Clayette Pierre
Pla Vincent
Racca Kevin
Stephan Kevin
Sarry Romain
Cortinovis Jonathan

I4 Groupe 2

Prédiction du risque de cancer

La conception du réseau de neurones a eu pour but de répondre à une problématique bien précise. En se basant sur un fichier de données contenant des informations sur des personnes atteintes ou non du cancer, il est capable de donner une estimation sur les chances d'avoir un cancer, suivant des critères bien précis.

Choix technique :

Le réseau de neurone est développé en C# en utilisant les avantages et les propriétés du langage orienté objet.

Nous pouvons remarquer une architecture précise qui permet de séparer les éléments, permettant une meilleur compréhension du fonctionnement du réseau, et une meilleur maintenance.

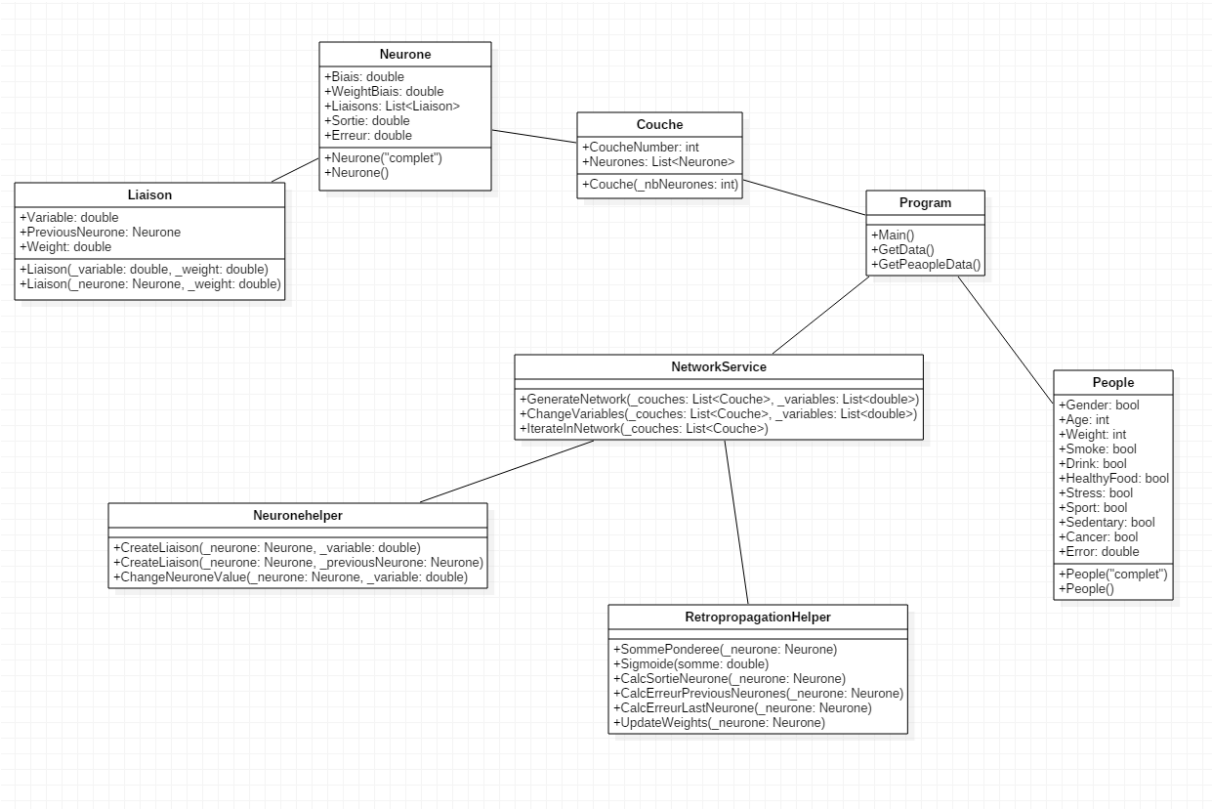

Figure 1: Diagramme de classe du RDN

Le diagramme de classe ci-dessus nous permet de voir que le programme de décompose en plusieurs classes « clés ».

Nous avons dans un premier temps la classe « Program » qui est le cœur de l'application. Elle permet d'ordonnancer le fonctionnement global et orchestre les différentes phases. Dans un premier temps la récupération des données (stockées dans un fichier JSON) puis la phase d'apprentissage (elle consiste à ajuster les poids des neurones jusqu'à ce que le taux total d'erreur atteigne le seuil d'acceptabilité fixé) et enfin la phase de prédiction qui permet à l'utilisateur de rentrer une série de critère et d'obtenir une estimation de la probabilité d'avoir un cancer.

Les données récupérées dans le JSON sont « parsées » pour être utilisable dans la classe « People ».

Nous remarquons aussi que l'architecture propre au réseau de neurone se découpe en trois classes :

- Couche : Le nombre de couche est défini dans la classe « Program », elle permet d'instancier un certain nombre de neurones.
- Neurone : Un neurone contient un poids de base (le biais) contenu dans une couche, et va permettre de créer des Liaisons ayant des poids qui leurs sont propres. Il a aussi une sortie et une erreur qui vont nous permettre d'effectuer la rétropropagation du gradient.
- Liaison : La liaison est l'élément qui va permettre de lier les neurones, entre eux et avec les variables d'entré.

Enfin, le fonctionnement propre au réseau de neurone se situe au niveau de la classe « NetworkService », qui contient les « helpers » permettant son bon fonctionnement. Ainsi, c'est elle qui permettra, en fonction des données renvoyées par l'architecture du RDN de :

- Généré le RDN
- Changer les variables d'entrée lors de la phase d'apprentissage
- Effectuer la rétropropagation du gradient
- Prédire la valeur de sortie du neurone pour calculer l'erreur

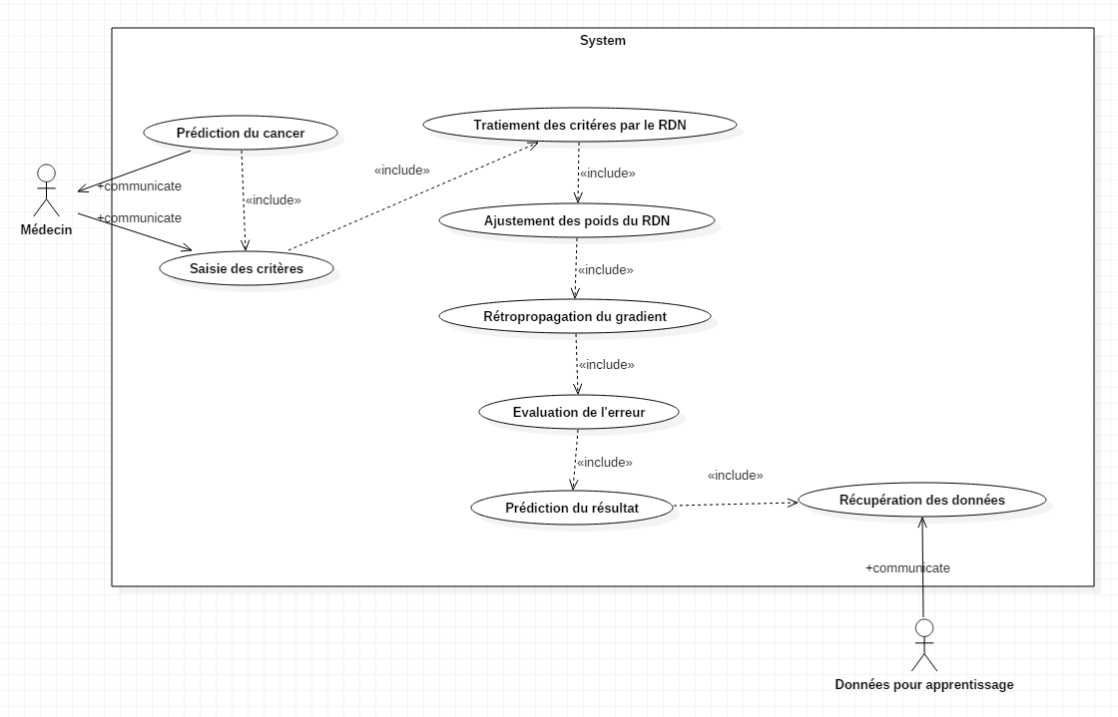

Figure 2:Use case

Le fonctionnement décrit par le « use case » est relativement simple.

Lorsque l'utilisateur saisi les critères, le réseau de neurones va les traiter selon les poids qu'il aura ajusté lors de la phase d'apprentissage. Cette phase d'apprentissage se défini par la rétropropagation du gradient basée sur un jeu de données généré en amont contenant 1000 cas différents.

Particularités :

- Le programme est défini avec 3 couches de neurones (5)(5)(1). En réalité, lors de l'exécution, le réseau comprend 4 couches puisque les variables d'entrée constituent une couche supplémentaire. Le RDN suit donc le schéma suivant (8)(5)(5)(1).
- L'incohérence des résultats lors de la prédiction provient de l'incohérence des données utilisées lors de la phase d'apprentissage.
  En effet, les données médicales étant très protégées, nous n'avons pas pu avoir accès à des données réelles et nous avons été obligé de les simuler.
  Nous avons eu recours à un script JS pour générer 1000 patients au format JSON.
  Pour chaque patient nous savons s'il a oui ou non un cancer. Ce critère est déterminé après avoir effectué une somme pondérée des autres critères afin de déterminer une probabilité de chance d'avoir ou non le cancer. Malgré ce procédé, les résultats sont souvent… surprenants.

  Le script utilisé se trouve dans le fichier .txt joint au dossier.

Algorithme :

Le programme sur base sur l'utilisation de la fonction sigmoïde pour la prédiction, le calcule de l'erreur et la rétropropagation du gradient comme expliqué dans la partie précédente.

Nous disposons d'un jeu de données de 1000 patients. A chaque itération, le programme prend un patient au hasard, calcule la prédiction au travers du réseau de neurone avec la fonction sigmoïde ($F(x) = 1 / (1 + \exp(-x))$).

L'erreur est ensuite calculée et utilisée pour effectuer la rétropropagation via la formule suivante :

Nouveau_poids_Xi = Ancien_poids_Xi + ConstanteApprentissage x Erreur en sortie x (prédiction_réseau)x(1-Prédiction du réseau) x Xi

L'apprentissage se termine lorsque le taux d'erreur TOTAL passe en dessous de 1%.

Résultats :

Pour une plus grande cohérence, nous avons choisi d'obtenir une estimation des risques de cancer par pourcentage. Ainsi, une personne ayant de fortes chances d'avoir un cancer obtiendra un pourcentage élevé (plus de 50%).

Au contraire une personne ayant une faible chance d'avoir un cancer (grâce à une bonne hygiène de vie par exemple) obtiendra un pourcentage bas (moins de 50%).

C’est le comportement qui est observé lors de l’utilisation de l’application, cependant, comme il s’agit de probabilitées, qu’il reste un pourcentage d’erreurs et que les données utilisées dans la phase d’apprentissage ne sont pas toujours cohérentes, certaines prédictions peuvent sembler surprenantes.