1 Introduction

Utilisation des composants deep learning de Keras (Tensorflow backend) sous KNIME. Implémentation de perceptrons simples et multicouches. Comparaison des performances.

« En dehors de R et Python, point de salut alors pour le deep learning ? » me demande, un brin inquiet, un internaute. J'ai compris quand j'ai voulu le rediriger vers d'autres langages (Java, C++) qu'il était surtout réfractaire à la programmation. Sa vraie question était plutôt de savoir s'il était possible d'exploiter les librairies de deep learning, comme le fameux tandem tensorflow / keras par exemple, sans passer par le codage informatique.

La réponse est oui. Dans ce tutoriel, nous verrons comment installer et rendre exploitables ces librairies dans le logiciel Knime, un de mes outils libres favoris pour la data science. Les opérations usuelles de manipulation de données, de modélisation et d'évaluation sont représentées par un « workflow » où les traitements sont symbolisés par des composants (nodes), et l'enchaînement des traitements par les liens entre ces nœuds. On parle alors de « programmation visuelle », moins traumatisante que l'écriture de ligne de code. Pour ce qui est de Knime, plus qu'une simple succession d'opérations, il s'agit bien de programmation puisqu'il est possible de définir des structures algorithmiques telles que les actions conditionnelles, des boucles, et même des fonctions sous forme de meta-nodes regroupant des actions.

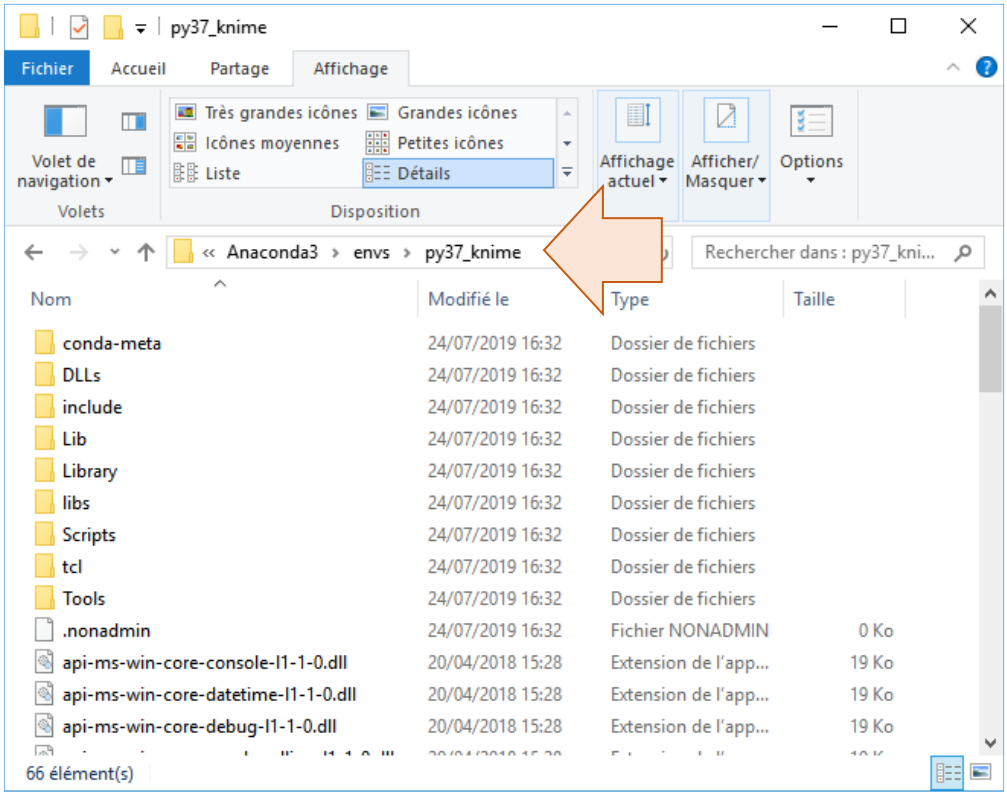
2 Installation des outils

L'intégration de Keras dans Knime nécessite une série d'installations assez délicates. Il faut être attentif pour que le tout fonctionne correctement.

1. Première information importante, Knime fait appel à Keras via Python c.-à-d. nos instructions sous Knime génèrent du code qui est exécuté par l'interpréteur Python. Il nous faut donc installer la distribution Anaconda - Python 3.7 (Python **3.7.3** au 24.07.2019, je travaille sous Windows 10 Education – 64 bits).
2. Le package Keras est en réalité une surcouche sur Tensorflow. Il faut tout d'abord installer ce dernier. J'ai opté pour la version stable via le gestionnaire de paquet pip (Tensorflow, version **1.13.1** au 24.07.2019).
3. Puis, le package Keras via conda cette fois-ci (Keras, version **2.2.4** au 24.07.2019).
4. Enfin, pour ce qui concerne Python tout du moins, nous installons un environnement d'exécution spécifique intégrant Keras pour Knime. Par rapport aux instructions que l'on trouve en ligne, je les ai adaptées comme ceci pour ma configuration :

```
conda create -y -n py37_knime python=3.7 pandas jedi keras=2.2.4
```

Un répertoire dédié est créé dans le sous-dossier « **envs** » de la distribution Anaconda comme nous pouvons le constater ci-dessous.



Nous passons à KNIME maintenant.

5. Nous récupérons et installons la version libre, Knime Analytics Platform (version 64 bits pour Windows en ce qui me concerne).
6. Il faut par la suite charger la libraire de traitement d'images via le menu HELP / INSTALL NEW SOFTWARE. Nous indiquons le dépôt « KNIME Community Extensions », puis nous sélectionnons « KNIME Community Extensions – Image Processing and Analysis » qui comporte - entres autres - les outils de deep learning « Deep Learning Extension » (Figure 1). Nous avions déjà exploré en partie ce package de traitement d'images dans un précédent tutoriel (Image Mining avec Knime, juin 2016).
7. Une branche « KNIME Labs » apparaît dans la collection de composants (Node Repository), avec l'extension « Deep Learning », lequel comporte 3 items (Figure 2) : « DL4J », une autre librairie de deep learning qui permet d'exploiter des réseaux de neurones sous KNIME ; « Keras », qui nous intéresse au premier chef dans ce tutoriel ; « Python » permet de faire l'interface avec l'interpréteur Python qui réalise les tâches en sous-main.

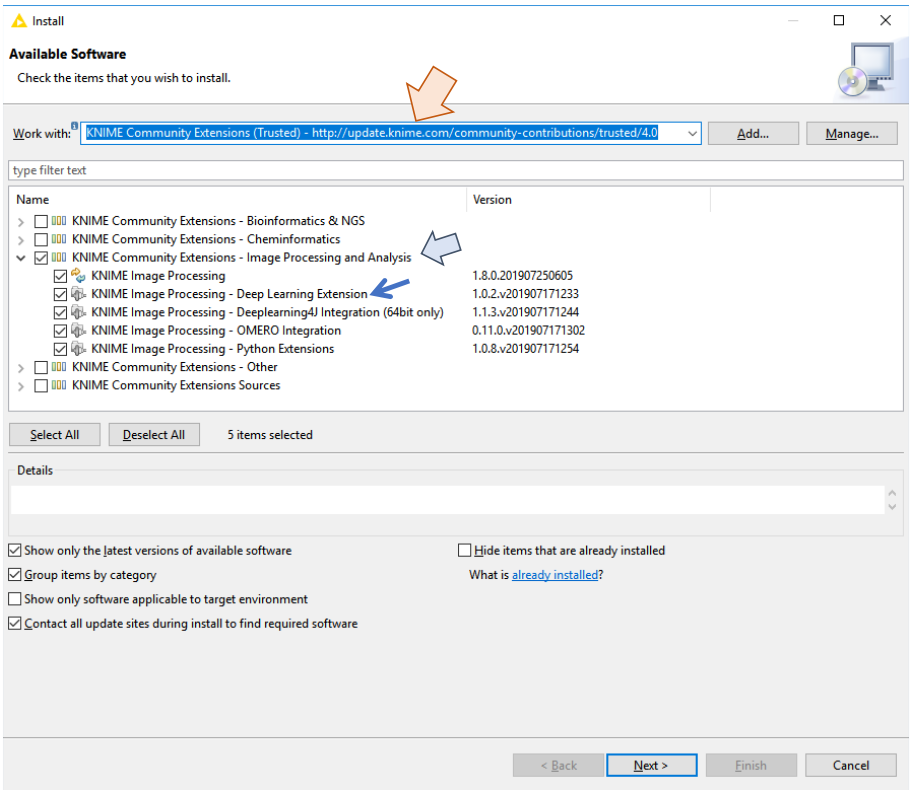


Figure 1 - Installation de l'extension "Image processing and Analysis" pour Knime

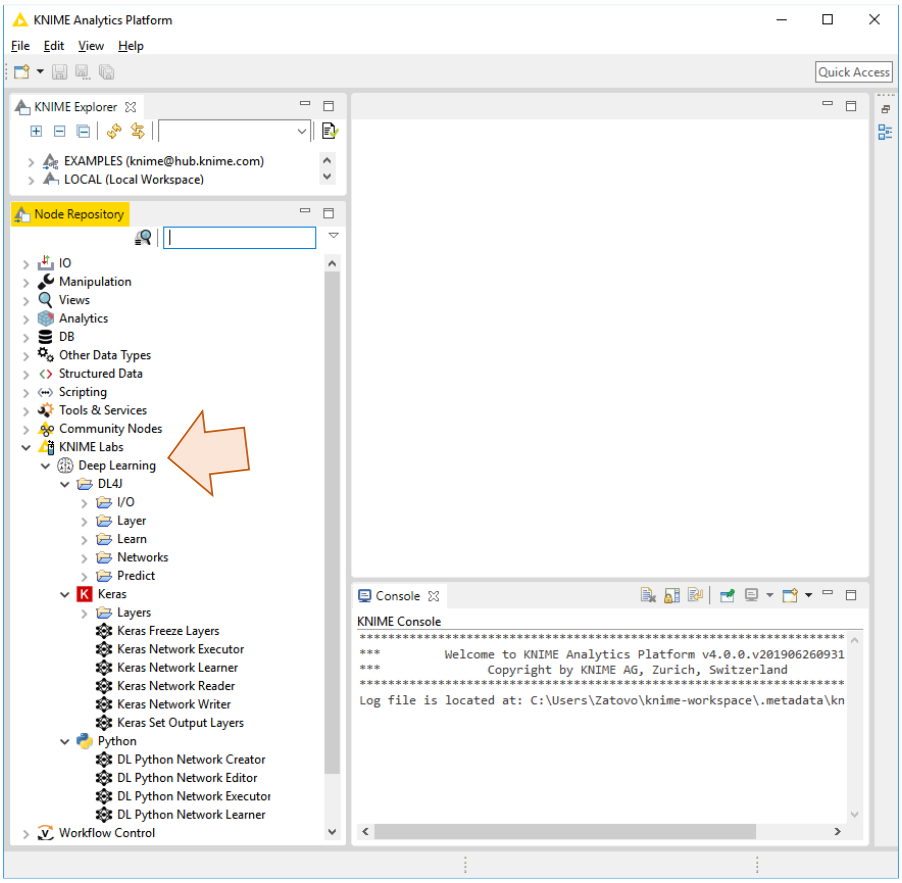


Figure 2 - L'extension "Deep Learning" de la branche "KNIME Labs" dans la liste des composants

8. Pour activer la connexion entre KNIME et l'interpréteur Python, nous actionnons le menu FILE / PREFERENCES. Pour Python 3 (Python **3.7.3**) que nous utilisons dans cette étude, nous indiquons la localisation de l'interpréteur (Figure 3).

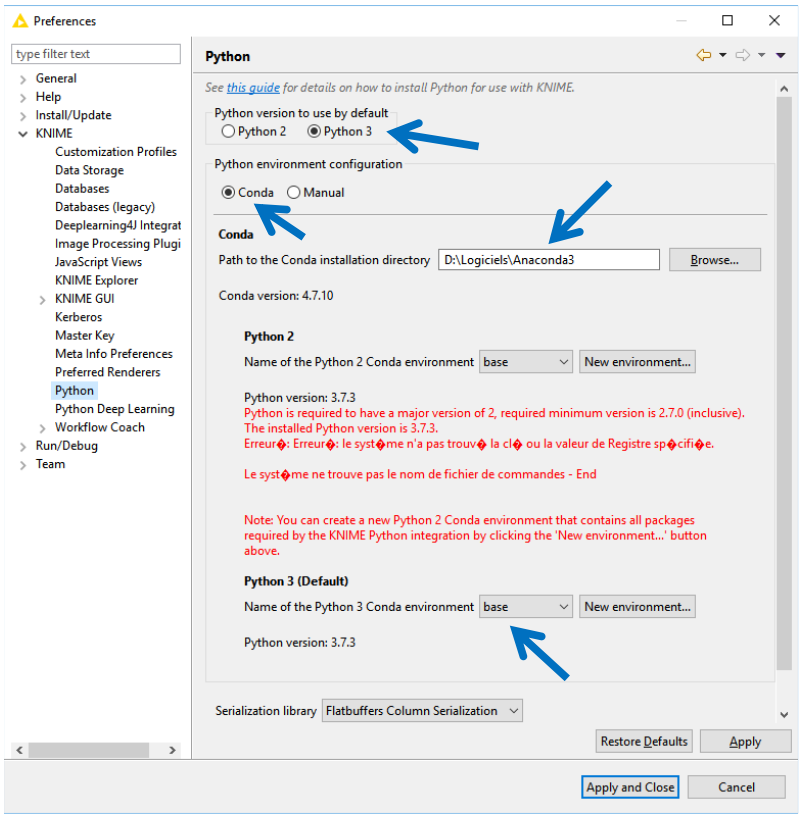


Figure 3 - Configuration de l'environnement pour Python

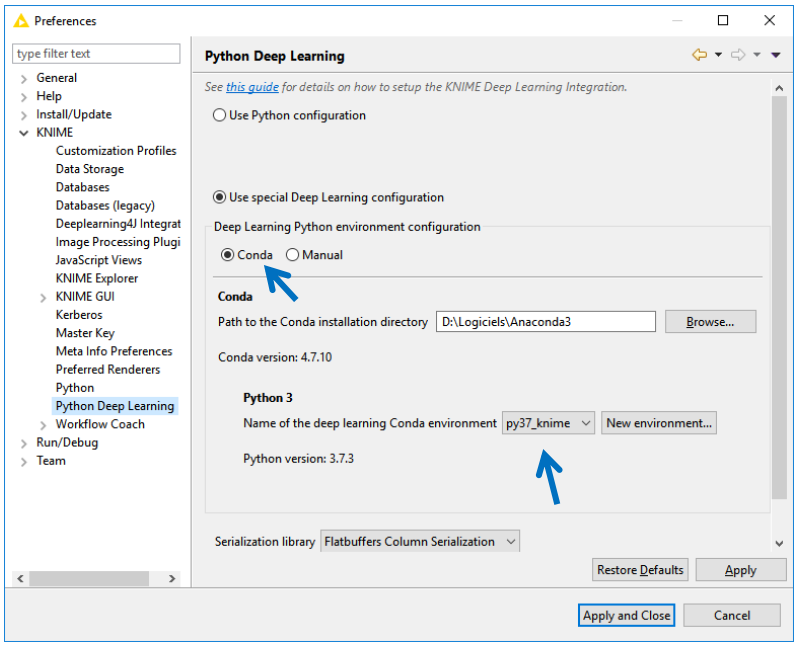


Figure 4 - Configuration de l'environnement "Deep Learning" sous Python

9. Puis nous précisons l'environnement « **py37_knime** », créé précédemment (étape 4), pour « Python Deep Learning ». Il nous permet d'accéder à la librairie Keras lorsque KNIME fera appel à l'interpréteur Python (Figure 4).

Lancement du logiciel KNIME. A la première exécution de KNIME, les composants deep learning fonctionnent correctement. Par la suite, j'ai dû, sans que je ne sache réellement pourquoi, lancer le logiciel via la ligne de commande DOS (Invite de commandes sous Windows) avec l'instruction…

```
> knime -clean
```

… pour que le dispositif (le lien avec Python pour l'exécution des réseaux de neurones, l'apprentissage des modèles et la prédiction en particulier) fonctionne à nouveau. J'ai mis du temps à trouver cette solution. A chaque utilisation, j'en étais réduit à désinstaller et réinstaller les outils, à redémarrer mon ordinateur, etc. Ça devenait passablement irritant à la longue.

3 Données

Nous traitons des données artificielles déjà utilisées dans un précédent tutoriel consacré au deep learning avec Keras / Tensorflow sous Python (Avril 2018). Les classes sont séparables par une parabole dans le plan (voir Figure 5, page 7). Nous disposons de 2 descripteurs (X1, X2) et de la variable cible binaire (Y ∈ {neg, pos}). Voici les premières lignes du fichier « **artificial2d_data2.xlsx** » :

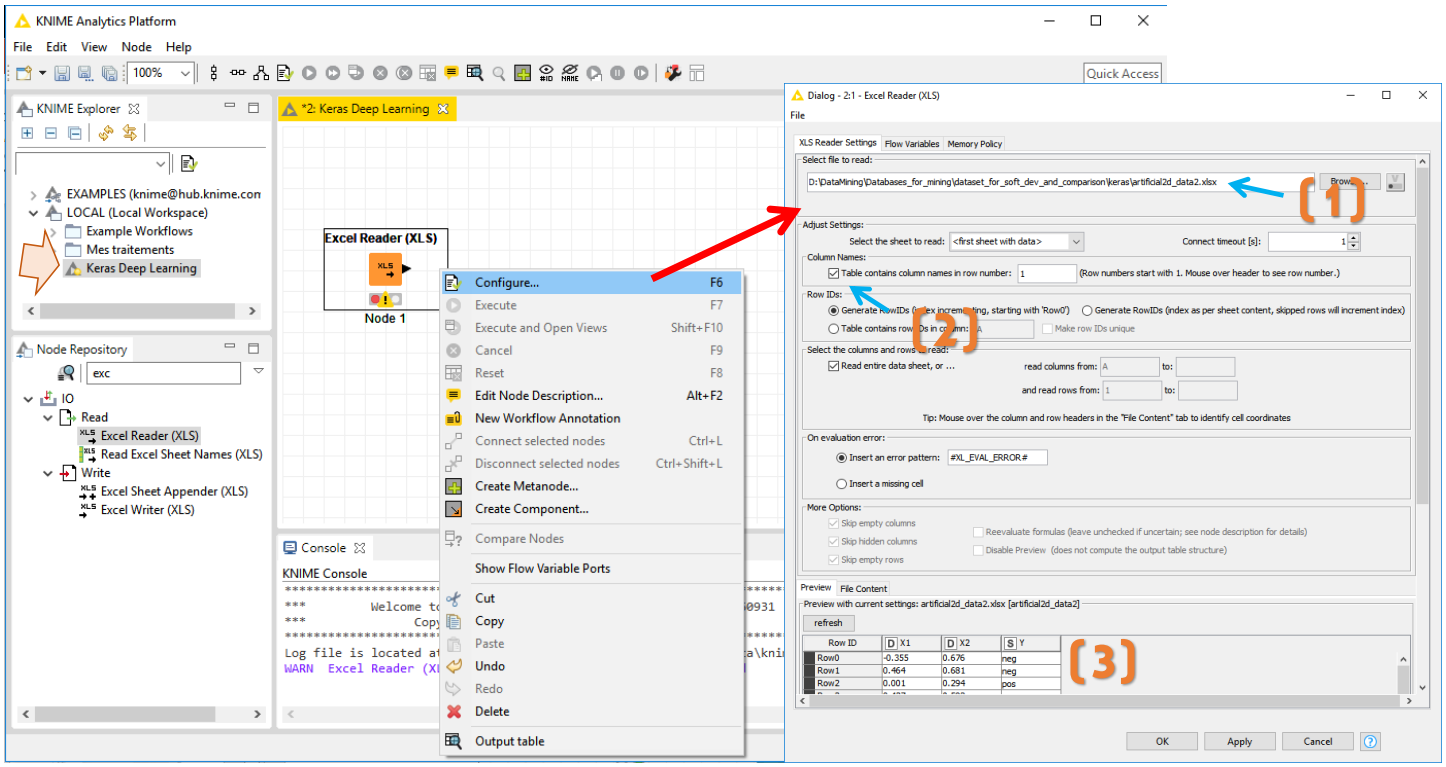
X1	X2	Y
-0.355	0.676	neg
0.464	0.681	neg
0.001	0.294	pos
0.427	0.592	neg
-0.391	0.823	neg
-0.425	0.305	neg
0.467	0.948	neg
0.295	0.266	neg
-0.392	0.336	neg

4 Deep Learning avec Keras sous KNIME

Nous allons directement à l'essentiel dans ce tutoriel. Pour la prise en main basique de KNIME (créer un projet, accéder aux composants dans le « Node Repository », les insérer dans le workflow, établir des liens entre les composants, etc.), je conseille la lecture d'un précédent document (Analyse prédictive sous KNIME, février 2016).

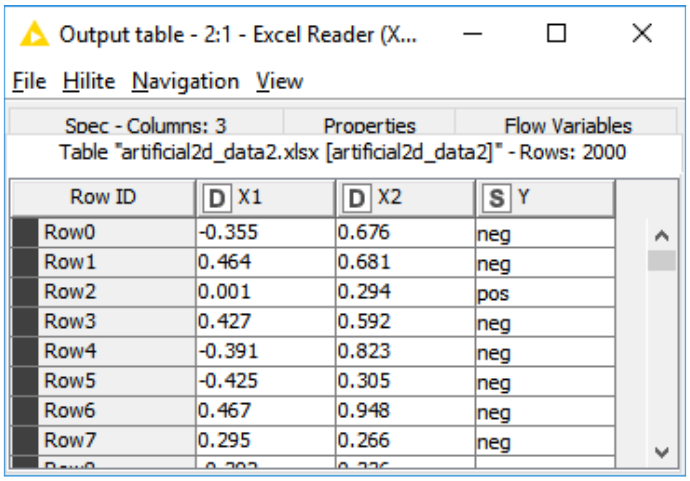
4.1 Importation et visualisation des données

Après avoir créé un workflow que nous nommons « **Keras Deep Learning** », nous insérons un composant EXCEL READER (XLS), nous le configurons (menu CONFIGURE) pour importer le fichier « **artificial2d_data2.xlsx** ».



Nous veillons à (1) indiquer le nom du fichier, (2) spécifier que la première ligne correspond aux noms des variables, (3) vérifier la bonne configuration de l'importation avec la prévisualisation.

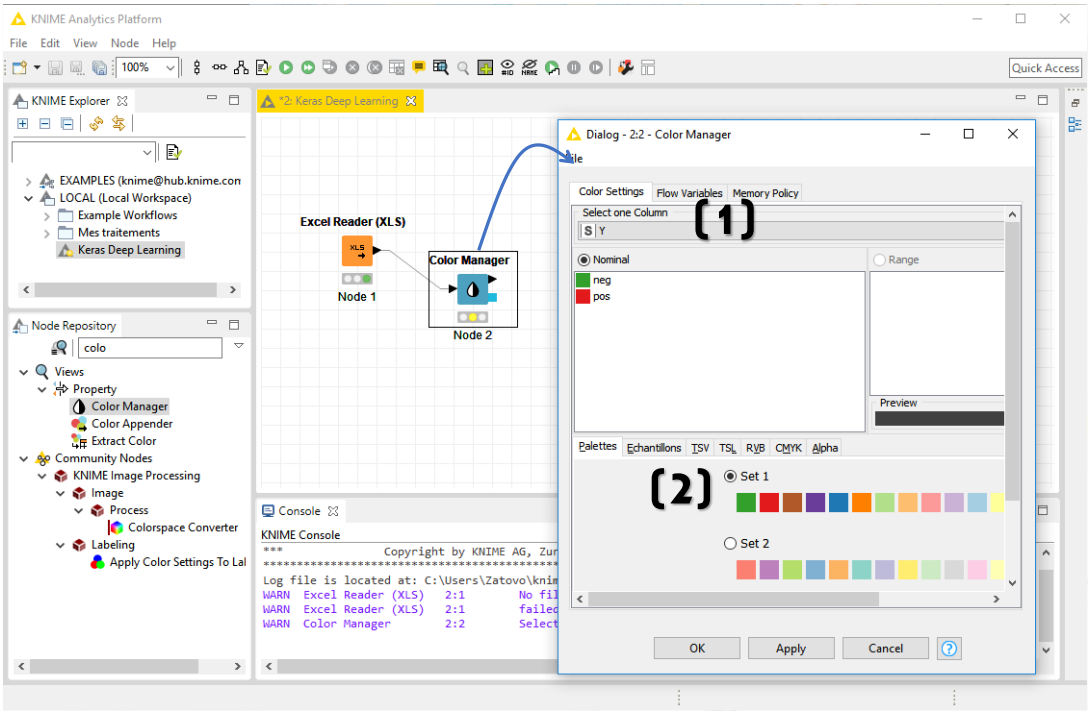
Pour examiner le dataset importé, nous lançons l'importation avec le menu contextuel EXECUTE, puis nous activons le menu contextuel OUTPUT TABLE.



Row ID	D X1	D X2	S Y
Row0	-0.355	0.676	neg
Row1	0.464	0.681	neg
Row2	0.001	0.294	pos
Row3	0.427	0.592	neg
Row4	-0.391	0.823	neg
Row5	-0.425	0.305	neg
Row6	0.467	0.948	neg
Row7	0.295	0.266	neg

Nous disposons de 2000 observations et 3 variables (X1, X2 et Y).

Puisque nous n'avons que deux variables explicatives potentielles, nous pouvons projeter les observations dans le plan en caractérisant les classes avec des couleurs différentes. Nous introduisons l'outil COLOR MANAGER dans le workflow, nous le connectons à la source de données puis nous le paramétrons (menu CONFIGURE).



Les couleurs sont définies par la variable Y (**1**). Nous optons pour la palette « Set 1 » (**2**).

Nous insérons l'outil SCATTER PLOT ensuite, les variables X1 et X2 sont automatiquement sélectionnées en abscisse et ordonnée, mais nous pouvons les paramétrer si besoin. Nous visualisons avec le menu EXECUTE AND OPEN VIEWS. Attention, le graphique met un peu de temps à s'afficher lors de la première exécution.

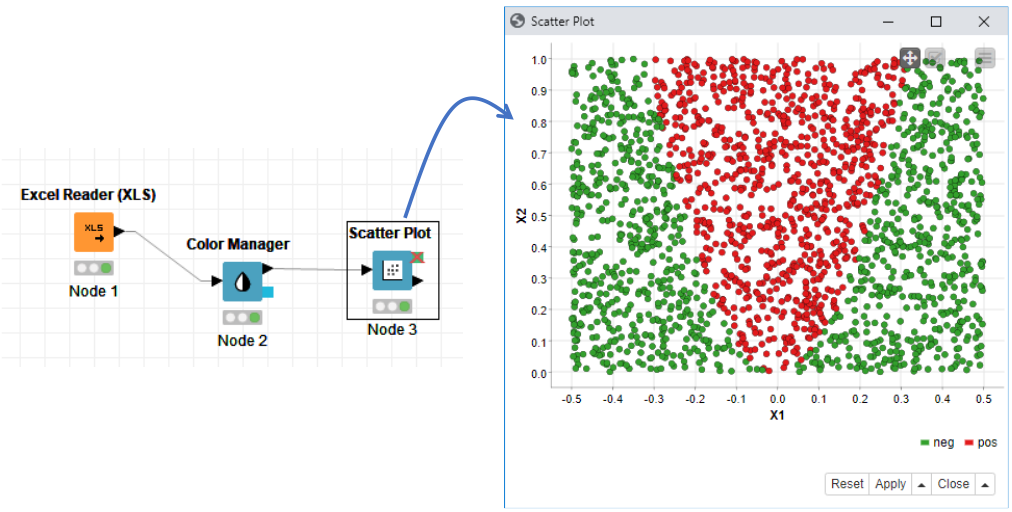
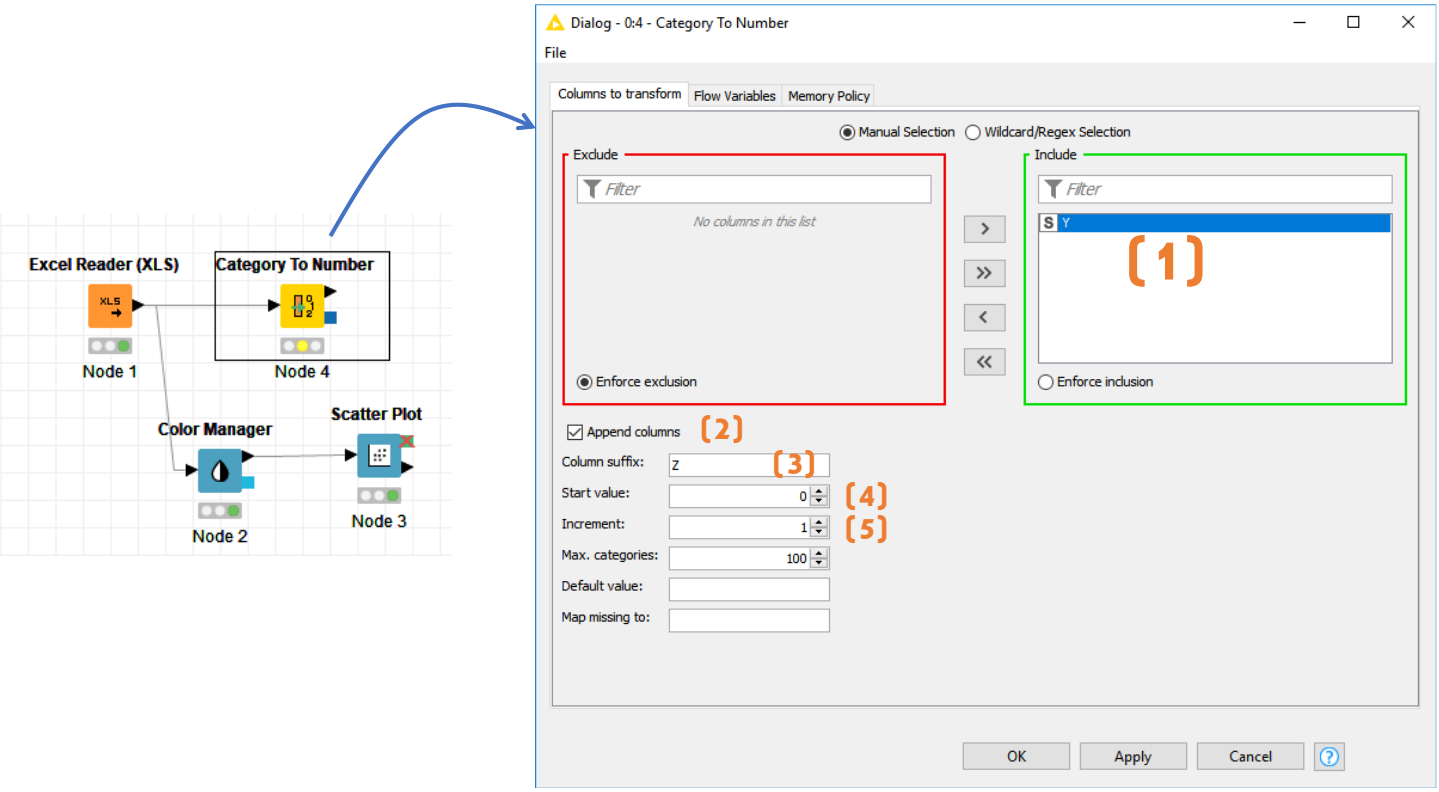


Figure 5 - Position des classes dans le plan

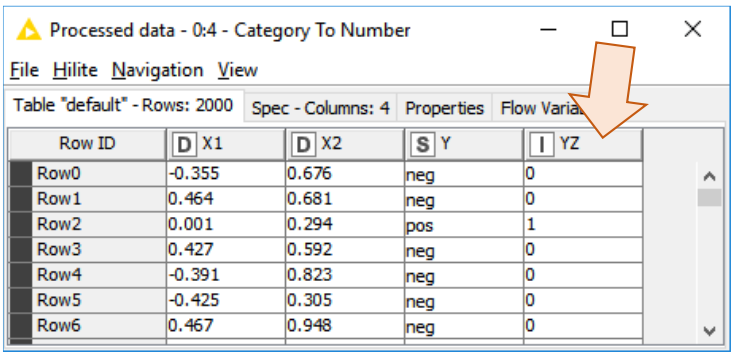
La frontière entre les classes présente la forme d'une parabole. Il est impossible de l'approcher avec une droite (perceptron simple), la tâche sera plus facile avec une combinaison de droites (perceptron multicouche).

4.2 Préparation des données

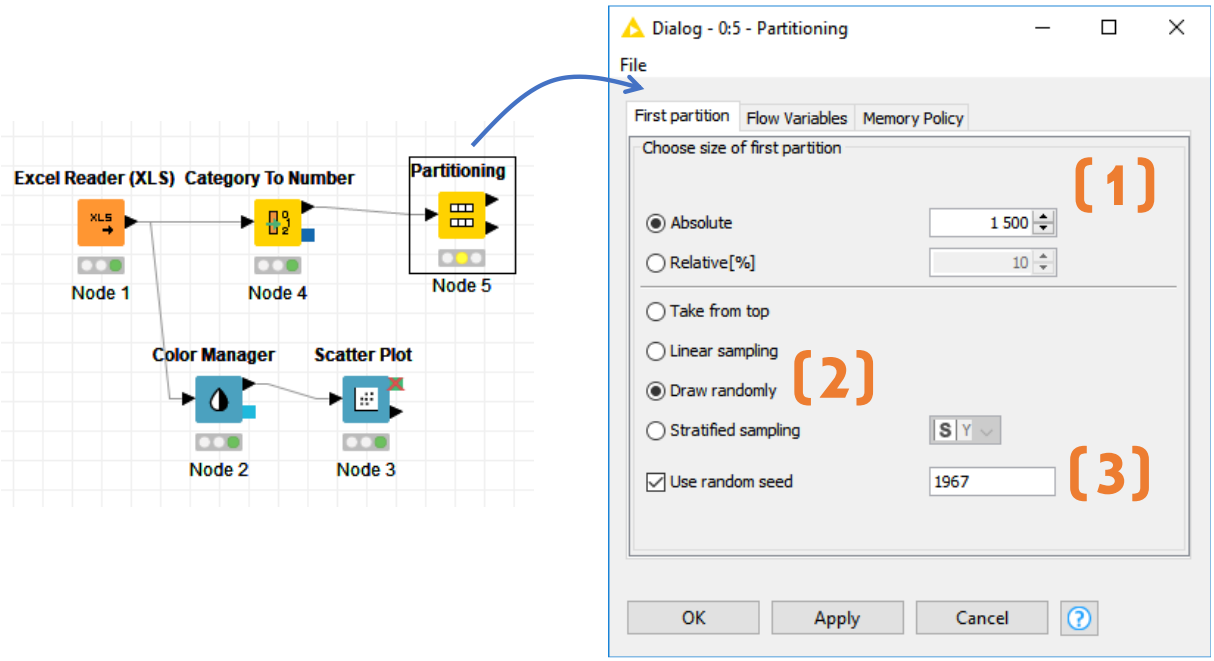
Recodage de la variable cible. Nous devons convertir la variable cible en binaire {0,1} pour être exploitable par les composants Keras. Nous utilisons l'outil CATEGORY TO NUMBER :



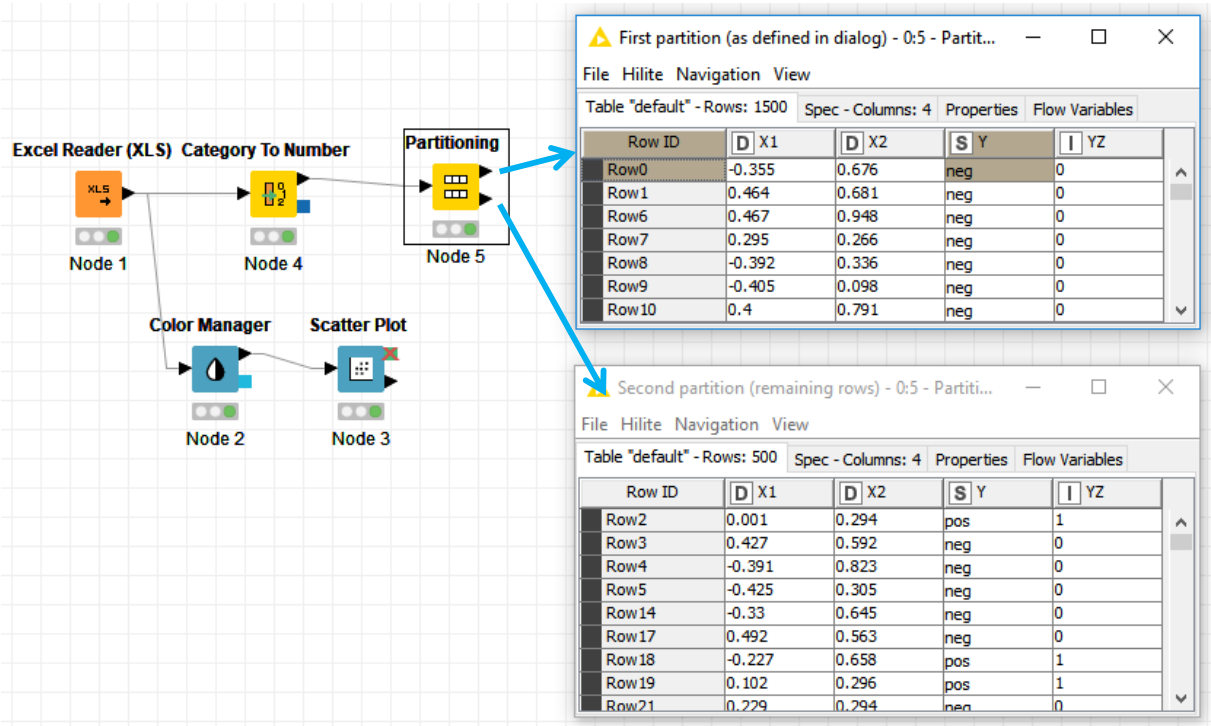
La variable à recoder est Y (1). Une nouvelle colonne est rajoutée au dataset (2) avec le suffixe Z (3) c.-à-d. pour Y, la variable YZ sera créée. La première modalité (« neg » par ordre alphabétique) prendra la valeur 0 (4), puis les valeurs seront incrémentées d'une unité pour les suivantes (5) c.-à-d. « pos » sera codée 1. Nous obtenons ainsi (menu PROCESSED DATA) :



Subdivision apprentissage-test. Comme de coutume en analyse prédictive, nous scindons les données en échantillons d'apprentissage, pour la modélisation, et de test, pour l'évaluation. Nous utilisons le composant PARTITIONING.



Nous réservons 1500 observations pour l'apprentissage (1), aléatoirement (2), la valeur d'initialisation du générateur de nombres aléatoires est 1967 (3) afin que vous puissiez reproduire à l'identique l'expérimentation.



Le premier échantillon est composé des observations n°0, 1, 6, 7, etc. ; le second des individus n°2, 3, 4, 5, etc.

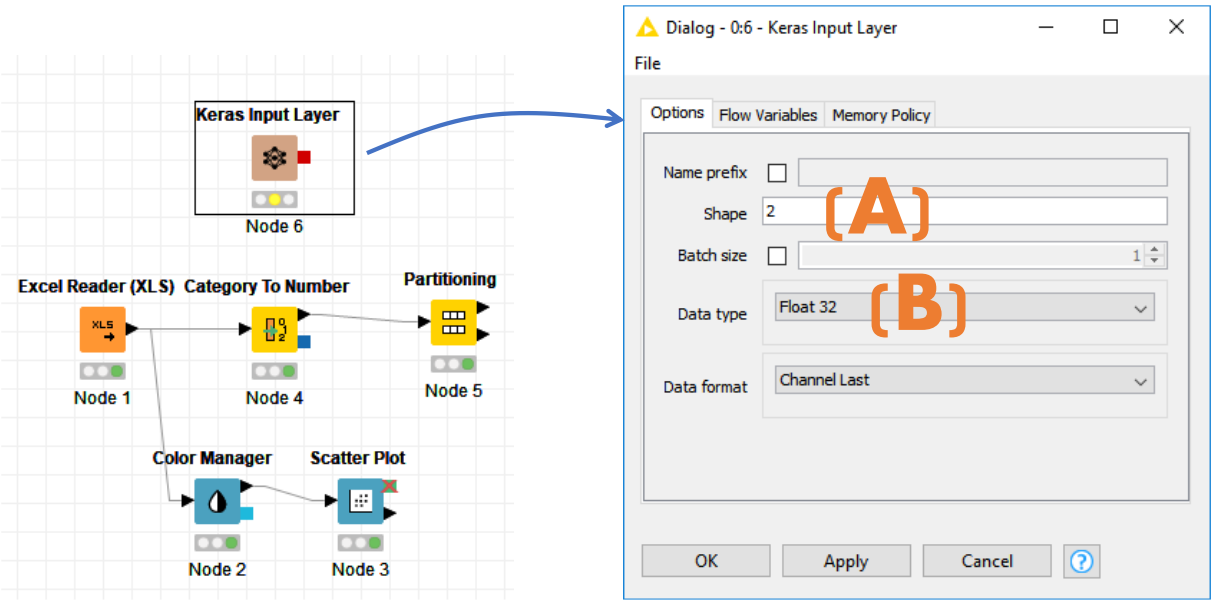
4.3 Perceptron simple

Nous sommes parés pour lancer la modélisation. Nous faisons le choix d'un perceptron simple dans un premier temps. Nous savons d'office qu'une droite ne permet pas de reproduire une frontière curvilinéaire. Il s'agit avant tout d'aller au plus simple pour comprendre le fonctionnement des composants Keras sous KNIME.

4.3.1 Définition du réseau

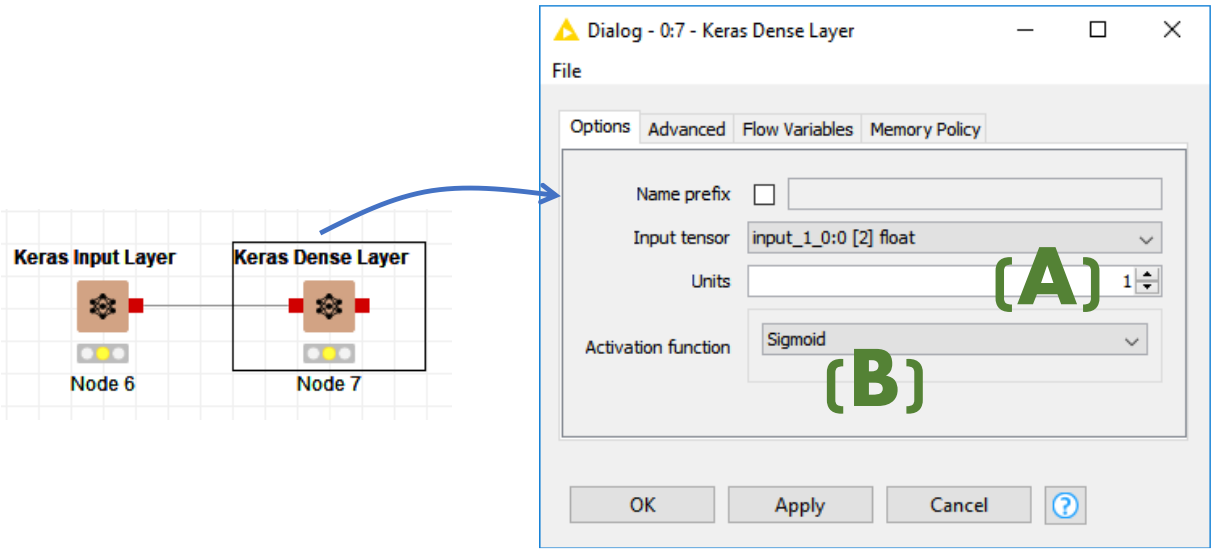
Notre réseau est un perceptron simple. Il est composé de deux couches : une d'entrée branchée sur les variables explicatives, une de sortie pour la variable cible YZ.

Nous plaçons tout d'abord le composant KERAS INPUT LAYER.



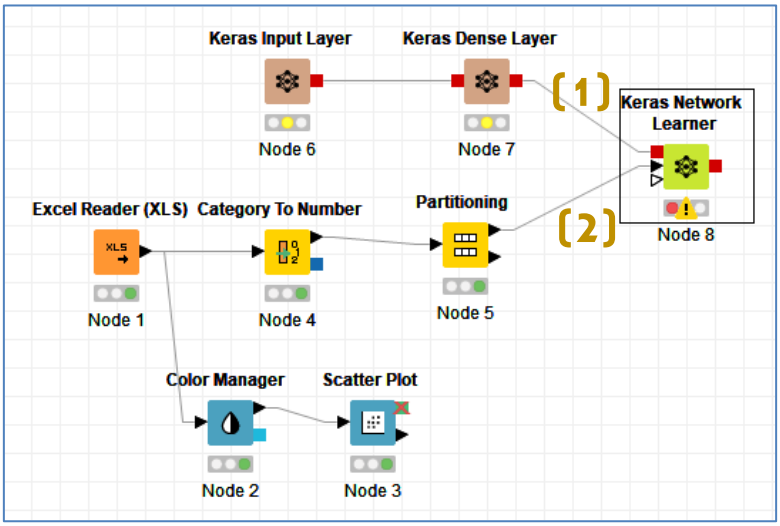
La couche (layer) est composée de deux neurones parce que nous avons 2 variables explicatives (X1, X2) **(A)**, il s'agit de variables numériques **(B)**.

Nous plaçons ensuite la couche suivante KERAS DENSE LAYER qui constitue la couche de sortie pour nous. C'est une couche dense c.-à-d. à chaque neurone de cette couche est connecté à tous les neurones de la couche précédente. Elle comporte un seul neurone **(A)** pour la variable YZ, nous optons pour la fonction d'activation sigmoïde **(B)**.



4.3.2 Processus d'apprentissage

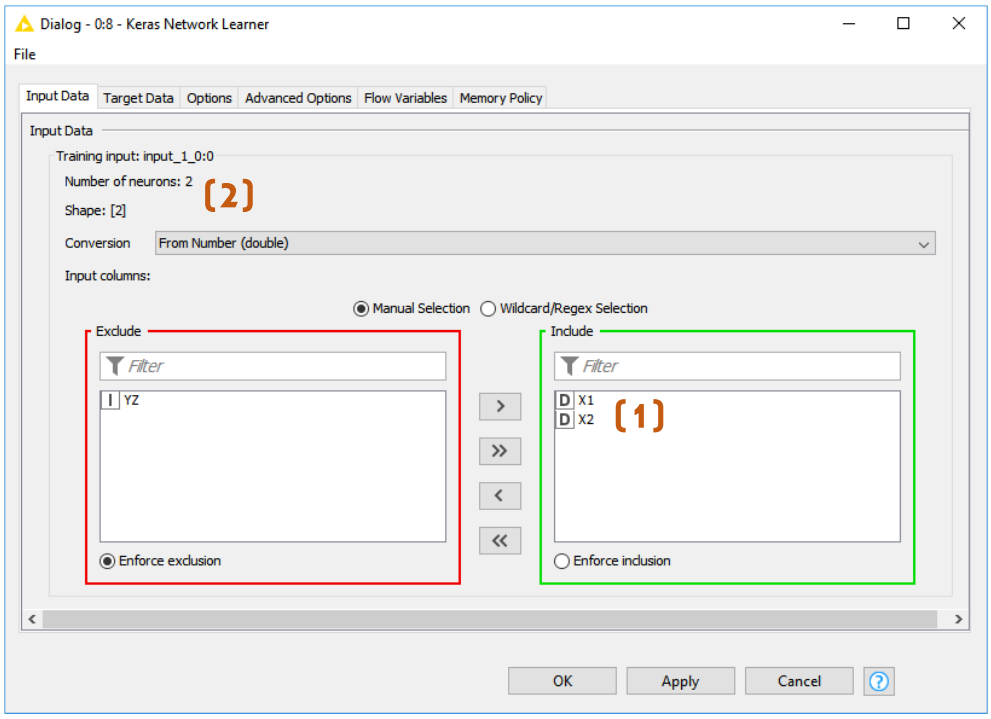
La modélisation – le calcul des poids synaptiques à partir des données – est réalisée avec le composant KERAS NETWORK LEARNER. Il prend en entrée (1) la structure du réseau et (2) l'échantillon d'apprentissage.



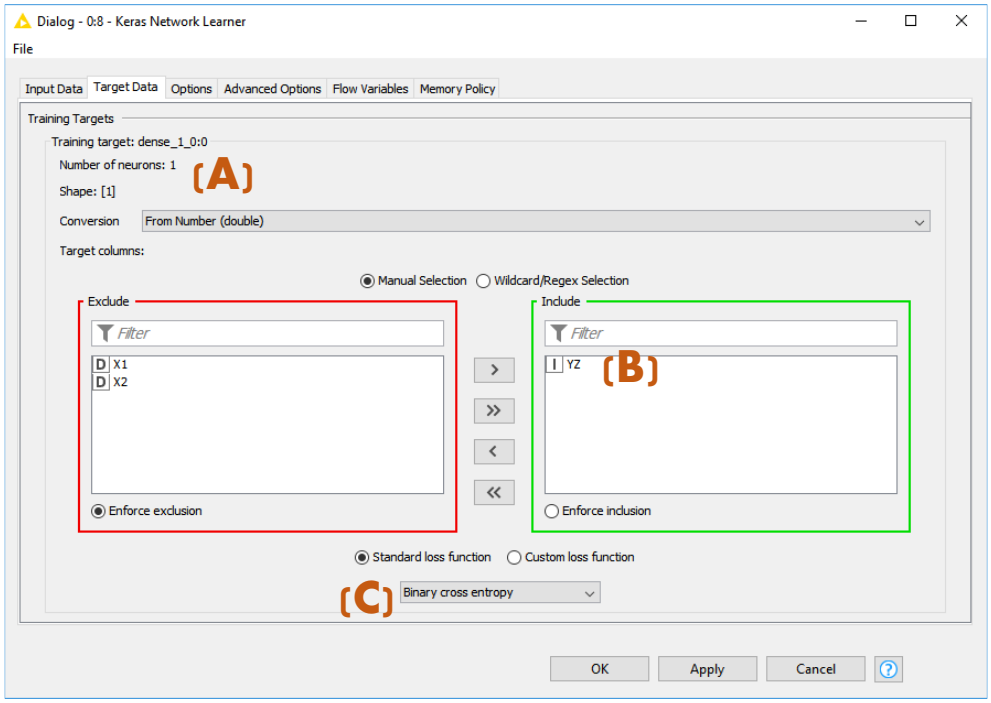
Remarque : Nous pouvons lui brancher éventuellement une troisième entrée (troisième slot non utilisé ici) représentant l'échantillon de validation pour le suivi du processus de modélisation. En effet, le calcul des indicateurs de performances sur l'échantillon (d'apprentissage) servant à l'estimation des poids synaptiques du réseau peut être entaché par le surapprentissage c.-à-d. fournir des indications trop optimistes, surtout lorsque le réseau est surdimensionné par rapport à la taille de la base utilisée. En utilisant des données à part, distinctes, nous nous prémunissons de cet éventuel excès d'euphorie. A contrario, cela veut dire aussi qu'une partie des données ne sont pas exploitées pour la construction du modèle prédictif. Ce qui peut être pénalisant.

Nous nous intéressons à trois onglets dans la boîte de dialogue de paramétrage.

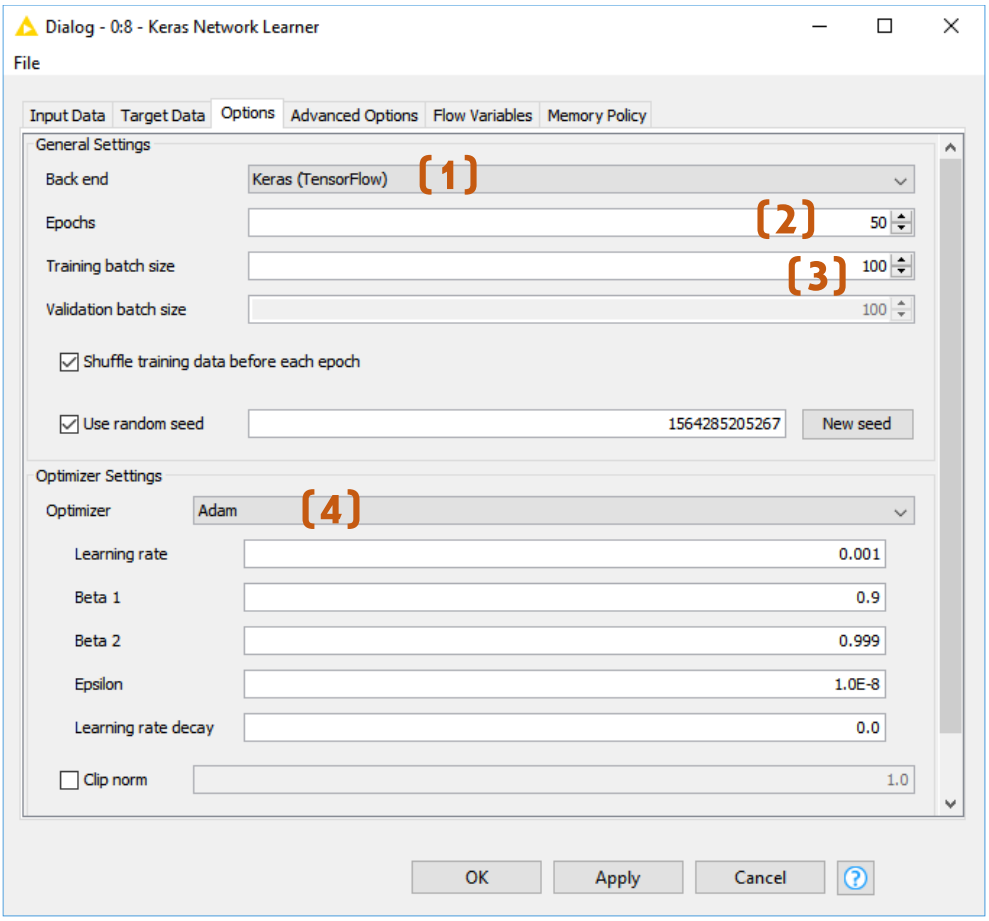
Input Data. Il permet de désigner les variables explicatives en entrée du réseau. Nous sélectionnons X1 et X2 (1), en correspondance avec le nombre de neurones (2).



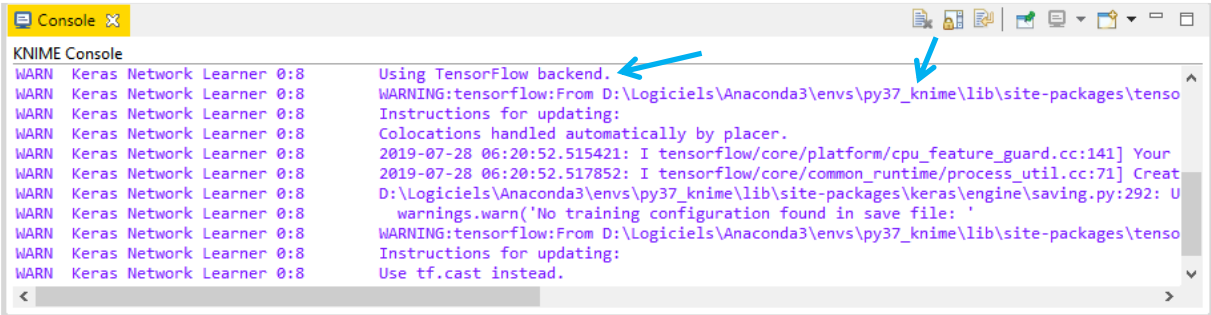
Target Data. Nous avons un seul neurone en sortie (A), il correspond à la variable YZ (B). Nous optons pour la fonction de perte « Binary Cross Entropy » (C) (cf. R. Rakotomalala, « Deep Learning : perceptrons simples et multicouches », novembre 2018, page 30).



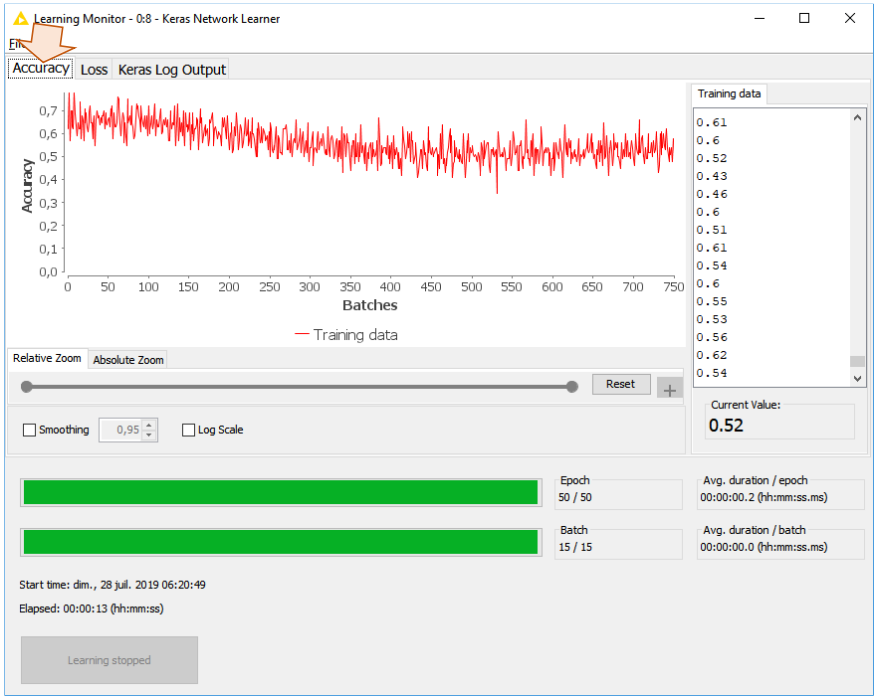
Options. Dans cet onglet, nous spécifions les caractéristiques de l'algorithme d'optimisation. Le moteur utilisé est bien le duo Keras / Tensorflow (1), le nombre d'epochs est de 50 c.-à-d. 50 passages sur la base entière (2), la taille du batch est de 100 c.-à-d. les poids sont mis à jour à chaque passage de 100 observations (3), l'algorithme d'optimisation ADAM (4) est mis à contribution. Nous avions opté pour des paramètres similaires lorsque nous avions étudié la librairie Keras sous Python sur les mêmes données (avril 2018).



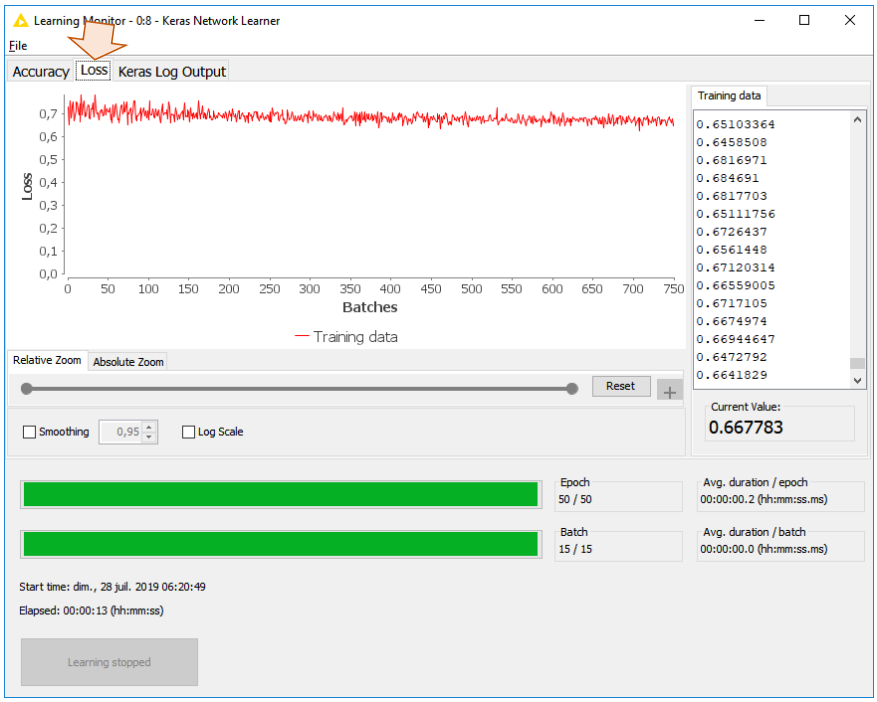
Nous lançons la modélisation en actionnant le menu EXECUTE AND OPEN VIEWS. La console KNIME affiche des informations qu'il faut inspecter pour s'assurer la bonne marche du processus d'apprentissage. Le lien avec l'interpréteur Python est souvent source d'erreur.



La fenêtre LEARNING MONITOR est automatiquement affichée. Nous disposons de l'évolution du taux de reconnaissance (Accuracy = 1 – taux d'erreur) sur l'échantillon d'entrainement (Training Data).



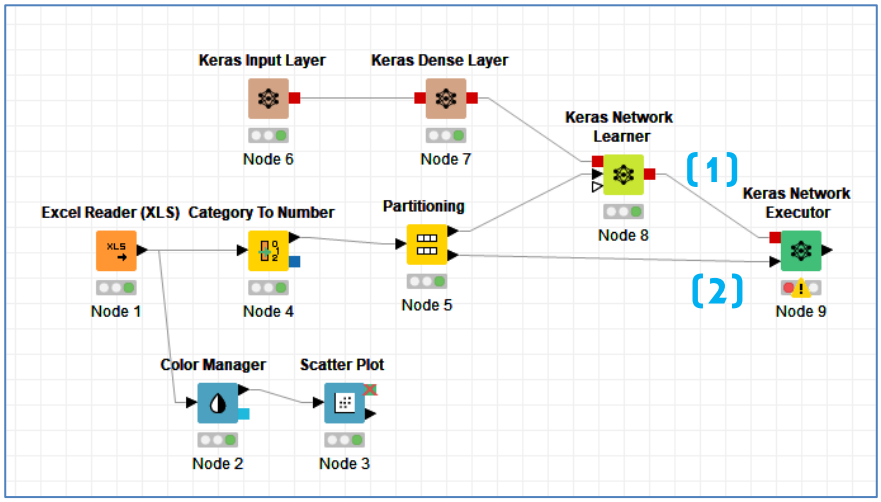
Et de l'évolution de la fonction de perte (Loss).



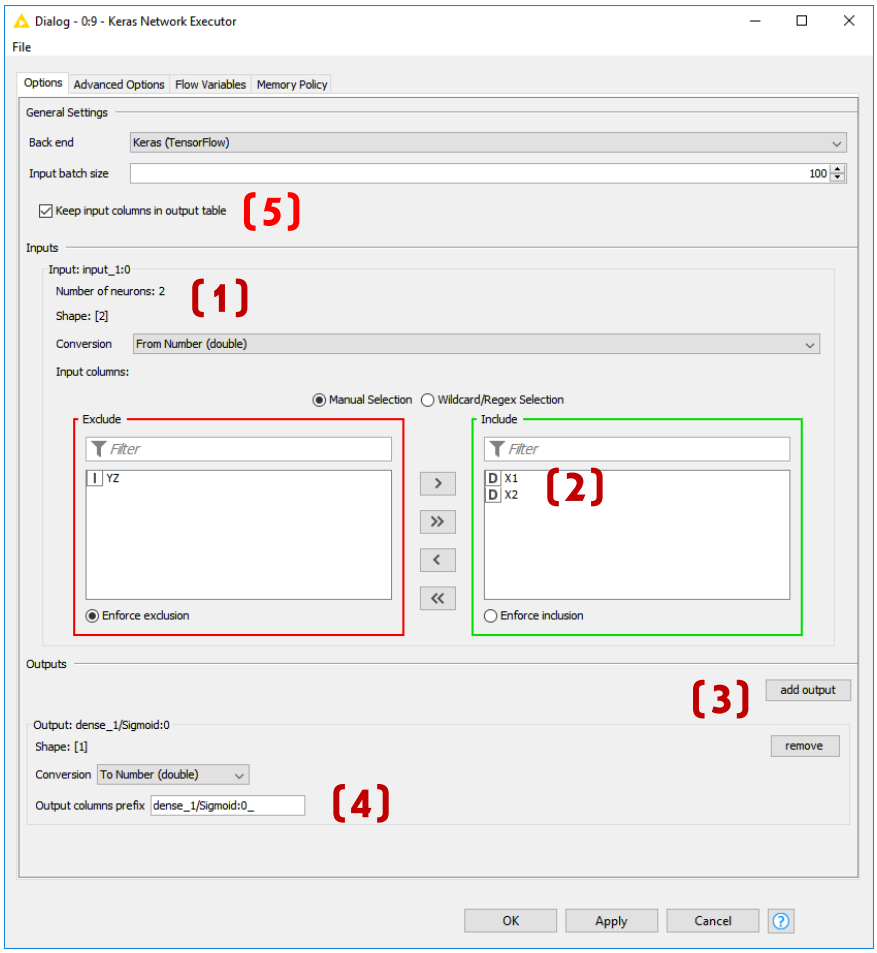
Un perceptron simple n'est manifestement pas adapté à notre problème. Le processus ne converge pas malgré les 50 itérations (epoch) sur la base d'entraînement.

4.3.3 Prédiction en test

Essayons quand-même d'évaluer la qualité du modèle sur l'échantillon test de 500 observations (section 4.2). Pour ce faire, nous effectuons la prédiction avec le composant KERAS NETWORK EXECUTOR. Nous lui branchons le modèle entrainé (1) et l'échantillon test (2).

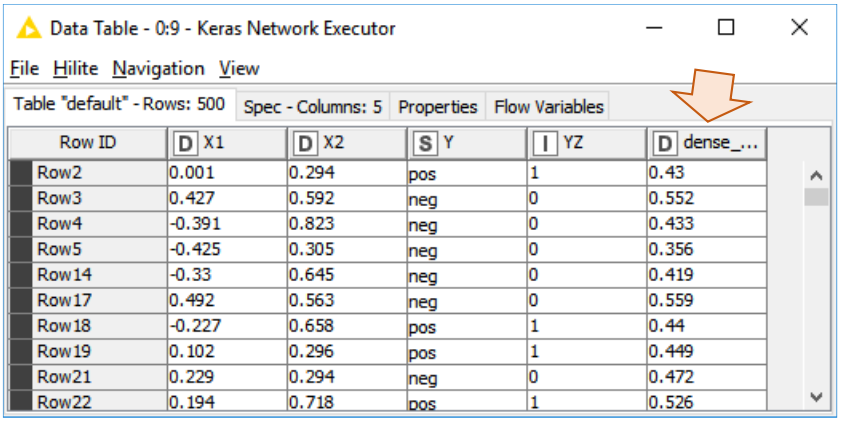


Voyons son paramétrage.



Aux deux neurones de la couche d'entrée **(1)** correspondent les variables (X1, X2) **(2)**. Nous devons rajouter manuellement la prédiction du réseau « add output » **(3)** et indiquer la sortie sigmoïde de la couche dense **(4)**. Enfin, il ne faut pas oublier de conserver les variables originelles dans le flux **(5)**. En effet, nous aurons à confronter l'étiquette observée Y avec la prédiction du réseau par la suite.

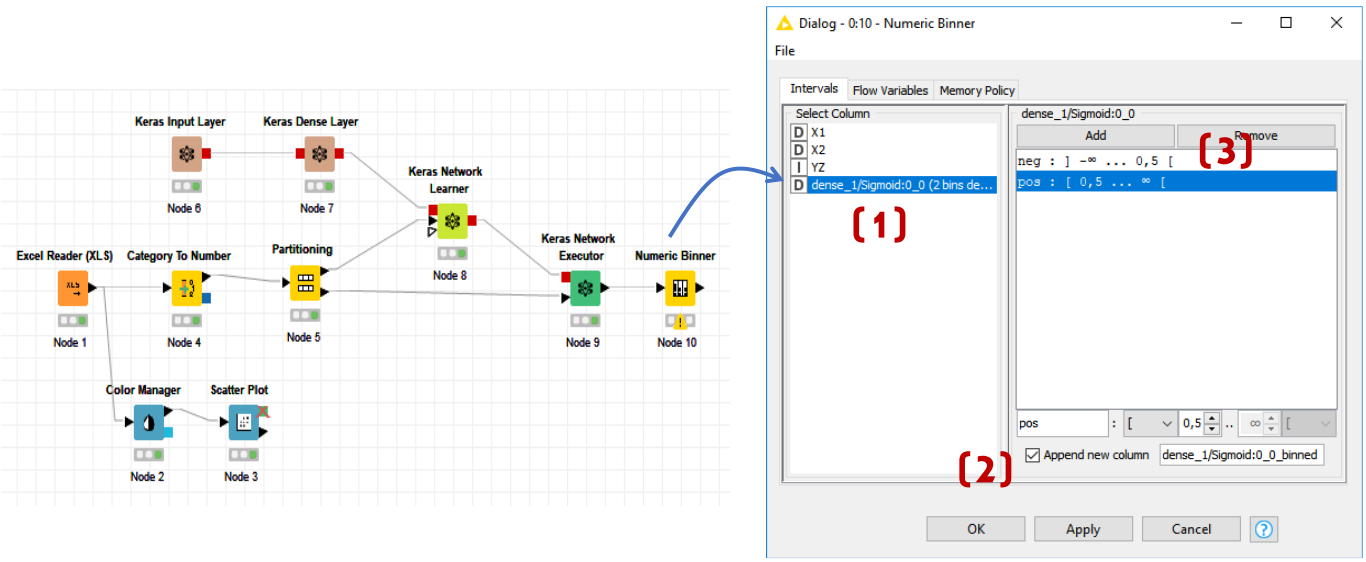
Après exécution, nous pouvons observer la table de données (menu contextuel DATA TABLE).



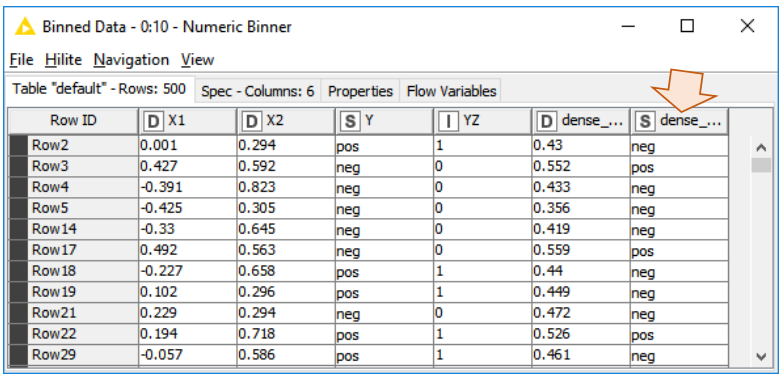
Row ID	D X1	D X2	S Y	I YZ	D dense_...
Row2	0.001	0.294	pos	1	0.43
Row3	0.427	0.592	neg	0	0.552
Row4	-0.391	0.823	neg	0	0.433
Row5	-0.425	0.305	neg	0	0.356
Row14	-0.33	0.645	neg	0	0.419
Row17	0.492	0.563	neg	0	0.559
Row18	-0.227	0.658	pos	1	0.44
Row19	0.102	0.296	pos	1	0.449
Row21	0.229	0.294	neg	0	0.472
Row22	0.194	0.718	pos	1	0.526

Nous avons bien l'échantillon test composé des individus n°2, 3, 4, etc.

La colonne « dense_... » correspond à la probabilité d'appartenance à la modalité « pos » fournie par le réseau pour ces observations. Elle est comprise entre 0 et 1, nous devons la confronter au seuil 0.5 pour produire la prédiction neg/pos. Nous introduisons le composant NUMERIC BINER dans le workflow. Nous paramétrons la conversion de la probabilité en classe prédite : à partir de la sortie du réseau « dense_1/Sigmoid :0_0 » **(1)**, nous créons une variable « dense_1/Sigmoid :0_0_binned » que nous ajoutons au dataset **(2)**, « neg » correspond aux probabilités strictement inférieures à 0.5, « pos » à celle supérieures à 0.5 **(3)**.



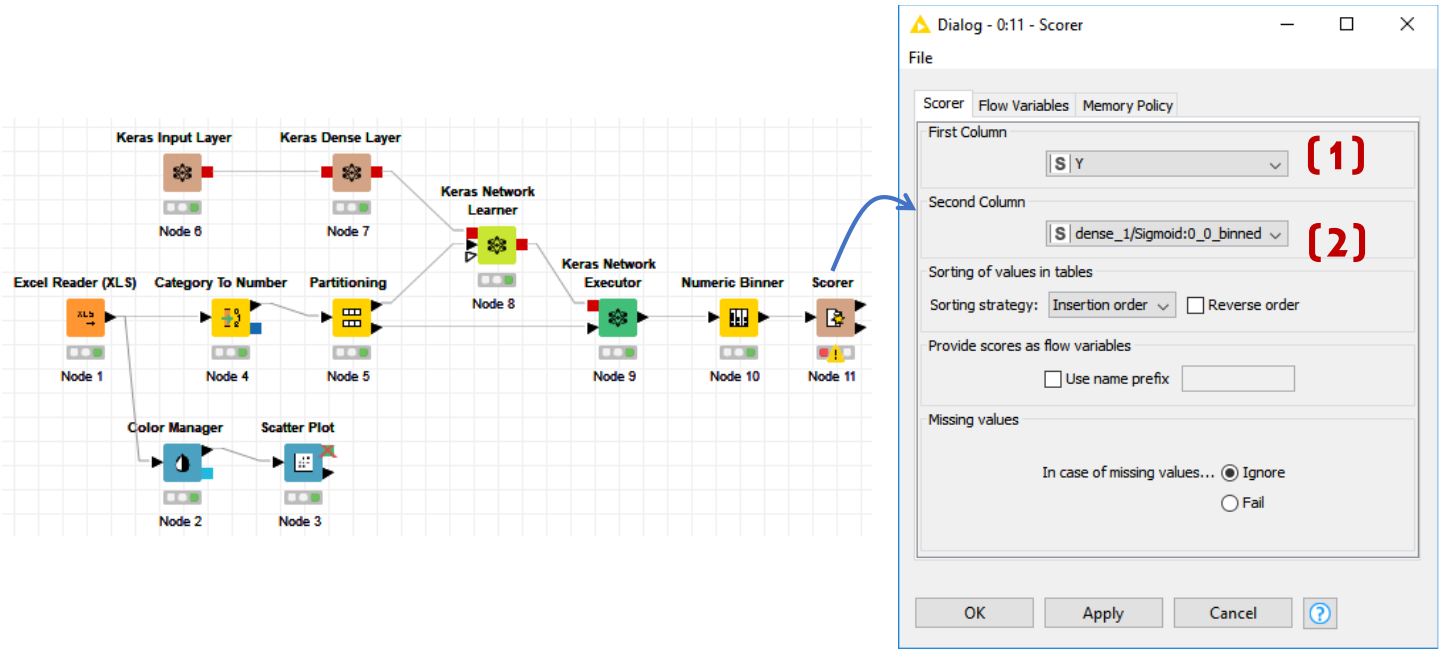
Après exécution, en cliquant sur le menu BINNED DATA, nous pouvons visualiser la nouvelle version du dataset avec la colonne de prédiction.



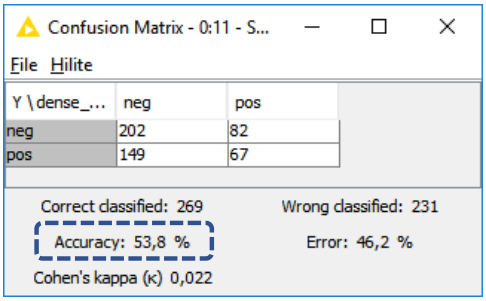
Row ID	D X1	D X2	S Y	I YZ	D dense_...	S dense_...
Row2	0.001	0.294	pos	1	0.43	neg
Row3	0.427	0.592	neg	0	0.552	pos
Row4	-0.391	0.823	neg	0	0.433	neg
Row5	-0.425	0.305	neg	0	0.356	neg
Row14	-0.33	0.645	neg	0	0.419	neg
Row17	0.492	0.563	neg	0	0.559	pos
Row18	-0.227	0.658	pos	1	0.44	neg
Row19	0.102	0.296	pos	1	0.449	neg
Row21	0.229	0.294	neg	0	0.472	neg
Row22	0.194	0.718	pos	1	0.526	pos
Row29	-0.057	0.586	pos	1	0.461	neg

4.3.4 Mesure des performances

Nous utilisons le composant SCORER pour construire la matrice de confusion.



Nous opposons l'étiquette observée Y **(1)** avec la prédiction **(2)** issue du recodage de la probabilité d'affectation. Nous obtenons la matrice de confusion suivante (menu EXECUTE AND OPEN VIEWS).



Y \ dense_...	neg	pos
neg	202	82
pos	149	67

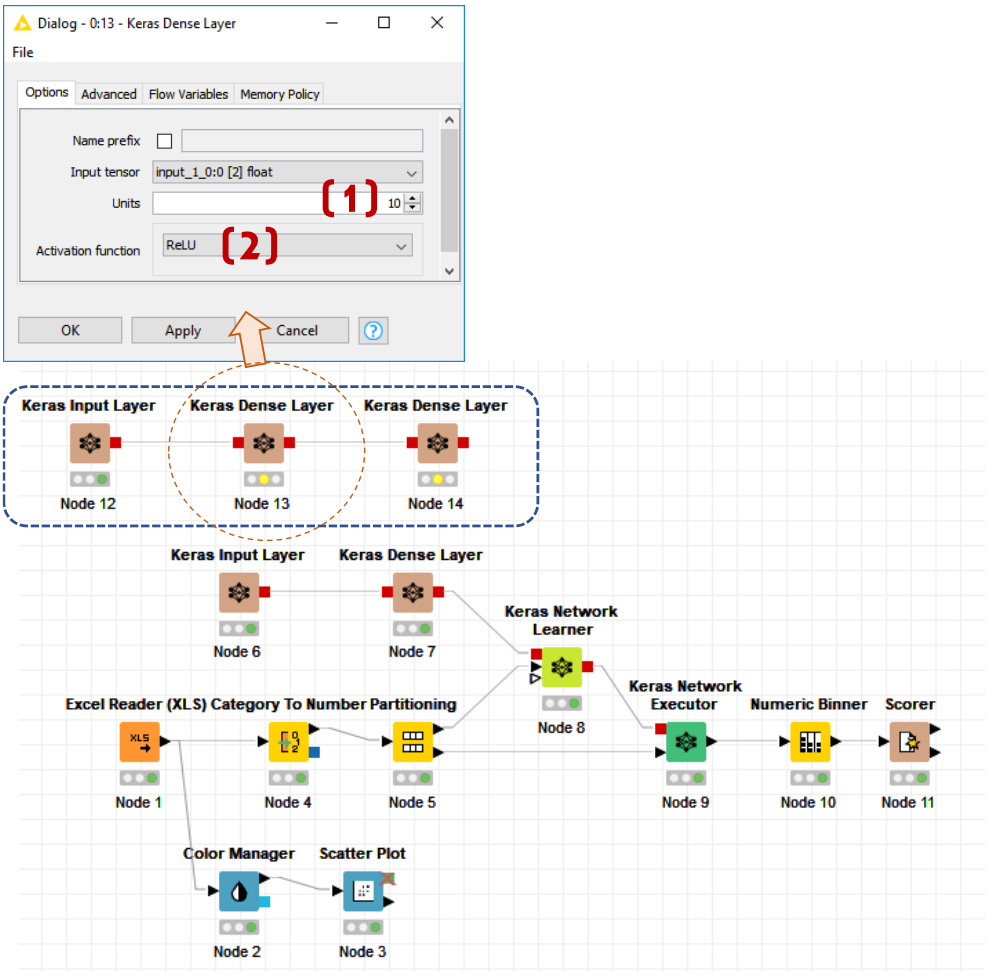
Le taux de succès du perceptron simple est de 53.8%. Pas terrible vraiment pour un problème à deux classes. Mais on s'y attendait au regard de la conformation des classes dans l'espace de représentation (Figure 5).

4.4 Perceptron multicouche

Nous nous essayons au perceptron multicouche dans cette section. Maintenant que nous savons manipuler les outils sous KNIME, je mettrai l'accent avant tout sur les différences par rapport au perceptron simple.

4.4.1 Configuration du réseau

Nous construisons toujours le réseau avec les mêmes outils KERAS INPUT LAYER et KERAS DENSE LAYER, sauf que nous dupliquons cette dernière pour matérialiser la couche cachée et la couche de sortie. Le réseau se présente comme suit. Les paramétrages pour les couches d'entrées et de sorties ne sont pas modifiée, il est spécifique pour la couche cachée.

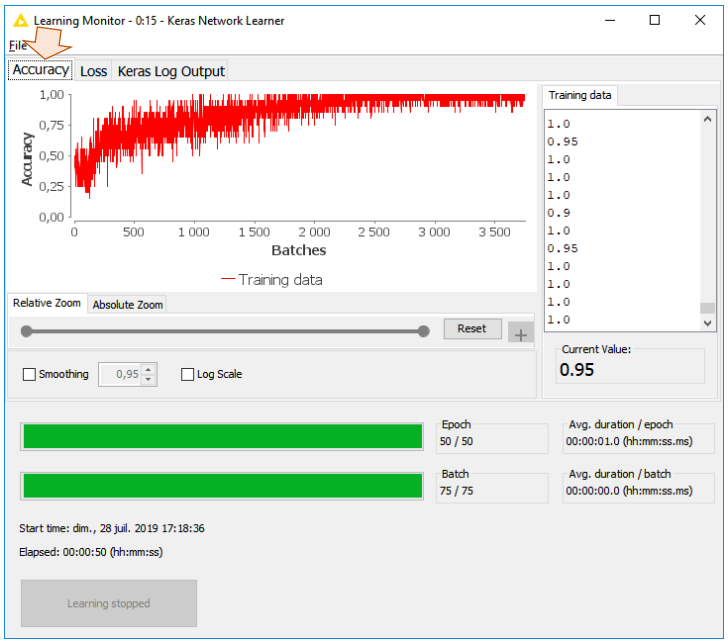


Nous affectons 10 neurones (1) à la couche intermédiaire, avec la fonction d'activation ReLU (2).

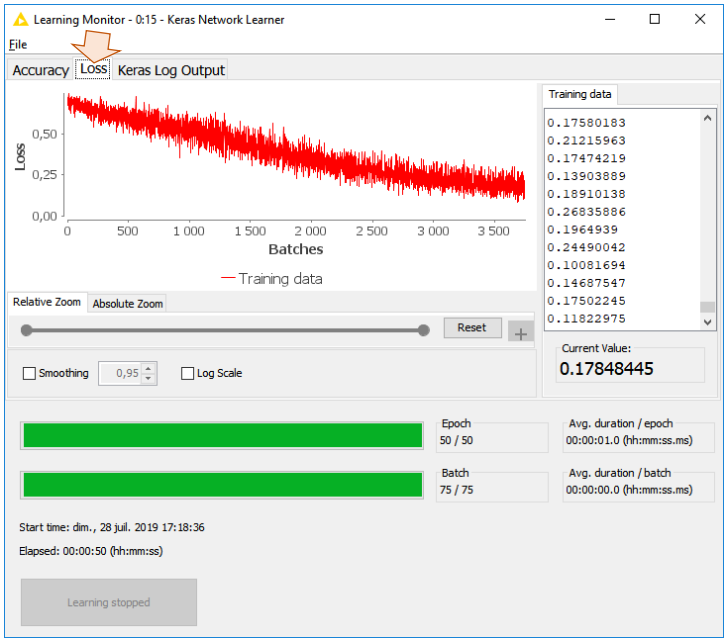
4.4.2 Apprentissage

Nous lançons la modélisation avec le composant KERAS NETWORK LEARNER dûment paramétré (voir section 4.3.2). Pour assurer une meilleure exploitation des données, j'ai réduit la taille du batch (Training Batch Size) à 20. Les mises à jour sont plus fréquentes durant l'apprentissage. Cela devrait améliorer la convergence.

Les fenêtres de visualisation « Accuracy » …



… et « Loss »…

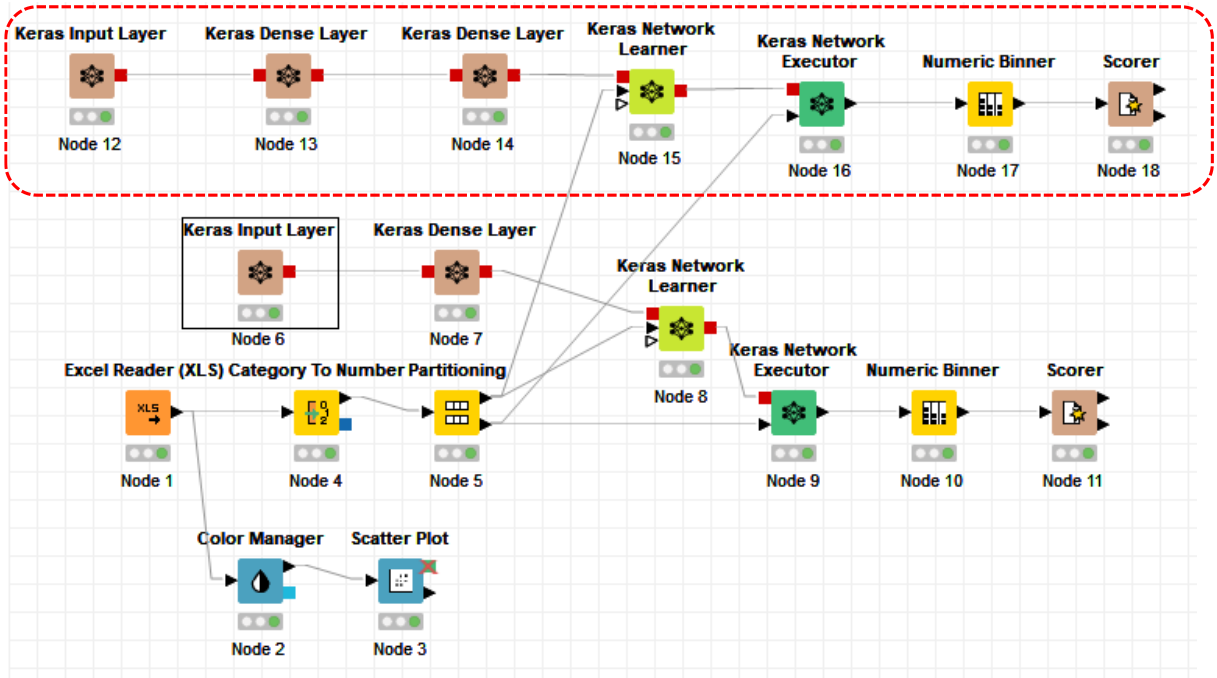


…nous autorisent à être un peu plus optimiste par rapport au perceptron simple.

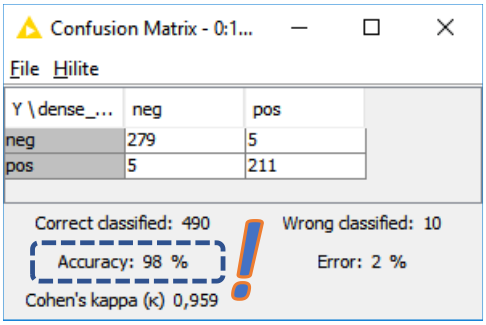
4.4.3 Prédiction et évaluation

Voyons ce qu'il en est avec l'échantillon test. Attention, c'est bien la sortie de la seconde couche dense « **dense_2/Sigmoid :0_** » que nous devons utiliser pour la prédiction (OUTPUT) en ce qui concerne le composant KERAS NETWORK EXECUTOR.

Voici le diagramme (encadré en rouge) pour cette seconde version du perceptron.



Sans surprise, le modèle s'avère nettement plus performant avec un **taux de reconnaissance de 98%** sur les 500 observations de l'échantillon test.



5 Conclusion

Ce tutoriel est avant tout un ballon d'essai pour vérifier l'opérationnalité des composants KERAS dans KNIME. Nous aurions pu utiliser les composants natifs de KNIME pour élaborer un perceptron (RPROP MLP LEARNER et MULTILAYERPERCEPTRON PREDICTOR). L'intérêt pour nous est d'ouvrir la porte à la pratique du deep learning sur un logiciel qui ne nécessite pas de

produire du code informatique, toujours plus ou moins abscons pour les réfractaires à la programmation. Maintenant que nous sommes en si bon chemin, il nous sera facile d'aller plus loin en explorant les autres applications du deep learning, la classification automatique d'images avec des réseaux de convolutions par exemple. Je vois que le fameux benchmark du classement des chiens et des chats fait partie des workflow d'illustration fournis avec le logiciel. Il va falloir étudier cela de près dans un futur proche.

Un mot enfin sur **KNIME** pour conclure. Je suis toujours épaté quand j'étudie de près ses fonctionnalités. J'ai fait le pari de R et Python pour mes étudiants, parce qu'ils peuvent combiner ainsi des compétences en programmation et en utilisation des librairies performantes de data science. Il reste que, pour avoir moi-même longtemps trimé dans l'élaboration de logiciels dans le domaine, je mesure parfaitement la quantité de travail monumentale nécessaire pour la réalisation d'un outil de cette qualité. Si je dois conseiller un **outil libre** qui permet de réaliser des projets performants de data science sans avoir à passer par la case programmation informatique, je le mettrai en avant sans aucune hésitation. Le seul bémol toujours concernant KNIME est la documentation. Elle est quasi-inexistante en français. En anglais, on trouve certes quelques références avec des requêtes ciblées sur Google, mais il faut pouvoir l'adapter à son problème, et ce n'est pas toujours évident. J'ai vraiment dû batailler ferme pour faire fonctionner le dispositif décrit dans ce tutoriel. Plus d'une fois, il m'a fallu fermer KNIME, le relancer proprement, voire redémarrer ma machine, pour rétablir la liaison avec Python. Mais après coup, je me dis que cet investissement en valait la peine.

6 Références

KNIME Deep Learning – Keras Integration -- https://www.knime.com/deeplearning/keras

R. Rakotomalala, « Deep Learning : perceptrons simples et multicouches », novembre 2018.

Tutoriel Tanagra, « Deep Learning avec Tensorflow et Keras (Python) », avril 2018.

Tutoriel Tanagra, « Deep Learning - Tensorflow et Keras sous R », avril 2018.