Institut Agro - Montpellier SupAgro

Ingenieur Agronome

Data Science pour l'Agronomie et l'Agroalimentaire

Réseaux de neurones pour graphe pour la prédiction de phénotypes

Auteur :
Guilhem HUAU

Encadrants :
Nathalie Vialaneix
Céline Brouard

Tuteur :
Philippe Vismara

Mars - Août 2021

Table des matières

Résumé

La prédiction de phénotypes d'intérêt représente un atout majeur en agronomie et en médecine, et les données d'expression de gènes sont de plus en plus disponibles de nos jours. Or les gènes à l'origine de ces phénotypes, interagissent entre eux, créant une structure entre ces données. Le développement récent des réseaux de neurones pour graphe donne l'opportunité d'utiliser la structure, en sus des données initiales d'expression pour la prédiction de phénotypes. Au cours de ce stage, on a cherché à comparer les performances des réseaux de neurones pour graphe entre eux et avec d'autres méthodes de machine learning, afin de voir leur intérêt dans la prédiction sur deux exemples : la réussite d'un régime hypo-calorique chez des patients atteints du syndrome métabolique, ainsi que la survenue d'événements métastatiques chez des patients atteints d'un cancer du sein. On a pu voir que les performances restent très liées à la tâche que l'on cherche à accomplir et peuvent à l'occasion se comparer à d'autres méthodes plus classiques, mais que la sélection de la bonne architecture du réseau de neurones pour graphe et des bons paramètres d'entrainement, revêt une importance particulière et mériterait d'être approfondie.

Mots-clefs :

Réseaux de neurones, graphe, données d'expression, gènes, prédiction, phénotype, syndrome métabolique

Chapitre 1

Présentation de la structure d'accueil

1.1 INRAE

L'Institut National de Recherche pour l'Agriculture, l'Alimentation et l'Environnement [1] (INRAE) est un organisme français de recherche agronomique. Il fait partie des organismes de recherche leaders dans le monde, en agriculture, alimentation et environnement : en sciences de l'agriculture, l'institut est classé au 3e rang mondial en nombre de citations et au 4e en nombre de publications [2]. Sous la double tutelle du Ministère de l'Enseignement Supérieur et de la Recherche et du Ministère de l'Agriculture et de l'Alimentation, et avec près d'un milliard d'euros de budget, INRAE compte 268 unités de recherche, de services et expérimentales, réparties sur 18 sites régionaux et dans 14 départements de recherche, ainsi que 10 000 ha de terres agricoles dédiées à l'expérimentation.

INRAE s'est donné comme orientation scientifique pour les prochaines années les 5 thématiques suivantes [1] :

- répondre aux enjeux environnementaux et gérer les risques associés
- accélérer les transitions vers des systèmes agricoles et alimentaires agroécologiques en tenant compte des enjeux économiques et sociaux
- une bioéconomie basée sur une utilisation sobre et circulaire des ressources
- favoriser une approche globale de la santé
- mobiliser la science des données et les technologies du numérique au service des transitions

et comme orientations de politique générale, les trois suivantes :

- placer la science, l'innovation et l'expertise au cœur de nos relations avec la société pour renforcer notre culture de l'impact
- être un acteur engagé dans les sites universitaires en France et un leader dans les partenariats européens et internationaux
- la stratégie « Responsabilité Sociale et Environnementale » (RSE) : une priorité collective

1. Site officiel d'INRAE.
2. Données InCites Clarivate Analytics – publications 2015-2019 - InCites dataset updated Mar 7, 2020.

De plus, l'Institut a pour mission de réaliser, d'organiser et de coordonner, à son initiative ou à la demande de l'État, tous travaux de recherche scientifique et technologique dans les domaines de l'agriculture, de l'alimentation, de la forêt, de l'environnement, de l'eau, de la biodiversité, de la bioéconomie, de l'économie circulaire, de la gestion durable des territoires et des risques dans les champs de compétence précités.

L'organigramme d'INRAE est donné dans la figure 1.1.

1.2 L'Unité de Mathématiques et Informatique Appliquées de Toulouse

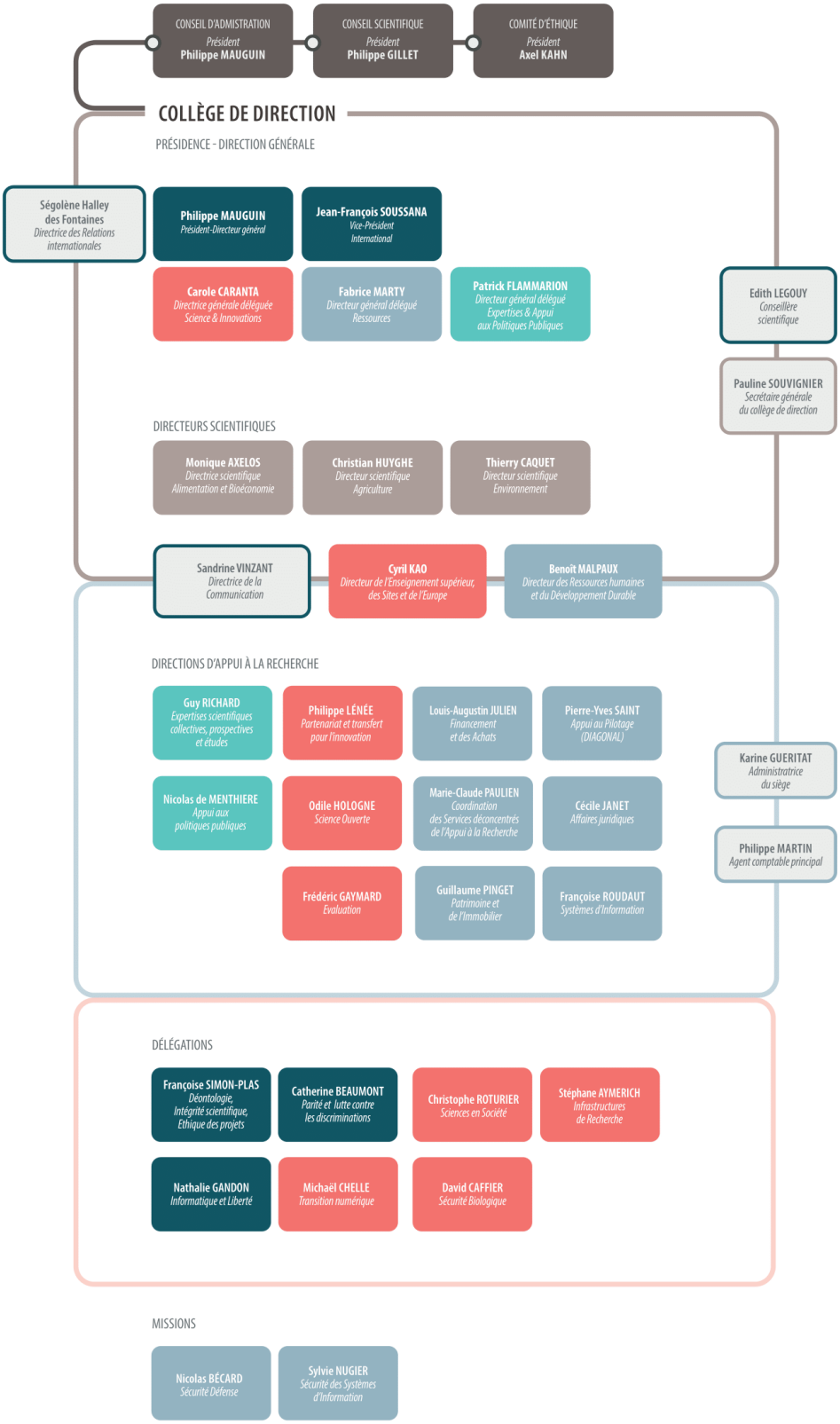
L'unité de rattachement du stage est l'Unité de Mathématiques et Informatique Appliquées de Toulouse (MIAT). C'est une unité du département MathNum d'INRAE et, elle a pour mission de développer et mettre en œuvre des méthodes mathématiques et/ou informatiques pertinentes pour résoudre des problèmes identifiés avec ses collaborateurs qui sont principalement issus d'autres départements d'INRAE.

L'unité comporte actuellement deux équipes de recherche (SCIDYN et SaAB) et trois équipes de service (Plateformes GENOTOUL Bioinfo, RECORD et SIGENAE) :

- SCIDYN (Simulation, Contrôle et Inférence de Dynamiques Agro-environnementales et Biologiques), ex équipe MAD (Modélisation des Agro-écosystèmes et Décision)
- SaAB (Statistiques et Algorithmique pour la Biologie)
- GENOTOUL Bioinfo (Plateforme bioinformatique du GIS GENOTOUL - Génopole Toulouse Midi-Pyrénées)
- RECORD (Plateforme de modélisation et de simulation des agro-écosystèmes)
- SIGENAE (Plateforme "Systèmes d'information des génomes des animaux d'élevage)

Mon stage a été encadré par Nathalie Vialaneix (Directrice de Recherche - HDR, Statistique, MIAT-SaAB, INRAE) et Céline Brouard (Chargée de Recherche, Informatique/Statistique, MIAT-SaAB, INRAE).

FIGURE 1.1 – Organigramme de l'INRAE



Chapitre 2

Objectif du stage et présentation des données

2.1 Objectif du stage

La prédiction de phénotypes est d'un intérêt croissant dans le domaine de la biologie, notamment en agronomie et en médecine, dans la mesure où la réalisation d'expériences *in vivo* est coûteuse en moyens et en temps. Ainsi, être en mesure de prévoir la réaction d'un patient à un traitement ou le rendement d'une plante dans différentes conditions climatiques, en fonction de son génome ou de diverses données cellulaires (données dites « omiques ») intermédiaires, et ce avant même qu'elle ne rentre en production, ouvre des perspectives pour mieux cibler les expériences terrains et accompagner les professionnels à l'aide d'outils d'aide à la décision.

En outre, la démocratisation des méthodes de phénotypage à haut débit et l'accessibilité croissante des techniques de séquencage, ont permis de produire de très nombreuses données omiques et phénotypiques. Cela, couplé à l'augmentation exponentielle des capacités de calcul, permet aux statisticiens et informaticiens l'utilisation de techniques d'apprentissage supervisées de plus en plus variées et performantes. On s'intéresse ici plus particulièrement à la prédiction de phénotypes à partir de données d'expression de gènes. Une approche standard pour aborder cette question consiste à utiliser une méthode d'apprentissage supervisée (comme par exemple les réseaux de neurones, dont les capacités de prédictions ont fait leurs preuves ces dernières années) ayant pour données d'entrée, les taux d'expression d'un ensemble de gènes et pour sortie, un phénotype d'intérêt.

Or, nous savons aujourd'hui que les processus biologiques résultent de multiples interactions entre gènes, ARN, protéines, etc. et que prendre en compte ces informations d'interaction permet d'améliorer la qualité de prédiction des modèles [2]. Par ailleurs, certaines méthodes de réseaux de neurones récentes reprennent les approches appliquées aux images (convolution, pooling, etc), pour les transférer à des données comportant des relations entre les variables, que l'on pourra modéliser sous la forme d'un graphe : elles s'appellent « réseaux de neurones pour graphes » (ou GNN)[3] [4].

Dans le cadre de ce stage, on cherchera dans un premier temps à tester les approches de réseaux de neurones pour graphe dans le contexte de la prédiction de phénotypes au travers de deux exemples, le premier sur la prédiction de la réponse à un régime hypo-calorique chez des patients atteints du syndrome métabolique, et le second sur la prédiction de la survenue de métastases dans les cinq ans chez des patients ayant eu un cancer du sein. Puis à comparer différents modèles de GNN à l'aide de deux librairies

Python (Pythorch Geometric et Spektral) implémentant des réseaux de neurones pour graphes. Ce travail avait initialement pour objectif une poursuite en thèse sur le sujet « réseaux de neurones à base de graphes pour la prédiction et la compréhension des mécanismes cellulaires impliqués dans le syndrome métabolique ».

2.2 Présentation des données

Au cours de ce stage, plusieurs jeux de données ont été utilisés, notamment le jeu de données DiOGene [5] ainsi que les données que nous appellerons « BreastCancer », qui sont un jeu de données public, déjà utilisé dans un article proche de mon sujet de stage [6].

Remarque 2.2.1. *Dans le contexte des réseaux de neurone pour graphe, les mots réseau et graphe peuvent prêter à confusion, c'est pourquoi, on utilisera autant que possible dans ce document le mot réseau pour parler du réseau de neurones et le mot graphe pour parler du réseau de co-dépendance entre gènes.*

2.2.1 Contexte et méthodes d'acquisition des données

Dans le cadre de ce stage, les données principalement utilisées correspondent à des expressions de gènes. Pour rappel, l'information génétique est stockée dans la cellule au travers de l'ADN (Acide DésoxyriboNucléique). Cette molécule séquentielle est composée de deux brins complémentaires formant une double hélice et, l'ensemble de l'ADN contenu dans une cellule représente son génome. Afin de transcrire l'information génétique contenue dans l'ADN pour produire des composés nécessaires au développement et au fonctionnement de l'organisme, une étape de transcription va avoir lieu, où un complexe enzymatique, l'ARN polymérase, va à partir de la lecture du brin d'ADN, synthétiser une molécule d'ARNm (ARN messager). L'ensemble de ces ARN transcrits représente le transcriptome de la cellule. Par la suite, une seconde étape, dite de traduction permettra le passage de l'ARNm à une protéine.

De nombreux facteurs régulent l'expression des gènes en réaction aux conditions du corps et de son environnement. Ainsi la mesure quantitative de l'activité de transcription de gènes d'intérêt peut apporter de nombreuses informations. Cette mesure de l'expression de gènes peut être faite au travers de ce que l'on appelle une puce à ADN. Elle s'effectue généralement au niveau d'un seul tissu à un moment donné.

Pour obtenir l'expression des gènes d'intérêt, on fixe sur une plaque des mono-brins d'ADN connus à des endroits précis, c'est la puce à ADN. Puis on met en contact cette puce à ADN avec des brins d'ADN transcrits à partir de l'ARN contenu dans le tissu que l'on souhaite étudier. Or ces derniers brins auront été traités par fluorescence/radioactivité et on pourra ainsi mesurer la quantité de fluorescence/radioactivité émise au niveau de chacun de nos brins connus. Cette méthode permet aujourd'hui de récupérer les taux d'expression pour plusieurs milliers de gènes à la fois. Néanmoins, même si son coût a grandement diminué, la plupart des jeux de données ne disposent de mesures que pour une centaine d'individus.

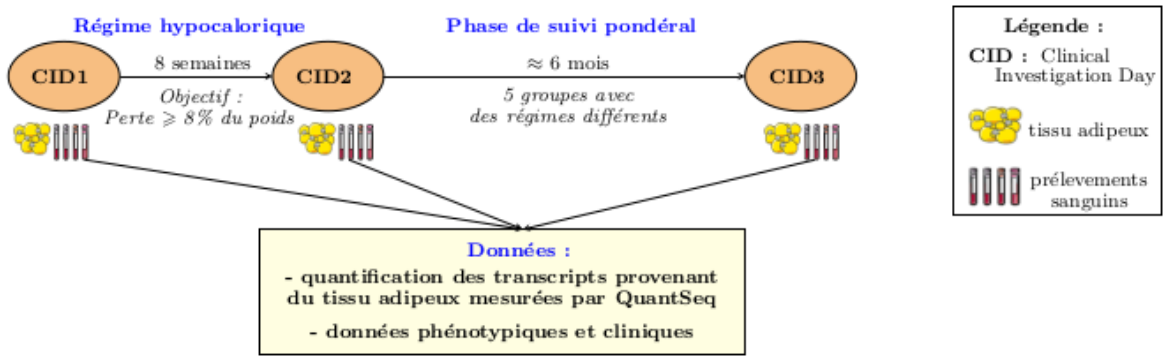
2.2.2 Données DiOGene

Le projet DiOGene [7] (pour Diet, Obesity and Genes) est une étude interventionelle et longitudinale réalisée à l'échelle européenne dans huit pays : Danemark (DK), Pays-Bas

(NL), Royaume-Uni (UK), Grèce (Crète) (GR), Allemagne (D), Espagne (SP), Bulgarie (BG), République Tchèque (CZ). Le but de cette étude était d'observer la réaction à différents régimes diététiques suite à un régime amaigrissant, chez des personnes obèses ou en surpoids. Ce projet à été réalisée dans un cadre contrôlé et randomisé.

Cette étude est composée de deux étapes, tout d'abord une étape de régime amaigrissant basse calorie où les patients avaient pour objectif de perdre 8% de leur poids en huit semaines. Suite à cela ils passaient dans un des groupes randomisés avec une observation et un suivi pendant six mois. Durant ce suivi, ils devaient suivre un régime particulier avec des taux de protéines et de glucides, ainsi qu'un index glycémique bien défini pour chacun des groupes. Le déroulement de l'étude est décrit dans la figure 2.1.

FIGURE 2.1 – Schéma du protocole du projet DiOGenes



Des données ont été récoltées aux trois étapes CID1, CID2 et CID3 comme on peut le voir sur la figure 2.1. Dans ce stage les données analysées correspondent aux données de 416 patients, entre les étapes CID1 et CID3. Elles comprennent des données phénotypiques (âge, genre, etc.), des données cliniques (IMC et Indice de Matsuda) et le comptage de transcrits (quantification des brins d'ARNm) pour 887 gènes. Ces données ont été prétraités pour ce stage par mes maîtresses de stage, notamment pour normaliser les données d'expression.

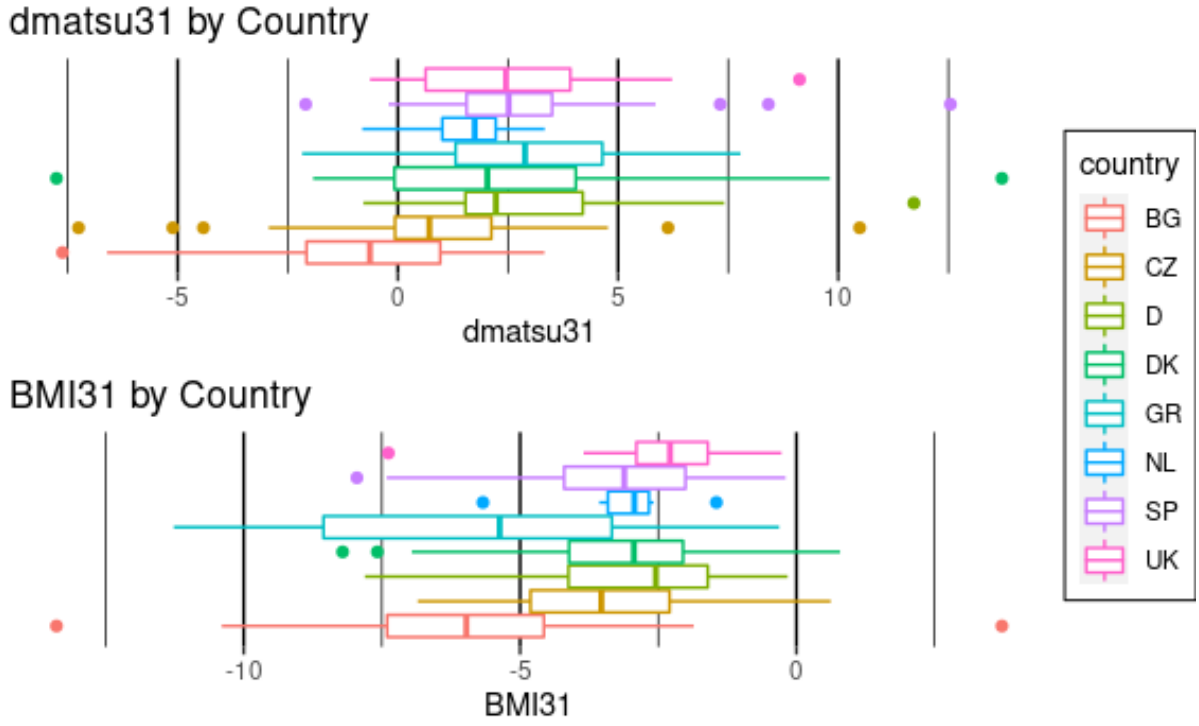
Les deux cibles de prédiction pour ces données seront l'Indice de Masse Corporelle (IMC) et l'indice de Matsuda. Le premier est un indicateur fréquemment utilisé pour évaluer les risques liés au surpoids chez les adultes. Sa formule est la suivante :

$$\text{IMC} = \frac{\text{POIDS}}{\text{TAILLE}^2} \tag{2.1}$$

L'indice de Matsuda [8] est, lui, utilisé pour calculer la sensibilité à l'insuline du corps à un moment donné, grâce à des mesures orales de tolérance au glucose. Des statistiques descriptives sont données pour ces deux indices dans les figures 2.2, 2.3 et 2.4.

L'objectif est ici de voir si il est possible de prédire la réponse des patients à un régime hypo-calorique, c'est à dire les variations des variables choisies à l'aide de réseaux de neurones pour graphe et de graphes inférés de co-dépendance de gène. L'IMC prend ici des valeurs négatives car il s'agit dans notre cas de la différence entre l'IMC au temps CID1 et l'IMC au temps CID3, et grâce au régime, l'IMC a diminué dans la grande majorité des cas. On peut voir qu'il est pertinent d'utiliser les deux indices car il peut y avoir des différences sur leur variation, comme pour la Grèce qui dans la figure 2.2 est

FIGURE 2.2 – Variation de l'IMC (BMI31) et de l'Indice de Matsuda (dmatsu31) en fonction de la localisation géographique



située dans la moyenne pour l'indice de Matsuda alors que la perte d'IMC est elle plus étalée et semble plus faible que la moyenne par rapport aux autres pays. On constate assez peu de différences au niveau de la réponse au régime hypo-calorique des patients qu'ils soient hommes ou femmes, de même il ne semble pas y avoir de tendance liée à l'age dans la réponse au régime d'après l'observation de ces deux indices.

2.2.3 Données BreastCancer

Le jeu de données BreastCancer correspond à un réseau d'interaction protéine-protéine (PPI) venant de l'Human Protein Reference Database (HPRD) et comprenant 6888 nœuds (gènes cartographiés) et des données d'expression pour ces gènes pour 969 patients ayant été atteints par un cancer du sein dans les 10 dernières années. Ces données d'expression ont été acquises à l'aide de puces à ADN et étaient déjà pré-traitées.

Le jeu de données comprend trois fichiers :

— `labels_GEO_HG` contient les labels des patients séparés en deux classes

- les patients **ayant été victimes de métastases** dans les cinq années ayant suivi leur cancer seront représentés par des 1s. Ils représentent 393 patients.
- les patients **n'ayant pas été victimes de métastases** dans les cinq années ayant suivi leur cancer seront représentés par des 0s. Ils représentent 576 patients.

— `GEO_HG_PPI` contient les données d'expression des patients pour les 6888 gènes

— `HPRD_PPI` contient la matrice d'adjacence du graphe PPI

FIGURE 2.3 – Variation de l'IMC (BMI31) et de l'Indice de Matsuda (dmatsu31) en fonction du genre

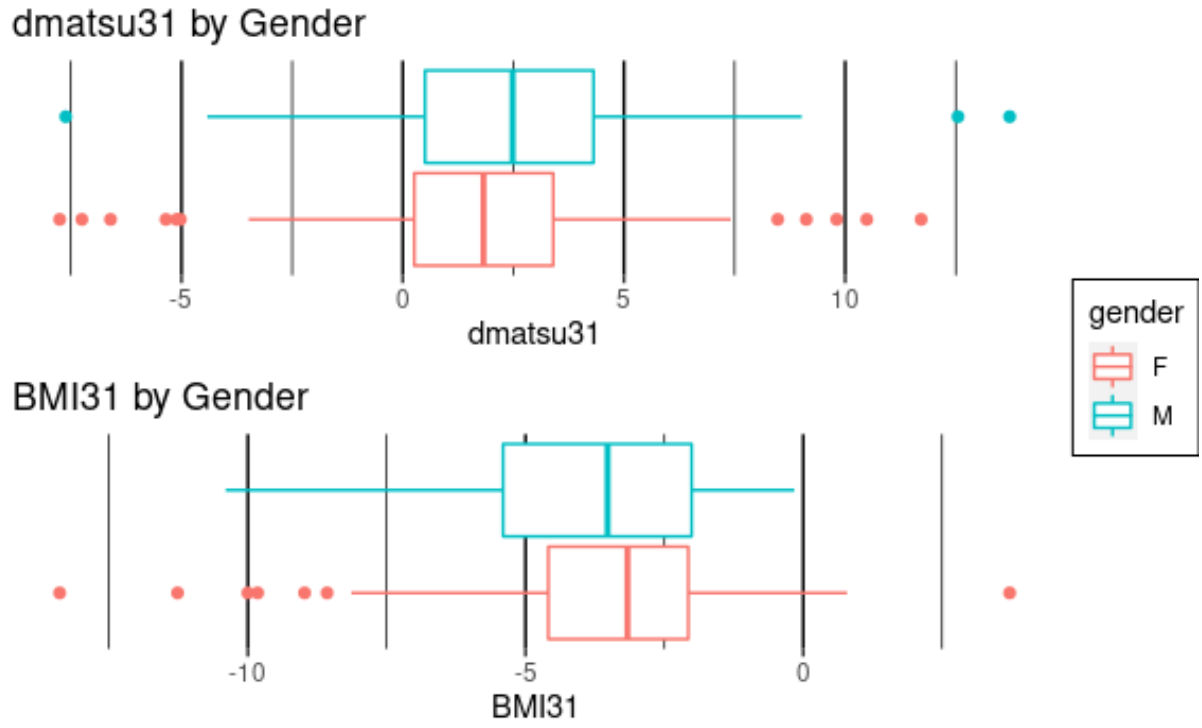
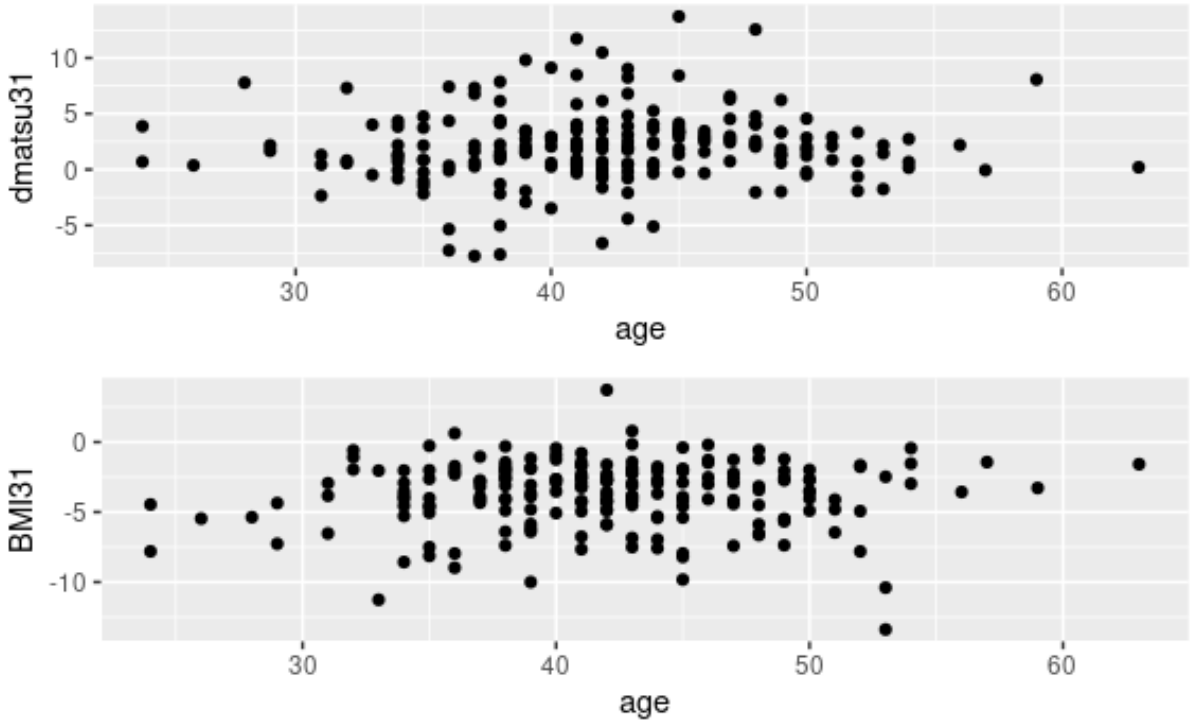


FIGURE 2.4 – Variation de l'IMC (BMI31) et de l'Indice de Matsuda (dmatsu31) en fonction de l'âge



L'objectif sera ici de voir si il est possible de prédire à l'aide d'un réseau de neurones pour graphe la survenue de métastases dans les cinq ans après un cancer.

Chapitre 3

Méthodes

3.1 Graphes

Les graphes sont des structures mathématiques que nous avons utilisées pour représenter les gènes et leurs relations.

Définition 3.1.1 (Graphe)**.** *Un* ***graphe*** G *est un couple* $G = (V, E(A))$ *composé de* V*, un ensemble de nœuds (ou sommets) reliés (ou non) deux à deux par des arêtes de l'ensemble* $E \subseteq V \times V$*. De plus, des poids peuvent être associés aux arêtes,* $A \subset (\mathbb{R}^+)^{p \times p}$ *matrice symétrique avec une diagonale nulle (où* p *est le nombre de nœuds du graphe* G*). On parlera, dans ce cas, d'un graphe pondéré.*

On peut associer, à un graphe $G = (V, E)$, une matrice d'adjacence A de dimension $p \times p$ définie par :

- $A_{ij} = 1$ si $(v_i, v_j) \in E$
- $A_{ij} = 0$ si $(v_i, v_j) \notin E$

On peut aussi, dans certains cas, associer à chaque nœud une étiquette catégorielle (type du nœud) ou numérique (sous la forme d'une matrice d'attributs X de dimension $p \times n$ où n est le nombre de caractéristiques numériques pour chacun des p sommets). Dans notre cas, ces attributs seront la matrice d'expression de gènes où chaque gène correspondra à un sommet et où le taux d'expression de chaque patient sera une caractéristique de chaque sommet. De ce fait dans notre cas d'étude, on aura k graphes (un par patient), chacun comportants p nœuds, et chacun de ces nœuds comportant 1 attribut : le taux d'expression du gène correspondant pour ce patient.

Remarque 3.1.1. *On se place dans le cadre où nous ne considérerons donc pas les effets d'auto-régulation des gènes, d'où la diagonale nulle synonyme d'un d'un graphe simple sans boucle, c'est à dire où un nœud ne peut pas avoir de relation (arête) avec lui même.*

On pourra définir, pour l'étude et la comparaison des graphes, plusieurs propriétés :

Définition 3.1.2. *Le* ***degré*** *d'un nœud correspond au nombre d'arêtes connectées à ce nœud dans le cas d'un graphe non-orienté.*

Définition 3.1.3. *La* ***densité****, D, d'un graphe est le rapport entre son nombre d'arêtes, e, et le nombre maximal d'arêtes d'un graphe ayant le même nombre de nœuds p. On peut la calculer à l'aide de la formule suivante :*

$$D = \frac{e}{\frac{1}{2}p(p-1)} \tag{3.1}$$

Définition 3.1.4. *La **transitivité** d'un graphe correspond à la fréquence à laquelle deux nœuds connectés à un troisième nœud, soient eux aussi connectés.*

Définition 3.1.5. *L'**indice de Moran** [9] (ou join-count pour des valeurs discrètes) est une mesure de la corrélation spatiale. Il mesure si les nœuds ont tendance à être connectés à des nœuds ayant des attributs similaires, aléatoires ou opposés.*

$$I = \frac{p}{\sum_j \sum_{j'} a_{jj'}} \frac{\sum_j \sum_{j'} a_{jj'}(X_j - \bar{X})(X_{j'} - \bar{X})}{\sum_j (X_j - \bar{X})^2} \tag{3.2}$$

avec $a_{jj'}$ le poids lié à l'arête entre les nœuds j et j' dans la matrice A, $\bar{X}$ la moyenne de X, et X_j et X'_j les vecteurs correspondants respectivement aux colonnes j et j' de la matrice X .

Définition 3.1.6. *Le Laplacien d'un graphe est une représentation du graphe souvent utilisée en analyse de graphe. L se calcule selon la formule suivante :*

$$L = D - A \tag{3.3}$$

avec D la matrice diagonale des degrés et A la matrice d'adjacence du graphe. Une forme normalisée existe aussi :

$$L' = I - D^{-1/2} A D^{-1/2} = D^{-1/2} L D^{-1/2} \tag{3.4}$$

3.2 Inférence de graphe

L'inférence de graphe correspond, pour des données d'expression, à la recherche d'un réseau de co-dépendance. Plus précisément, on cherche à retrouver quelles variables omiques ont des relations de dépendances entre elles. Il peut s'agir de relations de co-expression, de relations de corrélation, etc. On s'intéressera pour notre part à la co-expression des variables omiques.

Remarque 3.2.1. *Dans le cas d'un graphe orienté, on cherchera aussi à connaître le sens de la relation, c.à.d. si une variable omique a un effet inhibiteur ou activateur sur une seconde, on serait alors dans le cas d'un réseau de régulation et non de co-expression.*

Une approche naïve standard pour construire ce type de graphe est :

— le calcul des corrélations deux-à-deux entre variables omiques ;

— puis le choix d'un seuil t à partir duquel on considère que la corrélation traduit une relation entre les variables (ou au travers d'un test statistique) ;

— et enfin une définition du réseau : les arêtes sont définies comme les couples de variables ayant une corrélation supérieure au seuil t [10] .

On peut choisir de garder la valeur (absolue) de la corrélation comme poids de l'arête dans le cas d'un graphe pondéré, ou choisir une règle du « tout ou rien » comme décrit ci-dessus.

3.2.1 Modèle Graphique Gaussien

La mesure de la corrélation entre l'expression des gènes deux à deux permet donc d'inférer un réseau de co-expression génique.

Néanmoins la corrélation simple présente des limites, puisque, pour donner un exemple, deux variables a et b (telles que $a = c+\epsilon$ et $b = c+\epsilon$) seront très corrélées entre elles alors qu'elles ne sont liées qu'indirectement. La corrélation ne tient pas compte de l'influence possible de variables extérieures. De ce fait dans le cas qui nous intéresse, on utilisera la corrélation partielle pour calculer la corrélation entre chaque couple de gènes et éliminer l'influence des autres gènes.

On peut définir la corrélation partielle entre deux gènes j et j' comme :

$$\pi_{jj'} = \mathrm{Cor}(X_j, X_{j'}|(X_k)_{k\neq j,j'}) \tag{3.5}$$

On obtient ainsi une matrice Π des corrélations partielles de taille $p \times p$.

Pour la suite, nous allons nous placer dans le cadre du modèle graphique gaussien, où l'on considère que l'expression de chaque gène suit une loi normale : $X = [X_1, ..., X_p] \sim \mathcal{N}(0, \Sigma)$. On observe $X^{(i)}$, réalisations de X pour $i = 1...k$. Dans ce cadre, il a été montré [11] que les corrélations partielles entre deux gènes peuvent être reliées à l'inverse de la matrice de variance-covariance : la matrice de concentration $S = \Sigma^{-1}$:

$$\pi_{jj'} = -\frac{S_{jj'}}{\sqrt{S_{jj}S_{j'j'}}} \tag{3.6}$$

Dans la situation qui nous intéresse, c'est à dire où le nombre d'observations est faible devant le nombre de gènes étudiés, l'estimation de l'inverse de la matrice de variance-covariance est complexe. En effet, l'estimateur empirique de Σ, $\hat{\Sigma}$ est soit non inversible soit mal conditionné (et son inverse n'est donc pas stable) : l'estimation de S est donc impossible à réaliser de cette manière-là ou de mauvaise qualité.

Plusieurs solutions ont été proposées pour pallier à cette difficulté. Dans le cadre du modèle graphique gaussien, certaines sont basées sur le fait que le modèle graphique gaussien est équivalent à la résolution de p problèmes de régression linéaire :

$$X_j = \sum_{k\neq j} \beta_{jk} X_k + \epsilon \tag{3.7}$$

dans lesquels l'expression du gène j est exprimée comme une combinaison linéaire de l'expression de tous les autres gènes. En effet, on peut montrer qu'une relation explicite lie les coefficients du modèle linéaire précédent aux corrélations partielles :

$$\beta_{jk} = -\frac{S_{jk}}{S_{jj}}. \tag{3.8}$$

Ainsi, estimer β permet de pouvoir utiliser la régression pénalisée et de se placer ainsi dans un cadre où l'on peut estimer la matrice de concentration S et donc la matrice des corrélations partielles Π.

Lors de l'application de l'inférence du graphe sur les données DiOGene, trois librairies R reposant sur des méthodes différentes ont été utilisés : *GeneNet*, *rags2ridges* et *WGCNA*.

3.2.2 Inférence à l'aide de la librairie R *WGCNA*

La librairie *WGCNA* [12] se base sur l'analyse de graphe par corrélation pondérée (ou weighted correlation network analysis [13]). Cela se rapproche de l'approche dite naïve. La première étape consiste à construire une matrice de similarité $s_{jj'} = |cor(x_j, x'_j)|$ entre les gènes j et j', puis à calculer une matrice d'adjacence. La matrice d'adjacence peut être soit composée de 0 et de 1 ($a_{jj'} = 0$ si $s_{jj'} < \tau$ et $a_{jj'} = 1$ si $s_{jj'} > \tau$ avec τ un seuil défini par l'utilisateur), ou de poids, et dans ce second cas, elle sera égale à $\tilde{a}_{jj'} = s^{\beta}_{jj'}$, le β optimal étant choisi à l'aide de la librairie.

De plus la librairie *WGCNA* propose de nombreux outils d'analyse de réseau d'expression de gène, avec notamment de la détection de modules, de la sélection de gènes et de la visualisation et analyse du réseau suivant les tâches d'intérêt.

3.2.3 Inférence à l'aide de la librairie R *GeneNet*

La librairie *GeneNet*, et notamment la fonction `ggm.estimate.pcor` permet d'estimer la matrice des corrélations partielles, calculées selon la méthode de Schäfer et Strimmer [14]. Elle a été spécifiquement créée pour des données où le nombre d'échantillons est inférieur au nombre de variables. La méthode se base sur le calcul d'un estimateur alternatif de la matrice de covariance S^*. L'avantage de ce dernier est de toujours être positif, défini, bien conditionné et inversible, car il utilise une régularisation ridge (équation 3.9),

$$S^* = (\hat{\Sigma} + \lambda I)^{-1} \tag{3.9}$$

avec $\lambda \in \mathbb{R}^+$ l'hyper-paramètre de régularisation.

Une fois la matrice des corrélations partielles estimée, on peut utiliser la librairie pour sélectionner les arêtes à conserver en comparant la valeur du *local FDR* à un seuil. Seules les arêtes ayant une valeur inférieure à ce seuil, pour l'estimateur de la corrélation partielle entre les deux gènes qu'elles relient, seront conservées. Le *local FDR* est une statistique calculé automatiquement par la librairie.

3.2.4 Inférence à l'aide de la librairie R *rags2ridges*

La librairie *rags2ridges* est elle aussi basée sur une pénalisation ℓ_2 de l'estimation de la matrice de précision empirique, mais elle diffère de celle de la librairie *GeneNet* et est basé sur l'estimation donnée par [15]. Il estime tout d'abord le paramètre λ par validation croisée à 10 blocs puis calcule la matrice de précision associée au paramètre λ optimal [16]. De la même façon que pour la librairie *GeneNet*, une sélection d'arête peut être effectuer en comparant le *local FDR* à un seuil et en ne conservant que les arêtes ayant des valeurs inférieures à ce seuil.

3.3 Réseaux de neurones pour graphe

De nombreuses données, issues de domaines variés, peuvent être représentées sous forme d'un graphe : chimie (molécules), écologie (réseaux d'interactions), finances (réseaux de transactions), réseaux sociaux, etc. De plus, il est certain que les réseaux de neurones et le deep learning représentent une révolution en terme d'apprentissage, notamment pour la vision par ordinateur, et dans de nombreux autres domaines, bien que leur coté « boite noire » ne les rends pas utiles dans tout les cas. L'existence d'une quantité toujours plus

importante de données disponibles, sous des formes très hétérogènes, amène à adapter les solutions existantes. Les réseaux de neurones pour graphes font partie de ces adaptations, de par leur capacité à traiter des données représentées sous forme de graphes.

3.3.1 Réseaux de neurones

Les réseaux de neurones sont un ensemble de méthodes d'apprentissage supervisé automatique et profond. Le neurone formel est à l'origine créé pour modéliser les neurones biologiques et imiter, par analogie, leurs capacités [17]. Néanmoins, ces premiers modèles ne sont même pas capables d'apprendre et il faudra attendre l'apparition du perceptron pour cela [18]. De plus, il faudra attendre plusieurs innovations telles que la rétropropagation du gradient [19][20] et les modèles multi-couches [21], qui, couplés avec le développement des capacités informatiques, ont permis les premières utilisations pratiques des réseaux de neurones [22].

Plus concrètement, un neurone est composé d'une couche d'entrée acceptant des variables d'entrées $x \in \mathbb{R}^p$, d'un biais w_0, de poids w, d'une fonction d'activation σ et d'une ou plusieurs valeurs de sortie y. On a ainsi la formule suivante :

$$y = \sigma\left(w_0 + \sum_{j=0}^{p} w_j x_j\right) \tag{3.10}$$

Le perceptron multicouche est, lui, composé de plusieurs couches, chacune composée de plusieurs neurones comme on peut le voir dans la figure 3.1. En théorie, on peut simuler n'importe quelle fonction avec une seule couche et un nombre suffisant de neurones mais l'apprentissage est alors beaucoup plus long que pour un perceptron multi-couche. L'information parcourt ce réseau depuis la couche d'entrée vers la couche de sortie. Les neurones de chaque couche k sont reliés à tous les neurones des couches précédentes $(k-1)$ et suivantes $(k+1)$. Chaque neurone modifie les valeurs qu'il prend en entrée à l'aide de poids puis les assemble pour obtenir une valeur de sortie. L'apprentissage du perceptron passe par le calcul d'une fonction de perte qui mesure l'écart entre la prédiction du réseau de neurones (les valeurs de sa couche de sortie) et la réalité connue. Une fois cette différence calculée, le réseau de neurones corrigera les poids par descente de gradient stochastique.

Lors de la passe « en avant »(ou passe *forward*), le neurone va effectuer une combinaison linéaire des valeurs d'entrée, combinée à une fonction d'activation pour apporter de la non-linéarité. Cette fonction d'activation σ permet de passer d'une combinaison linéaire (où il n'y aurait pas d'intérêt à avoir plusieurs neurones puisque une combinaison linéaire de plusieurs combinaisons linéaires des mêmes valeurs d'origine peut s'exprimer comme une unique combinaison linéaire des valeurs d'origine) à une transformation non linéaire. Il en existe plusieurs différentes qui doivent remplir certaines conditions : dérivabilité, non-linéarité (dans la majorité des cas) et monotonie. Les principales sont données dans le tableau 3.3.1. Ces fonctions d'activation sont notamment adaptées à la descente de gradient, qui est au coeur de l'apprentissage du réseau de neurones.

Ensuite, l'erreur d'apprentissage est calculée au travers de la fonction de perte qui dépend de la tâche en cours : régression ou classification. Les plus communes sont l'erreur quadratique moyenne (MSE) dont la formule est donnée par l'équation 3.11 avec $\hat{Y}_i$ la prédiction pour l'individu i, et l'erreur absolue moyenne (MAE) pour la régression et l'entropie croisée (Cross-entropy) pour la classification. L'objectif d'apprentissage est de chercher à minimiser la fonction de perte grâce à une méthode de descente de gradient stochastique. Ainsi lorsque on ne constate plus de diminution de cette fonction de perte,

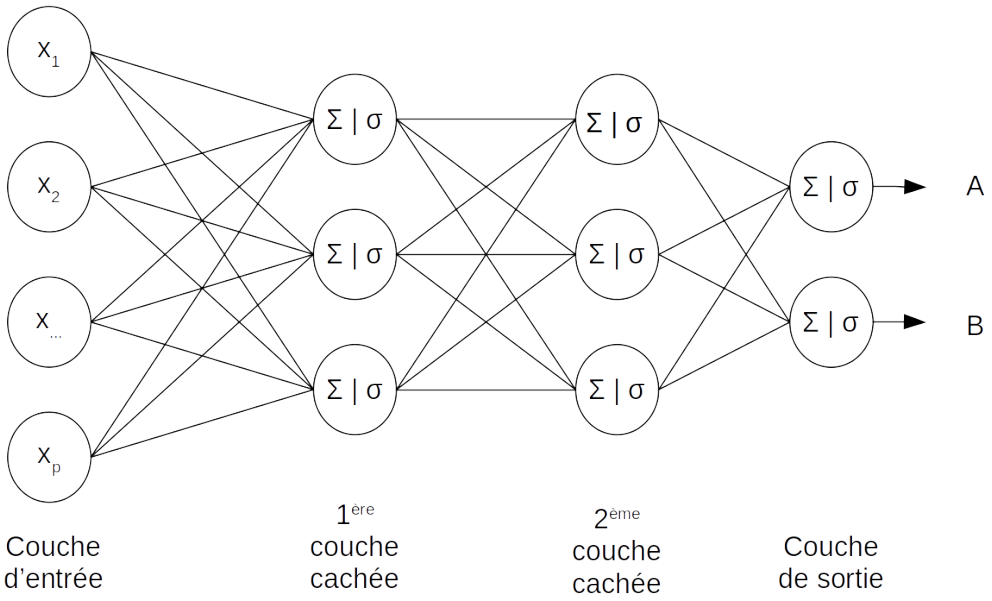


FIGURE 3.1 – Schéma simplifié d'un perceptron multicouche comportant deux couches cachées et effectuant une classification binaire

on peut arrêter prématurément les itérations d'apprentissage (on parlera alors d'un hyper-paramètre de patience).

$$\text{MSE} = \frac{1}{n}\sum_{i}(Y_i - \hat{Y}_i)^2 \tag{3.11}$$

Viens ensuite l'étape de la passe « en arrière » (ou passe *backward*) qui consiste en la mise à jour des poids w du réseau de neurones par descente de gradient stochastique. De façon itérative, on va ainsi calculer dans un premier temps l'erreur d'apprentissage pour une observation x_i (ou l'erreur moyenne sur un batch d'observation), puis on va mettre à jour chacun des poids en fonction de cette erreur, à partir du gradient de l'erreur pour l'observation x_i. On aura ainsi pour le neurone k de la couche l :

$$w_{ik}^{(l)} = w_{ik}^{(l-1)} - \eta e_k^{(l-1)} x_i^{(l-1)} \tag{3.12}$$

On remarquera aussi la présence du taux d'apprentissage η dans l'équation (3.12). Ce terme joue sur la vitesse à laquelle les connaissances apprises remplacent celles déjà connues. Il convient de bien le régler car pour converger vers le minimum de la fonction de perte, il est intéressant de progresser rapidement par soucis de rapidité, mais cela peut poser des problèmes lors des dernières étapes où il faut une correction plus fine des poids. Pour cela, il existe des algorithmes d'optimisation adaptatifs qui régleront le paramètre η au fur et à mesure des étapes, le plus utilisé actuellement étant l'algorithme Adam [23].

Fonction d'activation	Formule	Etendue
Heaviside	$\sigma(x) = \begin{cases} 0 \text{ si } x < 0 \\ 1 \text{ si } x > 0 \end{cases}$	$\{0\,;1\}$
Logistique	$\sigma(x) = \frac{1}{1+e^{-x}}$	(0,1)
ReLU	$\sigma(x) = \begin{cases} 0 \text{ si } x < 0 \\ x \text{ si } x > 0 \end{cases}$	$[0, \infty[$
ELU	$\sigma(x) = \begin{cases} \alpha(e^x - 1) \text{ si } x < 0 \\ x \text{ si } x > 0 \end{cases}$	$(-\alpha, \infty)$

TABLE 3.1 – Différentes fonctions d'activation utilisées dans les réseaux de neurones

En outre, cette descente de gradient ne s'effectue pas en général après chaque observation ni après une passe complète du jeu de donnée, mais toutes les m observations, ces paquets d'observations ainsi formés sont ce que l'on appelle des mini-batch. Enfin lors de l'apprentissage, on pourra jouer aussi sur la régularisation des poids (hyperparamètre souvent appelé *decay*) pour éviter le sur-apprentissage.

Pour résumer, de nombreux paramètres doivent être choisis dans un réseau de neurones :

— les variables d'entrée et les variables de sortie

— le nombre de couches du réseau et la taille de chaque couche

— le taux d'apprentissage η

— le nombre d'epochs (nombre de fois où l'on utilise le jeu de données en entier pour l'apprentissage du réseau de neurone)

— la présence de régularisation

— la taille des mini-batchs

Gardons en tête que cela ne prends en compte qu'une architecture simple : celle d'un perceptron multicouches, des architectures plus complexes entraînent souvent une augmentation du nombre d'hyper-paramètres à ajuster.

3.3.2 Extension aux données de graphe

Un des domaines majeur d'application des réseaux de neurones reste la vision par ordinateur, et ce notamment grâce à l'introduction d'un nouveau type de réseaux de neurones : les réseaux de neurones convolutionels [22]. Ces derniers sont composés de deux types de couches : les couches de convolution et les couches de pooling. La première consiste en la récupération de l'information proche, et permet de regarder non plus un pixel mais un pixel et ses voisins, afin de tenir compte de la structure des données. Suit une étape de pooling qui sert, elle, à agréger les résultats de la couche de convolution. Ainsi on peut, à partir de petites sous-étapes, créer des réseaux capables de reconnaître des formes complexes.

Néanmoins, ces techniques ne sont applicables qu'à des données respectant une géométrie euclidienne : signaux en 1D, images en 2D, vidéos ou volumes en 3D. Or les données

de type graphes ne respecte pas cette organisation et il a fallu adapter ces techniques pour pouvoir les utiliser dans ce cadre. C'est ainsi qu'ont émergé les réseaux de neurones pour graphe (GNN) [3], capable de fournir des prédictions à l'échelle du graphe ou de chacun des nœuds du graphe comme dans la figure 3.2.

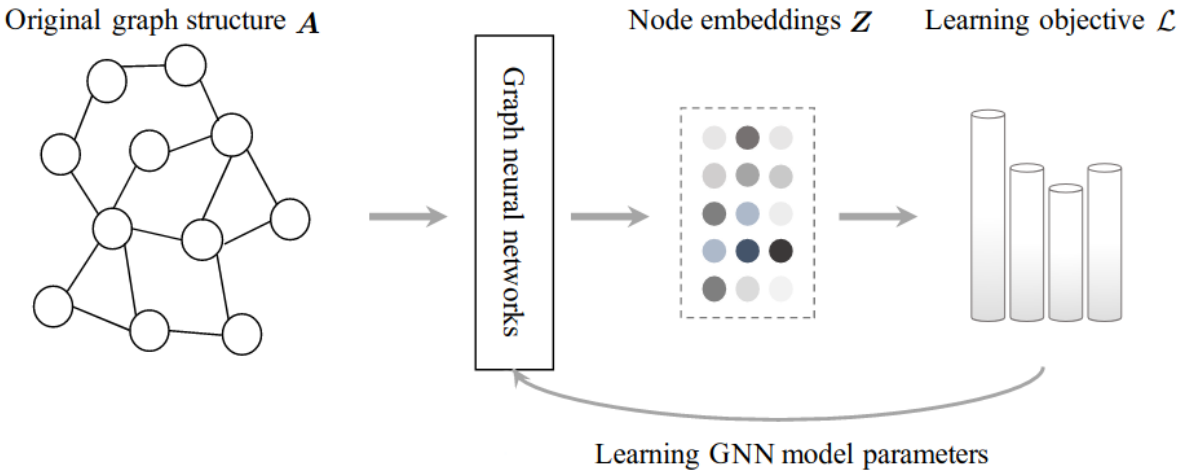


FIGURE 3.2 – Apprentissage d'un réseau de neurones pour graphe [24]

Dans la même logique d'utiliser l'information contenue dans les pixels voisins, dans un GNN, on va chercher à utiliser l'information contenue dans les nœuds voisins, et pour ce faire, on va passer par une étape de *message passing*. Mathématiquement une couche de *message passing* correspond à l'équation (3.13), avec H les labels des noeuds du graphe (une valeur les décrivant et qui pourra être agrégée par la suite pour avoir un label pour le graphe dans son ensemble si nécessaire).

$$H^{(k+1)} = \sigma(\tilde{L}H^{(k)}W^{(k)}) \tag{3.13}$$

Remarque 3.3.1. *Ici on utilisera une version* $\tilde{L} = \tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}$ *avec* $\tilde{A} = A + I_N$ *et* $\tilde{D}_{j'j'} = \sum_j \tilde{A}_{jj'}$

Par la suite un autre type d'étape est fréquemment ajouté lorsque la prédiction se fait au niveau du graphe. En effet, avec les réseaux de neurones pour graphe, il est possible de faire des prédictions aussi bien au niveau du nœud que du graphe : par exemple si notre graphe représente une molécule, on peut vouloir prédire des propriétés au niveau de l'atome ou de la molécule. Pour agréger l'information de différents nœuds, on pourra utiliser une ou plusieurs couches de pooling.

Chapitre 4

Application

4.1 Inférence de graphe

4.1.1 Comparaison des graphes inférés à partir des données DiOGene

À partir des données DiOGene, trois graphes ont été inférés, à l'aide des trois librairies R : *GeneNet*, *rags2ridges* et *WGCNA*. Chacun est basé sur une méthode différente mais *GeneNet* et *rags2ridges* sont assez proches contrairement à *WGCNA*.

Une des particularités des réseaux de co-expression est le fait qu'ils sont parcimonieux. Pour atteindre cet objectif, nous avons choisi d'utiliser les méthodes incluses dans les librairies *GeneNet* et *rags2ridges* pour réduire le nombre d'arêtes. La méthode consiste en la sélection d'un seuil à partir duquel on choisi de garder ou non l'arête. Ce seuil à été fixé à 0.2 pour les deux librairies comme recommandé dans le papier de B. Efron [25], et a été comparé au *local FDR* (ou local false discovery rate [26]) qui correspond à la probabilité postérieure qu'une arête soit nulle sachant l'estimateur $\tilde{r}_{ij}$ présenté dans l'équation (4.1). Ainsi pour *rags2ridges* et *GeneNet*, lorsque le *local FDR* est inférieur au seuil choisi (ici 0.2), alors on la considèrera comme existante, et à l'inverse, si le *local FDR* est supérieur au seuil alors on la considèrera comme nulle.

$$\tilde{r}_{jj'} = \frac{-\hat{\omega}_{jj'}}{\hat{\omega}_{jj}\hat{\omega}_{j'j'}} \tag{4.1}$$

avec

$$(\hat{\omega}_{jj'}) = \hat{S} = \hat{\Sigma}^{-1} \tag{4.2}$$

Pour le graphe obtenu à partir de *WGCNA*, il a été choisi de conserver un nombre d'arêtes égal au nombre d'arêtes moyen obtenu par les deux autres graphes (qui ont des valeurs proches) afin de garder une base de comparaison plus simple.

Les arêtes obtenues grâce aux trois librairies sont représentées dans le diagramme de Venn de la figure 4.1. On voit ici les différences entres les deux types de méthodes, produisant ainsi des graphes différents.

En complément, différentes statistiques ont été calculées comparer pour chacun des graphes obtenus dans la table et 4.1 afin de présenter leurs particularités.

On peut voir dans la table 4.1 que les graphes sont bien parcimonieux de par leur faible densité, comme on le souhaite puisque c'est généralement le cas dans les réseaux

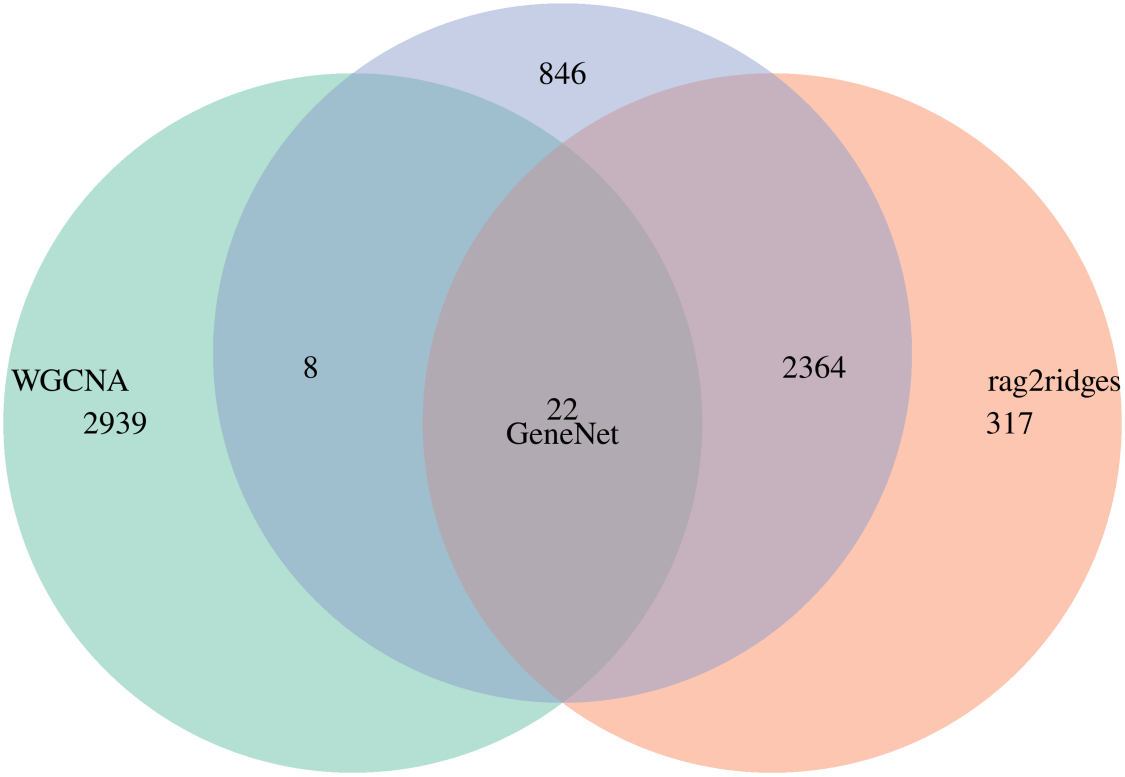


FIGURE 4.1 – Diagramme de Venn des arêtes en commun

de co-expression connus. De plus, on remarque ici la différence entre le graphe issu de *WGCNA* et ceux issus des deux autres librairies.

L'objectif de l'inférence de ces différents graphes est de pouvoir comparer les résultats obtenus afin d'observer l'influence du graphe sur les résultats que peuvent donner les réseaux de neurones pour graphe. Evidemment à cela il faut ajouter une comparaison avec des graphes complètement connectés (matrice d'adjacence remplie de 1 sauf sur la diagonale) et non connectés (matrice d'adjacence remplie de 0). Au vu des graphes obtenus, on s'attends à obtenir des résultats similaires pour les graphes inférés à partir de *GeneNet* et de *rags2ridges*, et probablement assez différents par rapport au graphe inféré à partir de la librairie *WGCNA*. Néanmoins la durée de ce stage n'a pas permis de tester ces différents graphes et pour les applications, c'est le graphe inféré à partir de la librairie *rags2ridges* qui a été utilisé pour les différentes implémentations de GNN avec les données DiOGene.

4.2 Benchmarking des réseaux de neurones pour graphe

4.2.1 Comparaison entre PyTorch Geometric et Spektral

Aujourd'hui, pour travailler avec les réseaux de neurones, les deux librairies les plus utilisées sont Pytorch et Tensorflow/Keras. L'arrivée des réseaux de neurones pour graphe

		GeneNet	rags2ridges	WGCNA
Nombre d'arêtes		3240	2703	2969
Densité		0.0082	0.0069	0.0076
Transitivité		0.030	0.024	0.392
Join-count	BMI	0.067	0.053	0.252
	Matsuda Index	0.069	0.053	0.218

TABLE 4.1 – Statistiques obtenues sur les trois graphes générés avec *GeneNet*, *rags2ridges* et *WGCNA*.

a conduit au développement d'extensions pour ces deux librairies : respectivement Pytorch Geometric [27] et Spektral [28].

Les deux implémentations sont assez similaires mais des différences existent. Là où Spektral implémente 18 couches de convolution et une dizaine de couches de pooling, Geometric propose lui près d'une cinquantaine de couches de convolution et une trentaine de couches de pooling[1]. Néanmoins l'utilisation de Spektral m'a semblé plus abordable dans un premier temps et les premiers tests pour la prise en main des GNN ont été effectués avec. Par la suite lorsque j'ai cherché à implémenter des réseaux plus complexes, il s'est avéré que Spektral comportait des limitations et que certaines couches qui n'étaient pas présentes (notamment les couches de pooling local), étaient trop compliquées à implémenter, et de ce fait le travail sur la partie traitant des données BreastCancer a été effectué à l'aide de Geometric. En effet la création de nouvelles couches non encore implémentées, est possible pour les deux librairies, mais il est nécessaire de se baser sur les implémentations des couches de base déjà existantes, et/ou d'avoir une très bonne maitrise du fonctionnement des librairies sous-jacentes. Il faut aussi ajouter le fait que la documentation des librairies n'est pas complètement à jour, étant des librairies récentes et maintenues principalement par de petites équipes (voire une seule personne).

Une autre différence notable entre les deux librairies provient de leurs librairies de base, Pytorch pour Geometric et Tensorflow/Keras pour Spektral, où entre autres, le multithreading (c'est à dire la capacité à effectuer des calculs en parallèle sur le processeur de la machine, n'est pas implémenté de base pour Geometric contrairement à Spektral. Ainsi même si les deux librairies présentent des temps de calculs similaires dans des conditions similaires, sur ma machine qui comportait un processeur 8 cœurs, les calculs s'effectuaient donc environ 8 fois plus vite avec Spektral par rapport à avec Geometric. Même s'il est possible d'implémenter le calcul en parallèle sur Geometric, les choix de design des scripts et le temps du stage n'ont pas permis de le mettre en place.

4.2.2 Comparaison entre modèles de GNNs

De la même façon que pour les réseaux de neurones plus classiques, il existe un nombre croissant d'implémentations de réseaux de neurones pour graphe. Dans ce stage, on s'est notamment intéressé à la prédiction de phénotype à partir de données d'expression génétique, on s'intéressera donc plutôt à la prédiction au niveau du graphe dans son ensemble plutôt que au niveau du nœud, bien que les deux soient possibles. Certaines méthodes

1. Au moment de l'écriture de ce mémoire (août 2021)

permettent d'ailleurs de travailler aux deux niveaux alors que d'autres non, la différence se situant souvent au niveau de l'étape de pooling et de l'architecture du réseau utilisé.

La création rapide de nombreuses implémentations de couches de réseaux de neurones pour graphe semble avoir conduit à la création de nombreuses architectures spécialisées sur une tâche, mais peu testées dans différentes conditions, de la même façon, les combinaisons possibles, et le choix des nombreux hyper-paramètres n'a actuellement pas reçu assez d'attention à l'exception de quelques papiers dont celui de You et al [29], qui pose des bases sur les différents paramètres sur lesquels on peut jouer lors de la construction de notre réseau de neurones pour graphe. En effet un concept important dans le cas qui nous intéresse, est celui de *transfer learning* : on va chercher un modèle ayant de très bonnes performances sur une tâche similaire à celle qui nous intéresse et on va l'utiliser, pré-entrainé pour aller au plus vite vers un modèle donnant de bons résultats. Dans le cas des réseaux de neurones profonds, ce concept est souvent utilisé dans les couches basses, c'est à dire les premières couches, où l'on utilise des couches pré-entrainées pour des tâches similaires. Pour faire une analogie avec les réseaux de neurones spécialisés dans la reconnaissance d'image, si on prend l'exemple d'un réseau entrainé à détecter un carré, les premières couches vont probablement servir à détecter chacun des quatre côtés de notre carré, or si l'on souhaite entrainer un réseau à reconnaitre des maisons, savoir reconnaitre des formes géométriques comme des lignes verticales et horizontales sera surement très pratique. On peut donc utiliser ces premières couches obtenues après l'entrainement pour les carrés, et ainsi accélérer la vitesse d'entrainement de notre réseau, et pourquoi pas pouvoir réduire le nombre d'exemples nécessaires pour l'apprentissage. C'est pour cette raison qu'il est important de pouvoir identifier quels paramètres vont être importants et quels designs/architectures peuvent être réutilisés pour notre tâche d'intérêt. Bien sûr il ne faut pas céder à la simplicité et garder à l'esprit le *no free lunch theorem* : Aucun modèle n'est bon pour toutes les tâches.

La complexité de mettre en place une architecture complète pour tester et benchmarker le design des réseaux de neurones pour graphe a conduit à une majeure partie d'analyse bibliographique et à des essais de reproduction des résultats rapportés dans divers papiers scientifique. A première vue, et en accord avec la littérature à ce sujet [30], les papiers traitant de nouvelles implémentations de réseaux de neurones pour graphes ne sont pas toujours reproductibles, car ils ne fournissent pas le code source pour pouvoir réimplémenter l'expérience exacte, ou bien plus souvent car ils ne donnent pas l'ensemble des hyperparamètres nécessaires pour pouvoir reproduire les résultats qui ont été obtenus. Dans certains cas il apparait que dans un contexte rigoureux (incluant notamment cross-validation et normalisation correcte des différents split de jeux de données), les résultats sont uniquement reproductibles sur certain splits du jeu de données ou pour une tâche très précise [31]. Au cours du stage, on a notamment chercher à reproduire les résultats de l'article de Chereda et al [6] qui utilise le jeu de données BreastCancer. Or cet article ne fournit pas le code source, et il manque certains paramètres comme l'algorithme d'optimisation (Adam ou SGD généralement), le taux d'apprentissage ou le nombre d'epochs utilisés. Heureusement la publication d'un papier au cours du stage [32], a permis de lever certaines interrogations, mais néanmoins pour cette expérience, les chercheurs ont créé une implémentation de GNN « maison » en partant de Keras/Tensorflow, et non en utilisant la librairie Spektral, or le manque de bases communes rend difficile la comparaison entre résultats puisque on a souhaité au cours de ce stage utiliser les librairies Spektral et Geometric. Il en ressort qu'il nous a été impossible d'obtenir une aussi bonne performance qu'attendue, avec une précision de prédiction de la survenue d'événements

métastatiques de 73.79% pour notre implémentation sous Spektral contre une précision de 76.18% pour les auteurs. La difficulté de reproduire ces résultats est un des problèmes rencontrés pour le benchmark des réseaux de neurones pour graphe pour la prédiction de phénotypes. Une analyse portant sur différents jeux de données et testant un panel plus large d'architectures serait intéressante à réaliser dans la continuité de ce travail.

4.2.3 Application sur le jeu de données DiOGenes

A l'origine, il était prévu pour ce stage (et pour le doctorat qui aurait dû suivre) d'utiliser les données DiOGenes et d'appliquer les réseaux de neurones pour graphe dessus. Comme cela a été dit précédemment, l'application, a été faite à l'aide de la librairie Spektral, même s'il était au début prévu de tester les deux librairies, il a été choisi de commencer avec Spektral et l'utilisation des données DIOGenes a été abandonnée avant qu'une implémentation fonctionnelle ne soit faites avec Geometric (bien qu'elle ait été commencée). L'implémentation s'est faites à l'aide d'un réseau « classique », composé de deux couches GCN de taille 32 et 64 suivi d'une couche de pooling et de deux couches dense de taille 64 avec des fonctions d'activation elu, ont aussi été utilisés un batch size de 32, un learning-rate de 0.001 avec un algorithme Nadam. Il est rapidement apparu que les résultats obtenus étaient peu encourageants, la plupart des modèles entrainés n'arrivant pas à prédire mieux qu'un modèle aléatoire. D'autres hyperparamètres ont été testés, jouant notamment sur la taille des couches, le learning rate et les fonctions d'activation mais aucun n'a permis une réelle amélioration des performances. Les raisons possibles de ces faibles performances sont présentées dans la discussion.

4.2.4 Comparaison avec d'autres méthodes d'apprentissage supervisées

Les performances des réseaux de neurones pour graphe ont été comparées avec différentes méthodes de machine learning qui sont déjà très utilisées pour la classification et qui n'ont plus à prouver leur efficacité. Les méthodes testées sont les RandomForest et la régression Ridge. La librairie Python Scikit-Learn [33] a été utilisée pour effectuer les différentes étapes de la modélisation et les données sont celles du jeu de donnée Breast-Cancer, où l'on va essayer de prévoir la survenue de métastases dans les cinq ans suivant un cancer du sein.

Remarque 4.2.1. *Comme dit ci-dessus, il était prévu d'utiliser les données DiOGenes pour comparer les performances des réseaux de neurones pour graphe avec d'autres méthodes d'apprentissage, mais plusieurs problèmes au cours du passage du jeu de données initial à un jeu de données exploitable et compréhensible par Spektral, ainsi que des performances très faibles ($R^2 \approx 0$), nous on conduit à essayer avec un jeu de données pour lequel on était sûr de l'existence d'un lien entre les données d'expression et la variable à prédire. Le jeu de donnée BreastCancer ayant déjà été utilisé dans l'article de Chereda et al [6], et avec une application sur les réseaux de neurones pour graphe, il nous a semblé pertinent de l'utiliser.*

Pour le réseau de neurones de graphe, une implémentation classique sera utilisée, elle est décrite dans la figure 4.2. Elle comporte deux couches de *message passing* GCN Conv [34] de taille 32 avec des fonctions d'activation Elu, suivies d'un pooling global max et

Méthode	Précision %	100*AUC
Réseau de neurones pour graphe	73.79	-
Perceptron avec pénalité ℓ_1 ($\lambda = 1e^{-3}$)	75.72	76.83
Random Forest ($n = 5000$)	76.54	74.21
Ridge Regression ($\lambda = 0.1$)	76.54	75.95

TABLE 4.2 – Comparaison des performances de prédiction de la survenue d'événements métastasiques par diverses méthodes de machine learning

de deux couches denses de taille 64 avec des fonctions d'activations Relu. Un taux d'apprentissage de 0.0005 avec un algorithme Nadam ont été utilisés. Au cours de la mise en place de cette implémentation, et pour toutes les autres du stage utilisant Spektral, un des problèmes rencontrés a consisté en l'impossibilité d'utiliser un pooling local au détriment d'un pooling global, à cause d'incompatibilités au niveau des librairies probablement. Le problème étant la perte d'informations, puisque l'utilisation d'un pooling global réduit grandement le nombre d'informations que l'on peut passer à la couche suivante (ici une couche dense). C'est probablement cette perte d'information qui explique les mauvais résultats du GNN par rapport aux autres méthodes, et même par rapport aux résultats rapportés dans les papiers de recherche. Il convient donc de nuancer ces résultats et, arriver à utiliser un pooling local serait surement un bon départ pour améliorer les performances de notre GNN.

Chacune des méthodes testées (or GNN), l'a été grâce à la librairie Scikit-Learn qui met à disposition de nombreux outils pour la sélection et la comparaison de modèles d'apprentissage. Pour la sélection des hyper-paramètres, il a été choisi de procéder par validation croisée à 10 blocs. Pour cela, le jeu de données a d'abord était séparé en un jeu de donnée d'entrainement (75% des échantillons) et un jeu de données de test (25% des échantillons) en prenant garde à garder la même proportion de chaque label (présence ou absence de métastases) dans les deux jeux de données. Puis le jeu de données d'entrainement a servi à la validation croisée pour sélectionner les hyper-paramètres, et le meilleur modèle a été testé sur le jeu de test. Les statistiques présentées dans la table 4.2 correspondent à celles calculées à partir du meilleur modèle sur le jeu de donnée de test. Les hyper-paramètres sélectionnés sont : pour les RandomForest, le nombre d'arbres de la forêt aléatoire, pour la régression Ridge (ou ℓ_2), le paramètre λ de régularisation, et de même pour le perceptron où une régularisation Lasso (ou ℓ_1) a été appliquée.

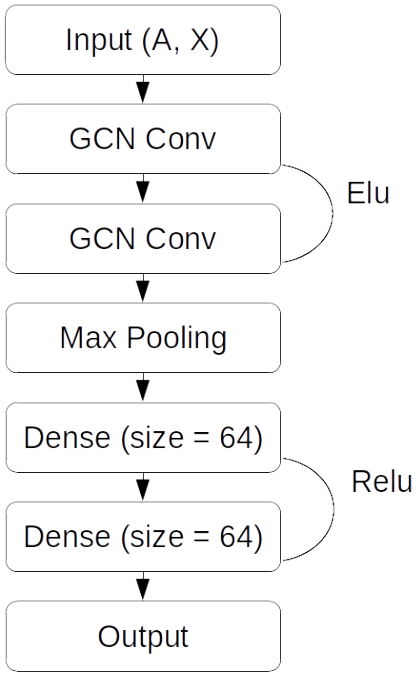


FIGURE 4.2 – Modèle de réseau de neurone pour graphe pour la prédiction de la survenue d'événements métastatiques suite à un cancer

Chapitre 5

Discussion et Ouverture

5.1 Résultats et performances des GNNs

Comme on a pu le constater, les réseaux de neurones pour graphe ne sembles pas donner des résultats satisfaisants pour les tâches demandées.

Néanmoins il faut garder à l'esprit que si les résultats peuvent sembler décevants pour la prédiction à partir du jeu de données DiOGene, le graphe utilisé n'était pas connu mais inféré et, il est possible que les performances aient été impactées à cause de cela. En effet, on voit que les performances semblent meilleures, en comparaison avec d'autres méthodes de machine learning, pour la prédiction à partir du jeu de donnée BreastCancer, pour lequel le graphe était connu puisque construit à partir d'un graphe d'interaction protéine-protéine (PPI) issu de la base de donnée Human Protein Reference Database (HPRD) [35]. Ainsi les différences de prédictions peuvent être expliquées par cette connaissance, basée sur d'autres études, et non uniquement sur une inférence à partir de nos données d'expression. En effet, il est probable que l'ajout de connaissances bruitées ou fausse au travers du graphe puisse desservir l'apprentissage. Une suite intéressante à ce travail, serait de comparer les résultats obtenus en utilisant soit le graphe PPI connu, soit un graphe inféré comme cela a été fait pour les données DiOGene, afin d'observer si il y a une différence dans la prédiction et s'il est possible d'utiliser efficacement l'inférence de graphe pour la prédiction de phénotype en se basant sur les données d'expression disponibles. Une autre cause possible reste le fait que si sur le papier les GNN semblent intéressant pour s'attaquer aux problèmes utilisant des données d'expression, ce n'est peut-être pas forcément le cas, ou des précautions spécifiques doivent être prises.

Par ailleurs l'analyse des résultats obtenus par Errica et al [30] montre que le design d'un réseau de neurones reste très spécifique à la tâche que l'on souhaite accomplir, et au vu du grand nombre de paramètres disponibles, il est donc possible que nous n'ayons pas utilisé les plus optimaux pour la tâche qui nous intéresse, et ce malgré le soin apporté pour les choisir. L'exploration des différentes dimensions accessibles pour le design d'un réseau de neurone pour graphe peut par exemple être testée à l'aide de l'implémentation *GraphGym* décrite dans le papier de You et al [29]. Elle propose par ailleurs la possibilité d'utiliser PyTorch Geometric mais nécessite des adaptations pour pouvoir utiliser des jeux de données qui ne sont pas proposés nativement avec la librairie.

5.2 Apprentissage simultané

Une autre piste d'amélioration de la capacité de prédiction de réseaux de neurones pour graphe est l'apprentissage simultané des paramètres du réseau de neurones et de la matrice d'adjacence du graphe. Ces nouvelles méthodes sont en train d'émerger et cherchent à améliorer la prédiction en partant du principe qu'il existe potentiellement des relations inconnues (la plupart du temps on va chercher à ajouter des relations plutôt qu'à en enlever) qui pourraient apporter de l'information et améliorer notre modèle pour améliorer ses performances de prédiction.

La mise en place de ce type d'approche passe souvent par l'ajout d'un terme à l'équation de la fonction de perte : $\mathcal{L} = \mathcal{L}_{graph}(A) + \mathcal{L}_{pred}(A, X)$. Il existe pour l'instant trois familles d'apprentissage simultané structure-paramètres [24] :

— celle basée sur l'apprentissage des paramètres d'une distribution des arêtes du graphe. Elle ne nous intéresse pas puisque nous cherchons à déterminer la matrice d'adjacence à partir des données que l'on connait, et ce genre de méthode n'apporte pas d'informations sur la relation entre deux nœuds mais au niveau des arêtes du graphe dans leur ensemble [36],

— celle basée sur l'apprentissage de paramètres à partir d'une métrique, qui vont eux-mêmes permettre de définir une nouvelle matrice d'adjacence [37],

— celle basée sur l'apprentissage direct d'une nouvelle matrice d'adjacence [38].

La troisième méthode semble la plus intéressante dans notre cas puisqu'elle permet d'apprendre une nouvelle matrice d'adjacence.

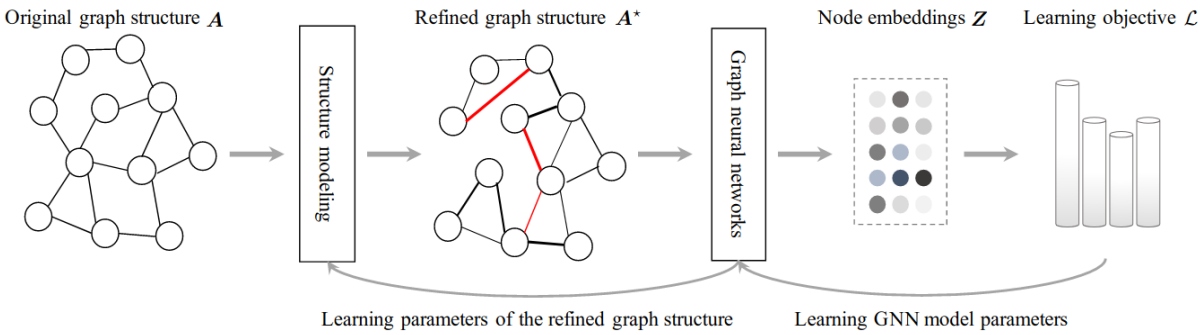


FIGURE 5.1 – Apprentissage simultané de la structure du graphe et des paramètres du réseau de neurones [24]

Comme on peut le voir sur la figure 5.1, en comparaison avec la figure 3.2, on a rajouté une étape où la structure de notre graphe, c'est à dire sa matrice d'adjacence, va être mise à jour et pourra être utilisée pour la prochaine passe. Ainsi on s'attend à ce que la prédiction s'améliore et à obtenir une nouvelle version de la matrice d'adjacence. Dans un contexte où l'on va chercher à comprendre le fonctionnement d'un système biologique en supplément de la prédiction que l'on a pu chercher à faire (prédiction de phénotype comme dans notre cas par exemple), l'étude de cette nouvelle matrice d'adjacence et la comparaison de cette structure du graphe avec celle d'origine présente un intérêt certain puisqu'il pourrait permettre de donner de nouvelles pistes de recherche sur l'interaction entre différents gènes ou fonctions biologiques.

Bibliographie

[1] Inrae2030, 2021.

[2] Franck Rapaport, Andrei Zinovyev, Marie Dutreix, Emmanuel Barillot, and Jean-Philippe Vert. Classification of microarray data using gene networks. *BMC Bioinformatics*, 8 :35, 2007.

[3] F. Scarselli, M. Gori, Ah Chung Tsoi, M. Hagenbuchner, and G. Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1) :61–80, January 2009.

[4] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks : A review of methods and applications. *AI Open*, 1 :57–81, 2020.

[5] Alyssa Imbert, Nathalie Vialaneix, Julien Marquis, Julie Vion, Aline Charpagne, Sylviane Metairon, Claire Laurens, Cédric Moro, Nathalie Boulet, Ondine Walter, Gregory Lefebvre, Jörg Hager, Dominique Langin, Wim H.M. Saris, Arne Astrup, Nathalie Viguerie, and Armand Valsesia. Network analyses reveal negative link between changes in adipose tissue GDF15 and BMI during dietary induced weight loss. Submitted (in revision for publication in The Journal of Clinical Endocrinology & Metabolism), 2021.

[6] Hryhorii Chereda, Annalen Bleckmann, Frank Kramer, Andreas Leha, and Tim Beissbarth. *Studies in Health Technology and Informatics*, page 281–286, Sep 2019.

[7] Thomas Meinert Larsen, Stine-Mathilde Dalskov, Marleen van Baak, Susan A. Jebb, Angeliki Papadaki, Andreas F.H. Pfeiffer, J. Alfredo Martinez, Teodora Handjieva-Darlenska, Marie Kunešová, Mats Pihlsgård, Steen Stender, Claus Holst, Wim H.M. Saris, and Arne Astrup. Diets with high or low protein content and glycemic index for weight-loss maintenance. *New England Journal of Medicine*, 363(22) :2102–2113, November 2010.

[8] M. Matsuda and R. A. DeFronzo. Insulin sensitivity indices obtained from oral glucose tolerance testing : comparison with the euglycemic insulin clamp. *Diabetes Care*, 22(9) :1462–1470, September 1999.

[9] P. A. P. Moran. Notes on continuous stochastic phenomena. *Biometrika*, 37(1/2) :17, 1950.

[10] A. Butte and I. Kohane. Mutual information relevance networks : functional genomic clustering using pairwise entropy measurements. In *Proceedings of the Pacific Symposium on Biocomputing*, pages 418–429, 2000.

[11] David Edwards. Introduction to graphical modelling. *Springer Texts in Statistics*, 1995.

[12] Peter Langfelder and Steve Horvath. WGCNA : an r package for weighted correlation network analysis. *BMC Bioinformatics*, 9(1), December 2008.

[13] Bin Zhang and Steve Horvath. A general framework for weighted gene co-expression network analysis. *Statistical Applications in Genetics and Molecular Biology*, 4(1), January 2005.

[14] J. Schafer and K. Strimmer. An empirical bayes approach to inferring large-scale gene association networks. *Bioinformatics*, 21(6) :754–764, 2005.

[15] Wessel N. van Wieringen and Carel F.W. Peeters. Ridge estimation of inverse covariance matrices from high-dimensional data. *Computational Statistics and Data Analysis*, 103 :284–303, Nov 2016.

[16] Wessel N. van Wieringen and Carel F.W. Peeters. Ridge estimation of inverse covariance matrices from high-dimensional data. *Computational Statistics & Data Analysis*, 103 :284–303, November 2016.

[17] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, 5(4) :115–133, December 1943.

[18] F. Rosenblatt. The perceptron : A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6) :386–408, 1958.

[19] Seppo Linnainmaa. Taylor expansion of the accumulated rounding error. *BIT*, 16(2) :146–160, Jun 1976.

[20] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088) :533–536, Oct 1986.

[21] Paul Werbos. *Beyond regression : new tools for prediction and analysis in the behavioral sciences*. PhD thesis, Harvard University, 1974.

[22] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4) :541–551, Dec 1989.

[23] Diederik P. Kingma and Jimmy Ba. Adam : A method for stochastic optimization, 2017.

[24] Yanqiao Zhu, Weizhi Xu, Jinghao Zhang, Qiang Liu, Shu Wu, and Liang Wang. Deep graph structure learning for robust representations : A survey. *CoRR*, abs/2103.03036, 2021.

[25] Bradley Efron. Local false discovery rates, 2005.

[26] Juliane Schäfer and Korbinian Strimmer. A shrinkage approach to large-scale covariance matrix estimation and implications for functional genomics. *Statistical Applications in Genetics and Molecular Biology*, 4(1), January 2005.

[27] Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[28] Daniele Grattarola and Cesare Alippi. Graph neural networks in tensorflow and keras with spektral, 2020.

[29] Jiaxuan You, Rex Ying, and Jure Leskovec. Design space for graph neural networks. *CoRR*, abs/2011.08843, 2020.

[30] Federico Errica, Marco Podda, Davide Bacciu, and Alessio Micheli. A fair comparison of graph neural networks for graph classification. *CoRR*, abs/1912.09893, 2019.

[31] Maurizio Ferrari Dacrema, Paolo Cremonesi, and Dietmar Jannach. Are we really making much progress ? A worrying analysis of recent neural recommendation approaches. *CoRR*, abs/1907.06902, 2019.

[32] Hryhorii Chereda, Annalen Bleckmann, Kerstin Menck, Júlia Perera-Bel, Philip Stegmaier, Florian Auer, Frank Kramer, Andreas Leha, and Tim Beißbarth. Explaining decisions of graph convolutional neural networks : patient-specific molecular subnetworks responsible for metastasis prediction in breast cancer. *Genome Medicine*, 13(1), March 2021.

[33] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn : Machine learning in Python. *Journal of Machine Learning Research*, 12 :2825–2830, 2011.

[34] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *CoRR*, abs/1609.02907, 2016.

[35] T. S. Keshava Prasad, R. Goel, K. Kandasamy, S. Keerthikumar, S. Kumar, S. Mathivanan, D. Telikicherla, R. Raju, B. Shafreen, A. Venugopal, L. Balakrishnan, A. Marimuthu, S. Banerjee, D. S. Somanathan, A. Sebastian, S. Rani, S. Ray, C. J. Harrys Kishore, S. Kanth, M. Ahmed, M. K. Kashyap, R. Mohmood, Y. L. Ramachandra, V. Krishna, B. A. Rahiman, S. Mohan, P. Ranganathan, S. Ramabadran, R. Chaerkady, and A. Pandey. Human protein reference database–2009 update. *Nucleic Acids Research*, 37(Database) :D767–D772, January 2009.

[36] Luca Franceschi, Mathias Niepert, Massimiliano Pontil, and Xiao He. Learning discrete structures for graph neural networks. *CoRR*, abs/1903.11960, 2019.

[37] Yu Chen, Lingfei Wu, and Mohammed J. Zaki. Iterative deep graph learning for graph neural networks : Better and robust node embeddings. *CoRR*, abs/2006.13009, 2020.

[38] Wei Jin, Yao Ma, Xiaorui Liu, Xianfeng Tang, Suhang Wang, and Jiliang Tang. Graph structure learning for robust graph neural networks. *CoRR*, abs/2005.10203, 2020.

Annexe A

Notations

La liste des différentes notations utilisées dans ce document est donnée ci-dessous :

— n : nombre de caractéristiques d'un nœud (par exemple le nombre de patients)
— p : nombre de nœuds du graphe (par exemple, le nombre de gènes)
— A : matrice d'adjacence du graphe
— X : matrice des attributs des nœuds du graphe
— H : vecteur des embedding des noeuds
— D : matrice diagonale symétrique des degrés du graphe
— L : Laplacien du graphe
— L' : Laplacien normalisé du graphe
— λ : hyper-paramètre pour une régression ℓ_2
— σ : fonction d'activation
— ω : poids du réseau de neurones
— ω_0 : biais du réseau de neurones
— l : étape (ou epoch)
— η : taux d'apprentissage
— m : taille des mini-batchs

Annexe B

Scripts

B.1 Scripts R

B.1.1 Première approche exploratoire des données DiOGenes

First look

Guilhem Huau

2021/08/24

Contents

Packages / R-options:

```
suppressPackageStartupMessages({
  library(caret) #Model training and testing
  library(doParallel) #Parallel computing
  library(plotly)
  library(ggplot2)
  library(gridExtra)
  library(ggpubr)
})

#  R options
options(max.print = 100)
```

Data summary

```
design <-
  read.table("../../data/Diogenes/prepared/design_cid1.csv", header = TRUE)
pseudoCounts <-
  t(read.table(
    "../../data/Diogenes/prepared/pseudoCounts_cid1.csv",
    header = TRUE
```

```
  ))
selclinical <-
  read.table("../../data/Diogenes/prepared/selclinical_cid1.csv",
             header = TRUE)
```

Design matrix

Information on each subject: - id - sample number - Plate number - Gender - Age - Country

```
design[, c(3:5, 7)] <-  lapply(design[, c(3:5, 7)], as.factor)
summary(design)
str(design)
```

```
##       id               sample           group         Plate     gender
##  Length:416         Length:416         cid1:416   P05    : 48   F:291
##  Class :character   Class :character              P04    : 43   M:125
##  Mode  :character   Mode  :character              P01    : 42
##                                                  P06    : 42
##                                                  P11    : 42
##                                                  P07    : 40
##                                                  (Other):159
##       age           country
##  Min.   :24.00   D       :83
##  1st Qu.:37.00   CZ      :67
##  Median :41.00   DK      :66
##  Mean   :41.01   GR      :62
##  3rd Qu.:45.00   SP      :56
##  Max.   :63.00   BG      :39
##                  (Other):43
## 'data.frame':    416 obs. of  7 variables:
##  $ id     : chr  "1_015_1_1" "1_029_2_1" "1_035_2_1" "1_056_2_1" ...
##  $ sample : chr  "1_015_1" "1_029_2" "1_035_2" "1_056_2" ...
##  $ group  : Factor w/ 1 level "cid1": 1 1 1 1 1 1 1 1 1 1 ...
##  $ Plate  : Factor w/ 11 levels "P01","P02","P03",..: 2 2 10 7 5 5 4 4 9 6 ...
##  $ gender : Factor w/ 2 levels "F","M": 2 1 1 1 2 1 1 1 1 1 ...
##  $ age    : int  56 33 31 40 41 38 49 36 34 38 ...
##  $ country: Factor w/ 8 levels "BG","CZ","D",..: 6 6 6 6 6 6 6 6 6 6 ...
```

Pseudo-Counts

```
dim(pseudoCounts)
head(pseudoCounts[, 1:5])
```

```
## [1] 416 887
##            ENSG00000000005.5 ENSG00000003249.13 ENSG00000004838.13
## X1_015_1_1          9.581878         1.20761225         0.01860643
## X1_029_2_1          7.324382         0.08872925         0.55337698
## X1_035_2_1          9.741932         0.25404600         0.25404600
## X1_056_2_1          9.486271         1.77105270         0.25850437
## X1_066_1_1          8.548115         1.53885262         0.74972339
## X1_066_2_1          9.632010         1.39864943        -0.51571703
##            ENSG00000004939.13 ENSG00000005884.17
## X1_015_1_1           3.166334           1.774375
## X1_029_2_1           3.315171           2.612230
```

```
## X1_035_2_1           3.886987           1.413066
## X1_056_2_1           5.469251           3.465797
## X1_066_1_1           5.403393           2.282230
## X1_066_2_1           4.693864           2.114870
```

Clinical results

A difference between the results at CID1 (begining of the experiment) and CID3 (after the diet and treatment) of two indices: BMI and Matsuda index

```
summary(selclinical)
```

```
##      BMI31             dmatsu31
##  Min.   :-13.388   Min.   :-7.7532
##  1st Qu.: -4.948   1st Qu.: 0.3335
##  Median : -3.441   Median : 1.9776
##  Mean   : -3.789   Mean   : 2.0472
##  3rd Qu.: -2.148   3rd Qu.: 3.5067
##  Max.   :  3.718   Max.   :13.7184
##  NA's   :167       NA's   :215
```

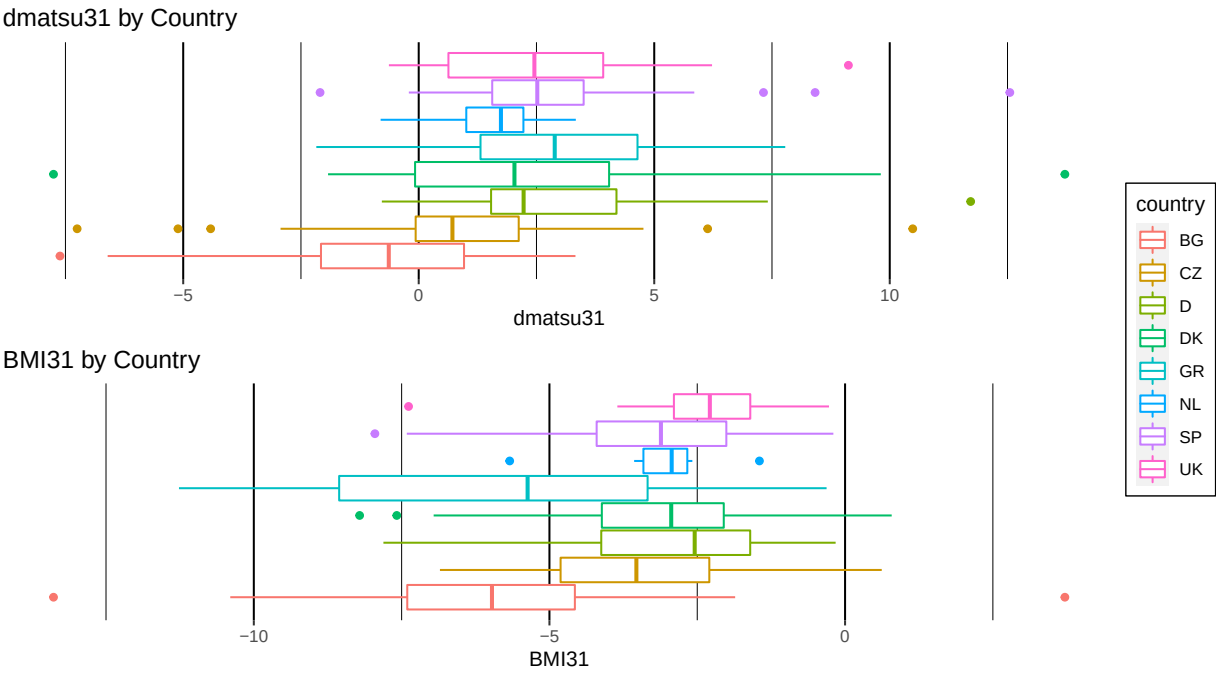
```
dat <- cbind(design, selclinical)
dat <- dat[order(dat$BMI31),]
dat <- na.omit(dat)
```

Visualisation

By country

```
plt_matsu_country <- ggplot(data = dat) +
  geom_boxplot(aes(x = dmatsu31, color = country)) +
  labs(title = "dmatsu31 by Country") +
  theme(
    legend.background = element_blank(),
    legend.box.background = element_rect(colour = "black"),
    axis.title.y=element_blank(),
    axis.text.y=element_blank(),
    axis.ticks.y=element_blank(),
    panel.grid.major.y =element_blank(),
    panel.grid.minor.y =element_blank(),
    panel.grid.minor.x =element_line(color = "black"),
    panel.grid.major.x =element_line(color = "black"),
    panel.background = element_rect(fill="white")
  )

plt_bmi_country <- ggplot(data = dat) +
  geom_boxplot(aes(x = BMI31, color = country)) +
  labs(title = "BMI31 by Country") +
  theme(
    legend.background = element_blank(),
    legend.box.background = element_rect(colour = "black"),
    axis.title.y=element_blank(),
    axis.text.y=element_blank(),
    axis.ticks.y=element_blank(),
```

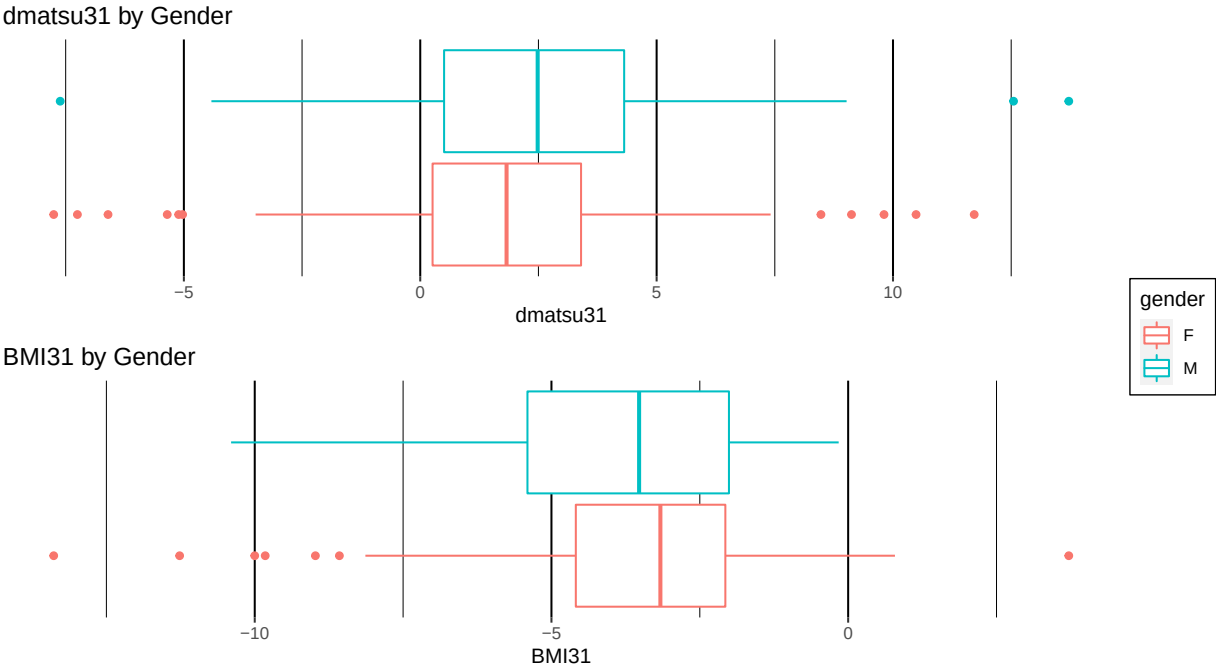
```
    panel.grid.major.y =element_blank(),
    panel.grid.minor.y =element_blank(),
    panel.grid.minor.x =element_line(color = "black"),
    panel.grid.major.x =element_line(color = "black"),
    panel.background = element_rect(fill="white")
  )

ggarrange(plt_matsu_country, plt_bmi_country, nrow = 2, common.legend = TRUE, legend = "right")
```



By gender

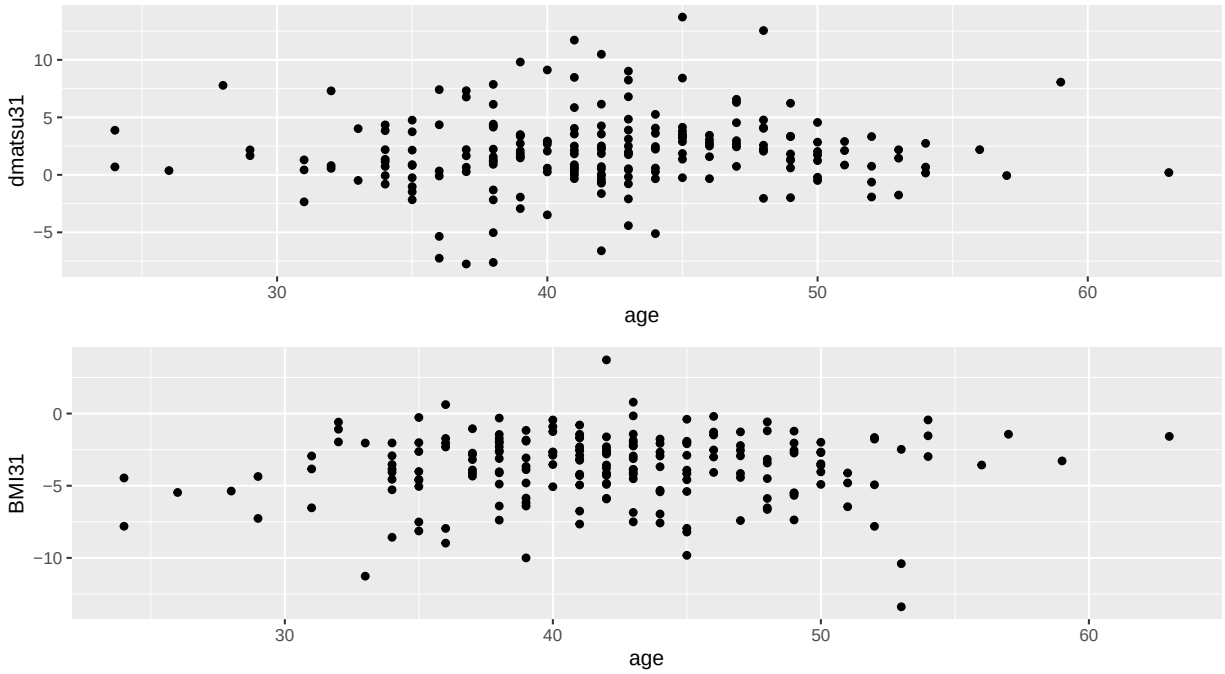
```
plt_matsu_gender <- ggplot(data = dat) +
  geom_boxplot(aes(x = dmatsu31, color = gender)) +
  labs(title = "dmatsu31 by Gender") +
  theme(
    legend.background = element_blank(),
    legend.box.background = element_rect(colour = "black"),
    axis.title.y=element_blank(),
    axis.text.y=element_blank(),
    axis.ticks.y=element_blank(),
    panel.grid.major.y =element_blank(),
    panel.grid.minor.y =element_blank(),
    panel.grid.minor.x =element_line(color = "black"),
    panel.grid.major.x =element_line(color = "black"),
    panel.background = element_rect(fill="white")
  )

plt_bmi_gender <- ggplot(data = dat) +
  geom_boxplot(aes(x = BMI31, color = gender)) +
  labs(title = "BMI31 by Gender") +
```

```
  theme(
    legend.background = element_blank(),
    legend.box.background = element_rect(colour = "black"),
    axis.title.y=element_blank(),
    axis.text.y=element_blank(),
    axis.ticks.y=element_blank(),
    panel.grid.major.y =element_blank(),
    panel.grid.minor.y =element_blank(),
    panel.grid.minor.x =element_line(color = "black"),
    panel.grid.major.x =element_line(color = "black"),
    panel.background = element_rect(fill="white")
  )

ggarrange(plt_matsu_gender, plt_bmi_gender, nrow = 2, common.legend = TRUE, legend = "right")
```



By age

```
r2 <- round(summary(lm(data = dat, formula = dmatsu31 ~ age))$r.squared, 5)
plt_matsu_age <- ggplot(data = dat, aes(x = age, y = dmatsu31)) +
  geom_point()

r2 <- round(summary(lm(data = dat, formula = BMI31 ~ age))$r.squared, 5)
plt_bmi_age <- ggplot(data = dat, aes(x = age, y = BMI31)) +
  geom_point()

grid.arrange(plt_matsu_age, plt_bmi_age)
```



Predicting Body Mass Index

```
dat <- data.frame(cbind(selclinical$BMI31, pseudoCounts))
names(dat)[1] <- "BMI31"
dat <- na.omit(dat)
```

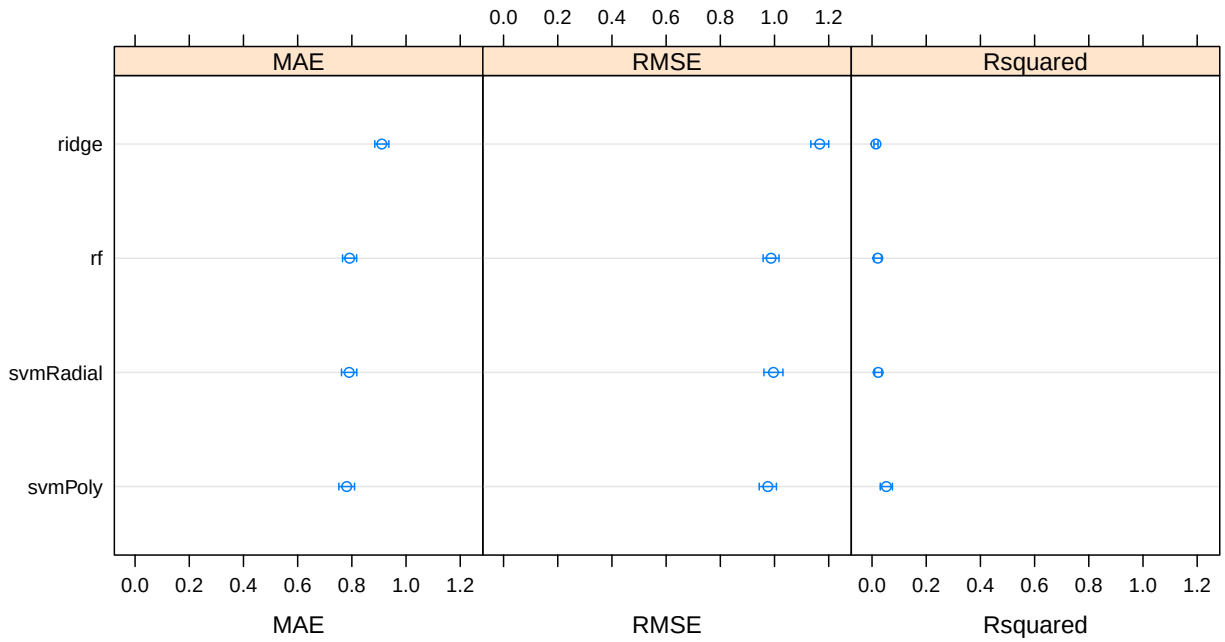
```
set.seed(42)

inTrain <- createDataPartition(y = dat$BMI31,
                               ## the outcome data are needed
                               p = .75,
                               ## The percentage of data in the
                               ## training set
                               list = FALSE)

training <- dat[inTrain, ]
validating  <- dat[-inTrain, ]

preProcValues <- preProcess(training, method = c("center", "scale"))

training <- predict(preProcValues, training)
validating <- predict(preProcValues, validating)
```

```
set.seed(42)
trainCtrl <- trainControl(method = "repeatedcv", number = 10, repeats = 3)
if (dir.exists("01_DiogeneFirstLook_files/model/")==FALSE) {
  dir.create("01_DiogeneFirstLook_files/model/")
}
# Parallel computing for the training
cl <- makePSOCKcluster(6)
registerDoParallel(cl)
```

```r
set.seed(42)
mod_ridge <- train(BMI31 ~ ., data = training, method="ridge")
saveRDS(mod_ridge, "01_DiogeneFirstLook_files/model/model_ridge.rds")
set.seed(42)
mod_rf <- train(BMI31 ~ ., data = training, method="rf")
saveRDS(mod_rf, "01_DiogeneFirstLook_files/model/model_rf.rds")
set.seed(42)
mod_svmRadial <- train(BMI31 ~ ., data = training, method="svmRadial")
saveRDS(mod_svmRadial, "01_DiogeneFirstLook_files/model/model_svmRadial.rds")
set.seed(42)
mod_svmPoly <- train(BMI31 ~ ., data = training, method="svmPoly")
saveRDS(mod_svmPoly, "01_DiogeneFirstLook_files/model/model_svmPoly.rds")

## When you are done with parallel computing
stopCluster(cl)
```

```r
mod_ridge <- readRDS("01_DiogeneFirstLook_files/model/model_ridge.rds")
mod_rf <- readRDS("01_DiogeneFirstLook_files/model/model_rf.rds")
mod_svmRadial <- readRDS("01_DiogeneFirstLook_files/model/model_svmRadial.rds")
mod_svmPoly <- readRDS("01_DiogeneFirstLook_files/model/model_svmPoly.rds")
```

```r
results <- resamples(list(
  ridge = mod_ridge,
  rf = mod_rf,
  svmRadial = mod_svmRadial,
  svmPoly = mod_svmPoly
))

summary(results)

dotplot(results)
```

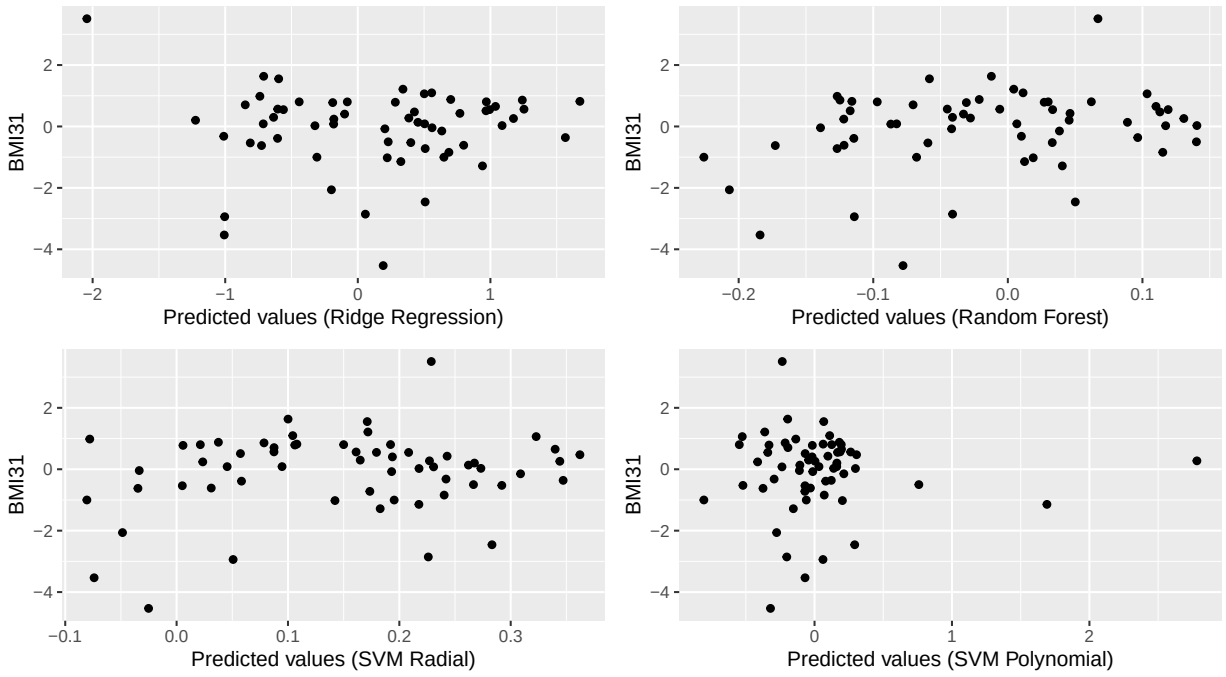


Confidence Level: 0.95

```
##
## Call:
## summary.resamples(object = results)
##
## Models: ridge, rf, svmRadial, svmPoly
## Number of resamples: 25
##
## MAE
##                Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## ridge     0.7686114 0.8712344 0.9148929 0.9106482 0.9664505 1.0264346    0
## rf        0.6331508 0.7548395 0.8007005 0.7916224 0.8290425 0.8991986    0
## svmRadial 0.5940179 0.7590721 0.7954850 0.7899890 0.8164408 0.9308720    0
## svmPoly   0.6099903 0.7279021 0.7981807 0.7809867 0.8197631 0.9235611    0
##
## RMSE
##                Min.   1st Qu.    Median      Mean  3rd Qu.     Max. NA's
## ridge     0.9655802 1.1206690 1.1658898 1.1669421 1.225465 1.322351    0
## rf        0.8011980 0.9563927 0.9884937 0.9872965 1.026637 1.095406    0
## svmRadial 0.7451222 0.9622136 0.9842073 0.9960360 1.028745 1.149176    0
## svmPoly   0.7691215 0.9379323 0.9904048 0.9752879 1.027371 1.118048    0
##
## Rsquared
##                   Min.     1st Qu.      Median       Mean    3rd Qu.       Max.
## ridge     8.492454e-06 0.002149662 0.007400924 0.01446433 0.01839386 0.06734633
## rf        3.589224e-06 0.001293066 0.004616721 0.02113145 0.02446626 0.14077918
## svmRadial 6.346792e-06 0.001106852 0.007503067 0.02264505 0.02453201 0.11022904
## svmPoly   1.358536e-03 0.012508350 0.034021275 0.05272073 0.05830120 0.18401623
##           NA's
## ridge        0
## rf           0
## svmRadial    0
```

```
## svmPoly      0
```

```
valid_ridge <- predict(mod_ridge, validating)
valid_rf <- predict(mod_rf, validating)
valid_svmRadial <- predict(mod_svmRadial, validating)
valid_svmPoly <- predict(mod_svmPoly, validating)
dat_valid <-data.frame(valid_ridge, valid_rf, valid_svmRadial, valid_svmPoly, validating$BMI31)

plt_ridge <- ggplot(data = dat_valid, aes(x = valid_ridge, y = validating.BMI31)) +
  geom_point() +
  xlab("Predicted values (Ridge Regression)") +
  ylab("BMI31")
plt_rf <- ggplot(data = dat_valid, aes(x = valid_rf, y = validating.BMI31)) +
  geom_point() +
  xlab("Predicted values (Random Forest)") +
  ylab("BMI31")
plt_svmRadial <- ggplot(data = dat_valid, aes(x = valid_svmRadial, y = validating.BMI31)) +
  geom_point() +
  xlab("Predicted values (SVM Radial)") +
  ylab("BMI31")
plt_svmPoly <- ggplot(data = dat_valid, aes(x = valid_svmPoly, y = validating.BMI31)) +
  geom_point() +
  xlab("Predicted values (SVM Polynomial)") +
  ylab("BMI31")

grid.arrange(plt_ridge, plt_rf, plt_svmRadial, plt_svmPoly, ncol = 2, nrow = 2)
```



Predictind Matsuda Index

```
dat <- data.frame(cbind(selclinical$dmatsu31, pseudoCounts))
names(dat)[1] <- "dmatsu31"
dat <- na.omit(dat)
```

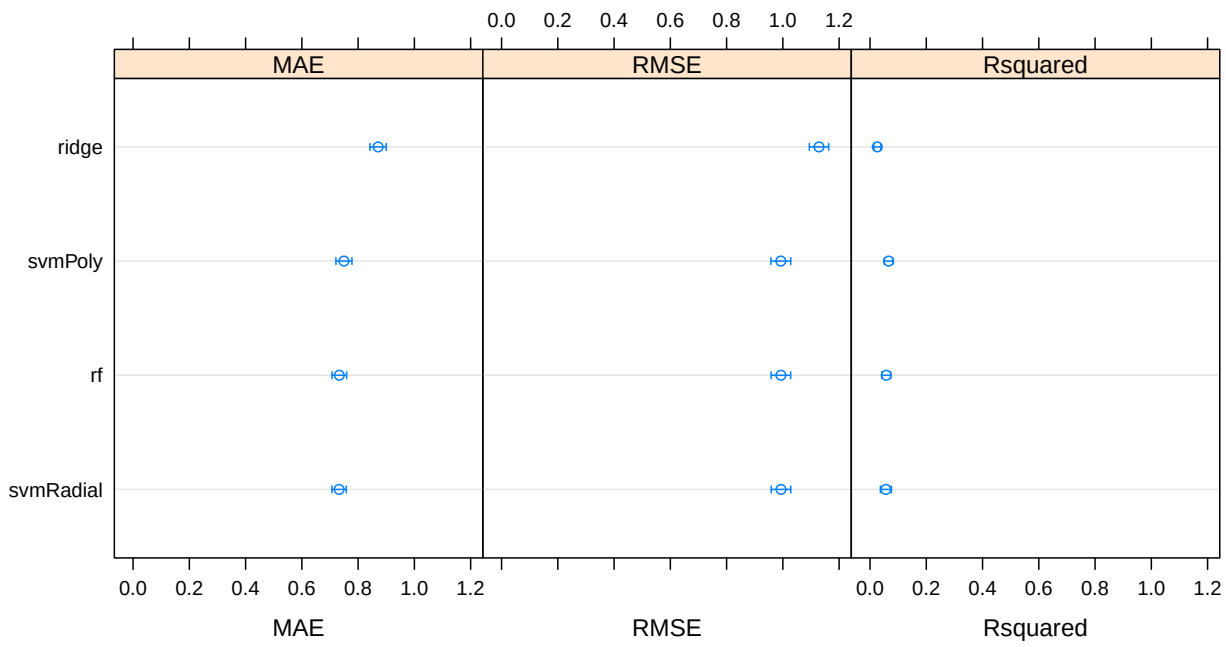
```
set.seed(42)

inTrain <- createDataPartition(y = dat$dmatsu31,
                               ## the outcome data are needed
                               p = .75,
                               ## The percentage of data in the
                               ## training set
                               list = FALSE)

training <- dat[inTrain, ]
validating  <- dat[-inTrain, ]

preProcValues <- preProcess(training, method = c("center", "scale"))

training <- predict(preProcValues, training)
validating <- predict(preProcValues, validating)
```

```
set.seed(42)
trainCtrl <- trainControl(method = "repeatedcv", number = 10, repeats = 3)
# Parallel computing for the training
cl <- makePSOCKcluster(5)
registerDoParallel(cl)

set.seed(42)
mod_ridge <- train(dmatsu31 ~ ., data = training, method="ridge")
saveRDS(mod_ridge, "01_DiogeneFirstLook_files/model/model_ridge.rds")
set.seed(42)
mod_rf <- train(dmatsu31 ~ ., data = training, method="rf")
saveRDS(mod_rf, "01_DiogeneFirstLook_files/model/model_rf.rds")
set.seed(42)
mod_svmRadial <- train(dmatsu31 ~ ., data = training, method="svmRadial")
saveRDS(mod_svmRadial, "01_DiogeneFirstLook_files/model/model_svmRadial.rds")
set.seed(42)
mod_svmPoly <- train(dmatsu31 ~ ., data = training, method="svmPoly")
saveRDS(mod_svmPoly, "01_DiogeneFirstLook_files/model/model_svmPoly.rds")

## When you are done with parallel computing
stopCluster(cl)
```

```
mod_ridge <- readRDS("01_DiogeneFirstLook_files/model/model_ridge.rds")
mod_rf <- readRDS("01_DiogeneFirstLook_files/model/model_rf.rds")
mod_svmRadial <- readRDS("01_DiogeneFirstLook_files/model/model_svmRadial.rds")
mod_svmPoly <- readRDS("01_DiogeneFirstLook_files/model/model_svmPoly.rds")
```

```
results <- resamples(list(
  ridge = mod_ridge,
  rf = mod_rf,
  svmRadial = mod_svmRadial,
  svmPoly = mod_svmPoly
```

```
))

summary(results)

dotplot(results)
```

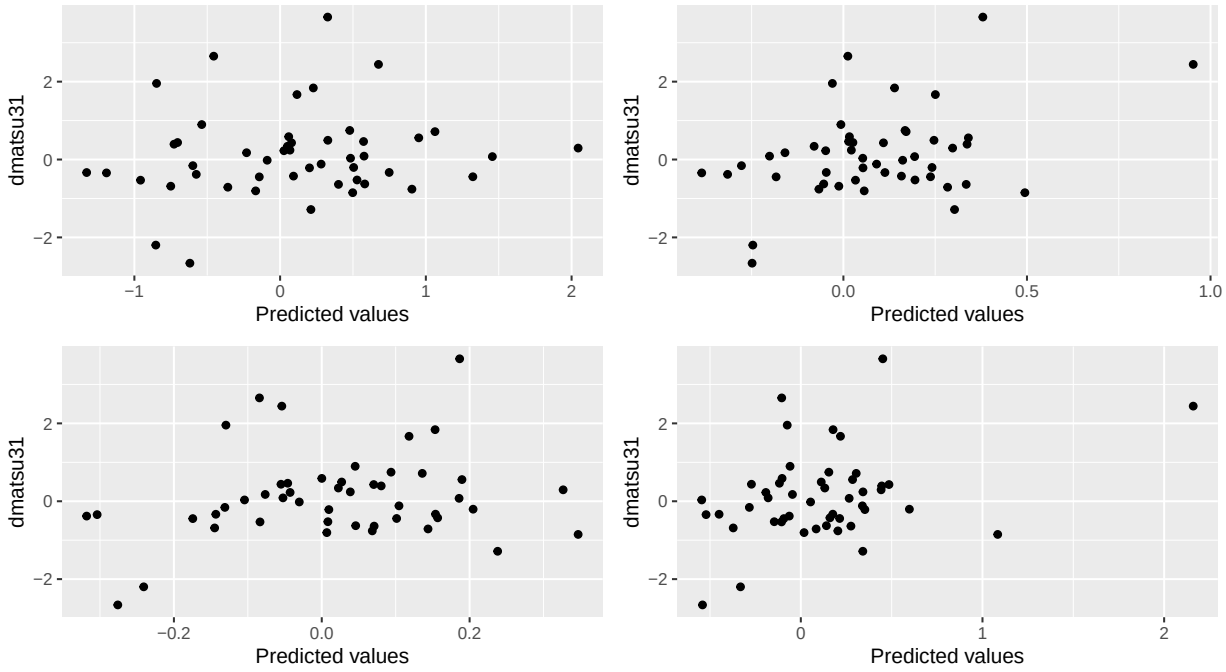


Confidence Level: 0.95

```
##
## Call:
## summary.resamples(object = results)
##
## Models: ridge, rf, svmRadial, svmPoly
## Number of resamples: 25
##
## MAE
##                Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## ridge     0.7416695 0.8207655 0.8619405 0.8710201 0.9119239 1.0865337    0
## rf        0.6131019 0.6884271 0.7252369 0.7331469 0.7700268 0.8421051    0
## svmRadial 0.6159221 0.6972006 0.7364521 0.7323974 0.7736860 0.8373714    0
## svmPoly   0.6361377 0.7044172 0.7443595 0.7496252 0.7865431 0.8786349    0
##
## RMSE
##                Min.   1st Qu.    Median      Mean  3rd Qu.     Max. NA's
## ridge     0.9807176 1.0854399 1.1173667 1.1285221 1.172047 1.357634    0
## rf        0.8035088 0.9365638 0.9882997 0.9930590 1.060051 1.137505    0
## svmRadial 0.8143972 0.9523530 0.9834295 0.9934545 1.069010 1.138871    0
## svmPoly   0.8479492 0.9322995 0.9806108 0.9927931 1.076132 1.154542    0
##
## Rsquared
##                   Min.     1st Qu.     Median       Mean    3rd Qu.      Max.
## ridge     4.992102e-05 0.008139674 0.01261229 0.02580431 0.03244780 0.1144048
```

```
## rf        1.768311e-03 0.030447334 0.04843092 0.05771267 0.07411269 0.1459651
## svmRadial 1.341462e-03 0.017702242 0.04859084 0.05637589 0.07563065 0.1975095
## svmPoly   2.792796e-04 0.037131908 0.07104220 0.06604232 0.09023189 0.1526728
##           NA's
## ridge        0
## rf           0
## svmRadial    0
## svmPoly      0
```

```
valid_ridge <- predict(mod_ridge, validating)
valid_rf <- predict(mod_rf, validating)
valid_svmRadial <- predict(mod_svmRadial, validating)
valid_svmPoly <- predict(mod_svmPoly, validating)
dat_valid <-data.frame(valid_ridge, valid_rf, valid_svmRadial, valid_svmPoly, validating$dmatsu31)

plt_ridge <- ggplot(data = dat_valid, aes(x = valid_ridge, y = validating.dmatsu31)) +
  geom_point() +
  xlab("Predicted values") +
  ylab("dmatsu31")
plt_rf <- ggplot(data = dat_valid, aes(x = valid_rf, y = validating.dmatsu31)) +
  geom_point() +
  xlab("Predicted values") +
  ylab("dmatsu31")
plt_svmRadial <- ggplot(data = dat_valid, aes(x = valid_svmRadial, y = validating.dmatsu31)) +
  geom_point() +
  xlab("Predicted values") +
  ylab("dmatsu31")
plt_svmPoly <- ggplot(data = dat_valid, aes(x = valid_svmPoly, y = validating.dmatsu31)) +
  geom_point() +
  xlab("Predicted values") +
  ylab("dmatsu31")

grid.arrange(plt_ridge, plt_rf, plt_svmRadial, plt_svmPoly, ncol = 2, nrow = 2)
```



Appendix

```
## Document generated in:
## Time difference of 11.86212 mins
##
## CPU: Intel(R) Core(TM) i7-10610U CPU @ 1.80GHz
## Memory total size: 32.54956 GB
##
##
## Session information:
## R version 4.1.1 (2021-08-10)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 20.04.2 LTS
##
## Matrix products: default
## BLAS:   /usr/lib/x86_64-linux-gnu/blas/libblas.so.3.9.0
## LAPACK: /usr/lib/x86_64-linux-gnu/lapack/liblapack.so.3.9.0
##
## attached base packages:
## [1] parallel  stats     graphics  grDevices utils     datasets  methods
## [8] base
##
## other attached packages:
## [1] ggpubr_0.4.0      gridExtra_2.3     plotly_4.9.4.1    doParallel_1.0.16
## [5] iterators_1.0.13  foreach_1.5.1     caret_6.0-88      ggplot2_3.3.5
## [9] lattice_0.20-44
##
## loaded via a namespace (and not attached):
##  [1] httr_1.4.2           tidyr_1.1.3          jsonlite_1.7.2
##  [4] viridisLite_0.4.0    splines_4.1.1        carData_3.0-4
##  [7] elasticnet_1.3       prodlim_2019.11.13   stats4_4.1.1
```

```
## [10] cellranger_1.1.0     yaml_2.2.1           ipred_0.9-11
## [13] pillar_1.6.2         backports_1.2.1      glue_1.4.2
## [16] pROC_1.17.0.1        digest_0.6.27        ggsignif_0.6.2
## [19] randomForest_4.6-14  colorspace_2.0-2     recipes_0.1.16
## [22] cowplot_1.1.1        htmltools_0.5.1.1    Matrix_1.3-4
## [25] plyr_1.8.6           timeDate_3043.102    pkgconfig_2.0.3
## [28] broom_0.7.9          haven_2.4.3          purrr_0.3.4
## [31] scales_1.1.1         openxlsx_4.2.4       gower_0.2.2
## [34] lava_1.6.9           rio_0.5.27           tibble_3.1.3
## [37] farver_2.1.0         generics_0.1.0       car_3.0-11
## [40] ellipsis_0.3.2       withr_2.4.2          nnet_7.3-16
## [43] lazyeval_0.2.2       readxl_1.3.1         survival_3.2-11
## [46] magrittr_2.0.1       crayon_1.4.1         evaluate_0.14
## [49] fansi_0.5.0          nlme_3.1-152         MASS_7.3-54
## [52] rstatix_0.7.0        forcats_0.5.1        foreign_0.8-81
## [55] class_7.3-19         tools_4.1.1          data.table_1.14.0
## [58] hms_1.1.0            lifecycle_1.0.0      stringr_1.4.0
## [61] kernlab_0.9-29       munsell_0.5.0        zip_2.2.0
## [64] compiler_4.1.1       rlang_0.4.11         grid_4.1.1
## [67] htmlwidgets_1.5.3    labeling_0.4.2       rmarkdown_2.10
## [70] lars_1.2             gtable_0.3.0         ModelMetrics_1.2.2.2
## [73] codetools_0.2-18     abind_1.4-5          curl_4.3.2
## [76] reshape2_1.4.4       R6_2.5.0             lubridate_1.7.10
## [79] knitr_1.33           dplyr_1.0.7          utf8_1.2.2
## [82] stringi_1.7.3        Rcpp_1.0.7           vctrs_0.3.8
## [85] rpart_4.1-15         tidyselect_1.1.1     xfun_0.24
```

B.1.2 Inférence des graphes pour les données DiOGenes

Comparison of the GeneNet, WGCNA and rags2ridges R packages for edges detection

Guilhem Huau

2021/08/24

Contents

Packages / R-options:

```
suppressPackageStartupMessages({
  library(GeneNet)
  library(graph)
  library(Rgraphviz)
  library(WGCNA)
  library(rags2ridges)
  library(igraph)
  library(ggplot2)
  library(spdep)
})

#  R options
options(max.print = 100)
```

```
pseudoCounts <-
  scale(t(read.table("../../data/Diogenes/prepared/pseudoCounts_cid1.csv",
                     header = TRUE)))
clinical <- scale(
  read.table("../../data/Diogenes/prepared/selclinical_cid1.csv",
```

```
             header = TRUE))
rownames(clinical) <- rownames(pseudoCounts)

if (dir.exists("02_GraphInference/src/")==FALSE) {
  dir.create("02_GraphInference/")
  dir.create("02_GraphInference/src/")
}
```

Package GeneNet

The edges are based on partial correlation. Inspired from the A. thaliana example of the package GeneNet.

```
set.seed(42)
# Compute Partial Correlations and Select Relevant Edges
pcor.stat = ggm.estimate.pcor(pseudoCounts, method = "static")
grph.edges = network.test.edges(pcor.stat,direct=TRUE)
dim(grph.edges)

# We extract only the 150 edges with the highest probability
grph.net = extract.network(grph.edges, method.ggm="number", cutoff.ggm=150)

# We create a graph object
# We drop the nodes that are not connected by our 150 edges)
node.labels = as.character(1:ncol(pseudoCounts))
grph = network.make.graph(grph.net, node.labels, drop.singles=TRUE)


# For a more beautiful plot of the network set node and edge parameters:
# Set global node and edge attributes:
globalAttrs = list()
globalAttrs$edge = list(color = "black", lty = "solid", lwd = 1, arrowsize=1)
globalAttrs$node = list(fillcolor = gray(.95), shape = "ellipse", fixedsize = FALSE)
# Set edge attributes:
edi = edge.info(grph) # get edge directions and correlations
edgeAttrs = list()
edgeAttrs$dir =  edi$dir # set edge directions
cutoff = quantile(abs(edi$weight), c(0.2, 0.8)) # thresholds for line width / coloring
edgeAttrs$lty = ifelse(edi$weight < 0, "dotted", "solid") # negative correlation
edgeAttrs$color = ifelse( abs(edi$weight <= cutoff[1]), "grey", "black") # lower 20% quantile
edgeAttrs$lwd = ifelse(abs(edi$weight >= cutoff[2]), 2, 1) # upper 20% quantile

grph <- plot(grph, attrs = globalAttrs, edgeAttrs = edgeAttrs, "fdp")
toFile(graph = grph, filename = '02_GraphInference/src/grph.svg', layoutType = 'fdp', fileType = 'svg')
```

```
## Estimating optimal shrinkage intensity lambda (correlation matrix): 0.0632
##
## Estimate (local) false discovery rates (partial correlations):
## Step 1... determine cutoff point
## Step 2... estimate parameters of null distribution and eta0
## Step 3... compute p-values and estimate empirical PDF/CDF
## Step 4... compute q-values and local fdr
## Step 5... prepare for plotting
##
```

```
## Estimate (local) false discovery rates (log ratio of spvars):
## Step 1... determine cutoff point
## Step 2... estimate parameters of null distribution and eta0
## Step 3... compute p-values and estimate empirical PDF/CDF
## Step 4... compute q-values and local fdr
## Step 5... prepare for plotting
##
## [1] 392941     10
##
## Significant edges:  150
##     Corresponding to  0.04 %  of possible edges
##
## Significant directions:  0
##     Corresponding to  0 %  of possible directions
## Significant directions in the network:  0
##     Corresponding to  0 %  of possible directions in the network
## NULL
```

Package WGCNA

The WGCNA package use **weighted correlation network analysis**. It first build an similarity matrix (level of concordance between the genes), and transform it into an adjacency matrix that express the edges of the network. But contrary to other packages, the adjacency matrix is not composed of 0 and 1 but instead of a weight $a_{ij} \in [0, 1]$. Then this adjacency matrix is used to compute a distance between nodes to perform some clustering and discover "modules" (cluster of nodes). We can then analyse the connection between nodes or modules and external traits.

Choosing a similarity function among power-law distributions

```
dat_WGCNA <- pseudoCounts

# Choose a set of soft-thresholding powers
powers = c(seq(1, 10, by = 1), seq(12, 20, by = 2))
# Call the network topology analysis function
sft = pickSoftThreshold(data = dat_WGCNA, powerVector = powers, verbose = 5, moreNetworkConcepts = TRUE
```

```
## Warning: executing %dopar% sequentially: no parallel backend registered
```

```
# Plot the results:
cex1 = 0.9
par(mfrow = c(1,2))
# Scale-free topology fit index as a function of the soft-thresholding power
plot(sft$fitIndices[,1], -sign(sft$fitIndices[,3])*sft$fitIndices[,2],
     xlab="Soft Threshold (power)",ylab="Scale Free Topology Model Fit,signed R^2",
     type="n",main = paste("Scale independence"))
text(sft$fitIndices[,1], -sign(sft$fitIndices[,3])*sft$fitIndices[,2],labels=powers,cex=cex1,col="red")
# this line corresponds to using an R^2 cut-off of h
abline(h=0.85,col="red")
# Mean connectivity as a function of the soft-thresholding power
plot(sft$fitIndices[,1], sft$fitIndices[,5],
     xlab="Soft Threshold (power)",ylab="Mean Connectivity",
     type="n",main = paste("Mean connectivity"))
text(sft$fitIndices[,1], sft$fitIndices[,5], labels=powers, cex=cex1,col="red")
```

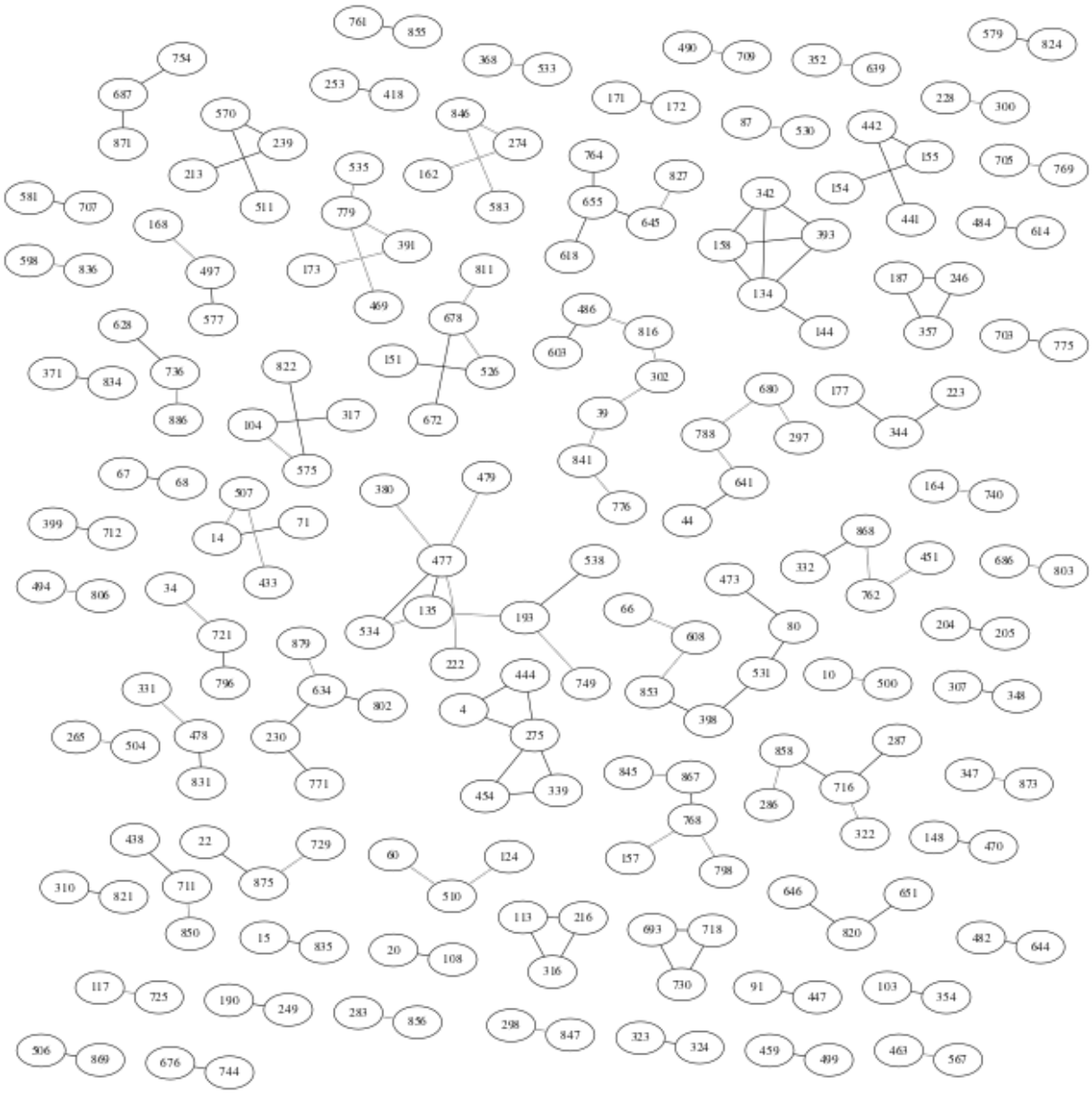
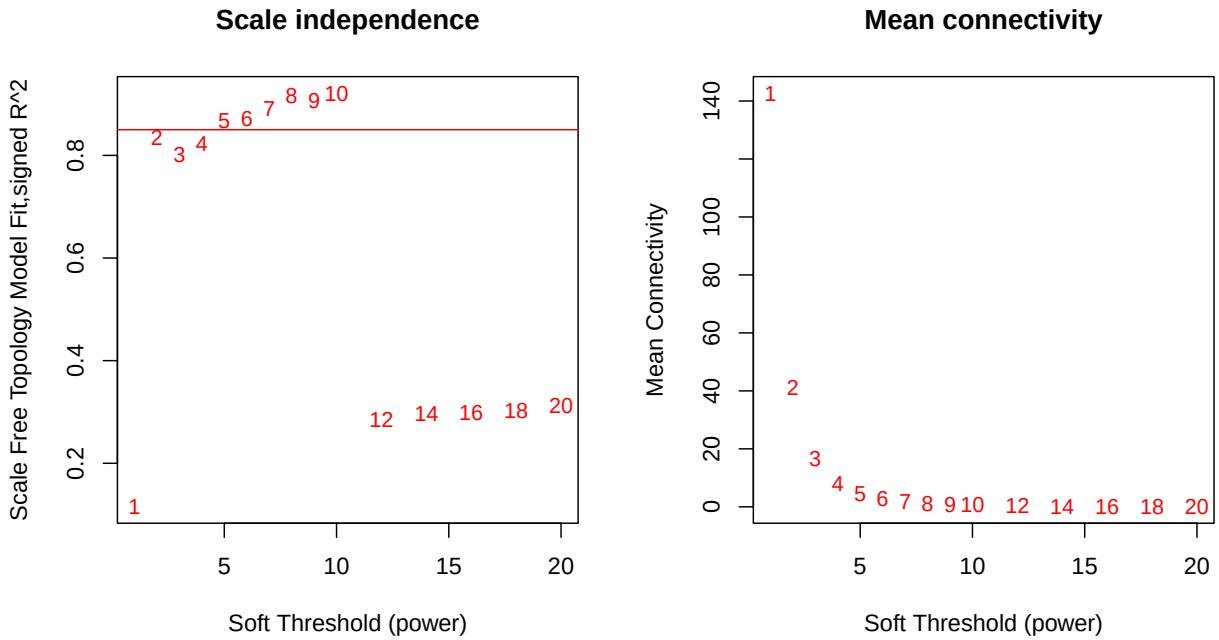


Figure 1: graph




```
## pickSoftThreshold: will use block size 887.
##  pickSoftThreshold: calculating connectivity for given powers...
##    ..working on genes 1 through 887 of 887
##    Power SFT.R.sq  slope truncated.R.sq mean.k. median.k. max.k.  Density
## 1      1    0.115 -0.328          0.838 143.000  133.0000  274.0 0.161000
## 2      2    0.836 -1.080          0.964  41.300   30.8000  139.0 0.046600
## 3      3    0.801 -1.370          0.864  16.500    9.3500   86.4 0.018600
## 4      4    0.824 -1.470          0.883   8.030    3.3400   59.5 0.009060
## 5      5    0.868 -1.440          0.909   4.460    1.3000   43.4 0.005030
## 6      6    0.872 -1.440          0.923   2.710    0.5280   32.9 0.003050
## 7      7    0.890 -1.390          0.929   1.750    0.2310   25.6 0.001970
## 8      8    0.916 -1.380          0.963   1.180    0.1050   20.2 0.001340
## 9      9    0.907 -1.360          0.946   0.830    0.0493   16.2 0.000936
## 10    10    0.921 -1.330          0.965   0.598    0.0234   13.2 0.000675
##    Centralization Heterogeneity
## 1          0.1490         0.401
## 2          0.1110         0.763
## 3          0.0791         1.090
## 4          0.0582         1.390
## 5          0.0441         1.660
## 6          0.0342         1.900
## 7          0.0269         2.120
## 8          0.0215         2.310
## 9          0.0174         2.490
## 10         0.0142         2.650
##  [ reached 'max' / getOption("max.print") -- omitted 5 rows ]
```

Calculation of [similarity -> adjacency] -> TOM matrix

```
adjMatrix <- adjacency(dat_WGCNA, power = sft$powerEstimate)

tomMatrix <- TOMsimilarity(adjMat = adjMatrix, verbose = 5)
```

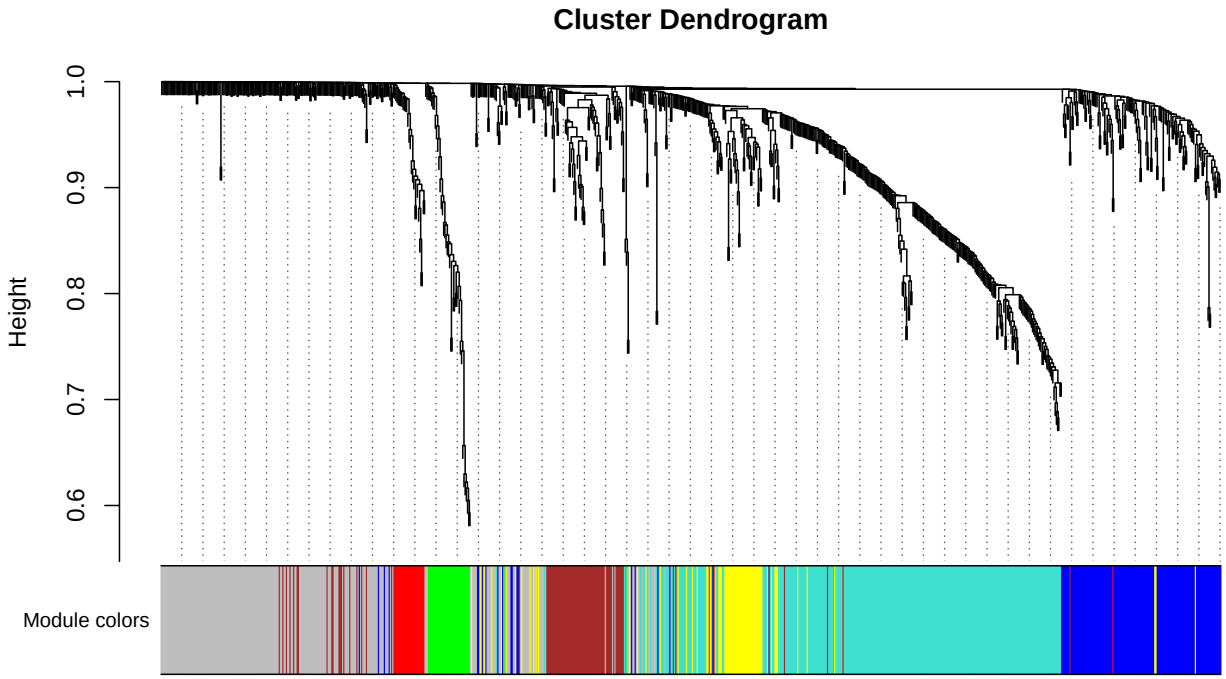
```
## ..connectivity..
## ..matrix multiplication (system BLAS)..
## ..normalization..
## ..done.
```

Construction of the network and module detection (automatic)

```
net = blockwiseModules(dat_WGCNA,
                       power = sft$powerEstimate,
                       TOMType = "unsigned",
                       numericLabels = TRUE,
                       pamRespectsDendro = FALSE,
                       saveTOMs = TRUE,
                       saveTOMFileBase = "02_GraphInference/src/WGCNA_net",
                       verbose = 3)
```

```
##  Calculating module eigengenes block-wise from all genes
##    Flagging genes and samples with too many missing values...
##     ..step 1
##  ..Working on block 1 .
##     TOM calculation: adjacency..
##     ..will not use multithreading.
##      Fraction of slow calculations: 0.000000
##     ..connectivity..
##     ..matrix multiplication (system BLAS)..
##     ..normalization..
##     ..done.
##    ..saving TOM for block 1 into file 02_GraphInference/src/WGCNA_net-block.1.RData
##  ....clustering..
##  ....detecting modules..
##  ....calculating module eigengenes..
##  ....checking kME in modules..
##      ..removing 30 genes from module 1 because their KME is too low.
##      ..removing 3 genes from module 3 because their KME is too low.
##      ..removing 1 genes from module 4 because their KME is too low.
##   ..reassigning 26 genes from module 1 to modules with higher KME.
##   ..reassigning 5 genes from module 2 to modules with higher KME.
##   ..reassigning 1 genes from module 4 to modules with higher KME.
##  ..merging modules that are too close..
##      mergeCloseModules: Merging modules whose distance is less than 0.15
##        Calculating new MEs...
```

```
table(net$colors)
# Convert labels to colors for plotting
mergedColors = labels2colors(net$colors)
# Plot the dendrogram and the module colors underneath
plotDendroAndColors(net$dendrograms[[1]], mergedColors[net$blockGenes[[1]]],"Module colors",dendroLabels
```




```
#Saving variables for after
moduleLabels = net$colors
moduleColors = labels2colors(net$colors)
MEs = net$MEs
geneTree = net$dendrograms[[1]]
```

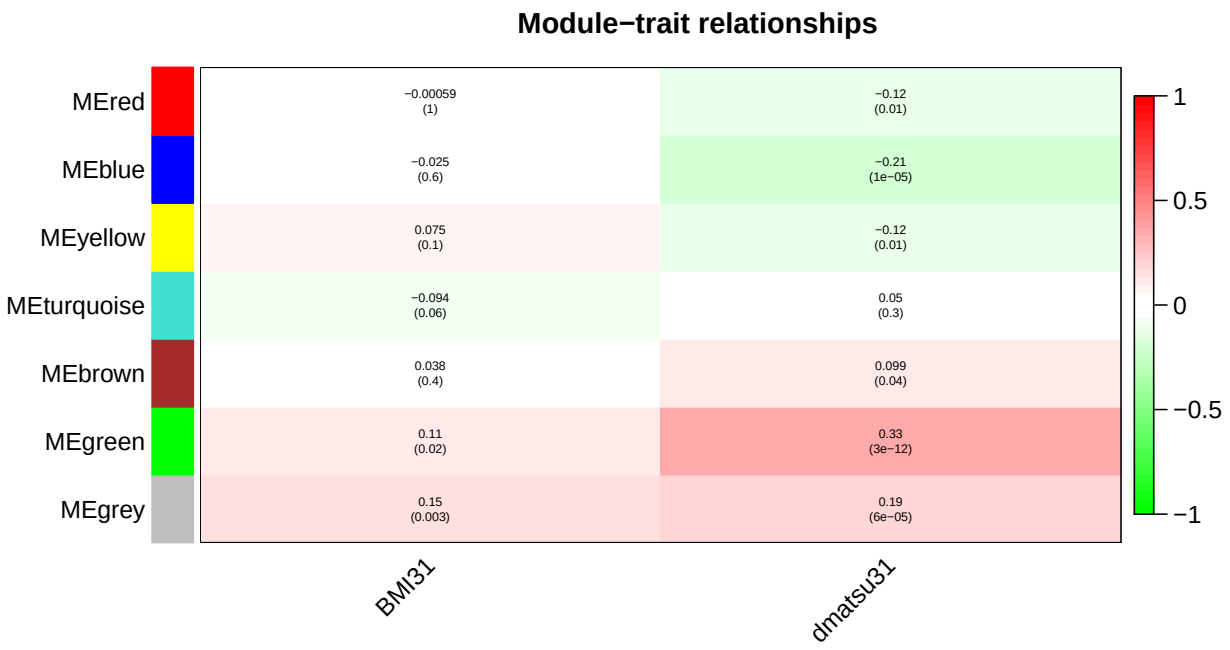
```
##
##   0   1   2   3   4   5   6
## 230 283 148  86  71  38  31
```

```
# Define numbers of genes and samples
nGenes = ncol(dat_WGCNA)
nSamples = nrow(dat_WGCNA)
# Recalculate MEs with color labels
MEs0 = moduleEigengenes(dat_WGCNA, moduleColors)$eigengenes
MEs = orderMEs(MEs0)
moduleTraitCor = cor(MEs, clinical, use = "p")
moduleTraitPvalue = corPvalueStudent(moduleTraitCor, nSamples)
```

```
# Will display correlations and their p-values
textMatrix = paste(signif(moduleTraitCor, 2), "\n(",signif(moduleTraitPvalue, 1), ")", sep = "")
dim(textMatrix) = dim(moduleTraitCor)
par(mar = c(6, 8.5, 3, 3))
# Display the correlation values within a heatmap plot
labeledHeatmap(
  Matrix = moduleTraitCor,
  xLabels = colnames(clinical),
  yLabels = names(MEs),
  ySymbols = names(MEs),
  colorLabels = FALSE,
  colors = greenWhiteRed(50),
  textMatrix = textMatrix,
  setStdMargins = FALSE,
```

```
  cex.text = 0.5,
  zlim = c(-1, 1),
  main = paste("Module-trait relationships")
)
```

```
## Warning in greenWhiteRed(50): WGCNA::greenWhiteRed: this palette is not suitable for people
## with green-red color blindness (the most common kind of color blindness).
## Consider using the function blueWhiteRed instead.
```



Package rag2ridges

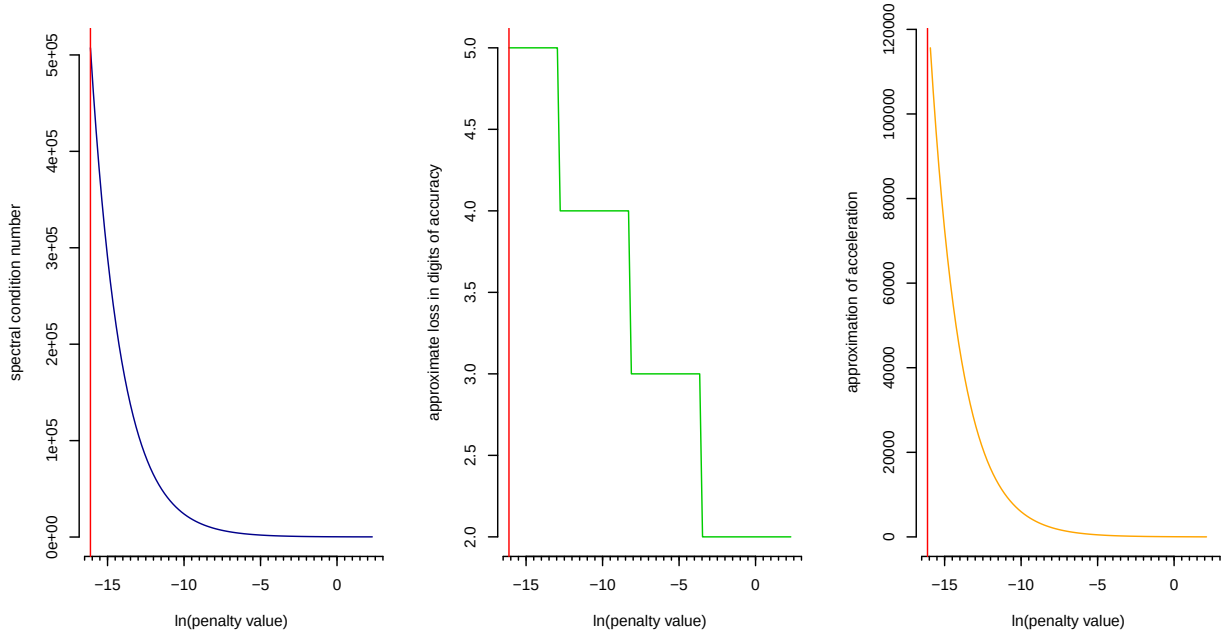
This package is based on L2-penalized maximum likelihood estimation of precision matrices. It support network extraction, network visualization and network analysis tools.

The first step is the estimation of the penalty parameter λ through 10-fold cross-validation. `rags2ridges` have also a tool to visualize the stability of this parameter and thus the conditioning of $\hat{\Omega}(\lambda)$.

```
set.seed(42)
lmin <- 0.0000001
lmax <- 10
stp <- 25
# Calculate the regularized precision matrix under optimal penalty
# OPT <- optPenalty.kCV(pseudoCounts,
                        # lambdaMin = lmin,
                        # lambdaMax = lmax,
                        # step = stp,
                        # fold = 10,
                        # target = default.target(covML(pseudoCounts),
                        #                         type = "DUPV"),
                        # graph = FALSE)

# saveRDS(OPT, '02_GraphInference/src/OPT')
```

```
OPT <- readRDS('02_GraphInference/src/OPT')
```

```
# Plot to visualize the spectral condition number against the regularization parameter
CNplot(covML(pseudoCounts),
       lambdaMin = lmin,
       lambdaMax = lmax,
       step = 100,
       target = default.target(covML(pseudoCounts), type = "DUPV"),
       Iaids = TRUE,
       vertical = TRUE,
       value = OPT$optLambda)
```




```
## Perform input checks...
## Calculating spectral condition numbers...
## Calculating interpretational aids...
## Plotting...
```

As we have used a ℓ_2 penalty, we need to sparsify the matrix to select edges of interest. This time we will use the edges with the strongest absolute partial correlation to choose the best edges (for an easier visualisation).

```
## Determine support regularized standardized precision matrix under optimal penalty (sparsification)
SPC0 <- sparsify(symm(OPT$optPrec), threshold = "top", top = 150)
PC0 <- SPC0$sparseParCor
edgeHeat(PC0, diag = FALSE, textsize = 1)
```

```
## Warning in type.convert.default(X[[i]], ...): 'as.is' should be specified by the
## caller; using TRUE

## Warning in type.convert.default(X[[i]], ...): 'as.is' should be specified by the
## caller; using TRUE
```


```
names(PC0) <- as.character(1:ncol(PC0))
row.names(PC0) <- as.character(1:ncol(PC0))
colnames(PC0) <- as.character(1:ncol(PC0))
```

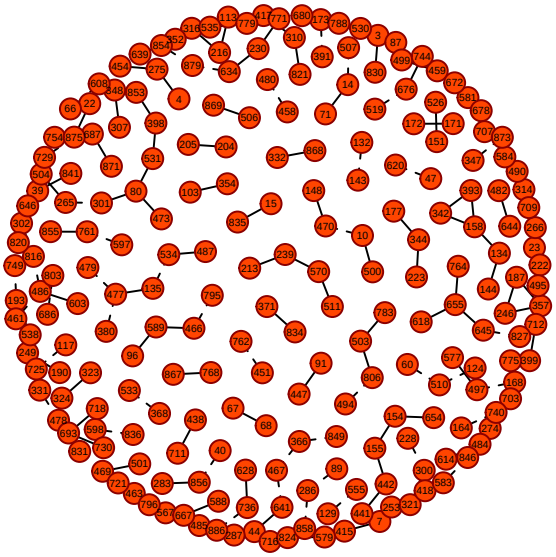
```
## - Retained elements:  150
## - Corresponding to 0.04 % of possible edges
##
```

We can now do a first visualisation of our graph with the `Ugraph()` fonction. The nodes with no connecting edges are pruned.

```
grph <- Ugraph(PC0,
               type = "fancy",
               prune = TRUE,
               lay = "layout_with_kk",
               Vsize = 8,
               Vcex = 0.3)
```

```
## Warning in type.convert.default(X[[i]], ...): 'as.is' should be specified by the
## caller; using TRUE
```

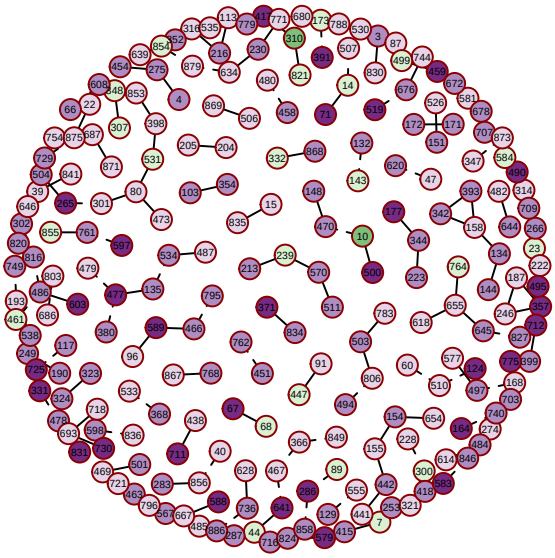
```
## Warning in type.convert.default(X[[i]], ...): 'as.is' should be specified by the
## caller; using TRUE
```



```
fus_dat <- cbind.data.frame(clinical, pseudoCounts)
correl <- cor(fus_dat$dmatsu31, fus_dat[,-c(1:2)], use = "pairwise.complete.obs")
COLORS <- c('#762a83','#af8dc3','#e7d4e8','#d9f0d3','#7fbf7b','#1b7837')
qq <- cut(correl, breaks=seq(min(correl),max(correl), length.out=6), include.lowest=T, labels=FALSE)
node_color<- labels2colors(qq, colorSeq = COLORS)
grph_matsu <- Ugraph(PC0,
                type = "fancy",
                prune = TRUE,
                lay = "layout_with_kk",
                Vsize = 8,
                Vcex = 0.3,
                Vcolor = node_color,
                main = "Correlation of selected nodes with Matsuda index")
```

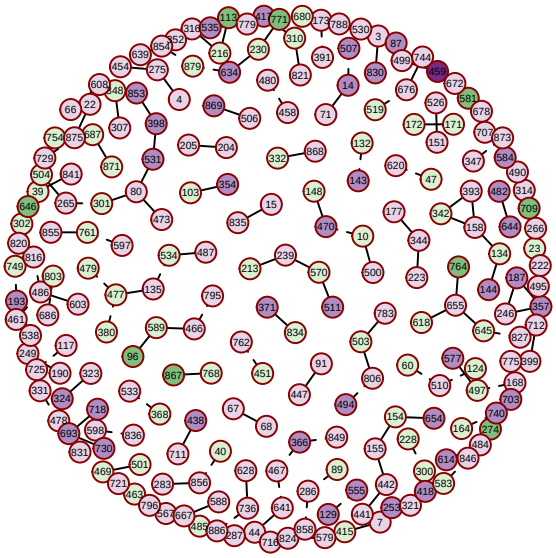
```
## Warning in type.convert.default(X[[i]], ...): 'as.is' should be specified by the
## caller; using TRUE

## Warning in type.convert.default(X[[i]], ...): 'as.is' should be specified by the
## caller; using TRUE
```

Correlation of selected nodes with Matsuda index



```
correl <- cor(fus_dat$BMI31, fus_dat[,-c(1:2)], use = "pairwise.complete.obs")
qq <- cut(correl, breaks=seq(min(correl),max(correl), length.out=6), include.lowest=T, labels=FALSE)
node_color<- labels2colors(qq, colorSeq = COLORS)

grph_bmi <- Ugraph(PC0,
                type = "fancy",
                prune = TRUE,
                lay = "layout_with_kk",
                Vsize = 8,
                Vcex = 0.3,
                Vcolor = node_color,
                main = "Correlation of selected nodes with BMI")
```

```
## Warning in type.convert.default(X[[i]], ...): 'as.is' should be specified by the
## caller; using TRUE

## Warning in type.convert.default(X[[i]], ...): 'as.is' should be specified by the
## caller; using TRUE
```



The package also provide some network stat tools.

```
PcorP <- pruneMatrix(PC0)
NwkSTATS <- GGMnetworkStats(PcorP)
```

```
## Warning in closeness(CIG, mode = "all"): At centrality.c:2784 :closeness
## centrality is not well-defined for disconnected graphs
```

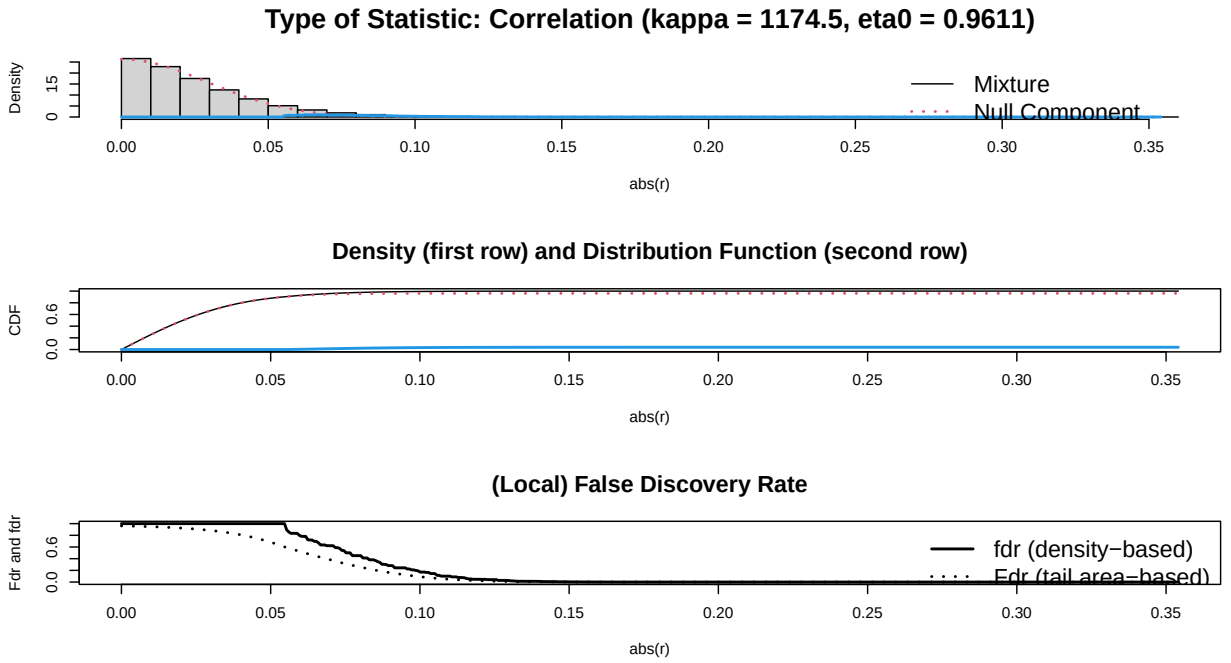
```
summary(NwkSTATS)
```

```
##                 Length Class  Mode
## degree          229    -none- numeric
## betweenness     229    -none- numeric
## closeness       229    -none- numeric
## eigenCentrality 229    -none- numeric
## nNeg            229    -none- numeric
## nPos            229    -none- numeric
## chordal           1    -none- logical
## mutualInfo      229    -none- numeric
## variance        229    -none- numeric
## partialVar      229    -none- numeric
```

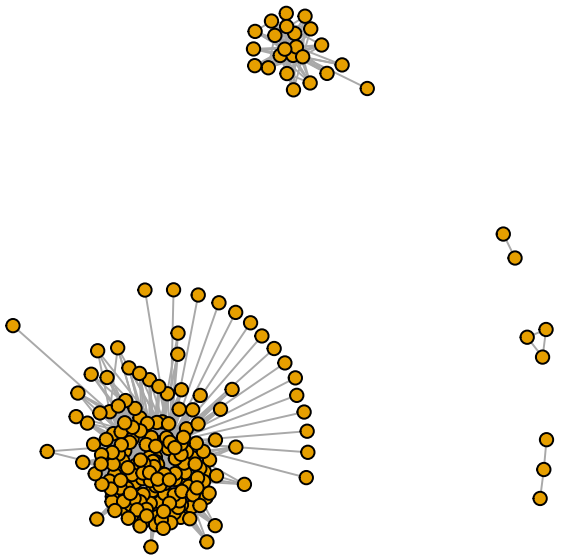
Comparison between packages

Comparison of the most importants edges in each graph with a criterion of 0.2 local FDR

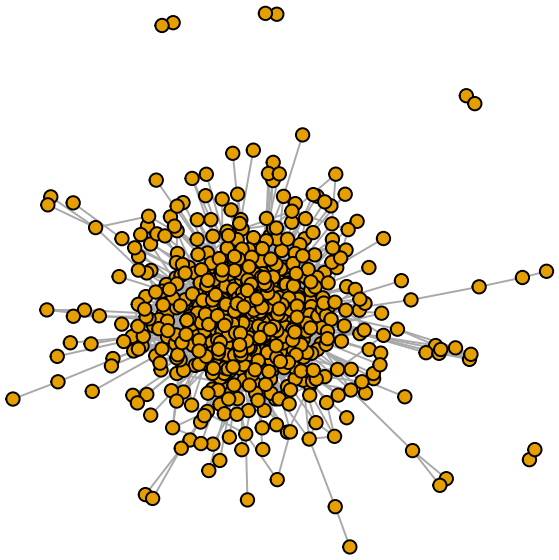
```
cutoff <- 0.80
edges_gn <-  extract.network(grph.edges, method.ggm="prob", cutoff.ggm = cutoff)[,2:3]
grph_gn <- graph_from_edgelist(as.matrix(edges_gn, ncol = 2), directed = FALSE)

edges_rag <- sparsify(symm(OPT$optPrec), threshold = "localFDR", FDRcut = cutoff)$sparseParCor
```



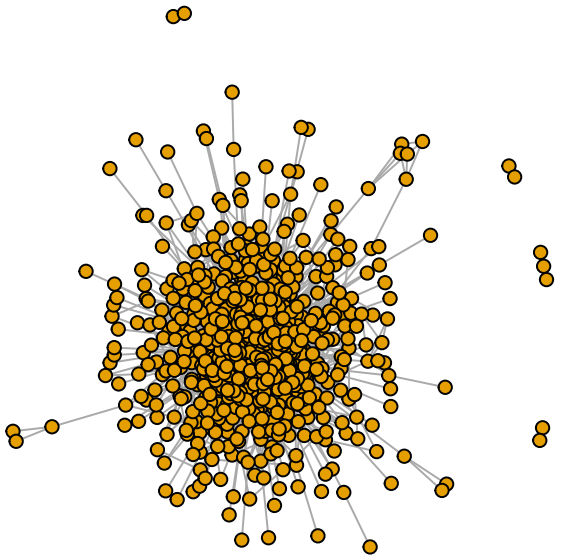

```
row.names(edges_rag) <- as.character(1:ncol(edges_rag))
colnames(edges_rag) <- as.character(1:ncol(edges_rag))
## Attention sur igraph, il faut que les poids de la matrice d'adjacence
## soit à 1 pour les calculs suivants (transitivité, etc)
edges_rag[edges_rag!=0] <- 1
edges_rag <- edges_rag - diag(1, nrow = nrow(edges_rag), ncol = ncol(edges_rag))
grph_rag <- graph_from_adjacency_matrix(adjmatrix = edges_rag, mode = "undirected")

mean_nbr_of_nodes <- (sum(get.adjacency(grph_rag)!=0)+sum(get.adjacency(grph_gn)!=0))/2
tomMatrix <- tomMatrix- diag(1,nrow(tomMatrix), ncol(tomMatrix))
cutoff_WGCNA <- seq(0,1,0.001)[which.min(lapply(seq(0,1,0.001), name <- function(x,y) abs(mean_nbr_of_n
adj_WGCNA <- matrix(0, nrow(tomMatrix), ncol(tomMatrix))
adj_WGCNA[tomMatrix>cutoff_WGCNA] <- 1
grph_WGCNA <- graph_from_adjacency_matrix(adjmatrix = adj_WGCNA, mode = "undirected")

Isolated = which(degree(grph_WGCNA)==0)
G2 = delete.vertices(grph_WGCNA, Isolated)
plot(G2, layout = layout_with_fr, vertex.size=5, vertex.label=NA, main = "Graph with WGCNA", sub = "Fru
```




```
Isolated = which(degree(grph_gn)==0)
G2 = delete.vertices(grph_gn, Isolated)
plot(G2, layout = layout_with_fr, vertex.size=5, vertex.label=NA, main = "Graph with Genenet", sub = "Fr
```




```
Isolated = which(degree(grph_rag)==0)
G2 = delete.vertices(grph_rag, Isolated)
plot(G2, layout = layout_with_fr, vertex.size=5, vertex.label=NA, main = "Graph with rags2ridges", sub =
```

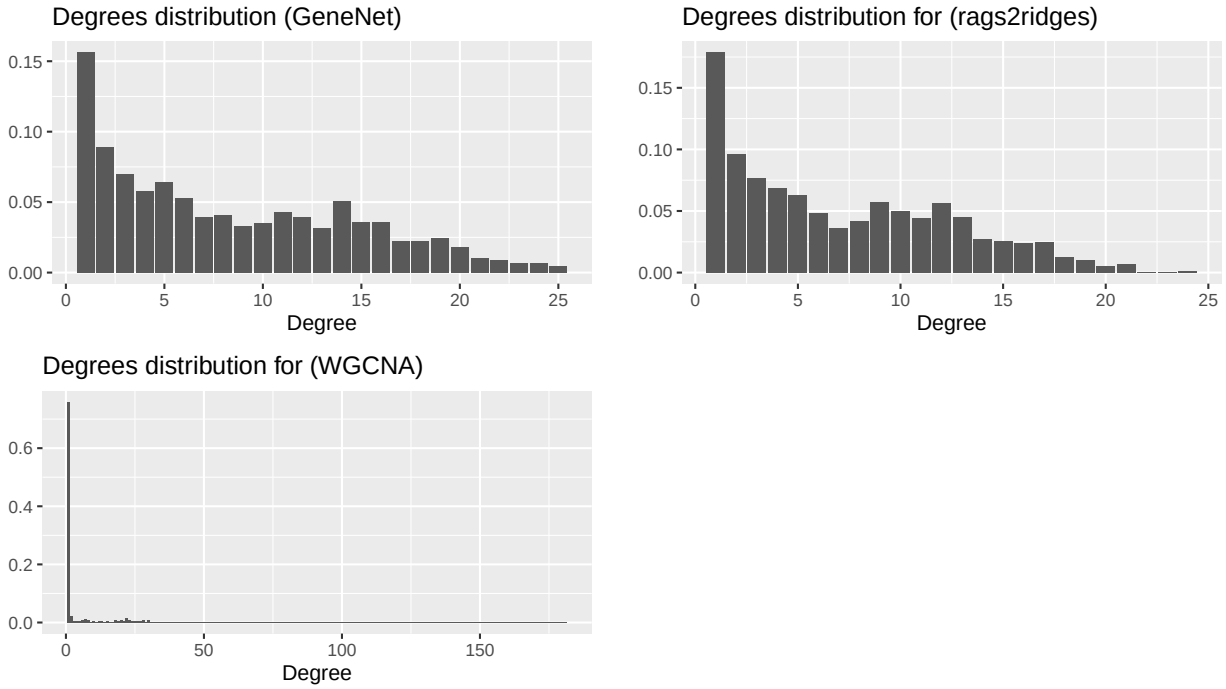



```
##
## Significant edges:  3240
##     Corresponding to  0.82 %  of possible edges
##
## Significant directions:  0
##     Corresponding to  0 %  of possible directions
## Significant directions in the network:  0
##     Corresponding to  0 %  of possible directions in the network
## Step 1... determine cutoff point
## Step 2... estimate parameters of null distribution and eta0
## Step 3... compute p-values and estimate empirical PDF/CDF
## Step 4... compute q-values and local fdr
## Step 5... prepare for plotting
##
## - Retained elements:  2703
## - Corresponding to 0.69 % of possible edges
##
```

Comparison of various graph features

```
feat <- data.frame(row.names = c("GeneNet", "rags2ridges", "WGCNA"))
feat$density[1] <- edge_density(grph_gn)
feat$transitivity[1] <- transitivity(grph_gn, type = "global")

feat$density[2] <- edge_density(grph_rag)
```

```
feat$transitivity[2] <- transitivity(grph_rag, type = "global", isolates = "zero")

feat$density[3] <- edge_density(grph_WGCNA)
feat$transitivity[3] <- transitivity(grph_WGCNA, type = "global", isolates = "zero")

print(feat)

same_edges <- sum(as.matrix(get.adjacency(grph_gn) +
                            get.adjacency(grph_rag) +
                            get.adjacency(grph_WGCNA)) ==3)
cat("\nThere is", same_edges, "identic edges between the two graphs")

deg_gn <- as.data.frame(degree_distribution(grph_gn))
deg_rag <- as.data.frame(degree_distribution(grph_rag))
deg_WGCNA <- as.data.frame(degree_distribution(grph_WGCNA))
names(deg_gn) <- "Degree" -> names(deg_rag) ->names(deg_WGCNA)
ggpubr::ggarrange(ggplot(deg_gn, aes(x=1:nrow(deg_gn), y=Degree)) +
                  geom_col()+xlab("Degree")+ylab("") +
                  ggtitle("Degrees distribution (GeneNet)"),
                  ggplot(deg_rag, aes(x=1:nrow(deg_rag), y=Degree)) +
                  geom_col()+ylab("")+xlab("Degree") +
                  ggtitle("Degrees distribution for (rags2ridges)"),
                  ggplot(deg_WGCNA, aes(x=1:nrow(deg_WGCNA), y=Degree)) +
                  geom_col()+ylab("")+xlab("Degree") +
                  ggtitle("Degrees distribution for (WGCNA)"))
```




```
##                density transitivity
## GeneNet     0.008245513   0.02966869
## rags2ridges 0.006878895   0.02378423
## WGCNA       0.007555842   0.39168163
```

```
##
## There is 44 identic edges between the two graphs
```

Join count calculation

```
igraph2nb <- function(grph) {
     Isolated = which(degree(grph)==0)
     grph_no_isolated = delete.vertices(grph, Isolated)
     weight<- mat2listw(as_adj(grph_no_isolated))
     return(list("weights" = weight, "isolated" = Isolated))
}

joincount_no_isolated <- function(cor_BMI, cor_Matsu, r) {
  cor_BMI_no_isolated <- cor_BMI[-r$isolated]
  cor_Matsu_no_isolated <- cor_Matsu[-r$isolated]
  joincount_BMI <- moran.mc(cor_BMI_no_isolated, r$weights, nsim = 100)
  joincount_Matsu <- moran.mc(cor_Matsu_no_isolated, r$weights, nsim = 100)
  return(list(joincount_BMI, joincount_Matsu))
}

cor_BMI <- apply(pseudoCounts, 2, function(x) stats::cor(x,clinical[,1], use='pairwise.complete.obs'))
cor_Matsu <- apply(pseudoCounts, 2, function(x) stats::cor(x,clinical[,2], use='pairwise.complete.obs'))

# Return the value on the form: list[weights, isolated]
r_gn <- igraph2nb(grph_gn)
r_rag <- igraph2nb(grph_rag)
r_wgcna <- igraph2nb(grph_WGCNA)

jc_gn <- joincount_no_isolated(cor_BMI,cor_Matsu, r_gn)
jc_rag <- joincount_no_isolated(cor_BMI,cor_Matsu, r_rag)
jc_wgcna <- joincount_no_isolated(cor_BMI,cor_Matsu, r_wgcna)

dat_jc <- data.frame(
  "joincount_BMI" = c(jc_gn[[1]]$statistic, jc_rag[[1]]$statistic, jc_wgcna[[1]]$statistic),
  "joincount_Matsu" = c(jc_gn[[2]]$statistic, jc_rag[[2]]$statistic, jc_wgcna[[2]]$statistic))
rownames(dat_jc) <- c("GeneNet", "rags2ridges", "WGCNA")

knitr::kable(dat_jc)
```

	joincount_BMI	joincount_Matsu
GeneNet	0.0672780	0.0690042
rags2ridges	0.0538727	0.0533884
WGCNA	0.2521190	0.2188646

Venn Diagram

```
library(VennDiagram)
```

```
## Le chargement a nécessité le package : futile.logger
```

```
library(RColorBrewer)
edge_list <- list(paste(as_edgelist(grph_WGCNA)[,1],
                        as_edgelist(grph_WGCNA)[,2])
```

```
                        ,paste(as_edgelist(grph_rag)[,1],
                               as_edgelist(grph_rag)[,2])
                        ,paste(as_edgelist(grph_gn)[,1],
                               as_edgelist(grph_gn)[,2]))
venn.diagram(x = edge_list,
             category.names = c("WGCNA" , "rag2ridges" , "GeneNet"),
             output = TRUE,
             filename = "02_GraphInference/src/venn_diagram.png",
             imagetype = "png",
             cat.default.pos = 'text',
             lty = 'blank',
             fill = c("#66c2a5", "#fc8d62", "#8da0cb"),
             main = "Shared edges between each graph",
             main.cex = 1.8)

unlink("02_GraphInference/src/venn_diagram.png*log")
```

```
## [1] 1
```

```
write.csv(as.matrix(as_adj(grph_gn)), "02_GraphInference/src/adj_mat_gn.csv")
write.csv(as.matrix(as_adj(grph_rag)), "02_GraphInference/src/adj_mat_rag.csv")
write.csv(as.matrix(as_adj(grph_WGCNA)), "02_GraphInference/src/adj_mat_wgcna.csv")
```

Appendix

```
## Document generated in:
## Time difference of 1.685884 mins
##
## CPU: Intel(R) Core(TM) i7-10610U CPU @ 1.80GHz
## Memory total size: 32.54956 GB
##
##
## Session information:
## R version 4.1.1 (2021-08-10)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 20.04.2 LTS
##
## Matrix products: default
## BLAS:   /usr/lib/x86_64-linux-gnu/blas/libblas.so.3.9.0
## LAPACK: /usr/lib/x86_64-linux-gnu/lapack/liblapack.so.3.9.0
##
## attached base packages:
## [1] grid      parallel  stats     graphics  grDevices utils     datasets
## [8] methods   base
##
## other attached packages:
##  [1] RColorBrewer_1.1-2    VennDiagram_1.6.20    futile.logger_1.4.3
##  [4] spdep_1.1-8           sf_1.0-2              spData_0.3.10
##  [7] sp_1.4-5              ggplot2_3.3.5         igraph_1.2.6
## [10] rags2ridges_2.2.5     WGCNA_1.70-3          fastcluster_1.2.3
## [13] dynamicTreeCut_1.63-1 Rgraphviz_2.36.0      graph_1.70.0
## [16] BiocGenerics_0.38.0   GeneNet_1.2.15        fdrtool_1.2.16
## [19] longitudinal_1.1.12   corpcor_1.6.9
```

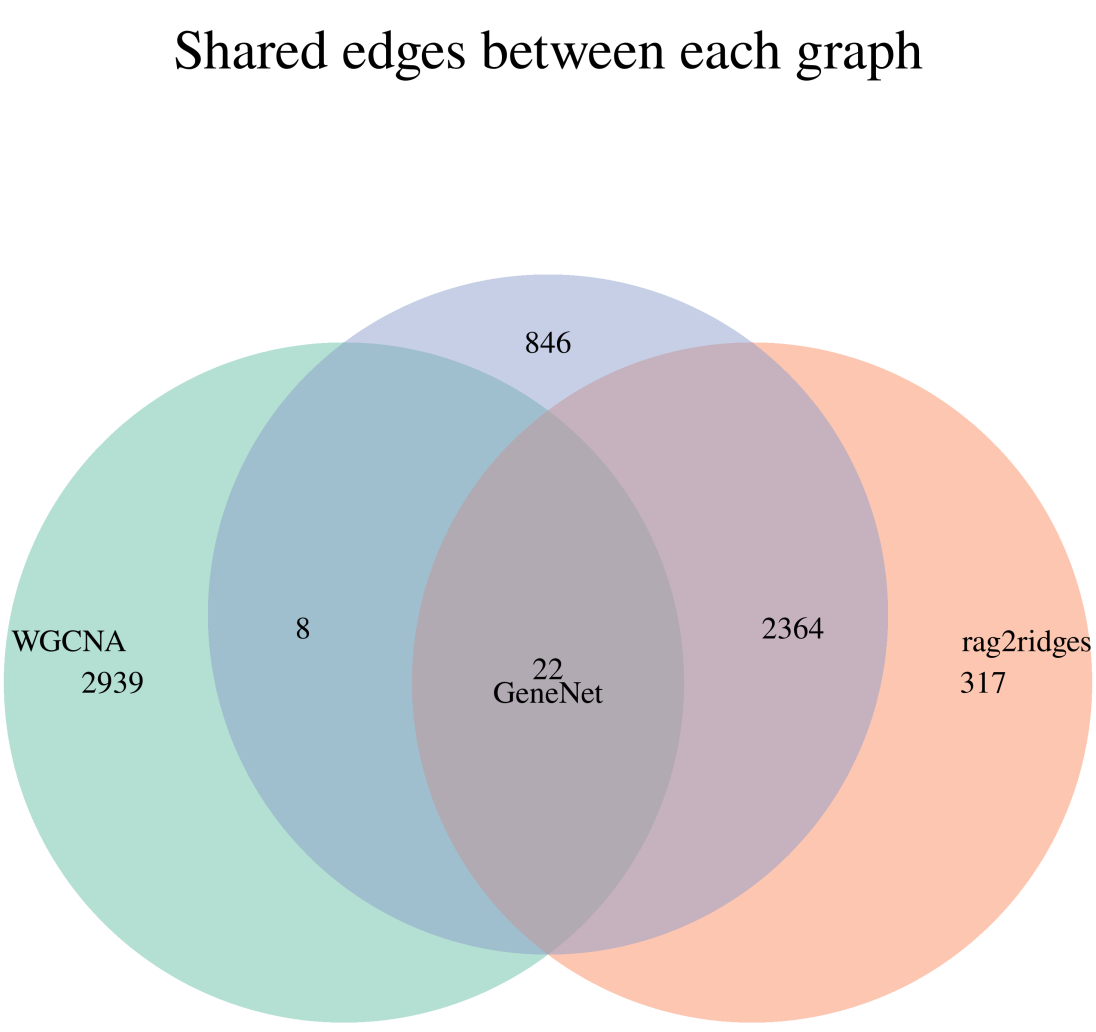


Figure 2: venn

```
##
## loaded via a namespace (and not attached):
##   [1] readxl_1.3.1           backports_1.2.1        Hmisc_4.5-0
##   [4] plyr_1.8.6             splines_4.1.1          GenomeInfoDb_1.28.1
##   [7] digest_0.6.27          foreach_1.5.1          htmltools_0.5.1.1
##  [10] GO.db_3.13.0           gdata_2.18.0           fansi_0.5.0
##  [13] magrittr_2.0.1         checkmate_2.0.0        memoise_2.0.0
##  [16] cluster_2.1.2          doParallel_1.0.16      sfsmisc_1.1-11
##  [19] openxlsx_4.2.4         Biostrings_2.60.1      matrixStats_0.60.0
##  [22] gmodels_2.18.1         jpeg_0.1-9             colorspace_2.0-2
##  [25] blob_1.2.2             haven_2.4.3            xfun_0.24
##  [28] dplyr_1.0.7            crayon_1.4.1           RCurl_1.98-1.3
##  [31] impute_1.66.0          survival_3.2-11        iterators_1.0.13
##  [34] glue_1.4.2             gtable_0.3.0           zlibbioc_1.38.0
##  [37] XVector_0.32.0         car_3.0-11             gRbase_1.8-6.7
##  [40] abind_1.4-5            scales_1.1.1           futile.options_1.0.1
##  [43] DBI_1.1.1              rstatix_0.7.0          Rcpp_1.0.7
##  [46] htmlTable_2.2.1        units_0.7-2            foreign_0.8-81
##  [49] bit_4.0.4              proxy_0.4-26           preprocessCore_1.54.0
##  [52] Formula_1.2-4          stats4_4.1.1           htmlwidgets_1.5.3
##  [55] httr_1.4.2             ellipsis_0.3.2         pkgconfig_2.0.3
##  [58] reshape_0.8.8          farver_2.1.0           nnet_7.3-16
##  [61] deldir_0.2-10          utf8_1.2.2             tidyselect_1.1.1
##  [64] labeling_0.4.2         rlang_0.4.11           AnnotationDbi_1.54.1
##  [67] cellranger_1.1.0       munsell_0.5.0          tools_4.1.1
##  [70] cachem_1.0.5           generics_0.1.0         RSQLite_2.2.7
##  [73] broom_0.7.9            evaluate_0.14          stringr_1.4.0
##  [76] fastmap_1.1.0          yaml_2.2.1             knitr_1.33
##  [79] bit64_4.0.5            zip_2.2.0              purrr_0.3.4
##  [82] KEGGREST_1.32.0        RBGL_1.68.0            nlme_3.1-152
##  [85] formatR_1.11           compiler_4.1.1         rstudioapi_0.13
##  [88] curl_4.3.2             png_0.1-7              e1071_1.7-8
##  [91] ggsignif_0.6.2         tibble_3.1.3           stringi_1.7.3
##  [94] highr_0.9              RSpectra_0.16-0        forcats_0.5.1
##  [97] lattice_0.20-44        Matrix_1.3-4           classInt_0.4-3
## [100] vctrs_0.3.8            pillar_1.6.2           LearnBayes_2.15.1
## [103] lifecycle_1.0.0        cowplot_1.1.1          data.table_1.14.0
## [106] bitops_1.0-7           raster_3.4-13          R6_2.5.0
## [109] latticeExtra_0.6-29    KernSmooth_2.23-20     gridExtra_2.3
## [112] rio_0.5.27             IRanges_2.26.0         codetools_0.2-18
## [115] lambda.r_1.2.4         boot_1.3-28            MASS_7.3-54
## [118] gtools_3.9.2           withr_2.4.2            S4Vectors_0.30.0
## [121] GenomeInfoDbData_1.2.6 hms_1.1.0              expm_0.999-6
## [124] rpart_4.1-15           tidyr_1.1.3            coda_0.19-4
## [127] class_7.3-19           rmarkdown_2.10         carData_3.0-4
## [130] ggpubr_0.4.0           Biobase_2.52.0         snowfall_1.84-6.1
## [133] base64enc_0.1-3
```

B.2 Scripts Python

B.2.1 Pre-processing des données Diogènes pour Spektral

```
import pandas as pd
import os

from sklearn.preprocessing import StandardScaler
import spektral
import numpy as np

if not os.path.exists('data'):
    os.mkdir('data')
if not os.path.exists('data/Diogene'):
    os.mkdir('data/Diogene')
if not os.path.exists('data/Diogene/raw'):
    os.mkdir('data/Diogene/raw')
adj_mat_gn = pd.read_csv("../../data/Diogenes/processed/adj_mat_gn.csv",
    index_col=1).to_csv('data/Diogene/raw/adj_mat_gn.csv')
adj_mat_rag = pd.read_csv("../../data/Diogenes/processed/adj_mat_rag.csv"
    , index_col=1).to_csv('data/Diogene/raw/adj_mat_rag.csv')
adj_mat_wgcna = pd.read_csv("../../data/Diogenes/processed/adj_mat_wgcna.
    csv", index_col=1).to_csv('data/Diogene/raw/adj_mat_wgcna.csv')

pseudoCount = pd.read_csv("../../data/Diogenes/prepared/pseudoCounts_cid1
    .csv", sep="\t", header=0, index_col=0).transpose()
clinical = pd.read_csv("../../data/Diogenes/prepared/selclinical_cid1.csv
    ", sep="\t", header=0)

pseudoCount_col_names = list(pseudoCount.columns)
clinical_col_names = list(clinical.columns)

pseudoCount_row_names = list(pseudoCount.index)
clinical_row_names = list(clinical.index)

########################################
## Suppression of na value in clinical (and corresponding graph in
    pseudoCount)
pseudoCount.columns = pseudoCount_col_names
pseudoCount.index = pseudoCount_row_names

clinical.columns = clinical_col_names
clinical.index = clinical_row_names

miss = clinical.isna()
miss_index = np.where(np.any(miss == True, axis=1))
clinical = clinical.drop(clinical.index[miss_index])
```

```
pseudoCount = pseudoCount.drop(pseudoCount.index[miss_index])


##########################################
## Standardization of nodes feature (pseudoCount) and graph labels (
   clinical)
scaler = StandardScaler()
pseudoCount_scale = scaler.fit_transform(pseudoCount)
pseudoCount_scale = pd.DataFrame(pseudoCount_scale)
scaler = StandardScaler()
clinical_scale = scaler.fit_transform(clinical)
clinical_scale = pd.DataFrame(clinical_scale)

pseudoCount_scale.to_csv('data/Diogene/raw/pseudoCount.csv', sep = '\t')
clinical_scale.to_csv('data/Diogene/raw/clinical.csv', sep = '\t')


class DiogeneTF(spektral.data.dataset.Dataset):
    r"""
␣␣␣␣Diogene␣dataset
␣␣␣␣Args:
␣␣␣␣nbr_of_graph␣(int):␣Number␣of␣graphs␣in␣the␣dataset

␣␣␣␣"""
    path = "data/Diogene/spektral/"


    def __init__(self, **kwargs):
        self.nbr_of_grph = len(pd.read_csv("data/Diogene/raw/clinical.csv"
            , sep="\t", header=0, index_col=0))
        self.a = None
        super().__init__(**kwargs)

    def download(self):
        os.mkdir(self.path)

        pseudoCount = pd.read_csv("data/Diogene/raw/pseudoCount.csv", sep=
            "\t", header=0, index_col=0)
        clinical = pd.read_csv("data/Diogene/raw/clinical.csv", sep="\t",
            header=0, index_col=0)

        for i in range(self.nbr_of_grph):
            x = pseudoCount.values[i]
            y = clinical.values[i]
            np.savez(os.path.join(self.path, 'graph_{}.pt'.format(i)), x=x
                , y=y)

    def read(self):
```

```
        ########################################
        ## Test with full adj mat
        ## indices_zero = adj == 0
        ## adj[indices_zero] = 1 # replacing 0s with 1s
        ## adj = adj - np.identity(adj.shape[0])
        ########################################
        ########################################
        ## Test with zero adj mat
        # indices_one = adj == 1
        # adj[indices_one] = 1 # replacing 0s with 1s
        ########################################
        self.a = pd.read_csv("data/Diogene/raw/adj_mat_rag.csv", index_col
            =1).to_numpy()

        # We must return a list of Graph objects
        output = []
        for i in range(self.nbr_of_grph):
            data = np.load(os.path.join(self.path, f'graph_{i}.pt.npz'),
                allow_pickle=True)
            output.append(spektral.data.graph.Graph(x=data['x'], y=data['y
                ']))
        return output
```

B.2.2 Pre-processing des données BreastCancer pour Spektral

```
import pandas as pd
import os

from sklearn.preprocessing import QuantileTransformer
import spektral
import numpy as np
from scipy.sparse import csr_matrix

if not os.path.exists('data'):
    os.mkdir('data')
if not os.path.exists('data/breast_cancer'):
    os.mkdir('data/breast_cancer')
if not os.path.exists('data/breast_cancer/raw'):
    os.mkdir('data/breast_cancer/raw')

pseudoCount = pd.read_csv("data/breast_cancer/raw/GEO_HG_PPI.csv")
labels = pd.read_csv("data/breast_cancer/raw/labels_GEO_HG.csv").
    transpose()

pseudoCount.index = list(pseudoCount.pop("probe"))
pseudoCount = pseudoCount.transpose()
```

```
########################################
## Standardization of nodes feature (pseudoCount)
scaler = QuantileTransformer(n_quantiles=10, random_state=0)
pseudoCount_scale = scaler.fit_transform(pseudoCount)
pseudoCount_scale = pd.DataFrame(pseudoCount_scale)

if not os.path.exists('data/breast_cancer/processed'):
    os.mkdir('data/breast_cancer/processed')
pseudoCount_scale.to_csv('data/breast_cancer/processed/GEO_HG_PPI.csv',
    index = False)
labels.to_csv('data/breast_cancer/processed/labels_GEO_HG.csv', index =
    False)


class BreastCancerTF(spektral.data.dataset.Dataset):
    r"""
␣␣␣␣Breast␣Cancer␣dataset
␣␣␣␣Args:
␣␣␣␣nbr_of_graph␣(int):␣Number␣of␣graphs␣in␣the␣dataset

␣␣␣␣"""
    path = "data/breast_cancer/spektral/"


    def __init__(self, **kwargs):
        self.nbr_of_grph = pd.read_csv("data/breast_cancer/processed/
            labels_GEO_HG.csv").shape[0]
        self.a = None
        super().__init__(**kwargs)

    def download(self):
        os.mkdir(self.path)

        pseudoCount = pd.read_csv('data/breast_cancer/processed/GEO_HG_PPI
            .csv', header=None)
        clinical = pd.read_csv("data/breast_cancer/processed/labels_GEO_HG
            .csv")

        for i in range(self.nbr_of_grph):
            x = pseudoCount.values[i]
            y = clinical.values[i]
            np.savez(os.path.join(self.path, 'graph_{}.pt'.format(i)), x=x
                , y=y)

    def read(self):
        ########################################
        ## Test with full adj mat
        ## indices_zero = adj == 0
```

```
        ## adj[indices_zero] = 1 # replacing 0s with 1s
        ## adj = adj - np.identity(adj.shape[0])
        ########################################
        ########################################
        ## Test with zero adj mat
        # indices_one = adj == 1
        # adj[indices_one] = 1 # replacing 0s with 1s
        ########################################
        self.a = pd.read_csv("data/breast_cancer/raw/HPRD_PPI.csv").
            to_numpy()
        # Use if you need a sparse version of the adjacency matrix
        # self.a = csr_matrix(self.a)
        # We must return a list of Graph objects
        output = []
        for i in range(self.nbr_of_grph):
            data = np.load(os.path.join(self.path, f'graph_{i}.pt.npz'),
                allow_pickle=True)
            output.append(spektral.data.graph.Graph(x=data['x'], y=data['y
                ']))
        return output
```

B.2.3 Pre-processing des données BreastCancer pour Geometric

```
 import pandas as pd
import os

from sklearn.preprocessing import QuantileTransformer
from torch_geometric.data import InMemoryDataset, Data
from torch_geometric.utils.sparse import dense_to_sparse
from torch import tensor, save, int64, load, from_numpy


class BreastCancerPytorch(InMemoryDataset):
    """
    Args:
        transform (callable, optional): A function/transform that takes in
             an
            :obj:`torch_geometric.data.Data` object and returns a
                transformed
            version. The data object will be transformed before every
                access.
            (default: :obj:`None`)
        pre_transform (callable, optional): A function/transform that
            takes in
            an :obj:`torch_geometric.data.Data` object and returns a
            transformed version. The data object will be transformed
                before
            being saved to disk. (default: :obj:`None`)
```

```
    """

    def __init__(self, root, transform=None, pre_transform=None):
        super(BreastCancerPytorch, self).__init__(root, transform,
            pre_transform)
        self.data, self.slices = load(self.processed_paths[0])

    @property
    def raw_file_names(self):
        return ['data/breast_cancer/raw/labels_GEO_HG.csv', 'data/
            breast_cancer/raw/GEO_HG_PPI.csv', 'data/breast_cancer/raw/
            HPRD_PPI.csv']


    @property
    def processed_file_names(self):
        return ['data.pt']

    def download(self):
        if not os.path.exists('data'):
            os.mkdir('data')
        if not os.path.exists('data/breast_cancer'):
            os.mkdir('data/breast_cancer')
        if not os.path.exists('data/breast_cancer/raw'):
            os.mkdir('data/breast_cancer/raw')

        pseudoCount = pd.read_csv("data/breast_cancer/raw/GEO_HG_PPI.csv")

        pseudoCount.index = list(pseudoCount.pop("probe"))
        pseudoCount = pseudoCount

        ##########################################
        # Standardization of nodes feature (pseudoCount)
        scaler = QuantileTransformer(n_quantiles=10, random_state=0)
        pseudoCount_scale = scaler.fit_transform(pseudoCount)
        pseudoCount_scale = pd.DataFrame(pseudoCount_scale)

        if not os.path.exists('data/breast_cancer/processed'):
            os.mkdir('data/breast_cancer/processed')
        pseudoCount_scale.to_csv('data/breast_cancer/processed/GEO_HG_PPI.
            csv', index=False)

    def process(self):

        x = pd.read_csv('data/breast_cancer/processed/GEO_HG_PPI.csv',
            header=0)
        labels = pd.read_csv("data/breast_cancer/raw/labels_GEO_HG.csv")
        a = pd.read_csv("data/breast_cancer/raw/HPRD_PPI.csv").to_numpy()
```

```
        nbr_of_grph = labels.shape[1]

        elist = dense_to_sparse(from_numpy(a))[0]
        data_list = []

        for i in range(nbr_of_grph):
            y = from_numpy(labels.values.transpose())[i]
            edge_index = tensor(elist, dtype=int64)

            data_list.append(Data(x=from_numpy(x.values.transpose()[i]).
               unsqueeze(1).double(), edge_index=edge_index, y=y))

        if self.pre_filter is not None:
            data_list = [data for data in data_list if self.pre_filter(
               data)]

        if self.pre_transform is not None:
            data_list = [self.pre_transform(data) for data in data_list]

        data, slices = self.collate(data_list)
        save((data, slices), self.processed_paths[0])
```

B.2.4 Exemple de réseaux utilisés avec Spektral

```
import spektral
from tensorflow import keras
from spektral.layers.pooling.pool import Pool
import tensorflow as tf

def model_simple(data, a_shape, x_shape, hidden, out):
    # Model
    # Input: Adjacency matrix (A) and Attributes matrix (X)
    # Hidden 1: GCN Conv (elu)
    # Hidden 2: GCN Conv (elu)
    # Hidden 3: Pooling sum
    # Hidden 4: Dense (relu)
    # Hidden 5: Dense (relu)
    # Output: Numeric variable

    input_a = keras.layers.Input(shape=a_shape)
    input_x = keras.layers.Input(shape=x_shape)

    hidden1 = spektral.layers.GCNConv(hidden, activation="elu")([input_x,
       input_a])
    hidden2 = spektral.layers.GCNConv(hidden, activation="elu")([hidden1,
       input_a])
    pool3 = spektral.layers.GlobalSumPool()(hidden2)
```

```
    hidden4 = keras.layers.Dense(hidden, activation="relu")(pool3)
    hidden5 = keras.layers.Dense(hidden, activation="relu")(hidden4)
    output = keras.layers.Dense(out, activation="linear")(hidden5)

    model = keras.Model(inputs=[input_x, input_a], outputs=[output])
    data.a = spektral.utils.gcn_filter(data.a)

    return model,data


class Pooling_chereda(Pool):

    def __init__(self, kernel_initializer="glorot_uniform",
        kernel_regularizer=None, kernel_constraint=None, **kwargs):
        super().__init__( kernel_initializer=kernel_initializer,
            kernel_regularizer=kernel_regularizer, kernel_constraint=
            kernel_constraint, **kwargs)

    def call(self, inputs):
        x = inputs
        # Pooling
        x = tf.expand_dims(x, 3) # N x M x F x 1
        x = tf.nn.max_pool(x, ksize=[1, 2, 1, 1], strides=[1, 2, 1, 1],
            padding='SAME')
        # tf.maximum
        output = tf.squeeze(x, [3]) # N x M/p x F

        return output
    @property
    def config(self):
        return {
        }


def model_classif(data, a_shape, x_shape, hidden, out):
    # Model
    # Input: Adjacency matrix (A) and Attributes matrix (X)
    # Hidden 1: GCN Conv (elu)
    # Hidden 2: GCN Conv (elu)
    # Hidden 4: Dense (relu)
    # Hidden 5: Dense (relu)
    # Output: Numeric variable

    input_a = keras.layers.Input(shape=a_shape)
    input_x = keras.layers.Input(shape=x_shape)

    hidden1 = spektral.layers.GCNConv(hidden, activation="elu")([input_x,
        input_a])
```

```
    hidden2 = spektral.layers.GCNConv(hidden, activation="elu")([hidden1,
        input_a])
    hidden3 = spektral.layers.GlobalMaxPool()(hidden2)

    hidden4 = keras.layers.Dense(hidden, activation="relu")(hidden3)
    hidden5 = keras.layers.Dense(hidden, activation="relu")(hidden4)
    output = keras.layers.Dense(out, activation="sigmoid")(hidden5)

    model = keras.Model(inputs=[input_x, input_a], outputs=[output])
    data.a = spektral.utils.gcn_filter(data.a)

    return model, data


def model_chereda(data, a_shape, x_shape, hidden, out):
    # Model from Chereda et al, SHTI 2019
    # Loss CrossEntropy
    # Activation ReLU
    # Input: Adjacency matrix (A) and Attributes matrix (X)
    # Hidden 1: Cheb Conv 32
    # Hidden 2: Cheb Conv 32
    # Hidden 3: Pooling max custom
    # Hidden 4: Dense 512 (relu)
    # Hidden 5: Dense 128 (relu)
    # Output: Numeric variable

    input_a = keras.layers.Input(shape=a_shape)
    input_x = keras.layers.Input(shape=x_shape)

    hidden1 = spektral.layers.ChebConv(32, K=8, activation="relu")([
        input_x, input_a])
    hidden2 = spektral.layers.ChebConv(32, K=8, activation="relu")([
        hidden1, input_a])
    pool3 = Pooling_chereda(p=2)(hidden2)
    hidden4 = keras.layers.Dense(512, activation="relu")(pool3)
    hidden5 = keras.layers.Dense(128, activation="relu")(hidden4)
    output = keras.layers.Dense(out, activation="sigmoid")(hidden5)

    model = keras.Model(inputs=[input_x, input_a], outputs=[output])
    data.a = spektral.utils.chebyshev_filter(data.a, k=8)[8]

    return model, data


def model_chereda_v2(data, a_shape, x_shape, hidden, out):
    # Model from Chereda et al, SHTI 2019
    # Loss CrossEntropy
    # Activation ReLU
```

```
    # Input: Adjacency matrix (A) and Attributes matrix (X)
    # Hidden 1: Cheb Conv 32
    # Hidden 2: Cheb Conv 32
    # Hidden 3: Pooling max
    # Hidden 4: Dense 512 (relu)
    # Hidden 5: Dense 128 (relu)
    # Output: Numeric variable

    # Same model as Chereda but with dropout and L1 regularization

    input_a = keras.layers.Input(shape=a_shape)
    input_x = keras.layers.Input(shape=x_shape)

    hidden1 = spektral.layers.ChebConv(32, K=8, activation="elu")([input_x
        , input_a])
    hidden2 = spektral.layers.ChebConv(32, K=8, activation="elu")([hidden1
        , input_a])
    pool3 = spektral.layers.GlobalMaxPool()(hidden2)
    drop4 = keras.layers.Dropout(0.4)(pool3)
    hidden4 = keras.layers.Dense(512, activation="elu", kernel_regularizer
        =keras.regularizers.L1(0.01))(drop4)
    drop6 = keras.layers.Dropout(0.4)(hidden4)
    hidden5 = keras.layers.Dense(128, activation="elu", kernel_regularizer
        = keras.regularizers.L1(0.01))(drop6)
    output = keras.layers.Dense(out, activation="sigmoid")(hidden5)

    model = keras.Model(inputs=[input_x, input_a], outputs=[output])
    data.a = spektral.utils.chebyshev_filter(data.a, k=8)[8]

    return model,data
```

B.2.5 Script d'entrainement pour de la regression sous Spektral

```
import sys
sys.path.insert(1, '')
from data_process_spektral_diogene import DiogeneTF
from model.model_simple import model_simple
# Library
import numpy as np
import tensorflow.keras as keras
import spektral
from spektral.models import GeneralGNN
import tensorflow as tf

# Global parameters
batch_size = 32
train_prop = 0.8
```

```
valid_prop = 0.2
hidden_channels = 64
learning_rate = 0.0005
epochs = 100

# Load data
data = DiogeneTF()
# The adjacency matrix is stored as an attribute of the dataset for
   spektral mixed mode.


# Optimizer: Nadam
# Loss: MSE
model, data = model_simple(data = data, a_shape=data.a.shape, x_shape=
   data[1].x.shape, hidden=hidden_channels, out=data.n_labels)
optimizer = keras.optimizers.Nadam()
model.compile(loss="mse")
loss_fn = keras.losses.MeanSquaredError()

# Train/valid/test split
np.random.seed(42)
np.random.shuffle(data)
tr_split = int(np.floor(data.n_graphs * train_prop))
data_tr, data_te = data[:tr_split], data[tr_split:]
va_split = int(np.floor(data_tr.n_graphs * valid_prop))
data_tr, data_va = data_tr[:va_split], data_tr[va_split:]


# We use a MixedLoader since the dataset is in mixed mode
loader_tr = spektral.data.MixedLoader(data_tr, batch_size=batch_size,
   epochs=epochs)
loader_va = spektral.data.MixedLoader(data_va, batch_size=batch_size)
loader_te = spektral.data.MixedLoader(data_te, batch_size=batch_size)

# Training function
@tf.function
def train_on_batch(inputs, target):
    with tf.GradientTape() as tape:
        predictions = model(inputs, training=True)
        loss = loss_fn(target, predictions) + sum(model.losses)
    gradients = tape.gradient(loss, model.trainable_variables)
    optimizer.apply_gradients(zip(gradients, model.trainable_variables))
    return loss


# Evaluation function
def evaluate(loader):
    step = 0
```

```
    results = []
    for batch in loader:
        step += 1
        inputs, target = batch
        predictions = tf.squeeze(model(inputs, training=False))
        loss = keras.losses.MSE(target, predictions)
        results.append(loss)
        if step == loader.steps_per_epoch:
            test = np.array([])
            for elem in results:
                test = np.concatenate((elem.numpy(), test))
            return np.average(test, 0, weights=test)


# Setup training
best_val_loss = 9999999
patience = 10
current_patience = patience
step = 0
ep = 0

# Training loop
results_tr = []
results_te = None
history_loss = []
for batch in loader_tr:
    step += 1
    # Training step
    inputs, target = batch
    loss = train_on_batch(inputs, target)
    results_tr.append((loss))

    if step == loader_tr.steps_per_epoch:
        results_va = evaluate(loader_va)
        ep += 1
        if results_va < best_val_loss:
            best_val_loss = results_va
            current_patience = patience
            results_te = evaluate(loader_te)
        else:
            current_patience -= 1
            if current_patience == 0:
                print("Early␣stopping")
                break
        # Print results
        results_tr = np.array(results_tr, dtype=object)
        test = np.array([])
        for elem in results_tr:
```

```
            print(elem.numpy())
            test = np.concatenate(([elem.numpy()], test))
        results_tr = np.average(test, 0, weights=test)
        history_loss.append(results_tr)
        print("Step:␣", ep, "\n"
                         "Train␣loss:␣", results_tr, "␣|␣\n",
              "Valid␣loss:␣", results_va, "␣|␣\n",
              "Test␣loss:␣", results_te, "\n\n"
              )
        # Reset epoch
        results_tr = []
        step = 0
```

B.2.6 Script d'entrainement pour de la regression sous Spektral

```
 import sys
sys.path.insert(1, '')
from data_process_spektral_breast_cancer import BreastCancerTF
from model.model_classif import model_classif
# Library
from sklearn.metrics import roc_auc_score
import numpy as np
import tensorflow.keras as keras
import spektral
import tensorflow as tf

# Global parameters
batch_size = 32
train_prop = 0.8
valid_prop = 0.2
hidden_channels = 64
learning_rate = 0.0005
epochs = 100

# Load data
data = BreastCancerTF()
# The adjacency matrix is stored as an attribute of the dataset for
   spektral mixed mode.

# Optimizer: Nadam
# Loss: Binary CrossEntropy
model, data = model_classif(data = data, x_shape=data[1].x.shape, a_shape
   =data.a.shape, out=data.n_labels, hidden = hidden_channels)
optimizer = keras.optimizers.Nadam()
model.compile(loss="binary_crossentropy")
loss_fn = keras.losses.BinaryCrossentropy()
m1 = keras.metrics.AUC()
```

```
# Train/valid/test split
np.random.seed(42)
np.random.shuffle(data)
tr_split = int(np.floor(data.n_graphs * train_prop))
data_tr, data_te = data[:tr_split], data[tr_split:]
va_split = int(np.floor(data_tr.n_graphs * valid_prop))
data_tr, data_va = data_tr[:va_split], data_tr[va_split:]


# We use a MixedLoader since the dataset is in mixed mode
loader_tr = spektral.data.MixedLoader(data_tr, batch_size=batch_size,
   epochs=epochs)
loader_va = spektral.data.MixedLoader(data_va, batch_size=batch_size)
loader_te = spektral.data.MixedLoader(data_te, batch_size=batch_size)


# Training function
@tf.function
def train_on_batch(inputs, target):
    with tf.GradientTape() as tape:
        predictions = model(inputs, training=True)
        loss = loss_fn(target, predictions) + sum(model.losses)
    gradients = tape.gradient(loss, model.trainable_variables)
    optimizer.apply_gradients(zip(gradients, model.trainable_variables))
    return loss


# Evaluation function
def evaluate(loader):
    step = 0
    results = []
    acc = []
    auc_res = []
    for batch in loader:
        step += 1
        inputs, target = batch
        predictions = model(inputs, training=False)
        predictions = tf.squeeze(predictions)
        loss = loss_fn(target, predictions)

        # acc1 = keras.metrics.binary_accuracy(target, predictions).numpy
           ()
        # acc.append((acc1.sum()) / (acc1.__len__()))
        auc_res.append(roc_auc_score(target, predictions))
        results.append(loss)
        if step == loader.steps_per_epoch:
            test = np.array([])
```

```
            # print(sum(acc) / len(acc))
            print(sum(auc_res)/len(auc_res))
            for elem in results:
                test = np.concatenate((tf.expand_dims(elem,0).numpy(), test
                    ))
            return np.average(test, 0, weights=test)



# Setup training
best_val_loss = 9999999
patience = 10
current_patience = patience
step = 0
ep = 0

# Training loop
results_tr = []
results_te = None
history_loss = []
for batch in loader_tr:
    step += 1
    # Training step
    inputs, target = batch
    loss = train_on_batch(inputs, target)
    results_tr.append((loss))

    if step == loader_tr.steps_per_epoch:
        results_va = evaluate(loader_va)
        ep += 1
        if results_va < best_val_loss:
            best_val_loss = results_va
            current_patience = patience
            results_te = evaluate(loader_te)
        else:
            current_patience -= 1
            if current_patience == 0:
                print("Early␣stopping")
                break
        # Print results
        results_tr = np.array(results_tr, dtype=object)
        test = np.array([])
        for elem in results_tr:
            test = np.concatenate((tf.expand_dims(elem,0).numpy(), test))
        results_tr = np.average(test, 0, weights=test)
        history_loss.append(results_tr)
        print("Step:␣", ep, "\n"
             "Train␣loss:␣", results_tr, "␣|␣\n",
```

```
            "Valid␣loss:␣", results_va, "␣|␣\n",
            "Test␣loss:␣", results_te, "\n\n"
            )
        # Reset epoch
        results_tr = []
        step = 0
```

B.2.7 Script d'entrainement pour de la regression sous Spektral

```
 import torch
import torch.nn.functional as F
from torch.nn import Linear
from sklearn.model_selection import KFold
from torch import nn
from torch_geometric.data import DataLoader
from torch_geometric.nn import ChebConv
from torch_geometric.nn import max_pool_neighbor_x
from data_process_geometric_breast_cancer import BreastCancerPytorch

batch_size = 32
train_prop = 0.9
valid_prop = 0.9
hidden_channels = 128
learning_rate = 0.001
num_epochs = 4
loss_function = nn.CrossEntropyLoss()
results = {}

dataset = BreastCancerPytorch(root='data/breast_cancer')


class Net(torch.nn.Module):
    def __init__(self, hidden_channels):
        super(Net, self).__init__()
        torch.manual_seed(12345)
        self.conv1 = ChebConv(dataset.num_node_features, 32, K=8)
        self.conv2 = ChebConv(32, 512, K=8)
        self.lin1 = Linear(512, 128)
        self.lin2 = Linear(128, 2)

    def forward(self, x, edge_index, batch):
        # 1. Obtain node embeddings
        x = self.conv1(x, edge_index)
        x = x.relu()
        x = self.conv2(x, edge_index)
        x = x.relu()
        x = max_pool_neighbor_x(x)
        # 3. Apply a final classifier
```

```
        x = self.lin1(x)
        x = x.relu()
        x = self.lin2(x)
        return x


def reset_weights(m):
    """
    Try resetting model weights to avoid
    weight leakage.
  """
    for layer in m.children():
        if hasattr(layer, 'reset_parameters'):
            print(f'Reset␣trainable␣parameters␣of␣layer␣=␣{layer}')
            layer.reset_parameters()


kfold = KFold(n_splits=10, shuffle=True)

# Start print
print('--------------------------------')

# K-fold Cross Validation model evaluation
for fold, (train_ids, test_ids) in enumerate(kfold.split(dataset)):

    # Print
    print(f'FOLD␣{fold}')
    print('--------------------------------')

    # Define data loaders for training and testing data in this fold
    trainloader = DataLoader(dataset[train_ids.tolist()], batch_size=
       batch_size, shuffle=True)
    testloader = DataLoader(dataset[test_ids.tolist()], batch_size=
       batch_size)

    # Init the neural network
    model = Net(hidden_channels=hidden_channels).double()
    model.apply(reset_weights)

    # Initialize optimizer
    optimizer = torch.optim.Adam(model.parameters(), lr=1e-4)

    # Run the training loop for defined number of epochs
    for epoch in range(0, num_epochs):

        # Print epoch
        print(f'Starting␣epoch␣{epoch␣+␣1}')
```

```
    # Set current loss value
    current_loss = 0.0

    # Iterate over the DataLoader for training data
    for i, data in enumerate(trainloader, 0):
        # Get inputs
        targets = data.y

        # Zero the gradients
        optimizer.zero_grad()

        # Perform forward pass
        outputs = model(data.x, data.edge_index, data.batch)

        # Compute loss
        loss = loss_function(outputs, targets)

        # Perform backward pass
        loss.backward()

        # Perform optimization
        optimizer.step()

# Process is complete.
print('Training process has finished. Saving trained model.')

# Saving the model
save_path = f'./model-fold-{fold}.pth'
torch.save(model.state_dict(), save_path)

# Print about testing
print('Starting testing')

# Evaluation for this fold
correct, total = 0, 0
with torch.no_grad():

    # Iterate over the test data and generate predictions
    for i, data in enumerate(testloader, 0):
        # Get inputs
        targets = data.y

        # Generate outputs
        outputs = model(data.x, data.edge_index, data.batch)

        # Set total and correct
        _, predicted = torch.max(outputs.data, 1)
        total += targets.size(0)
```

```
            correct += (predicted == targets).sum().item()

        # Print accuracy
        print('Accuracy for fold %d: %d %%' % (fold, 100.0 * correct /
            total))
        print('--------------------------------')
        results[fold] = 100.0 * (correct / total)

    # Print fold results
print(f'K-FOLD CROSS VALIDATION RESULTS FOR 10 FOLDS')
print('--------------------------------')
somme = 0.0
for key, value in results.items():
    print(f'Fold {key}: {value} %')
    somme += value
print(f'Average: {somme / len(results.items())} %')
```

B.2.8 Comparaison avec d'autres méthodes de machine learning

Comparison_machine_learning

August 24, 2021

[1]:
```
import pandas as pd
import os

from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
import spektral
import numpy as np

from sklearn.metrics import confusion_matrix
from matplotlib import pyplot as plt
from seaborn import heatmap
```

[2]:
```
pseudoCount = pd.read_csv("data/breast_cancer/raw/GEO_HG_PPI.csv")
labels = pd.read_csv("data/breast_cancer/raw/labels_GEO_HG.csv").transpose().
 ↪to_numpy().ravel()

pseudoCount.index = list(pseudoCount.pop("probe"))
pseudoCount = pseudoCount.transpose()

##############################################
## Standardization of nodes feature (pseudoCount)
scaler = StandardScaler()
pseudoCount_scale = scaler.fit_transform(pseudoCount)
pseudoCount_scale = pd.DataFrame(pseudoCount_scale)

X_train, X_test, y_train, y_test = train_test_split(
    pseudoCount_scale, labels, stratify=labels, random_state=42, shuffle = True)
```

1 RandomForestClassifier

[3]:
```
param_grid = [{'n_estimators': [50, 500, 5000]}]
rnd_clf = RandomForestClassifier()

grid_search = GridSearchCV(rnd_clf, param_grid, cv=10, scoring='accuracy',␣
 ↪n_jobs = -1, verbose = 3)
grid_search.fit(X_train, y_train)
```

```
Fitting 10 folds for each of 3 candidates, totalling 30 fits
```

[3]:
```
GridSearchCV(cv=10, estimator=RandomForestClassifier(), n_jobs=-1,
             param_grid=[{'n_estimators': [50, 500, 5000]}], scoring='accuracy',
             verbose=3)
```

[4]:
```
grid_search.best_estimator_
```

[4]:
```
RandomForestClassifier(n_estimators=5000)
```

[5]:
```
y_pred_rf = grid_search.predict(X_test)

accuracy_score(y_test, y_pred_rf)
```

[5]:
```
0.757201646090535
```

[6]:
```
roc_auc_score(y_test, y_pred_rf)
```

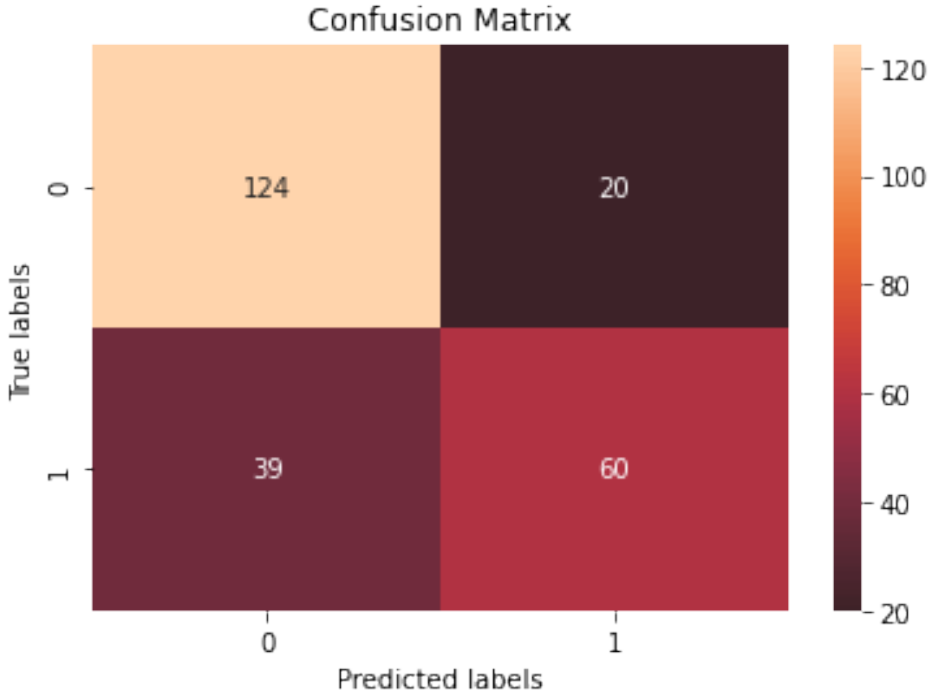
[6]:
```
0.7335858585858586
```

[7]:
```
pd.DataFrame(y_pred_rf).describe()
```

[7]:

	0
count	243.000000
mean	0.329218
std	0.470899
min	0.000000
25%	0.000000
50%	0.000000
75%	1.000000
max	1.000000

[8]:
```
tn, fp, fn, tp = confusion_matrix(y_test, y_pred_rf).ravel()
ax= plt.subplot()
heatmap([[tn, fp], [fn, tp]], annot=True, fmt='g', ax=ax, center=1)
ax.set_xlabel('Predicted labels');ax.set_ylabel('True labels');
ax.set_title('Confusion Matrix');
ax.xaxis.set_ticklabels(['0', '1']); ax.yaxis.set_ticklabels(['0', '1']);
```



2 ExtraTreesClassifier

[9]:
```
param_grid = [{'n_estimators': [50, 500, 5000]}]
rnd_clf = ExtraTreesClassifier()

grid_search = GridSearchCV(rnd_clf, param_grid, cv=10, scoring='accuracy',␣
 ↪n_jobs = -1, verbose = 3)
grid_search.fit(X_train, y_train)
```

```
Fitting 10 folds for each of 3 candidates, totalling 30 fits
```

[9]:
```
GridSearchCV(cv=10, estimator=ExtraTreesClassifier(), n_jobs=-1,
             param_grid=[{'n_estimators': [50, 500, 5000]}], scoring='accuracy',
             verbose=3)
```

[10]:
```
grid_search.best_estimator_
```

[10]:
```
ExtraTreesClassifier(n_estimators=5000)
```

[11]:
```
y_pred_rf = grid_search.predict(X_test)

accuracy_score(y_test, y_pred_rf)
```

[11]:
```
0.7530864197530864
```

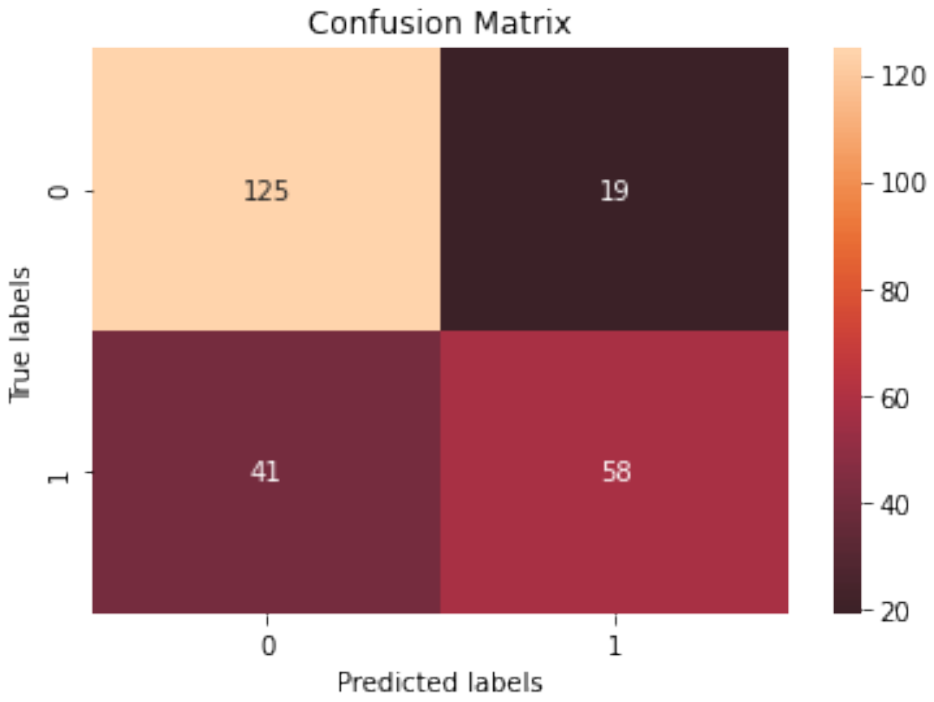
[12]:
```
roc_auc_score(y_test, y_pred_rf)
```

[12]:
```
0.7269570707070707
```

[13]:
```
pd.DataFrame(y_pred_rf).describe()
```

[13]:
```
                0
count  243.000000
mean     0.316872
std      0.466218
min      0.000000
25%      0.000000
50%      0.000000
75%      1.000000
max      1.000000
```

[14]:
```
tn, fp, fn, tp = confusion_matrix(y_test, y_pred_rf).ravel()
ax= plt.subplot()
heatmap([[tn, fp], [fn, tp]], annot=True, fmt='g', ax=ax, center=1)
ax.set_xlabel('Predicted labels');ax.set_ylabel('True labels');
ax.set_title('Confusion Matrix');
ax.xaxis.set_ticklabels(['0', '1']); ax.yaxis.set_ticklabels(['0', '1']);
```



3 Ridge regression

[15]:
```
from sklearn.linear_model import RidgeClassifier

param_grid = [{'alpha': [0.1, 1]}]
rdg_clf = RidgeClassifier()

grid_search_rdg = GridSearchCV(rdg_clf, param_grid, cv=10, scoring='accuracy',␣
 ↪n_jobs = -1, verbose = 3)
grid_search_rdg.fit(X_train, y_train)
```

```
Fitting 10 folds for each of 2 candidates, totalling 20 fits
```

[15]:
```
GridSearchCV(cv=10, estimator=RidgeClassifier(), n_jobs=-1,
             param_grid=[{'alpha': [0.1, 1]}], scoring='accuracy', verbose=3)
```

[35]:
```
grid_search_rdg.best_params_
```

[35]:
```
{'alpha': 0.1}
```

[17]:
```
y_pred_ridge = grid_search_rdg.predict(X_test)

accuracy_score(y_test, y_pred_ridge)
```

[17]:
```
0.7654320987654321
```

[18]:
```
roc_auc_score(y_test, y_pred_ridge)
```

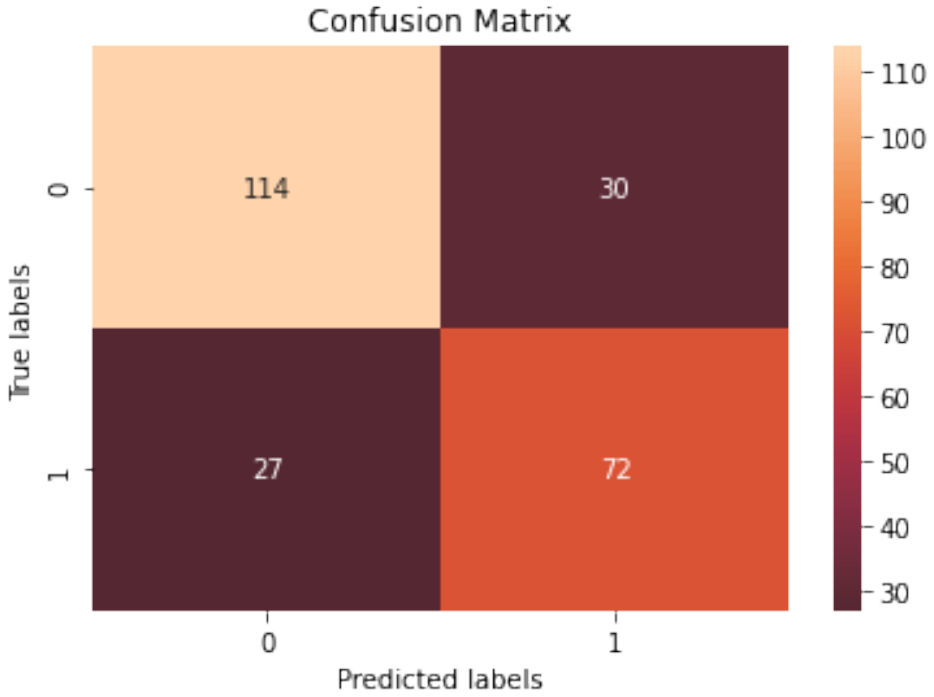
[18]:
```
0.759469696969697
```

[19]:
```
pd.DataFrame(y_pred_ridge).describe()
```

[19]:
```
                0
count  243.000000
mean     0.419753
std      0.494537
min      0.000000
25%      0.000000
50%      0.000000
75%      1.000000
max      1.000000
```

[20]:
```
tn, fp, fn, tp = confusion_matrix(y_test, y_pred_ridge).ravel()
ax= plt.subplot()
heatmap([[tn, fp], [fn, tp]], annot=True, fmt='g', ax=ax, center=1)
```

```
ax.set_xlabel('Predicted labels');ax.set_ylabel('True labels');
ax.set_title('Confusion Matrix');
ax.xaxis.set_ticklabels(['0', '1']); ax.yaxis.set_ticklabels(['0', '1']);
```



4 Perceptron

[30]:
```
from sklearn.linear_model import Perceptron

param_grid = [{'penalty' : ['l1'],'alpha': [0.0001, 0.001, 0.01, 0.1, 1]}]
pcp_clf = Perceptron()

grid_search_pcp = GridSearchCV(pcp_clf, param_grid, cv=10, scoring='accuracy',␣
 ↪n_jobs = -1, verbose = 3)
grid_search_pcp.fit(X_train, y_train)
```

```
Fitting 10 folds for each of 5 candidates, totalling 50 fits
```

[30]:
```
GridSearchCV(cv=10, estimator=Perceptron(), n_jobs=-1,
             param_grid=[{'alpha': [0.0001, 0.001, 0.01, 0.1, 1],
                          'penalty': ['l1']}],
             scoring='accuracy', verbose=3)
```

[34]:
```
grid_search_pcp.best_params_
```

[34]:
```
{'alpha': 0.0001, 'penalty': 'l1'}
```

[23]:
```
y_pred_perceptron = grid_search_pcp.predict(X_test)

accuracy_score(y_test, y_pred_perceptron)
```

[23]:
```
0.757201646090535
```

[24]:
```
roc_auc_score(y_test, y_pred_perceptron)
```

[24]:
```
0.7683080808080808
```

[25]:
```
tn, fp, fn, tp = confusion_matrix(y_test, y_pred_perceptron).ravel()
ax= plt.subplot()
heatmap([[tn, fp], [fn, tp]], annot=True, fmt='g', ax=ax, center=1)
ax.set_xlabel('Predicted labels');ax.set_ylabel('True labels');
ax.set_title('Confusion Matrix');
ax.xaxis.set_ticklabels(['0', '1']); ax.yaxis.set_ticklabels(['0', '1']);
```