter le surajustement

La modélisation basée sur le Machine Learning est une compétence difficile à acquérir mais utile pour tous les utilisateurs qui travaillent avec des données. Quelle que soit la raison pour laquelle vous utilisez le Machine Learning, vous avez probablement été confronté à des problèmes de classification ou de surajustement (« overfitting »). Ce document vous explique comment réduire les effets de ces problèmes à l'aide de MATLAB®.

Modèles de classification

Choisir le bon modèle de classification

Qu'est-ce qu'un modèle de classification de données ?

Les modèles de classification permettent d'attribuer des éléments à un groupe discret ou une classe discrète sur la base d'un ensemble spécifique de caractéristiques.

Pourquoi est-ce si difficile de faire le bon choix ?

Chaque modèle a ses points forts et ses points faibles dans un scénario donné. Aucun schéma ne permet de déterminer avec certitude quel modèle vous devez utiliser sans simplifier à l'extrême les éléments à prendre en compte. Le choix d'un modèle de classification est également étroitement lié à l'étude de cas et implique une bonne compréhension de ce que vous voulez faire.

Comment pouvez-vous vous assurer de choisir le bon modèle ?

Pour commencer, vérifiez que vous pouvez répondre aux questions suivantes :

- Quel est votre volume de données, et ces données sont-elles continues ?
- De quel type de données s'agit-il ?
- Que voulez-vous en faire ?
- Dans quelle mesure est-il important de visualiser le processus ?
- De quel niveau de détail avez-vous besoin ?
- Le stockage est-il un facteur contraignant ?

Une fois que vous comprenez mieux le type de données à utiliser et ce que vous allez en faire, vous pouvez commencer à examiner les points forts des différents modèles. Il existe quelques règles générales qui peuvent vous aider à choisir le meilleur modèle de classification, mais ce ne sont que de simples points de départ. Si vous travaillez avec un grand volume de données (dans ce cas de figure, une faible variation dans la performance ou la précision peut avoir un effet important), alors le choix de la bonne approche requiert souvent de travailler par essai et erreur pour atteindre le bon équilibre entre complexité, performances et précision. Les sections suivantes décrivent certains modèles courants que vous devez connaître.

Modèle bayésien naïf

Si vos données ne sont pas complexes et si votre tâche est relativement simple, vous pouvez essayer d'opter pour un algorithme bayésien naïf. Il s'agit d'un classificateur de type biais élevé/variance faible, ce qui présente des avantages par rapport aux algorithmes de régression logistique et des plus proches voisins si vous travaillez avec un volume de données disponibles limité pour entraîner un modèle.

Le modèle bayésien naïf constitue également un choix adapté quand le processeur et les ressources mémoire sont un facteur contraignant. Du fait de sa grande simplicité, il n'a pas tendance à surajuster les données, et il peut être entraîné très rapidement. Il convient également parfaitement lorsque de nouvelles données continues sont utilisées pour mettre à jour le classificateur.

Si le volume et la variance des données augmentent, et si vous avez besoin d'un modèle plus complexe, d'autres classificateurs seront probablement plus adaptés. En outre, son analyse simple n'est pas adaptée aux hypothèses complexes.

L'algorithme bayésien naïf est souvent le premier que les scientifiques essaient d'utiliser lorsqu'ils travaillent avec du texte (systèmes anti-spam et analyse des sentiments, par exemple). Il est recommandé de tester cet algorithme avant de l'écarter.

Modèle *k*-plus proches voisins

Le fait de catégoriser des points de données en fonction de leur distance par rapport aux autres points d'un jeu de données d'apprentissage peut constituer une méthode simple mais efficace pour classifier des données. Les k-plus proches voisins (*k*NN) est l'algorithme de type « coupable par association ».

Le *k*NN est un classificateur de type « lazy learner » basé sur les instances, ce qui signifie qu'il n'y a pas de véritable phase d'apprentissage. Vous chargez les données d'apprentissage dans le modèle, puis vous n'y touchez plus jusqu'au moment où vous voulez vraiment commencer à utiliser le classificateur. Lorsque vous avez une nouvelle instance de requête, le modèle *k*NN recherche le nombre *k* spécifié de plus proches voisins ; ainsi, si *k* est égal à 5, vous obtenez les 5 plus proches voisins. Si vous souhaitez appliquer une étiquette ou une classe, le modèle choisit à quel emplacement elle doit être classée. Si vous résolvez un problème de régression et souhaitez identifier un nombre continu, prenez la moyenne des valeurs *f* des *k* plus proches voisins.

Même si le temps d'apprentissage du modèle *k*NN est court, le temps de requête réel (et l'espace de stockage) peut être plus long que celui des autres modèles. Cela se vérifie particulièrement quand le nombre de points de données augmente, car vous conservez l'ensemble des données d'apprentissage, et pas uniquement un algorithme.

Le principal désavantage de cette méthode est le fait qu'elle peut être faussée par des attributs non pertinents qui masquent les attributs importants. D'autres modèles comme les arbres de décision sont plus efficaces pour ignorer ces distractions. Il existe des méthodes pour corriger ce problème, par exemple appliquer des pondérations à vos données. Vous devrez donc faire preuve de jugement au moment de décider quel modèle utiliser.

Arbres de décision

Pour savoir comment un arbre de décision prédit une réponse, suivez les décisions dans l'arbre à partir du nœud racine (début), jusqu'à atteindre un nœud feuille qui contient la réponse. Les arbres de classification donnent des réponses nominales, de type vrai ou faux. Les arbres de régression donnent des réponses numériques.

Les arbres de décision sont relativement faciles à suivre : vous pouvez voir une représentation complète du chemin entre la racine et la feuille. Ceci est utile si vous devez partager les résultats avec des personnes qui souhaitent savoir comment une conclusion a été atteinte. Ils sont également relativement rapides.

Le principal désavantage des arbres de décision est leur tendance à surajuster, mais il existe des méthodes ensemblistes qui permettent de remédier à cette situation. Dans un exemple pertinent (rédigé pour une compétition Kaggle), Toshi Takeuchi utilise un ***« bagging » d'arbres de décision*** pour déterminer la probabilité de survie d'une personne à la catastrophe du Titanic.

Machines à vecteurs de support

Vous pouvez utiliser une machine à vecteurs de support (SVM) lorsque vos données comportent exactement deux classes. Une SVM classifie les données en identifiant le meilleur hyperplan qui sépare tous les points de données d'une classe et ceux de l'autre classe (le meilleur hyperplan pour une SVM est celui qui offre la plus grande marge entre deux classes). Vous pouvez utiliser une SVM avec plus de deux classes, auquel cas le modèle créera un ensemble de sous-problèmes de classification binaire (avec un système d'apprentissage SVM pour chaque sous-problème).

L'utilisation d'une SVM présente un certain nombre d'avantages non négligeables : pour commencer, une SVM est extrêmement précise et n'a pas tendance à surajuster les données. Ensuite, une fois entraînée, elle constitue une solution rapide car il lui suffit de choisir entre l'une des deux classes. Les modèles SVM étant très rapides, une fois votre modèle entraîné, vous pouvez vous débarrasser des données d'apprentissage si vous disposez de ressources de mémoire limitées. Ils présentent également la particularité de très bien gérer les classifications complexes et non linéaires.

Toutefois, les SVM doivent être entraînées et réglées à l'avance, ce qui signifie que vous devez passer du temps sur le modèle avant de pouvoir commencer à l'utiliser. À noter également que le fait d'utiliser le modèle avec plus de deux classes a un impact important en termes de vitesse.

Réseaux de neurones

Un réseau de neurones artificiels peut apprendre et, après un apprentissage approprié, il saura donc trouver des solutions, reconnaître des modèles, classifier des données et prévoir les événements à venir. Les réseaux de neurones artificiels sont régulièrement utilisés pour résoudre des problèmes plus complexes, notamment en ce qui concerne la reconnaissance de caractères, la prédiction boursière et la compression d'image.

Le comportement d'un réseau neuronal est défini par la connexion entre ses éléments de calculs individuels, ainsi que par la force de ces connexions, ou poids. Les poids sont ajustés automatiquement en assurant l'apprentissage du réseau sur la base d'une règle d'apprentissage spécifiée jusqu'à ce qu'il exécute correctement la tâche voulue.

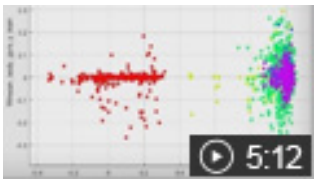
Pour les utilisateurs expérimentés, les réseaux de neurones artificiels constituent une solution idéale pour modéliser des données non linéaires avec un nombre élevé de caractéristiques d'entrée. S'ils sont correctement utilisés, ils peuvent résoudre des problèmes qui sont trop difficiles à traiter avec un simple algorithme. Cependant, les réseaux de neurones exigent des calculs complexes, il est difficile de comprendre comment un réseau de neurones artificiels est arrivé à une solution (et donc d'en déduire un algorithme), et l'ajustement d'un réseau de neurones artificiels s'avère souvent peu pratique, car vous pouvez uniquement modifier les entrées de votre jeu d'apprentissage et recommencer celui-ci.

Validation croisée des classifications

La validation croisée est une technique d'évaluation des modèles qui permet d'évaluer les performances d'un algorithme de Machine Learning lorsque vous faites des prédictions concernant de nouveaux jeux de données sur lesquels il n'a pas été entraîné. Cette opération s'effectue en partitionnant un jeu de données et en utilisant un sous-ensemble pour entraîner l'algorithme et les données restantes à des fins de test. Cette technique est présentée de façon plus détaillée dans la section intitulée Éviter le surajustement.

L'application Classification Learner

MATLAB vous permet de déterminer beaucoup plus facilement quel modèle fonctionne le mieux grâce à son application Classification Learner. Vous pouvez également utiliser MATLAB pour évaluer les performances de votre modèle et vérifier vos résultats.



Données classifiées à l'aide de l'application Classification Learner

L'application Classification Learner vous permet d'entraîner des modèles et de classifier les données dans le cadre du Machine Learning supervisé.

L'application Classification Learner permet d'effectuer les tâches du Machine Learning courantes, comme l'exploration interactive des données, la sélection des caractéristiques, la spécification de schémas de validation, l'apprentissage des modèles et l'évaluation des résultats. Plusieurs types de classifications vous sont proposés, parmi lesquels les arbres de décision, les machines à vecteurs de support et les k-plus proches voisins. Vous pouvez ensuite faire votre choix parmi plusieurs méthodes ensemblistes, notamment le bagging, le boosting et le sous-espace aléatoire.

Classification Learner vous aide à choisir le meilleur modèle pour vos données en vous permettant d'évaluer et de comparer des modèles à l'aide de matrices de confusion et de courbes de caractéristiques de performance (Receiver Operating Characteristic, ROC). Vous pouvez exporter des modèles de classification dans l'espace de travail MATLAB pour générer des prédictions sur de nouvelles données, ou générer du code MATLAB pour intégrer des modèles dans des applications telles que la vision par ordinateur, le traitement du signal et l'analytique de données.

L'application Classification Learner supporte les éléments suivants :

- Arbres de décision : arbre profond, arbre moyen et arbre peu profond
- Machines à vecteurs de support : SVM linéaire, SVM à profil gaussien fin, SVM à profil gaussien moyen, SVM à profil gaussien grossier, SVM quadratique et SVM cubique
- Classificateurs des plus proches voisins : *k*NN fin, *k*NN moyen, *k*NN grossier, *k*NN cosinus, *k*NN cubique et *k*NN pondéré
- Classificateurs ensemblistes : arbres de type « boosting » (AdaBoost, RUSBoost), arbres de type « bagging », *k*NN de type « sous-espace » et sous-espace discriminant

Cette application vous permet d'effectuer les opérations suivantes :

- Évaluer les performances du classificateur à l'aide de matrices de confusion, de courbes ROC ou de nuages de points
- Comparer la précision des modèles en utilisant le taux de classification erronée sur le jeu de validation
- Améliorer la précision des modèles grâce à des options avancées et à la sélection des caractéristiques
- Exporter le meilleur modèle dans l'espace de travail afin de réaliser des prédictions concernant les nouvelles données
- Générer du code MATLAB pour entraîner les classificateurs sur de nouvelles données

Surajustement

Éviter le surajustement

Qu'est-ce que le surajustement ?

Le surajustement signifie que votre modèle est si étroitement aligné sur les jeux de données d'apprentissage qu'il ne sait pas comment réagir aux nouvelles situations.

Pourquoi est-il difficile d'éviter le surajustement ?

Bien souvent, la difficulté à éviter le surajustement est due à des données d'apprentissage insuffisantes. La personne responsable du modèle n'est pas forcément la même que la personne chargée de recueillir les données, et cette dernière peut ne pas savoir quel volume de données est nécessaire pour les phases de test et d'apprentissage.

Un modèle surajusté renvoie très peu d'erreurs, ce qui en fait une solution intéressante au premier abord. Malheureusement, le modèle contient trop de paramètres par rapport au système sous-jacent. L'algorithme d'apprentissage règle ces paramètres pour réduire la fonction de perte, mais au lieu d'obtenir le comportement attendu, à savoir une généralisation des tendances sous-jacentes, il en résulte que le modèle est surajusté par rapport aux données d'apprentissage. Lorsque des volumes importants de nouvelles données sont introduits sur le réseau, l'algorithme ne peut pas faire face, et des erreurs importantes commencent à apparaître.

Idéalement, votre modèle de Machine Learning doit être aussi simple que possible et suffisamment précis pour produire des résultats pertinents. Plus votre modèle est complexe, plus il aura tendance à surajuster.

Comment éviter le surajustement ?

La meilleure façon d'éviter le surajustement consiste à vérifier que vous utilisez suffisamment de données d'apprentissage. Il est généralement admis que 10 000 points de données au minimum sont nécessaires pour entraîner un modèle, mais ce nombre dépend en grande partie du type de tâche que vous effectuez (la classification bayésienne naïve et la classification *k*-plus proches voisins nécessitent beaucoup plus de points d'échantillonnage). Ces données doivent également refléter avec précision la complexité et la diversité des données avec lesquelles le modèle devrait travailler.

Utilisation de la régularisation

Si vous avez déjà commencé à travailler et pensez que votre modèle surajuste les données, vous pouvez essayer de remédier à cette situation à l'aide de la ***régularisation***. La régularisation pénalise les paramètres élevés afin d'empêcher le modèle de trop dépendre des points de données individuels et de devenir trop rigide. La fonction objectif est modifiée et devient $Error+\lambda f(\theta)$, où $f(\theta)$ augmente en même temps que les composants de (θ) et λ représente la force de la régularisation.

La valeur que vous sélectionnez pour λ détermine le niveau de protection contre le surajustement. Si $\lambda=0$, cela signifie que vous ne souhaitez pas du tout corriger le surajustement. En revanche, si la valeur de λ est trop élevée, votre modèle fera en sorte que la valeur de θ soit aussi peu élevée que possible (au lieu d'avoir un modèle qui fonctionne correctement sur votre jeu d'apprentissage). L'identification de la meilleure valeur pour λ peut prendre un certain temps.

Validation croisée

Lorsque vous utilisez le Machine Learning, une étape importante consiste à contrôler les performances de votre modèle. La validation croisée est l'une des méthodes disponibles pour évaluer les performances d'un algorithme de Machine Learning. Cette technique consiste à faire faire des prédictions à l'algorithme en utilisant des données qui n'ont pas été employées pendant la phase d'apprentissage. La validation croisée partitionne un jeu de données et utilise un sous-ensemble pour entraîner l'algorithme et les données restantes à des fins de test. La validation croisée n'utilise pas l'ensemble des données pour créer un modèle, c'est pourquoi cette méthode est couramment utilisée pour empêcher le surajustement pendant l'apprentissage.

Chaque session de validation croisée implique de partitionner de façon aléatoirele jeu de données d'origine en un jeu d'apprentissage et un jeu de test. Le jeu d'apprentissage est ensuite utilisé pour entraîner un algorithme ***d'apprentissage supervisé***, et le jeu de test est utilisé pour évaluer ses performances. Ce processus est répété plusieurs fois, et l'erreur de validation croisée moyenne est utilisée comme indicateur de performance.

Les techniques courantes de validation croisée incluent notamment :

- **k-fold :** Partitionne les données en un nombre k de sous-ensembles choisis de façon aléatoire et de taille à peu près équivalente. Un sous-ensemble est utilisé pour valider le modèle entraîné à l'aide des sous-ensembles restants. Ce processus est répété un nombre k de fois, de façon à ce que chaque sous-ensemble soit utilisé exactement une fois pour la validation.
- **Holdout :** Partitionne les données en exactement deux sous-ensembles au ratio spécifié pour l'apprentissage et la validation.
- **Leaveout :** Partitionne les données en utilisant l'approche k-fold, où k est égal au nombre total d'observations dans les données. Également appelée validation croisée leave-one-out.
- **Sous-échantillonnage aléatoire répété :** Applique des répétitions ***Monte Carlo*** de données de partitionnement aléatoires et agrège les résultats à l'issue de toutes les exécutions.
- **Stratification :** Partitionne les données de façon à ce que les jeux d'apprentissage et de test aient à peu près les mêmes proportions de classe dans la réponse ou la cible.
- **Resubstitution :** Ne partitionne pas les données ; utilise les données d'apprentissage pour la validation. Cette méthode produit souvent des estimations de performances trop optimistes, et doit être évitée si les données disponibles sont suffisantes.

Fonctions MATLAB permettant de contrôler le surajustement

Des techniques de régularisation sont employées pour éviter le surajustement statistique dans un modèle prédictif. En introduisant des informations supplémentaires dans le modèle, les algorithmes de régularisation peuvent gérer la multicolinéarité et les prédicteurs redondants, contribuant ainsi à simplifier le modèle et à le rendre plus précis. Ces algorithmes fonctionnent généralement par application d'une pénalité pour complexité. Ils peuvent par exemple ajouter les coefficients du modèle dans la minimisation ou en incluant une pénalité de rudesse.

Pour les réseaux de neurones, la régularisation peut être effectuée en toute simplicité dans Neural Network Toolbox™ via la fonction **`trainbr`**. `trainbr` est une fonction d'apprentissage de réseaux qui met à jour les valeurs de la pondération et du biais en fonction de l'optimisation de Levenberg-Marquardt. Elle minimise une combinaison d'erreurs et de paramètres quadratiques, puis détermine la bonne combinaison permettant de produire un réseau qui généralise bien. Ce processus est appelé régularisation bayésienne.

La méthode par défaut pour améliorer la généralisation est appelée ***arrêt anticipé***. Cette technique est automatiquement proposée pour toutes les fonctions de création de réseaux supervisés, y compris les fonctions de création de réseaux rétropropagateurs telles que `feedforwardnet`.

Avec la technique de l'arrêt anticipé, les données disponibles sont divisées en trois sous-ensembles. Le premier sous-ensemble est le jeu d'apprentissage, qui est utilisé pour calculer le gradient et mettre à jour les paramètres et les biais du réseau. Le deuxième sous-ensemble est le jeu de validation. L'erreur sur le jeu de validation est contrôlée pendant le processus d'apprentissage. Normalement, l'erreur de validation est réduite pendant la phase initiale de l'apprentissage, à l'instar de l'erreur sur le jeu d'apprentissage. Toutefois, quand le réseau commence à surajuster les données, l'erreur sur le jeu de validation commence généralement à augmenter Lorsque l'erreur de validation augmente pour un nombre spécifié d'itérations (`net.trainParam.max_fail`), l'apprentissage est arrêté, et les paramètres et les biais au minimum de l'erreur de validation sont retournés. Le troisième sous-ensemble est le jeu de test, qui est utilisé sur le classificateur totalement entraîné, après les phases d'apprentissage et de validation, pour comparer les différents modèles.

Appliquées correctement, les techniques de l'arrêt anticipé et de la régularisation peuvent garantir la généralisation du réseau. En ce qui concerne l'arrêt anticipé, vous devez veiller à ne pas utiliser un algorithme qui converge trop rapidement. Si vous utilisez un algorithme rapide (par exemple `trainlm`), définissez les paramètres d'apprentissage de façon à ce que la convergence soit relativement lente. Vous pouvez par exemple définir mu avec une valeur relativement élevée comme 1, puis définir `mu_dec` et `mu_inc` avec des valeurs proches de 1 comme 0,8 et 1,5, respectivement. Les fonctions d'apprentissage **`trainscg`** et `trainbr` fonctionnent généralement bien avec l'arrêt anticipé.

Avec l'arrêt anticipé, le choix du jeu de validation est également important. Le jeu de validation doit être représentatif de tous les points contenus dans le jeu d'apprentissage.

La validation croisée dans MATLAB

Statistics and Machine Learning Toolbox™ offre deux fonctions particulièrement utiles lorsque vous effectuez une validation croisée : `cvpartition` et `crossval`.

La fonction `cvpartition` permet de créer une partition de validation croisée pour des données. Avec une validation croisée k-fold, la syntaxe serait la suivante :

`c = cvpartition(n,'KFold',k)` crée un objet `c` appartenant à la classe `cvpartition` définissant une partition aléatoire pour la validation croisée k-fold sur un nombre n d'observations.

`c = cvpartition(group,'KFold',k)` crée une partition aléatoire pour une validation croisée k-fold stratifiée.

Utilisez `crossval` pour effectuer une estimation des pertes à l'aide de la validation croisée. Voici un exemple de syntaxe pour ce cas de figure :

`vals = crossval(fun,X)` effectue une validation croisée pour la fonction fun, appliquée aux données dans X.

`fun` est un handle de fonction associée à une fonction qui comporte deux entrées, le jeu d'apprentissage de `X, XTRAIN`, et le jeu de test de `X, XTEST`, comme indiqué ci-dessous :

```
testval = fun(XTRAIN,XTEST)
```

Pour plus d'informations, consultez ***Améliorer la généralisation des réseaux de neurones et éviter le surajustement*** et ***La validation croisée dans MATLAB***.

Conclusion

La maîtrise des techniques de Machine Learning requiert une combinaison de compétences diverses, mais les applications, les fonctions et les exemples d'apprentissage proposés dans MATLAB peuvent contribuer à rendre cet objectif atteignable. Ce document décrit quelques-uns des défis qui sont couramment rencontrés par les utilisateurs du Machine Learning. Pour découvrir des techniques plus utiles, visionnez le webinar ***Introduction au Machine Learning***, qui couvre plusieurs exemples de processus type pour l'apprentissage supervisé (classification) et l'apprentissage non supervisé (clustering).