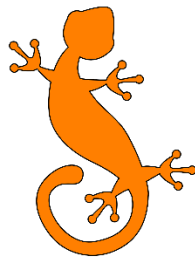


PROJET DE GROUPE

IGUANA



Réalisateurs : *Khalil Boulkenafet, Jinhui Liu, Pierre Le Jeune, Jules Paris
et Justin Voïnéa*

Encadrants : *Carito Guziolowski et Bertrand Miannay*

Remerciements

Nous tenons à remercier Mme Carito Guziolowski et M. Bertrand Miannay pour nous avoir accompagné pendant toute la durée de ce projet.

Nous tenons également à remercier l'ensemble du corps enseignant qui s'est toujours montré disponible et réactif pour nous conseiller.

Nous remercions enfin Julien Simon, expert en Machine Learning chez AWS pour ses conseils avisés.

TABLE DES MATIERES

| | |
|--|-----------|
| 1. Présentation du projet | 5 |
| 1.1 Contexte | 5 |
| 1.2 Iguana au début du projet | 6 |
| 1.3 Fonctionnalités attendues | 7 |
| 1.3.1 Calcul de similarité | 7 |
| 1.3.2 Classificateur | 8 |
| 1.3.3 Portage MacOS | 8 |
| 2. Planification et organisation | 9 |
| 2.1 Planification des tâches | 9 |
| 2.2 Organisation de l'équipe | 9 |
| 3. Environnement | 10 |
| 3.1 Environnement de développement | 10 |
| 3.2 Création de l'interface graphique | 10 |
| 4. Calcul de similarité | 13 |
| 4.1 Principe du calcul de similarité | 13 |
| 4.2 Algorithme | 13 |
| 4.3 Implémentation dans Iguana | 14 |
| 4.4 Visualisation des scores de similarité | 16 |
| 4.4.1 Contexte général | 16 |
| 4.4.2 Création du graphe des composants | 17 |
| 4.4.3 Création du graphe de similarité dans Cytoscape | 18 |
| 5. Classificateur/Prédiction | 19 |
| 5.1 Choix de l'algorithme et fonctionnement | 19 |
| 5.1.1 Fonctionnement de l'algorithme | 19 |
| 5.1.2 Choix de l'algorithme | 21 |
| 5.2 Phase de test | 21 |
| 5.2.1 Tests avant implémentation | 21 |
| 5.2.2 Tests sur les données et pistes d'amélioration du modèle | 23 |
| 5.3 Implémentation dans Iguana | 25 |

| | | |
|-----------|---|-----------|
| 5.3.1 | Module de création | 25 |
| 5.3.2 | Module de test | 26 |
| 5.3.3 | Module de prédiction | 28 |
| 6. | Création d'un exécutable Windows | 29 |
| 6.1 | L'exécutable | 29 |
| 6.2 | Rédaction d'un guide d'utilisation | 29 |
| 7. | Version Mac OS | 30 |
| 7.1 | Portage vers plateforme Mac OS | 30 |
| 7.2 | Les difficultés rencontrées | 33 |
| 8. | Conclusion | 35 |
| 9. | Annexes | 37 |

1. PRESENTATION DU PROJET

1.1 Contexte

Lors de ses travaux de doctorant, Bertrand Miannay a travaillé, entre autres, sur une méthode d'identification des sous-réseaux par colorations dites parfaites de graphes. Cette méthode permet, à partir d'un réseau modélisé sous forme de graphe, d'identifier les ensembles de nœuds dont les signes seront corrélés entre eux et de les assembler en sous graphes qui sont appelés composants. Dans sa thèse, il démontre l'efficacité de cette méthode pour identifier différents profils de patients atteints du myélome multiple. Dans ce modèle, les nœuds représentent des protéines, des gènes, des complexes ou autres phénomènes biologiques et les arcs représentent les interactions qu'il peut y avoir entre eux (i.e. l'activation ou l'inhibition). L'algorithme permettant d'identifier ces composants a été baptisé IGGY-poc.

Cet algorithme d'identification des colorations de graphes pouvant demander un temps de calcul important, il a également mis au point un algorithme de réduction de graphe permettant de diminuer considérablement le temps d'exécution nécessaire afin d'obtenir un résultat exploitable avec IGGY-poc. En effet, le temps d'exécution de l'algorithme d'identification des composants passe de 1h30 à 10 min une fois le graphe réduit sur l'exemple utilisé par M. Miannay dans sa thèse. Ce dernier quant à lui est composé de plusieurs scripts permettant à la fois d'identifier les colorations mais aussi de pouvoir reconstruire le graphe à partir du graphe réduit et des colorations trouvées.

Trouver des colorations dans des graphes se révèle très utile en pratique, car cela conduit à la détection de composants au sein du graphe. C'est-à-dire à des ensembles de nœuds qui dépendent les uns des autres de manière indépendante du reste du graphe.

Dans le cadre du myélome multiple, avec la représentation sous forme de graphe des gènes qui ont un rôle dans le développement de cette pathologie, on peut donc modéliser l'état d'un patient en ne regardant que les composants du graphe et non l'ensemble des gènes qui le composent. Ainsi, M. Miannay a créé des modèles de classification, permettant, à partir des niveaux d'expression de certains gènes, de faire des prédictions sur l'état du patient. Ces prédictions sont binaires, elles indiquent si un patient est à risque ou non. Grâce à cela, il sera possible de déterminer plus rapidement la gravité d'un cas. Les traitements qui lui seront prescrits seront d'autant plus efficaces.

Une interface graphique baptisée Iguana a été réalisée dans le cadre d'un projet d'application cette année. Elle permettait d'implémenter les premières parties du travail de M. Miannay, à savoir, l'affichage de graphes, la réduction de graphes, et la recherche de composants au sein de graphes. Ce projet s'inscrit dans la continuité d'Iguana. Notre objectif sera donc d'intégrer les nouvelles fonctionnalités à Iguana, afin de pouvoir réaliser l'ensemble du processus défini par M. Miannay, de la réduction du graphe à la prédiction de l'état d'un patient.

1.2 Iguana au début du projet

Comme nous l'avons évoqué précédemment, Iguana est une interface graphique permettant de réaliser des opérations sur des graphes. Pour cela, l'application se décompose en trois onglets.

- Un onglet d'affichage, qui permet de charger des graphes au format .sif et de les afficher dans Cytoscape. Cytoscape est une application externe qui permet de réaliser l'affichage de tout type de graphes (cf. Annexe 1).

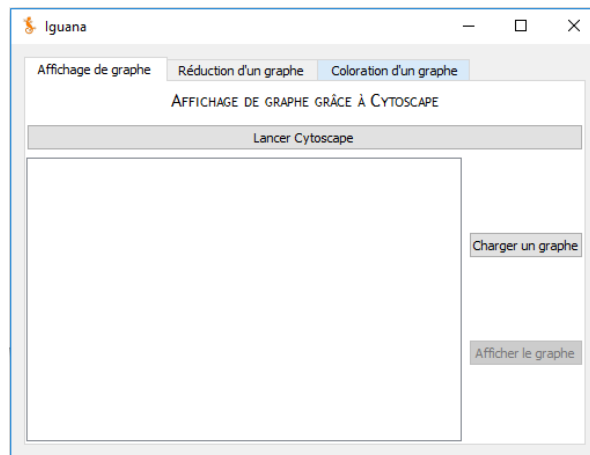


Figure 1: Onglet d'affichage de Iguana (version 1)

- Un onglet de réduction, qui permet de charger également des graphes et de les réduire. Pour cela, Iguana lance un des scripts d'IGGY-poc, basés à partir des travaux de M. Miannay. Pour plus de détails, se reporter au rapport du projet d'application, disponible sur le git de déploiement (cf. Annexe 2).

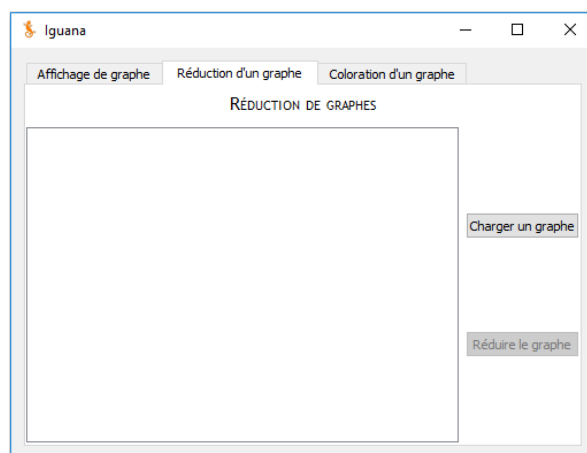


Figure 2 : Onglet de réduction de Iguana (version 1)

- Un onglet d'identification des colorations. Cet onglet est le plus complet, il comprend d'abord l'identification des colorations. Pour cela, Iguana utilise un programme logique et le solveur ASP clingo pour trouver les colorations. Ensuite, un autre script Python identifie les composants à partir des colorations. On peut alors choisir d'afficher les différents composants dans Cytoscape grâce à un code couleur.

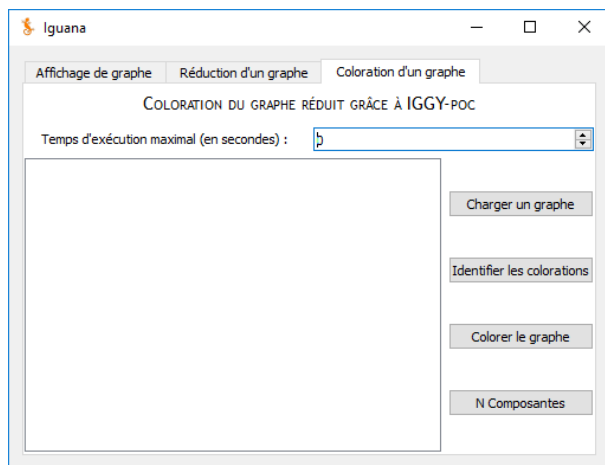


Figure 3 : Onglet de coloration de Iguana (version 1)

1.3 Fonctionnalités attendues

1.3.1 Calcul de similarité

La première fonctionnalité attendue était la réalisation d'un module permettant de calculer un score pour chaque composant du graphe, à partir des niveaux d'expression des gènes pour un patient. Pour cela, nous devons nous baser sur les travaux de M. Miannay, qui avait déjà implémenté ce *scoring* en Python et en bash. Il convenait donc surtout de l'intégrer à Iguana de manière cohérente.

Une autre fonctionnalité qui nous a été demandée au début de ce projet était de créer une fonction permettant de visualiser le résultat du calcul de similarité. Pour cela, plusieurs solutions ont été envisagées et discutées avec nos encadrants.

- Une heatmap : il s'agit d'une représentation en couleur des différentes interactions ou corrélations entre les gènes. Nous aurions pu adapter ce principe pour représenter la similarité de chaque composant. Cependant il n'y a pas vraiment de corrélation entre les différents composants et donc le principe de la heatmap perd un peu de son sens. C'est pourquoi nous avons opté pour la deuxième solution.

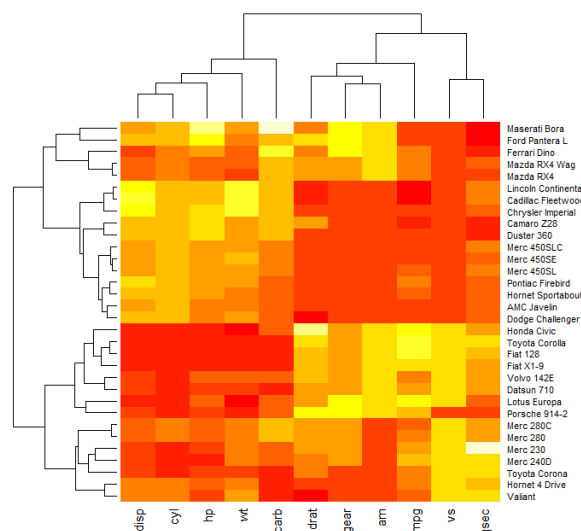


Figure 4 : Exemple d'une heatmap montrant la classification de modèles de voiture selon leurs caractéristiques

- Le graphe des composants : on crée un nœud par composant dans un graphe et on donne une couleur à chaque nœud en fonction de son score de similarité. Ensuite il suffit d'ajouter les liens entre les composants. Cette solution a l'avantage de montrer également l'influence des composants entre eux, même si ces influences ne sont pas aussi claires qu'entre deux nœuds simples d'un graphe.

1.3.2 Classificateur

Cette partie est la suite directe du calcul de similarité. En effet, une fois les scores calculés pour chaque composant, nous pouvons exploiter les données. Pour cela, l'idée de départ était de réaliser un classificateur à partir de ces scores grâce à un algorithme de *Random Forest*. Ce classificateur sera directement entraîné à partir des fichiers générés par le calcul de similarité. L'objectif de ce classificateur est évidemment de donner les réponses à la question : ce(s) patient(s) est(sont)-il(s) à risque ? Mais on doit également pouvoir le tester sur des jeux de données de validation.

Nous devons également essayer d'améliorer les résultats du classificateur, notamment en jouant sur les différents paramètres de création de celui-ci.

1.3.3 Portage MacOS

La dernière partie de notre projet a été le portage d'Iguana sur Mac OS. Cela voulait dire reprendre l'ensemble du code réalisé pendant la première partie du projet (PAPPL) et apporter les modifications nécessaires au passage sous MacOS. Mais aussi, une fois les autres développements réalisés sous Windows, les intégrer dans la version Mac OS.

2. PLANIFICATION ET ORGANISATION

2.1 Planification des tâches

Notre premier contact avec ce sujet a été lors d'une réunion avec Mme Guziolowski. Lors de cette réunion, elle nous a présenté le sujet ainsi que tous les outils mis à notre disposition pour le réaliser soit :

- Le code source d'Iguana (version 1)
- Les scripts python et R de Bertrand Miannay
- Les données utilisées par M Miannay

Suite à cette réunion nous avons ensuite défini les objectifs afin de s'assurer que nous avions bien intégré les attentes de nos encadrants.

Suite à cette réunion nous avons pu procéder à la rédaction du cahier des charges (cf. Annexe 3). Ce cahier des charges a permis de formaliser et découper les tâches à réaliser, puis de planifier le projet à l'aide d'un Gantt. Cette planification a d'ailleurs été bien respectée tout au long du projet, en partie parce que deux membres du groupe avaient déjà travaillé sur Iguana pendant le projet d'application et que ceux-ci connaissaient donc très bien le sujet.

2.2 Organisation de l'équipe

Sur ce projet nous avons travaillé avec une méthode agile. Pour cela, nous avons convenu de faire des réunions toutes les deux semaines avec les clients pour leur présenter notre état d'avancement. A chaque réunion nous faisons en sorte de présenter un prototype fonctionnel lors de la réunion, un peu à la manière de Scrum. Lors de ces réunions, nous définissons également les avancements à réaliser pendant les semaines suivantes, nous parlons des difficultés rencontrées et comment nous y faisons face. La bonne communication au sein du groupe et avec les clients nous a permis d'avancer assez rapidement puisque lorsque l'un de nous était indisponible, un autre pouvait prendre le relai.

En général nous séparons les tâches de manière à pouvoir avancer en parallèle, mais nous avons toujours conservé un regard sur ce que faisaient les autres de sorte à pouvoir s'entraider sans perdre trop de temps à rentrer dans le travail des autres le cas échéant.

Nous avons également utilisé Git pour partager notre code. Ça a été un peu difficile au début de tous prendre le réflexe de bien mettre son code sur le git en faisant les *commit* et les *push*, mais après un court temps d'adaptation, nous nous sommes habitués à cet environnement de travail.

3. ENVIRONNEMENT

3.1 Environnement de développement

L'environnement de travail était le même que lors de la première partie du projet. Ainsi nous n'avons pas eu de difficultés à installer les outils de développement sur Windows. Nous travaillions sous Python 3.6 avec un ensemble de bibliothèques (cf. Annexe 4) nous permettant aussi bien d'utiliser une interface graphique que de faire le lien avec Cytoscape ou encore d'utiliser les algorithmes de Machine Learning déjà présents dans ces bibliothèques.

C'est d'ailleurs sur le conseil d'un intervenant travaillant dans le domaine du Machine Learning que nous avons choisi de travailler avec XGBoost (cf. Annexe 5), dont la bibliothèque homonyme est disponible pour Python.

La plus grande difficulté a été d'arriver à travailler avec les membres de l'équipe possédant des systèmes MacOS. En effet, l'application ne fonctionnant pas sous MacOS au début du projet, il était difficile pour eux de faire des essais sur l'application directement. Ils ont dû se contenter de faire des essais sur des scripts sur leur ordinateur personnel puis d'utiliser un PC Windows afin de pouvoir faire les tests dans l'application et d'intégrer leur code. Par la suite, la version MacOS a commencé à être développée et il a été plus facile de travailler chacun sur nos propres ordinateurs.

En ce qui concerne Cytoscape, une version pour MacOS existe, il n'y a donc pas eu de problèmes à ce niveau-là. Il fallait juste faire attention à choisir la bonne version pour que tout le monde travaille là-aussi avec la même version.

3.2 Création de l'interface graphique

L'interface graphique avait déjà été bien définie lors du projet d'application du premier semestre. Elle avait été pensée de manière à faire avancer l'utilisateur dans les onglets au fur et à mesure qu'il avance dans les processus. Nous avons donc choisi de continuer sur cette lancée. Nous voulions l'interface la plus claire et intuitive possible. Cette interface épurée et instinctive les avait vraiment séduits et il paraissait logique de poursuivre. Cependant, de nombreuses fonctionnalités étaient à rajouter et il a été difficile de garder la simplicité que pouvaient présenter nos premiers onglets.

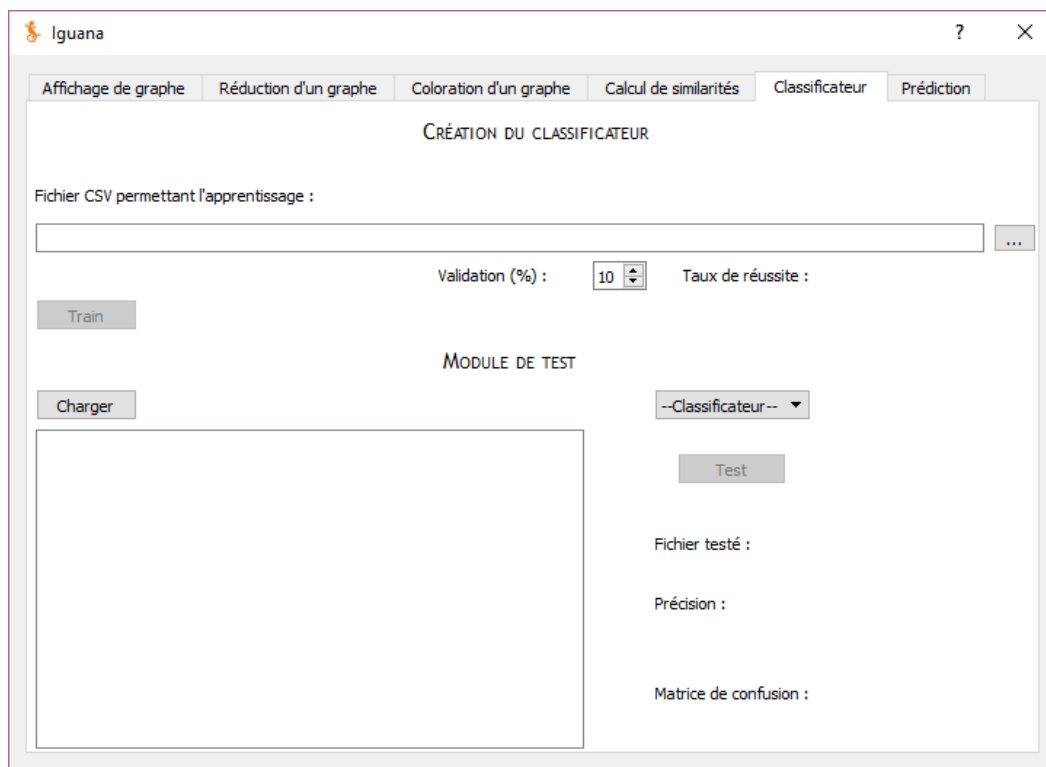


Figure 5 : Onglet de classification

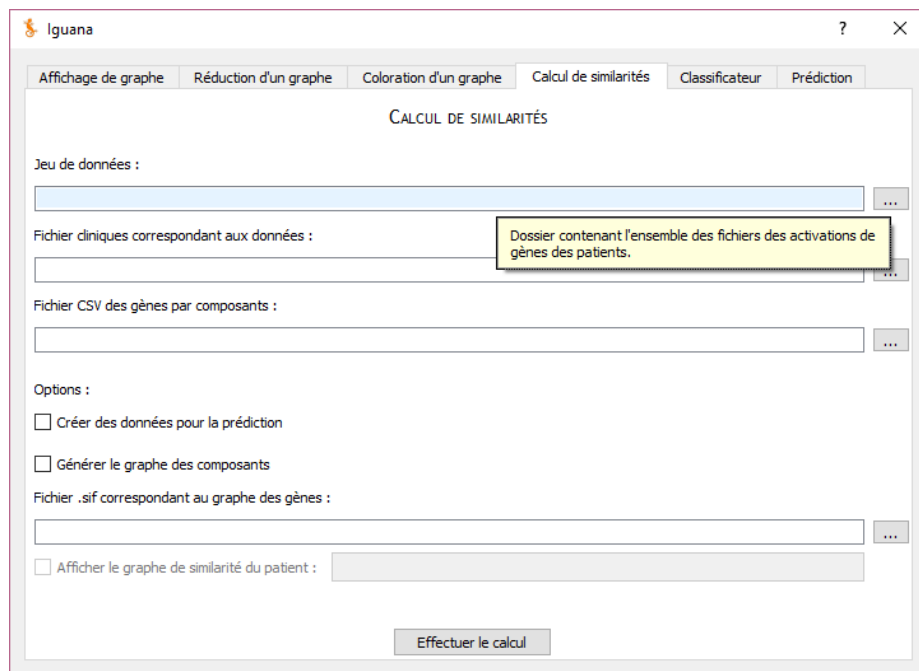
Le développement de cette interface a été effectué sur QtDesigner, une application permettant de *designer* l'interface en plaçant tous les boutons de manière précise et simple permettant de faire des interfaces assez complexes de manière simplifiée. Cela évite par exemple de devoir placer tous les boutons en codant précisément leur emplacement et sans avoir à recompiler le code souvent afin d'avoir un aperçu.

Depuis la première version de l'interface dont nous disposons au début de ce projet, trois onglets ont été ajoutés :

- Calcul de similarité : réalise le calcul de similarité entre les données des patients et les composants du graphe de gènes
- Génération du classificateur : à partir du résultat du calcul de similarité, crée un classificateur grâce aux données d'entraînement que peut fournir l'utilisateur
- Prédiction : en utilisant un des classificateurs créés, émet des prédictions quant à l'état des patients placés en entrée

Ces fonctionnalités nécessitent de nombreux boutons et de zones d'informations, ainsi il a été assez compliqué de rendre ces onglets aussi intuitifs que les premiers onglets réalisés lors du précédent projet. Suite à une discussion avec nos encadrants, nous avons choisi d'ajouter un mode d'aide dans l'application permettant de préciser le fonctionnement de chacun des boutons et la fonction des zones de textes présents dans l'application. Nous en avons profité pour ajouter ce mode pour les onglets déjà réalisés. Si l'utilisateur désire des informations sur un bouton présent sur l'interface, il lui suffit de cliquer sur le point

d'interrogation présent dans la barre avec la croix rouge et ensuite de cliquer sur la zone dont il souhaite recevoir une information.



Iguana ? X

Affichage de graphe Réduction d'un graphe Coloration d'un graphe **Calcul de similarités** Classificateur Prédiction

CALCUL DE SIMILARITÉS

Jeu de données :

Fichier cliniques correspondant aux données : Dossier contenant l'ensemble des fichiers des activations de gènes des patients.

Fichier CSV des gènes par composants :

Options :

☐ Créer des données pour la prédiction

☐ Générer le graphe des composants

Fichier .sif correspondant au graphe des gènes :

☐ Afficher le graphe de similarité du patient :

Effectuer le calcul

Figure 6 : Onglet de similarité

Dans l'exemple ci-dessus, il est indiqué brièvement quel type de données l'utilisateur doit placer dans ce champ.

Cette interface a beaucoup bougé pendant ces trois mois. En effet, nous profitons des réunions régulières avec nos clients pour leur proposer nos idées et leur montrer ce que nous avons choisi de faire. En fonction de leurs retours, nous modifions ou non l'interface. De plus, les fonctionnalités ont été ajoutées au fur et à mesure. Ainsi, il est arrivé de revenir un peu en arrière pour modifier certains points déjà fixés de l'interface car l'implémentation des nouvelles options n'était pas commode à ce niveau.

4. CALCUL DE SIMILARITE

Pour cette fonctionnalité, nous devons réaliser un module permettant de calculer un score de similarité pour chaque composant du graphe, à partir des informations d'activation des gènes de patients.

4.1 Principe du calcul de similarité

Les données génétiques de M individus sont contenues dans une matrice de taille $n * M$. Pour chacun de ces individus, n gènes sont donc concernés. Ces données représentent l'état génétique du patient. De plus, un diagnostic a été établi pour ces patients. Nous disposons également d'un graphe représentant les interactions entre N gènes du génome humain ($N \geq n$). Celui-ci a été réduit, sans perte d'information, en k sous-graphes de taille inférieure plus facilement exploitables.

Il va donc s'agir d'appliquer, pour chaque individu, l'algorithme de similarité entre les n gènes (qui correspondent à n nœuds du graphe dans un état particulier) et les k sous-graphes. Nous obtenons alors une matrice $k * M$ dont chaque colonne représente la similarité c'est-à-dire, l'écart entre l'expression des gènes du patient et l'expression théorique contenue dans un composant.

4.2 Algorithme

L'algorithme de cette fonctionnalité de *scoring* est tiré des travaux de M. Miannay sur lesquels nous nous sommes appuyés.

On récupère les observations sur un patient à partir du fichier d'expression de ses gènes. L'information de l'expression de chaque gène est un nombre réel entre 0 et 1.

```
NR2E3 = 0.346850229182481
CDH3 = 0.537973022322387
TANK = 0.319874882900231
CD24 = 0.17184001130133
HDAC5 = 0.424391832314019
BCL2L11 = 0.243834496876767
HCN4 = 0.397148868725062
```

Figure 7 : Exemple de niveaux d'expression de gènes dans un fichier patient

On récupère ensuite à partir d'un second fichier les composants du graphe réduit représentant les interactions entre différents gènes du génome humain. Chaque gène est suivi d'un signe qui traduit son activation au sein de la coloration parfaite trouvée par Iguana.

```

1 ABCB1 +, ABL1 +, ACHE -, ACKR3 +, ADAMTS1 +, ADRB2 +, AHR +, AHRH +, AIP +, AKR1C2 +, AKT2 +, ALDH1A1 +, ALOX15 +, ALPI +, APC +, APEX1 +, AR +, AREG +, ARNTL2 -, ASF1B
-, ASPM -, ATF4 +, ATF5 -, ATM -, AURKA -, AXL -, B4GALT1 -, BAMBI -, BAX +, BCL2 -, BCL2L1 +, BCL6 +, BIRC5 -, BLK -, BMI1 +, BMP2 +, BRCA1 -, BRCA2 -, BTG2 -, CA9
-, CALB1 -, CASP8 +, CCL17 +, CCL26 +, CCNA1 -, CCNA2 -, CCNB1 -, CCNB2 -, CCND1 -, CCND2 -, CCND3 -, CCNE1 -, CCNG2 +, CCR4 -, CCR7 -, CD14 -, CD19 +, CD4 +, CD40
+, CD44 +, CD46 +, CD79A +, CDC20 -, CDC25B -, CDC42 -, CDC47 -, CDH1 -, CDH3 +, CDK1 -, CDKN1A +, CDKN1B -, CDKN2B +, CDKN3 +, CD01 -, CD01 -, CD01 -, CEBPB +, CHEK1
-, CIITA +, CLEC4G -, CNR1 +, COL10A1 -, COL1A1 -, COL1A2 -, CREB1 +, CREM +, CRH -, CRX +, CSF3R +, CST5 -, CTBP1 +, CTSP1 +, CTSD +, CXCL1 +, CXCL8 -, CXCR4
+, CYP11B2 -, CYP19A1 -, CYP19A1 -, CYP2E1 +, CYP3A4 -, CYP7A1 +, DACT3 +, DDIT3 -, DHFR -, DKK4 +, DLX4 +, DNMT3 +, DNMT1 -, DNMT3B -, DOK1 -, E2F1 -, E2F2
-, E2F3 -, E2F4 -, E2F6 +, E2F7 -, E2F8 +, E4F1 +, EFNA1 +, EGFR +, EGR1 +, ELK1 -, ENO1 +, EP300 -, EPAS1 -, EPHA2 +, ERBB3 -, ERCC1 -, ERG +, ESR1 -, ESR2 +, ETS1
-, ETV6 +, EZH2 -, F2R -, FABP4 +, FASLG +, FBXO32 +, FCB2 +, FCB3 +, FGF2 +, FHL2 +, FLI1 -, FLT1 -, FLT4 -, FN1 +, FOS +, FOSL2 -, FOXO1 +, FOXO2 +, FOXO3 -, FOXM1
-, FOXO1 +, FOXO3 +, FOXO3 -, FST -, GAD1 +, GALT +, GDF15 +, GFER -, GSTM1 -, GYPA -, HCN2 -, HDAC1 +, HDAC2 -, HDAC5 +, HDAC9 -, HECA +, HIC1 -, HIF1A +, HLA-DDB
+, HMG1 +, HNF4A +, HNRNP2B1 +, HOXA10 -, HOXB13 +, HR +, HSD3B2 +, HSF1 -, HSPA5 +, ICAM1 +, ICAM3 -, ID3 -, ID4 +, IFNA1 -, IGF1 +, IGF2 +, IGF2BP1 -, IL10 +, IL12B +, IL15
-, ILIR1 +, IL2 +, IL2RA -, IL3 +, IL32 +, IL4 +, IL6 +, IL7 -, ILF3 -, ILK -, INSR -, IRF1 +, IRF7 -, IRF9 -, ITGB1 +, JUN -, KAT5 -, KDR -, KHSRP +, KIF2C -, KLF10
+, KLF15 +, KLF2 +, KLF4 +, KLF5 -, L3MBTL1 -, LCK -, LEF1 +, LGALS1 -, LM02 +, MACROD1 -, MAD2L1 -, MAFA +, MAPK1 +, MAPK3 +, MAT2A -, MBD2 +, MCM10 +, MCM2
+, MCM3 +, MCM4 +, MCM5 +, MCM6 +, MCM7 +, MED1 -, MICB -, MKI67 +, MMP1 -, MMP14 -, MSC -, MSH2 -, MSL1 -, MYB -, MYBL2 -, MYC -, MYCN +, MYOCD -, NCOA3 +, NCOA4
-, NDC80 +, NF1 -, NFATC1 +, NFE2L2 -, NFKB1 -, NPM1 +, NR2E3 +, NR2F1 +, NR3C1 -, NR4A1 +, NR5A1 +, NRL +, NUPR1 +, OTX2 -, PAX5 +, PC +, PCNA -, PEBP1 +, PEG10 +, PER2
-, PFDN4 -, PGR +, PIP5K1C +, PLK1 +, POLA1 -, POU1F1 -, POU2AF1 -, POU2F1 -, POU2F2 -, PPARG +, PRDM1 -, PRKCA +, PROX1 -, PTGS2 +, PTH +, PTHLH -, PTPN6 +, PTTG1
-, PTTG1P -, PURA +, RAC1 -, RAD51 +, RAP1A -, RARA +, RB1 +, RBBP8 -, RBL1 +, RBL2 -, RBP3 -, REEP5 -, RELA -, RELB -, RELN -, REST -, RFC3 +, RHO +, RNASEL -, RPLP0
-, RRM2 -, RUNX1 +, RUNX2 -, RUNX3 +, S100A10 -, S100A4 -, SATB1 +, SDC1 -, SELE -, SELP +, SERPINB1 -, SERPINE1 -, SFPQ -, SFTPB -, SIRT1 -, SIRT6 +, SKP2 +, SLC2A1
+, SMAD3 -, SMAD4 -, SNAI1 +, SNAI2 +, SOX2 -, SOX6 +, SP1 -, SP2 -, SP1 +, SRPX -, SSTR1 +, STAT1 -, STAT2 -, STAT3 +, STAT5A -, STAT6 +, STMN1 -, TAF1A +, TAL1
-, TCF19 +, TCF7L2 -, TERT -, TF -, TFAP2A +, TFCP2 -, TFDP1 -, TFF1 +, TGFA +, THBD +, TIMP2 +, TIMP3 +, TK1 -, TNFRSF10B +, TNFRSF13C -, TNFRSF21 -, TNFSF10 +, TOP2A
-, TOPBP1 +, TP53 +, TP53BP1 -, TP63 -, TPT1 -, TTK +, TUBB3 +, TWIST1 +, TWIST2 -, TXNIP +, TYMS -, UBE2C -, UGT1A1 +, UHRF1 -, VDR -, VEGFA -, WEE1 -, WT1 -, WWP1
+, XBP1 -, YBX1 +, ZBTB16 +, ZEB1 -, ZFXH3 +, ZHX2 +, ZNF224 -, ZNF239 +, ZNRD1 -
2 ACLY +, SREBF1 +
3 ATP6V0E1 +, CREB5 +
4 BUB1B +, ZNF143 +
5 C4BPB +, NFIC +
6 CAT +, EGR3 +, NAB2 +
7 CCL11 +
8 CD3G +
9 CDKN2A +, EAPP -, HDAC3 -, HDAC4 -, MTA1 +, SMARCA1 +, SMARCA4 +, SMARCB1 +, TBX2 -, TBX3 -, TBX5 +, ZBTB7A -

```

Figure 8 : Aperçu du fichier des composants

On applique à chaque composant la procédure qui suit :

On initialise à 0 deux variables : **nb** et **somme**.

Pour chaque gène du fichier patient présent dans le composant, on incrémente **nb**. Si le signe “+” est associé à ce gène dans le graphe des composantes, **somme** est augmentée de la valeur d’expression du gène. Si le signe “-” est associé à ce gène dans le graphe des composantes, **somme** est augmentée de 1 moins la valeur d’expression du gène.

Si **nb** est nul (i.e. aucun gène du fichier patient n’est dans la composante) la valeur de similarité est 0.5. Sinon la valeur de similarité est donnée par le max de **MS** et **1-MS** où $MS = \text{somme}/nb$.

4.3 Implémentation dans Iguana

Les ressources fournies par M. Miannay consistaient en un script Shell (pour la lecture des fichiers patients et l’écriture des fichiers de résultats) appelant un script python (réalisant les traitements explicités dans l’algorithme).

Comme nous poursuivions un ancien projet codé en python, nous avons jugé pratique d’implémenter la fonctionnalité entièrement en python. Nous avons donc traduit les scripts Shell de M. Miannay, et adapté le script python.

L’intérêt initial de cette fonctionnalité est la génération d’un fichier reprenant les valeurs de similarité pour les gènes de chaque patient du jeu de données. La nomenclature de ce fichier est la suivante : NOM_DU_PATIENT LISTE_DES_VALEURS_DE_SIMILARITE RESULTAT_BOOLEAN pour une ligne.

L'interface se présente comme ceci :

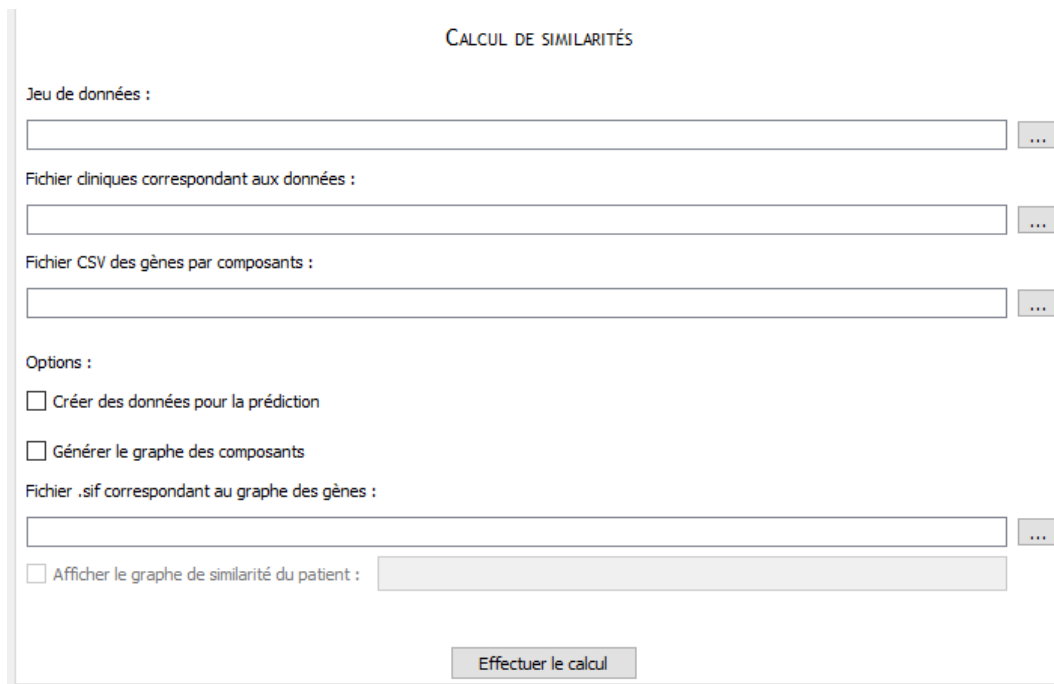


Figure 9 : Interface de l'onglet de similarité

L'utilisateur doit fournir plusieurs fichiers pour ce calcul :

- Tout d'abord le jeu de données concerné, c'est-à-dire l'ensemble des fichiers patients contenant l'expression des gènes d'intérêt étudiés ;
- Ensuite, le fichier clinique pour ce jeu de données, qui correspond simplement au diagnostic effectué pour chacun des patients ;
- Enfin, l'ensemble des gènes regroupés par composant.

Si l'une de ces informations venaient à manquer, l'utilisateur en serait informé et le calcul ne pourrait pas s'effectuer :

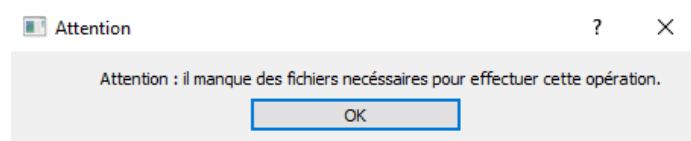


Figure 10 : Message d'erreur pour fichiers manquants

Des données optionnelles peuvent être fournies, ce point est développé dans le paragraphe 4.4.

Le fichier généré va ensuite pouvoir être utilisé par le classificateur : il suffira de donner les valeurs des scores de similarité pour chaque patient et le résultat (TRUE ou FALSE) à un module d'une bibliothèque (XGBoost) qui construit un classificateur.

Il va également pouvoir être utilisé pour la prédiction. Cependant, puisque l'objectif de la prédiction est effectivement de prédire le résultat, le fichier généré ne doit pas comprendre cette information booléenne. Une check box est donc présente dans l'onglet "Calcul de similarité" si l'utilisateur souhaite générer un fichier qui sera utilisé pour la prédiction.

Options :

☐ Créer des données pour la prédiction

Dans ce cas, il n'est pas nécessaire de spécifier un fichier clinique correspondant au jeu de données.

4.4 Visualisation des scores de similarité

4.4.1 Contexte général

La matrice de similarité produite est stockée en mémoire sous la forme d'un fichier. En pratique, ce calcul n'a d'intérêt que lors de l'utilisation du classificateur ou de la prédiction, qui vont réutiliser ces données. Néanmoins, il nous est apparu que l'utilisateur pouvait chercher à se représenter graphiquement cette matrice. Ce serait une façon rapide de visualiser, pour un patient, la similarité pour chaque composante. Nous avons décidé d'ajouter une option à l'interface : en plus d'effectuer le calcul de similarité, l'utilisateur pourrait choisir de créer le graphe des composants, de le colorer pour qu'il corresponde à un des patients de la matrice et de l'afficher grâce à Cytoscape. Le code couleur de ce graphe serait donc assez intuitif pour différencier les composantes similaires des composantes non-similaires (utilisation de couleurs chaudes et de couleurs froides).

Ces fonctionnalités sont donc optionnelles par rapport à l'objectif premier de l'utilisateur. C'est pourquoi nous avons choisi de modifier l'onglet concerné dans Iguana afin de permettre à l'utilisateur d'activer facilement et intuitivement ces fonctionnalités en fonction de ses attentes. Ce choix se présente sous forme de check boxes, initialement décochées :

Options :

☐ Créer des données pour la prédiction

☐ Générer le graphe des composants

Fichier .sif correspondant au graphe des gènes :

☐ Afficher le graphe de similarité du patient :

Figure 11 : Options disponibles dans l'onglet de similarité

Notons que l'affichage du graphe de similarité pour un patient X n'est possible que si la génération du graphe des composants a été effectuée. C'est pourquoi cette check box est initialement désactivée. Elle ne s'active donc que lorsque la check box "Générer le graphe des composants" est cochée.

☒ Générer le graphe des composants

Fichier .sif correspondant au graphe des gènes :

☒ Afficher le graphe de similarité du patient :

Figure 12 : Options sélectionnées dans l'onglet de similarité

Les fonctionnalités de contrôle de saisie restent évidemment actives pour ces options.

4.4.2 Création du graphe des composants

Par manque de temps, la fonctionnalité de création du graphe des composants n'avait jamais été implémentée. La visualisation des scores de similarités passe donc par ce premier travail.

Il s'agit, pour chaque paire de composants, de parcourir le graphe des gènes initial afin de déterminer s'il existe au moins un arc entre un nœud du premier composant et un nœud du second composant de la paire. Si c'est le cas, alors ce même arc existera entre ces deux composants dans le graphe des composants. Il faut faire attention à continuer le parcours tant qu'il reste des arcs ou tant que les deux types d'arcs potentiels n'ont pas été trouvés (inhibiteur et activateur).

Ces données sont ensuite enregistrées sous la forme d'un fichier .sif pour permettre une visualisation via le logiciel Cytoscape.

4.4.3 Création du graphe de similarité dans Cytoscape

Cette fonctionnalité est distincte de la création du graphe des composants mais ne peut s'effectuer sans cette dernière. La visualisation ne se fait que pour un patient, dont le nom est renseigné par l'utilisateur et doit correspondre à un des fichiers du jeu de données.

La première tâche consiste à colorer les nœuds du graphe des composants selon les valeurs de similarité du patient choisi. Ces valeurs se situent entre 0 et 1, mais nous choisissons d'appliquer une transformation affine sur les scores de similarité pour que l'échelle de couleur soit située entre 0 et 100, dans un souci de faciliter de traitement par Cytoscape.

La bibliothèque py2cytoscape crée ensuite automatiquement une échelle continue de couleur se situant entre la couleur la plus froide et la couleur la plus chaude. Chaque valeur de similarité du patient est ensuite associée à une couleur de cette échelle. Il suffit ensuite d'associer chaque nœud du graphe des composants à la valeur de similarité correspondante : le graphe est alors coloré.

Une fois que ces tâches sont effectuées, le graphe va s'afficher directement dans le logiciel Cytoscape, à condition qu'une instance de celui-ci soit lancée.

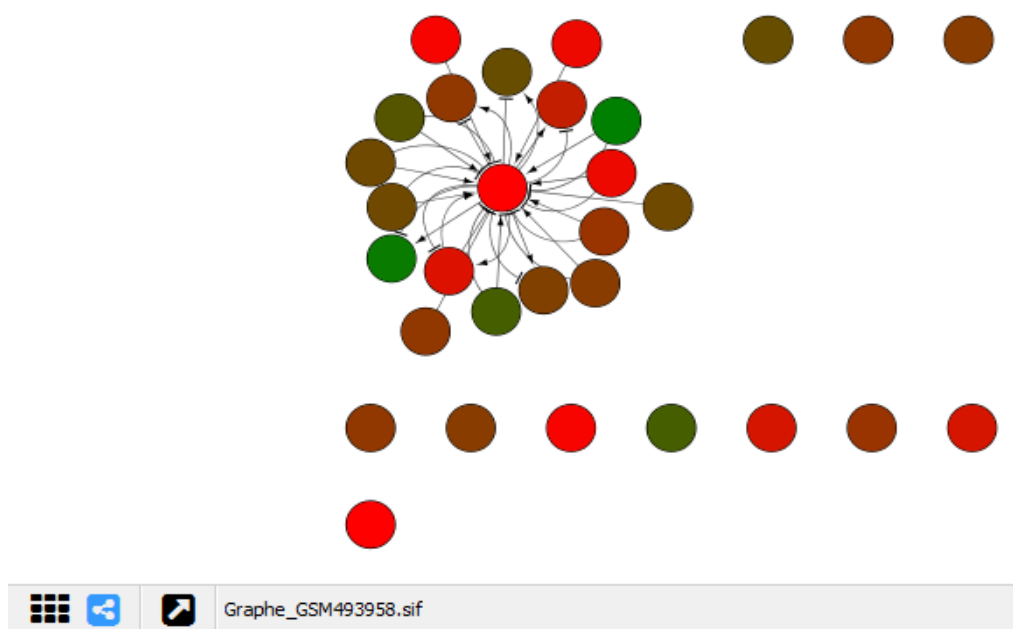


Figure 13 : Graphe de similarité

On visualise alors à la fois le graphe des composants et les valeurs de similarité pour le patient choisi (nuance de rouge pour une similarité moins élevée, nuance de vert pour une similarité plus élevée).

5. CLASSIFICATEUR/PREDICTION

5.1 Choix de l'algorithme et fonctionnement

5.1.1 Fonctionnement de l'algorithme

La première chose que nous avons dû choisir pour mettre en place un classificateur était l'algorithme qui permettait de créer un modèle de classification. Un tel modèle correspond concrètement à une fonction mathématique, qui, à partir d'un jeu de donnée relatif à un patient, prédit l'état de la personne. Une telle fonction mathématique peut être définie de nombreuses manières mais dans tous les cas, elle est basée sur un ensemble de paramètres que l'on peut ajuster de sorte que cette fonction se trompe le moins possible. Dans notre cas, le modèle que l'on cherche doit séparer l'espace des données en deux : "à risque" ou "non à risque".

Pour cela nous allons prendre pour exemple un problème simplifié où les données ne correspondent qu'à deux entiers, de sorte à pouvoir les placer sur un plan.

Voici le jeu de données :

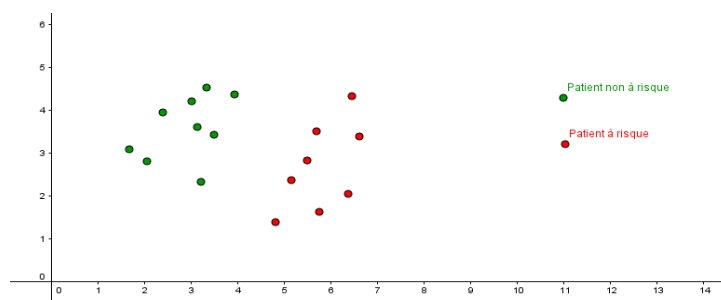


Figure 14 : Exemple simplifié de classification

On voit clairement que les des groupes de patients sont dissociés. Il suffit donc de trouver une fonction qui permette de les séparer. Pour cela, une simple droite suffit, comme suit.

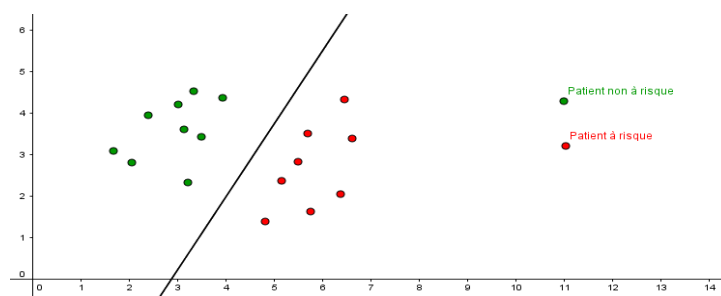


Figure 15 : Exemple simplifié de classification

On peut réaliser la même chose dans des espaces de plus grandes dimensions en cherchant un hyperplan qui sépare les données. Cependant, cela n'est valable uniquement lorsque les données sont linéairement séparables. Dans le cas contraire, une simple droite (ou hyperplan) ne suffit plus. On va alors construire des modèles plus complexes, comme des arbres de décision ou des réseaux de neurones.

Au début de notre projet, nous avons choisi d'utiliser des arbres de décision et plus précisément l'algorithme Random Forest, sur les conseils de Mme Guziolowski et M. Miannay. Un arbre de décision fonctionne un peu comme une cascade de classificateur linéaire à une dimension. En effet un classificateur à une dimension revient à sélectionner un point de l'axe des réel et à dire, si la donnée est plus grande, le patient est à risque (ou inversement), sinon il n'est pas à risque (ou inversement). Ainsi, un arbre de décision est un ensemble de classificateur linéaire 1D, que l'on représente dans les nœuds de l'arbre. Chaque bifurcation dans l'arbre permet de séparer un peu plus les données.

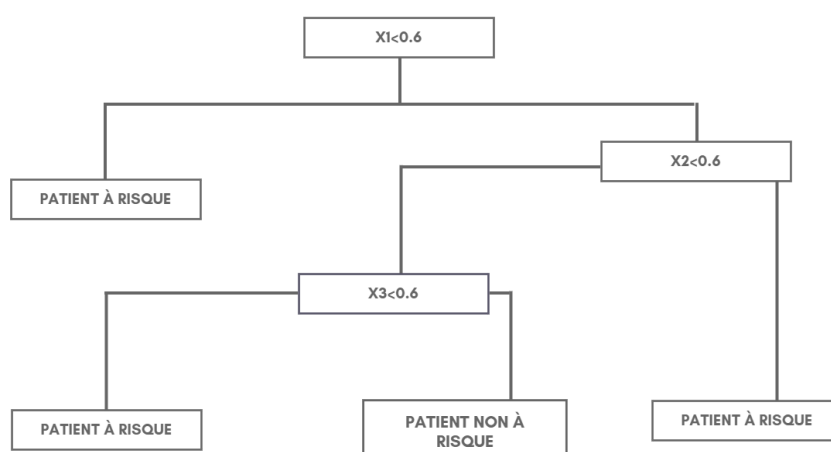


Figure 16 : Exemple d'arbre de décision

Maintenant que nous savons comment fonctionne un arbre de décision, nous pouvons comprendre comment fonctionne l'algorithme de Random Forest. Celui-ci consiste à créer un certain nombre d'arbre de décision pour notre jeu de données (une centaine nous a semblé un bon compromis, après expérimentation, entre précision et temps de calcul). En effet, pour un ensemble de variables, il existe un grand nombre d'arbre de décision. Pour une structure d'arbre donnée, on doit choisir quelle variable classifera sur quel nœud de l'arbre. Il y a alors $n!$ arrangements possibles (n est le nombre de variables) et chacun d'entre eux aura des résultats différents sur les prédictions. On construit alors des arbres à l'aide d'algorithmes classiques (CART, C4.5, ID3, ...) sur des sous-ensembles aléatoires du jeu de données initial. Cela permet d'avoir des arbres différents mais toutefois bien adaptés au problème. Lorsqu'on veut ensuite réaliser une prédiction, l'ensemble des arbres créés donnent leur "avis" et "votent" pour une des deux réponses : "patient à risque" ou "patient non à risque".

5.1.2 Choix de l'algorithme

Ce système a été testé, avec succès, par M. Miannay lors de sa thèse, sur le problème de classification des patients atteints du myélome multiple. Cependant, celui-ci ne s'est pas réellement concentré sur l'optimisation du classificateur et cette tâche nous revenait donc au début du projet. Après une phase de test et d'expérimentation que nous détaillerons dans la partie suivante, nous n'avons pas obtenu de résultats significativement meilleurs que ceux de M. Miannay. Nous nous sommes donc orientés vers d'autres techniques de classification, en essayant notamment de réseaux de neurone, sans franc succès.

En revanche, nous avons assisté à une conférence de Julien Simon, spécialiste Machine Learning chez Amazon, et nous avons pu nous entretenir à ce sujet avec lui. Il nous a conseillé d'essayer d'implémenter un classificateur de type XGBoost, qui signifie "Extreme Gradient Boosting". Ce classificateur est également basé sur des arbres de décision à la manière de l'algorithme de Random Forest, mais les arbres sont construits d'une manière différente. Notamment, lors de la création, un score de structure est défini pour chaque arbre et on optimise ce score grâce à la méthode de descente du gradient, pour plus de détails, voir Annexe 5.

Une fois mis en place, ce classificateur s'est révélé un peu plus précis que celui de M. Miannay. Cependant, la faible taille du jeu de donnée que nous possédions ne nous a pas permis de réaliser des tests très poussés.

5.2 Phase de test

5.2.1 Tests avant implémentation

Comme nous l'avons évoqué dans la partie précédente, nous avons d'abord réalisé une phase de test, pour reproduire les résultats que M. Miannay avait avec son classificateur. En effet nous avons eu accès à son code qui était en R. Nous avons donc traduit ce code en python dans un souci d'homogénéité. Ensuite nous avons comparé ses résultats sur les jeux de données fournis. Cependant, chaque modèle de Random Forest est tiré aléatoirement, il n'est donc pas possible d'avoir un résultat fixe, nous n'avons donc pu que comparer les ordres de grandeur de ses résultats avec les nôtres. Et nous avons trouvé sensiblement la même chose.

Nous avons deux jeux de données, et nous avons testé, en prenant exemple sur M. Miannay, d'entraîner un modèle avec un jeu et de tester sa précision sur l'autre jeu de donnée. Voici les résultats que nous avons obtenus :

| Modèle | Jeu d'entraînement | Jeu n°1 | Jeu n°2 |
|-----------------------------|--------------------|---------|---------|
| Random Forest M. Miannay | Jeu n°1 | | 0.68 |
| | Jeu n°2 | 0.78 | |
| Random Forest | Jeu n°1 | | 0.66 |
| | Jeu n°2 | 0.77 | |
| XGBoost | Jeu n°1 | | 0.75 |
| | Jeu n°2 | 0.71 | |

Nous ne testons pas les modèles sur les jeux d'entraînement, car cette donnée n'est pas pertinente, en effet un modèle est entraîné de sorte à coller le plus possible au jeu d'entraînement, les résultats sont alors fortement influencés.

Le modèle avec XGBoost semble plus régulier que les Random Forest et c'est pourquoi nous avons décidé de conserver cet algorithme.

De plus, à la suite de la conférence, nous avons également fait des tests sur nos modèles, appelés *Cross Validation*. Cela consiste à diviser aléatoirement notre jeu de données en deux, selon un certain pourcentage (entre 10% et 20% généralement), ensuite on entraîne notre modèle sur la plus grande partie des données et on le teste sur ce qu'il reste. On répète cette opération un grand nombre de fois (nous avons choisi 100) pour éviter tout bruit statistique. On obtient alors un taux de réussite que l'on peut espérer de notre modèle.

Nous n'avons pas réalisé ce test pour les Random Forest, mais pour XGBoost, on obtient avec nos deux jeux de données des valeurs autour de 0.8, ce qui est très satisfaisant.

5.2.2 Tests sur les données et pistes d'amélioration du modèle

Afin d'améliorer notre modèle, nous avons besoin de plus de données. La première chose qui nous est venue à l'esprit était de simplement fusionner les deux jeux de données. Cependant, ce n'est pas une chose aussi simple que cela en a l'air, en effet, les deux jeux de données ont été créés à l'aide de deux machines différentes (ces machines sont capables à partir de quelques gouttes de sang de mesurer le niveau d'expression de certains gènes). On peut donc imaginer que les données ne sont pas distribuées de manière exactement identique, en effet, les machines ne sont pas parfaitement identiques et les capteurs qui les composent ne donnent certainement pas exactement les mêmes valeurs. Les traitements de normalisation ne sont pas identiques non plus ce qui rajoute de la différence dans ces ensembles de données. Et cela peut avoir un impact non négligeable sur l'apprentissage de notre modèle.

Il convenait donc de mesurer les caractéristiques des distributions de données en comparant pour les deux jeux.

Pour cela, nous avons calculé l'espérance ainsi que l'écart-type pour les distributions des scores de similarité pour chaque composant. En toute rigueur, il faudrait réaliser cette étude sur chaque gène dont on mesure l'activité, de sorte que les données soient les plus similaires possibles. Cependant, cela revient à faire beaucoup plus de comparaisons (les machines calculent les activités de 2642 gènes, alors que l'on a 30 composants). De plus, le calcul du score de similarité est une fonction linéaire des activités des gènes (cf. 4.2) et la combinaison linéaire de lois normales reste une loi normale et donc les comparaisons que nous pourrions faire sur les distributions des scores des composants seraient valables pour les activations des gènes. Du fait de la normalisation sur les données brutes, les distributions des activations des gènes suivent des lois normales.

Ainsi on voit que pour la plupart des composants, les distributions sont assez similaires.

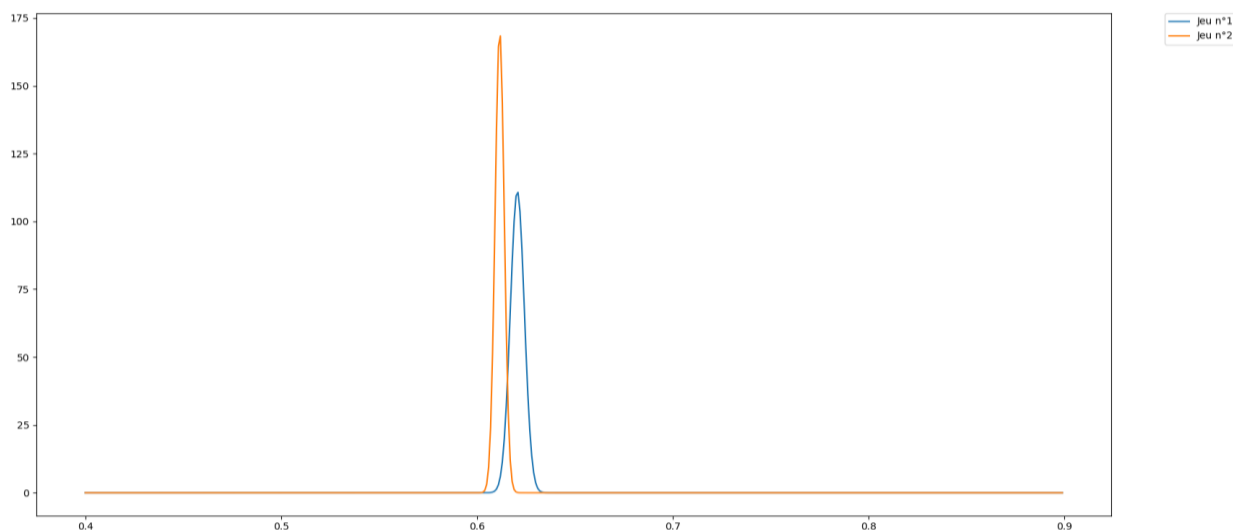


Figure 17 : Distribution des scores pour le composant 2

En revanche certains semblent présenter des différences significatives, le composant 7 par exemple. Ici, on observe que la moyenne varie d'environ 0.1 ce qui est assez énorme lorsque l'on compare deux jeux de données.

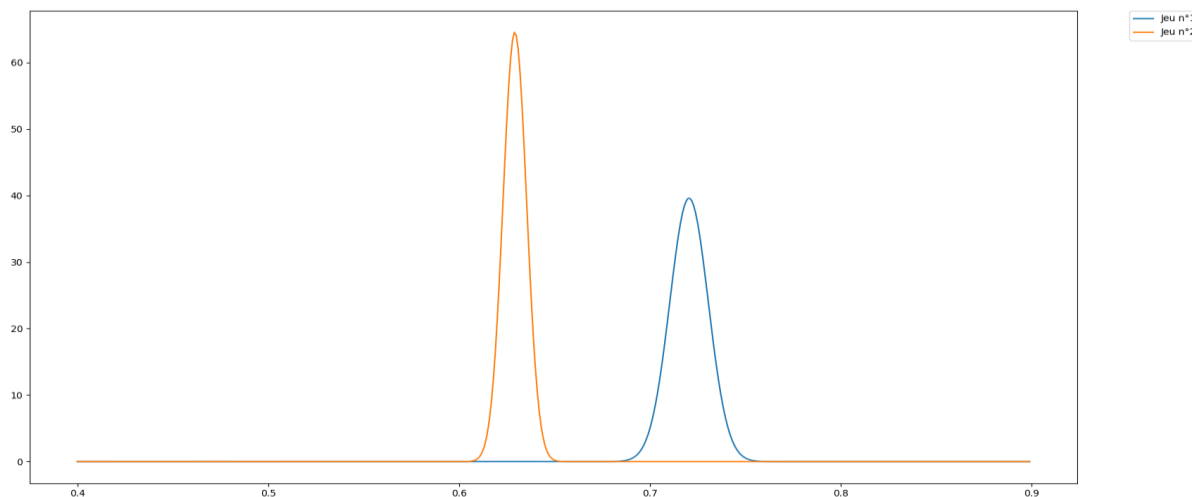


Figure 18 : Distribution des scores pour le composant 7

On souhaite évidemment que nos modèles puissent être utilisés sur des jeux de données différents et qui proviennent de machines différentes. Pour cela, on aura besoin d'utiliser les données de manière totalement indifférente de leur source, de sorte à construire de grands jeux de données d'entraînement et d'améliorer la précision des modèles. Cependant, comme certains composants semblent avoir des distributions relativement différentes d'une machine à l'autre, il pourrait être intéressant d'approfondir la comparaison entre les distributions pour déterminer quels composants semblent être indépendants de la solution matérielle utilisée pour collecter les données. On pourrait alors définir un modèle sur moins de composants, mais qui se généraliseraient mieux sur différents jeux de données.

Pour cela, il faudrait obtenir plus de données, provenant de machines encore différentes et de les comparer entre eux et avec celles que nous avons déjà. On pourrait aussi faire passer les mêmes tests à une même population de patients, sur deux machines différentes et comparer les distributions. Cela montrera, à priori, uniquement l'écart lié au matériel utilisé puisque la population ne change pas. De même, la taille des jeux de données que nous manipulons est assez restreinte, il conviendrait d'en construire de plus gros et de réaliser les mêmes comparaisons pour s'assurer que les différences que nous observons entre nos deux jeux ne sont pas simplement du bruit statistique.

Les deux derniers paragraphes posent les bases de recherches possibles en vue d'améliorer le modèle. Nous n'avons pas eu le temps de réaliser ces études, mais nous avons bon espoir que ces recherches débouchent sur des progrès significatifs.

5.3 Implémentation dans Iguana

Une fois la phase de test terminée, nous avons implémenté le classificateur dans l'application. Pour cela, nous avons ajouté des onglets à l'interface existante. Tout d'abord, un onglet pour la création et le test du classificateur. Un autre module est quant à lui réservé à la classification.

5.3.1 Module de création

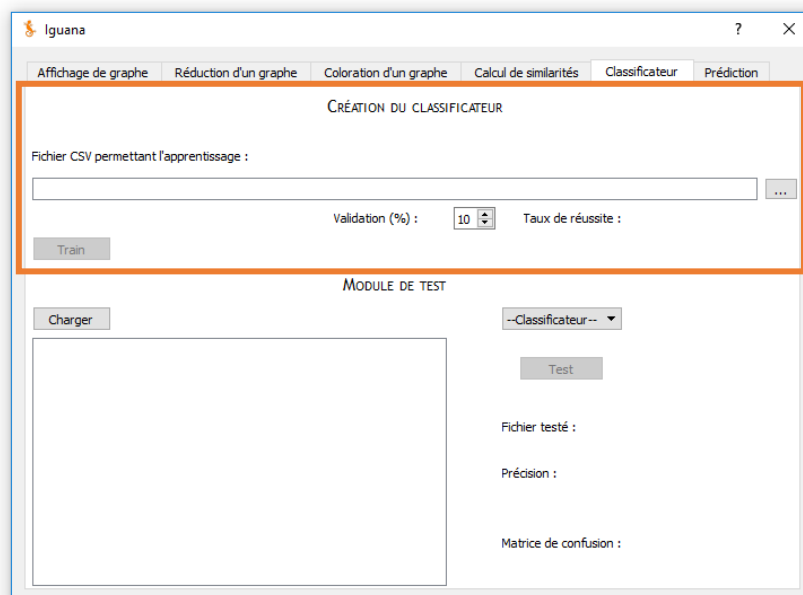


Figure 19 : Module de création

Le module de création est assez simple, il suffit d'aller chercher un fichier d'entraînement avec le bouton sur la droite du champ du fichier et de cliquer sur *Train*. Le fichier d'entraînement est un fichier .csv qui comporte sur chaque ligne, le nom du patient, les scores de similarité de chaque composant pour ce patient et son résultat clinique (TRUE pour "à risque" et FALSE pour non "à risque"), comme suit :

| | A | B | C | D | E | F | ... | AB | AC | AD | AE | AF |
|---|-----------|-------------|-------------|-------------|-------------|-------------|-----|-------------|-------------|-------------|-------------|-------|
| 1 | GSM493958 | 0.502115926 | 0.681063454 | 0.636294921 | 0.648431015 | 0.631680234 | | 0.672865657 | 0.658944587 | 0.707311709 | 0.682078206 | FALSE |

Figure 20 : Exemple d'une ligne d'un fichier d'entraînement

On peut ensuite régler le pourcentage de données qui sont utilisées dans la validation du modèle (le reste étant évidemment utilisé pour l'entraînement). Une fois que l'entraînement est terminé, le taux de réussite

est mis à jour. Cela correspond, au taux de réussite moyen que l'on peut obtenir avec le modèle créé. Pour plus de détails sur la création du classificateur et la validation de celui-ci, se reporter à la partie 5.1.1.

5.3.2 Module de test

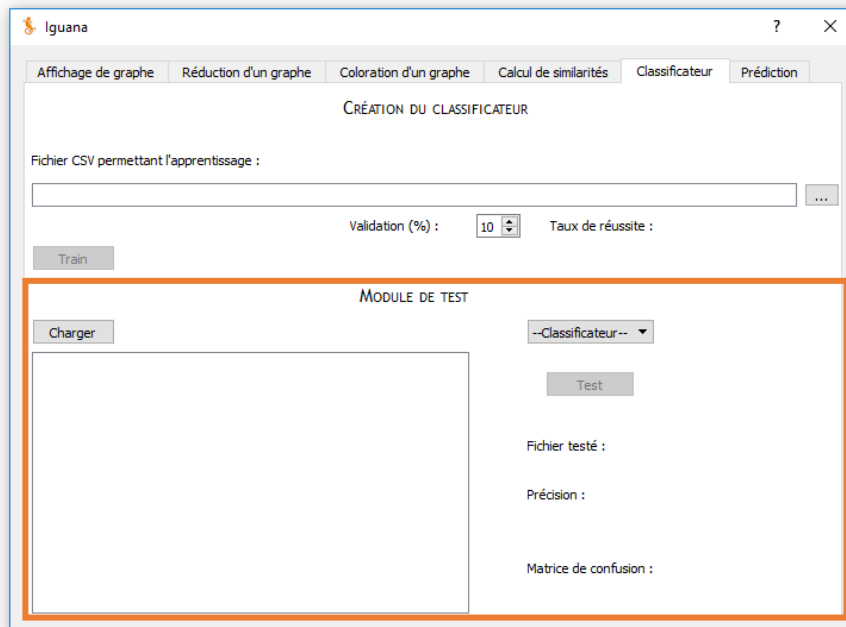


Figure 21: Module de test

Ce module est essentiellement présent dans le but de valider notre modèle sur d'autres jeux de données. Pour cela, on peut charger des fichiers de tests, ces fichiers sont identiques à ceux utilisés pour l'entraînement. La différence ici est que l'on va sélectionner un classificateur qui a été créé par le module précédent grâce à une liste déroulante.

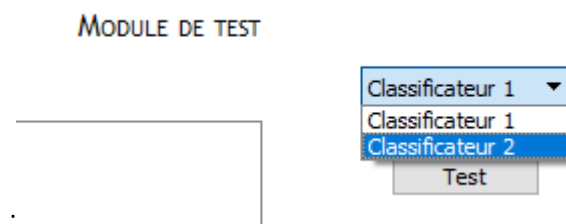
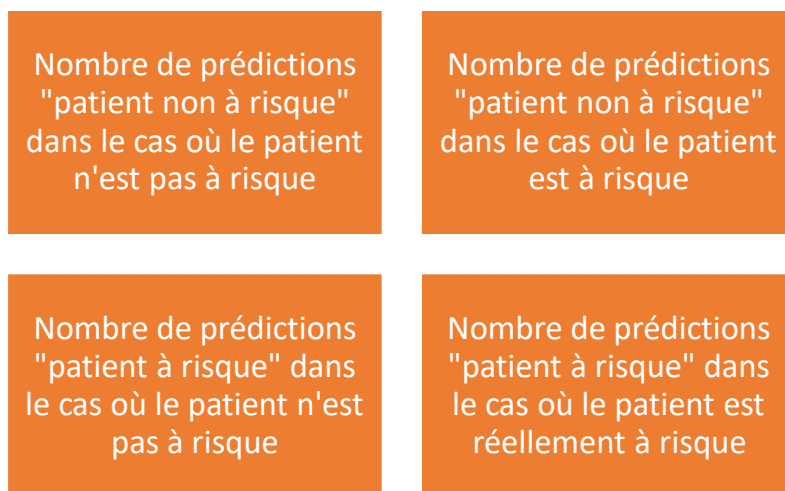


Figure 22 : Liste déroulante de sélection du classificateur

Ensuite, lorsque l'on clique sur *Test*, le classificateur réalise ses prédictions, affiche la précision obtenue (nombre de prédictions justes sur le nombre de prédictions total) ainsi que la matrice de confusion. Cette matrice permet de représenter les prédictions du modèle de manière synthétique en les divisant en 4 classes, voir ci-dessous.



MODULE DE TEST

resultat_dataPatientGSE19784HOVON65.csv

Classificateur 1 ▼

Fichier testé : resultat_dataPatientGSE19784HOVON65.csv

Précision : 0.91

Matrice de confusion :

| | Negative | Positive |
|----------|----------|----------|
| Negative | 177 | 5 |
| Positive | 19 | 73 |

Figure 23 : Exemple d'un test de classificateur

5.3.3 Module de prédiction

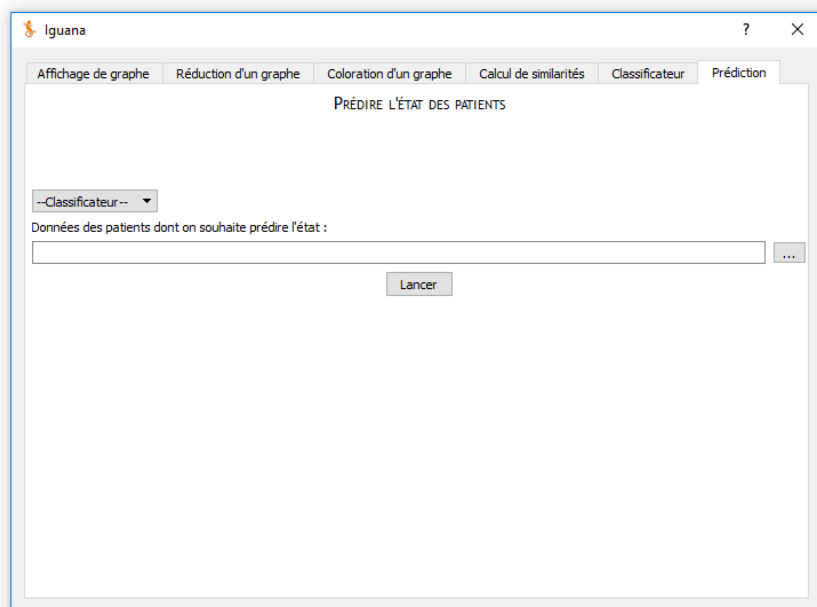


Figure 24 : Onglet de prédiction

Enfin, le dernier module d'Iguana, le module de création. Il se résume à un champ permettant de sélectionner les fichiers contenant les données des patients, d'une liste déroulante permettant de sélectionner le classificateur que l'on veut utiliser et d'un bouton permettant de lancer la prédiction.

Tout d'abord, les fichiers utilisés pour la prédiction sont très similaires à ceux utilisés pour créer ou tester les classificateurs. La seule différence est que dans le cas de la prédiction, nous ne connaissons pas l'état des patients (c'est ce que nous recherchons). La dernière colonne du fichier est donc vide (cf. figure 20). Ensuite, les classificateurs qui apparaissent dans la liste déroulante sont exactement les mêmes que dans l'onglet précédent. A chaque fois qu'un classificateur est créé, celui-ci est ajouté aux deux listes déroulantes.

Lorsque la prédiction est terminée, un fichier .csv est créé et contient le résultat de la prédiction. Ces résultats sont très simples, sur chaque ligne, on a le nom du patient suivi du résultat TRUE ou FALSE, comme défini plus tôt.

| | A | B |
|---|-----------|-------|
| 1 | GSM592391 | FALSE |
| 2 | GSM592392 | FALSE |
| 3 | GSM592393 | FALSE |

Figure 25 : Exemple de fichier de prédiction

6. CREATION D'UN EXECUTABLE WINDOWS

6.1 L'exécutable

Ce logiciel a vocation à être le plus simple possible et n'est pas destiné à des personnes sachant déjà programmer. Ainsi, il nous a semblé indispensable de créer un exécutable ne nécessitant aucune installation de Python ni de bibliothèque. Tout cela afin de rendre l'utilisation très simple. Pour ce faire, nous nous sommes documentés sur internet et avons finalement choisi d'utiliser la bibliothèque `cx_Freeze` (sous sa version : 5.1.1). Son utilisation est simple :

On crée un fichier `setup.py` qui sera utilisé pour compiler l'exécutable à placer dans le même dossier que le fichier principal.

Dans le fichier `setup.py` on écrit la base du code contenant notamment le fichier à compiler en `.exe`, les fichiers complémentaires et bibliothèques à inclure ou bien un icône pour le `.exe`.

En dernier lieu on lance la commande « `python setup.py build` » à partir d'un terminal Windows pour lancer la compilation

Une fois ces étapes réalisées, on obtient un dossier « Build » contenant tous les fichiers nécessaires à l'exécutable et l'exécutable lui-même.

6.2 Rédaction d'un guide d'utilisation

Pour aller avec notre exécutable, Mme Guziolowski nous a demandé de rédiger un guide d'utilisation expliquant l'installation et l'utilisation de l'application. Dans ce guide d'utilisation, nous avons expliqué chaque étape du processus au travers d'une brève explication, toujours accompagnée d'une capture d'écran afin de rendre cela le plus clair possible.

Nous avons également indiqué en particulier quelle version de Cytoscape installer ainsi que quelques conseils d'utilisation permettant de faciliter l'utilisation de cette application. L'ensemble des fichiers de l'application ainsi que le guide d'utilisation sont disponibles sur un dépôt git (cf. Annexe 2).

7. VERSION MAC OS

7.1 Portage vers plateforme Mac OS

Les motivations pour la réalisation d'une version MacOS est l'augmentation du nombre d'utilisateurs. La première étape a donc été de faire un portage de la première version d'Iguana puisque les nouvelles fonctionnalités n'étaient pas encore bien définies. Ces nouvelles fonctionnalités ont ensuite été rajoutées au fur et à mesure de leur réalisation. Toutes les fonctionnalités ont été correctement retranscrites dans l'application MacOS mis à part la partie d'aide en conservant la même interface graphique. Ainsi, l'interface est bien composée des 6 onglets prévus et elle garde une esthétique proche de la version Windows, ce qui permet de conserver l'ergonomie ainsi que la clarté des informations affichées.



Figure 26 : Interface Iguana MacOS



Figure 27 : Interface Iguana MacOS

Pour cette migration de plateforme, nous avons dû installer un environnement de travail sur MacOS. Pour cela, nous avons utilisé l'IDE Pyzo qui existe aussi bien sous Windows que sous MacOS, nous avons installé Cytoscape qui existe également sur MacOS et nous avons téléchargé l'ensemble des bibliothèques utilisées grâce à pip, un outil de gestionnaire de paquets.

La principale différence entre Windows et MacOS sont les chemins d'accès. En effet, ces chemins d'accès ne peuvent pas être réutilisés sur un autre système d'exploitation et il faut donc faire attention à bien tous les modifier. Voici par exemple l'une des modifications que nous avons dû effectuer, ici pour lancer Cytoscape (le commentaire correspond à la version Windows) :

```
def lancement(self):
    #os.startfile(r'C:\Program Files\Cytoscape_v3.5.1\Cytoscape.exe')
    opener = "open" if sys.platform == "darwin" else "xdg-open"
    subprocess.call([opener, "/Applications/Cytoscape_v3.6.0/Cytoscape.app"])
```

Figure 28 : Exemple de modification du script pour le portage MacOS

Pour l'utilisation du fichier optimizationComponent.lp et du module clingo permettant de faire la corrélation, nous avons téléchargé la version MacOS de clingo et effectué les changements suivant (la partie commentée correspond à la version Windows) :

```

print(input))
'''
    if (self.time.value()==0):
        command=dir_path+"\clingo.exe " +str(nbColor)+" "+dir_path +"\optimizationComponent.lp "+input+
os.path.splitext(input)[0] +"-colorations.txt"
    else:
        command=dir_path+"\clingo.exe " +str(nbColor)+" "+dir_path +"\optimizationComponent.lp "+input+
> "+ os.path.splitext(input)[0] +"-colorations.txt"
'''
    print(command)if (self.time.value()==0):
        command=dir_path+"/clingo " +str(nbColor)+" "+dir_path +"/optimizationComponent.lp "+input+
os.path.splitext(input)[0] +"-colorations.txt"
    else:
        command=dir_path+"/clingo " +str(nbColor)+" "+dir_path +"/optimizationComponent.lp "+input+
"+ os.path.splitext(input)[0] +"-colorations.txt"
    print(command)

```

Figure 29 : Autre exemple de modification pour le portage

Nous avons dû de plus changer des expressions qui sont différentes entre ces deux systèmes d'exploitation. Par exemple, pour la partie qui consiste à afficher le graphe de similarité du patient, nous ne pouvions pas obtenir correctement les fichiers à traiter sous MacOS. Nous avons donc dû effectuer les changements suivants :

```

def grapheSimilarite(self, rfile):
    resultFile = open(rfile, 'r')

    strings=[]

    lines=resultFile.readlines()
    for line in lines:
        temp=line.split(' ')

        if temp[0]==self.nomPatient.toPlainText():
            strings=temp.copy()
            print(temp)
    if strings != []:
        patientName = strings[0]
        similVector = strings[1:-1]
        resultFile.close()

```

Figure 30 : Extrait d'un script pour Windows


```
def grapheSimilarite(self, rfile):
    resultFile = open(rfile, 'r')

    strings=[]

    lines=resultFile.readlines()
    for line in lines:
        temp=line.split(' ')
        temp2=temp[0].split('/')[0]

        if temp2==self.nomPatient.toPlainText():
            strings=temp.copy()
            #print(temp[0])
            #print(temp2)
            #print(self.nomPatient.toPlainText())
            print(strings)
        if strings != []:
            patientName = strings[0].split('/')[0]
            similVector = strings[1:-1]
            resultFile.close()
```

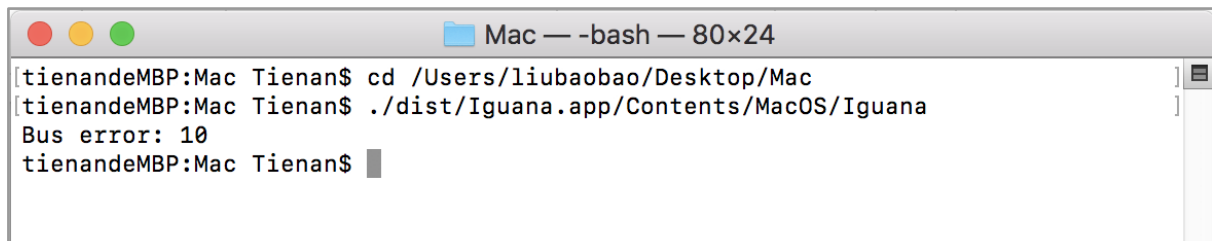
Figure 31 : Extrait d'un script pour MacOS

Ainsi, suite à de nombreux changements du même ordre, nous avons fini par retranscrire toutes les fonctionnalités de manière exacte sous MacOS. L'application peut donc maintenant être utilisée sous MacOS à condition d'installer Python et les bibliothèques correspondantes. Cependant, cette étape est beaucoup plus simple que sous Windows puisque sous MacOS les bibliothèques sont beaucoup plus simples à installer, nous n'avons eu aucune difficulté liée à des versions non compatibles par exemple.

7.2 Les difficultés rencontrées

Notre plus gros échec a été la réalisation d'un exécutable sous MacOS. Nous avons essayé plusieurs méthodes mais aucune n'a fonctionné.

Premièrement nous avons décidé d'utiliser py2app, une bibliothèque Python permettant de réaliser un exécutable en quelques étapes. Cependant, nous n'avons pas réussi à nous débarrasser d'une erreur qui survient à chaque fois lorsqu'on essayait de lancer l'application créée (voir ci-dessous). Malgré des recherches sur internet, nous n'avons pas malheureusement trouvé de solution étant donné que c'est une erreur très vague.



```
Mac — -bash — 80x24
[tiendeMBP:Mac Tienan$ cd /Users/liubaobao/Desktop/Mac
[tiendeMBP:Mac Tienan$ ./dist/Iguana.app/Contents/MacOS/Iguana
Bus error: 10
tiendeMBP:Mac Tienan$
```

Figure 32 : Erreur obtenue lors de la création de l'exécutable

Nous avons donc décidé d'utiliser le même outil que sous Windows c'est-à-dire la bibliothèque `cx_Freeze`. Cependant, là non plus nous n'avons pas réussi à faire fonctionner la création de l'application car nous n'avons pas réussi à trouver les fichiers présents dans `tcl` et `tk`, deux répertoires nécessaires à la compilation de l'application. En effet, ce sont ces fichiers qui permettent de dessiner l'interface graphique à l'écran. Là encore nous avons cherché sur internet mais nous n'avons pas trouvé de réponse.

Etant donné que cet exécutable Mac OS était la dernière chose sur notre planning, il fallait en effet avoir d'abord fini la partie Windows puis faire les modifications nécessaires sur le code pour l'adapter à MacOS avant de pouvoir créer un exécutable, nous n'avons pas eu le temps de nous pencher plus sur le problème. Cependant, sur la demande de Mme Guziolowski nous avons rédigé un guide d'installation très précis sous MacOS permettant d'installer tous les outils nécessaires au bon fonctionnement de l'application sous MacOS.

8. CONCLUSION

A travers ce projet, nous avons découvert les enjeux du travail en groupe. En effet, la plupart d'entre nous n'avaient pas, jusqu'ici, participé à des projets informatiques en collaboration avec plus de deux personnes (TP ou PAPPL). Ce projet nous a fait découvrir les difficultés organisationnelles du travail en équipe. Nous les avons affrontées en utilisant une communication efficace et régulière ainsi que des outils de versionnement de code et de partage de fichiers.

Ensuite, ce projet s'inscrivait dans la continuité d'un projet déjà établi. Bien que cela allège la charge de travail au niveau de la conception du système ; au niveau du design ou des choix de développement par exemple ; intégrer un projet déjà commencé nécessite beaucoup d'investissement pour appréhender les travaux déjà réalisés. Deux personnes dans le groupe étaient à l'origine du projet initial et cela nous a aidé à mieux comprendre les tenants et aboutissants et de commencer plus vite l'ajout de nouvelles fonctionnalités.

De plus, les bases du projet ayant déjà été posées, nous n'avons pas été confronté à certaines difficultés qui ont été rencontrées lors de la première phase. En effet, le choix et l'installation des bibliothèques Python a été compliqué pour Jules et Pierre. Or ces bibliothèques étaient déjà mises en place dans le projet, il n'a fallu qu'apprendre à les utiliser. Nous avons pu nous concentrer sur la réalisation et l'implémentation des nouvelles fonctionnalités. Ainsi nous avons pu ajouter une partie de classification, basée sur les travaux de M. Miannay. Cette classification est décomposée en trois étapes ; le calcul du score de similarité de chaque composant, à partir du graphe des gènes ; la création de classificateur à partir des données de similarité et la prédiction à partir de données patient.

Iguana est maintenant une solution logicielle complète, permettant, à partir simplement de données patients (expression de leurs gènes et résultats cliniques) ainsi qu'un graphe d'interaction entre les gènes de créer un modèle complet, basé sur les travaux de M. Miannay. Nous avons donc fourni un produit totalement fonctionnel, qui plus est, disponible sur deux plateformes différentes. Nous trouvons cela gratifiant que notre travail puisse être utilisé et diffusé largement dans le monde médical et notamment dans la recherche de nouveaux traitement contre le myélome multiple.

Nos tests sur ce modèle nous ont conduit à explorer un peu le monde de la recherche, notamment lorsque nous avons réalisé les comparaisons entre les différents jeux de données qui étaient à notre disposition. Cela nous a permis de dégager des axes d'améliorations possibles pour Iguana et le modèle créé par M. Miannay.

Pour conclure, voici un schéma récapitulatif du fonctionnement complet d'Iguana, des différentes interactions entre les fichiers créés au fil du processus de traitement du graphe puis de la classification.

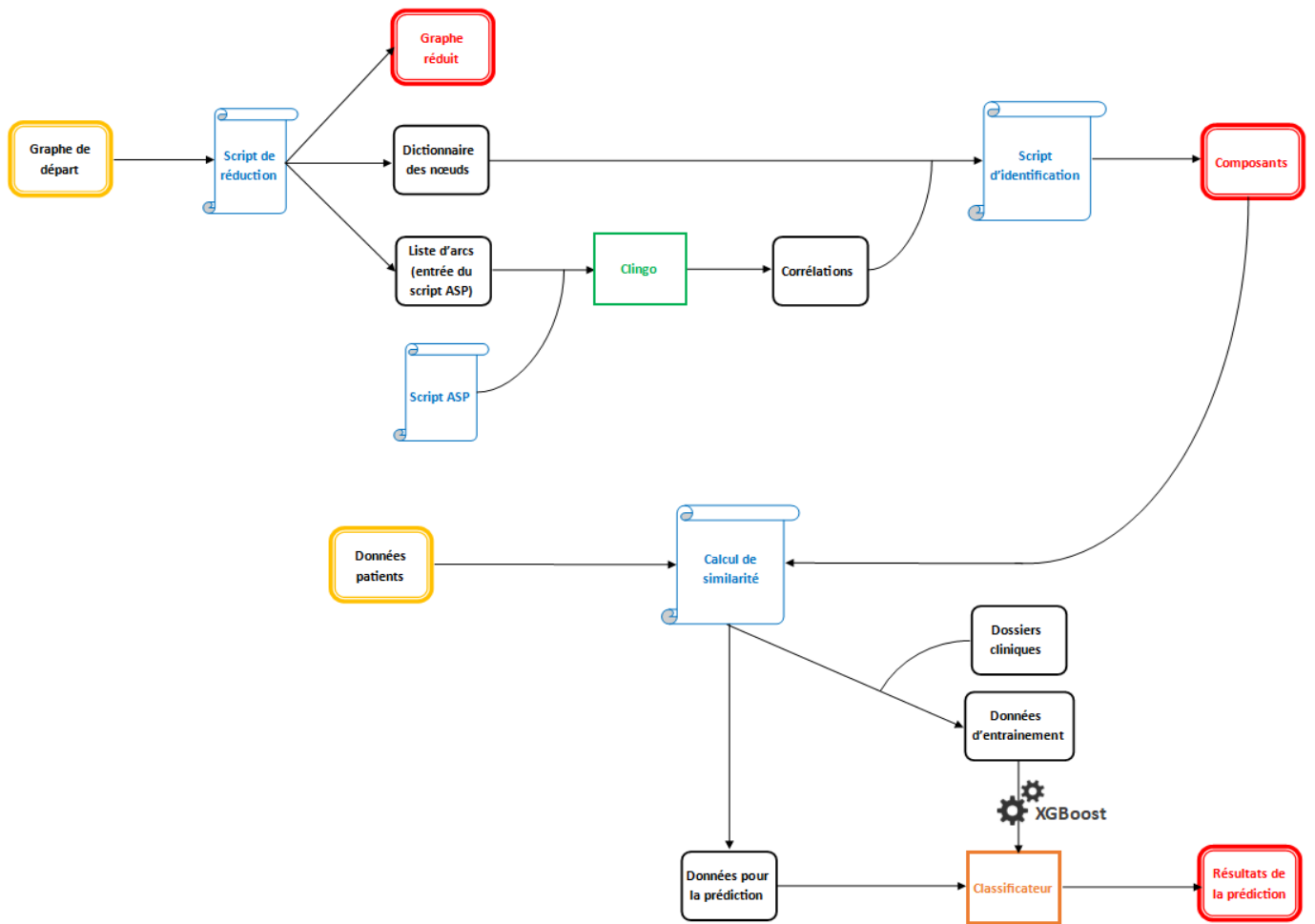


Figure 33 : Schéma récapitulatif du fonctionnement d'Iguana

9. ANNEXES

Annexe 1 : Présentation de Cytoscape

Annexe 2 : Lien vers l'application et le guide

Annexe 3 : Cahier des charges

Annexe 4 : Bibliothèque et logiciels

Annexe 5 : XGBoost

Annexe 6 : Distribution des données

Annexe 1 : Présentation de Cytoscape

Cytoscape est un logiciel de visualisation et de traitement de graphe. Voici une vue de son interface.

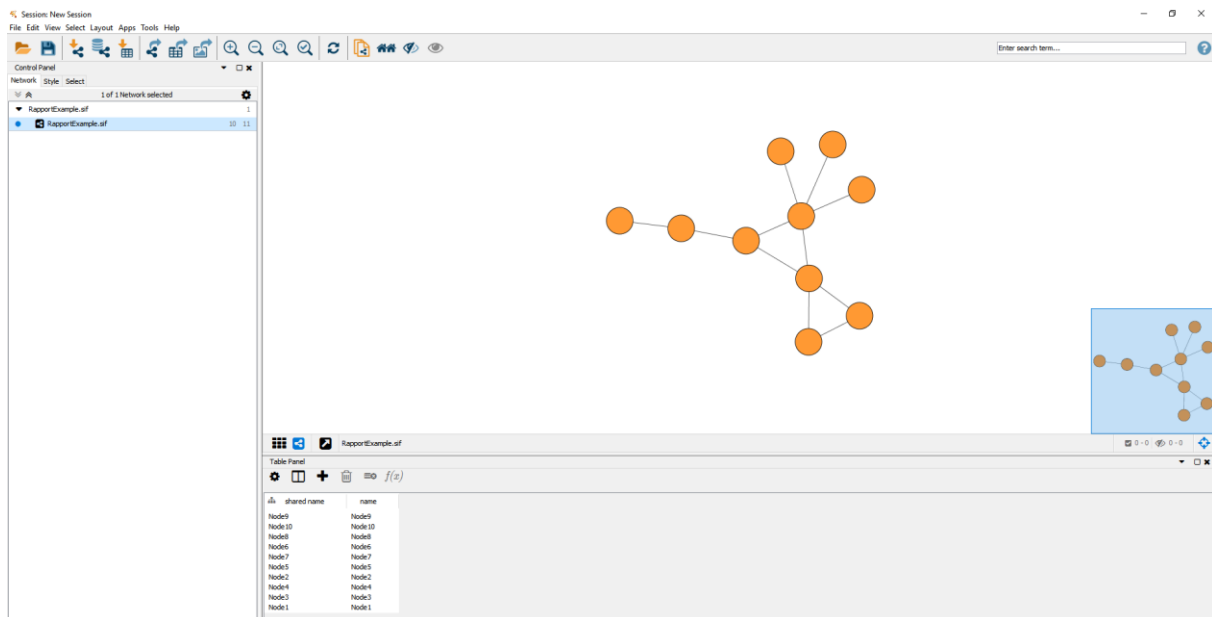


Figure 34 : Cytoscape

Annexe 2 : Lien vers l'application et le guide

Nous avons déposé sur un dépôt Git l'ensemble des sources du projet, ainsi qu'un guide d'utilisation pour Windows et MacOS.

Voici le lien :

<https://github.com/ipeter50/Iguana>

Annexe 3 : Cahier des charges

1. Présentation du sujet :

Ce projet fait suite au projet d'application de cette année qui a conduit à la création d'une IHM s'appelant Iguana. Lors de ce projet, les élèves ont utilisé les travaux de M Miannay et Mme Guziolowski sur le myélome multiple afin de réutiliser une partie de leurs scripts et les intégrer dans une interface graphique permettant à la fois de simplifier le processus de traitement des graphes mais aussi d'afficher ces graphes au fur et à mesure. Ce nouveau projet a pour but d'ajouter de nouvelles fonctionnalités à Iguana permettant de catégoriser le patient selon les données qui ont été récupérées. En effet, il existe plusieurs catégories de malade et il est important de bien les catégoriser afin d'adapter le traitement au mieux. Cette fonctionnalité supplémentaire utilise les graphes créés avec la première version d'Iguana et des méthodes de Deep Learning afin d'établir une classification la plus juste possible.

2. Tâches à effectuer :

Importation et structure des données.

Objectif : Permettre à Iguana de charger une matrice transcriptomique (matrice de dimension $n \times m$ à valeur dans $[0,1]$ représentant l'expression de n gènes pour m patients) à partir d'un fichier source. Ce fichier, qui aura un format bien défini, contiendra des données cliniques sur ces patients.

Description : Le fichier de données est chargé par l'utilisateur via une interface de chargement (explorateur de fichiers). Les données sont ensuite structurées sous la forme d'une matrice.

Contraintes : Vérifier que le fichier respecte la nomenclature établie.

Tâches à réaliser :

- Interface de chargement des fichiers patient
- Mise en forme des données pour les utiliser dans le script Python

Calcul de similarité.

Objectif : Ajouter la fonctionnalité de calcul de similarité à Iguana.

Description : Les données génétiques de M individus sont contenues dans une matrice de taille $n \times M$. Cela signifie que, pour chacun de ces individus, seuls n gènes sont concernés. Ces données représentent l'état génétique du patient. De plus, un diagnostic pour ces patients a déjà été établi et est connu. Nous disposons également d'un graphe représentant les interactions entre N gènes du génome humain ($N \geq n$). Celui-ci a été réduit, sans perte d'information, en k sous-graphes de taille inférieure plus facilement exploitables.

Il va donc s'agir d'appliquer, pour chaque individu, l'algorithme de similarité entre les n gènes (qui correspondent à n nœuds du graphe dans un état particulier) et les k sous-graphes.

Nous obtenons alors une matrice $k \times M$ dont chaque colonne représente la similarité entre l'état du patient et l'état qu'il doit avoir au vu de son information génétique.

Contraintes : L'utilisateur doit avoir réduit un graphe et importé une matrice transcriptomique avant de pouvoir lancer le calcul de similarité.

Comment les résultats du calcul sont transmis à l'utilisateur ? Ils apparaissent à l'écran ou sont écrits dans un fichier ?

Tâches à réaliser :

- Prise en main des scripts Iggy-POC pour le calcul et de l'environnement
- Ajout de l'interface graphique dans Iguana
- Implémentation du calcul en lui même
 - Faire le lien entre les données et le graphe (en fonction du nombre de gènes)
- Affichage du résultat de la similarité (vertue pédagogique)
 - Création du graphe des composants avec des couleurs/chiffres pour représenter la similarité
 - Si possible générer des images (à la manière de heatmap)
- Deux modes de fonctionnement :
 - Calcul de similarité sur l'ensemble de la base de donnée patient
 - Calcul de similarité sur un seul patient avec affichage dans Cytoscape du graphe des composants correspondant.

Classificateur :

Objectif : Récupérer les données fournies par le calcul de similarité et les utiliser afin d'établir une classification en utilisant les travaux de data mining déjà créés par M Miannay.

Description : Pour cela, il faudra utiliser les données de recherche déjà utilisées lors du concours de classification de 2017. Ces données seront récupérables grâce à la machine virtuelle Docker qu'il faudra prendre en main puis implémenter ces fonctionnalités dans Iguana ainsi que tester plusieurs méthodes de Data Mining afin d'affiner le modèle. On pourra entre faire varier le nombre de données disponibles pour l'apprentissage.

Tâches à réaliser :

- Insertion d'un nouveau module dans Iguana
 - Fonctionnalité de chargement de données d'apprentissage
 - Création du classificateur à partir de Random Forest
 - Prise en main des scripts en R dans Iggy-POC
 - Intégration de ces de ces scripts dans Iguana (Soit R soit Python)
 - Module de test
 - Chargement des données de test
 - Affichage des résultats de test
- Amélioration du modèle de départ

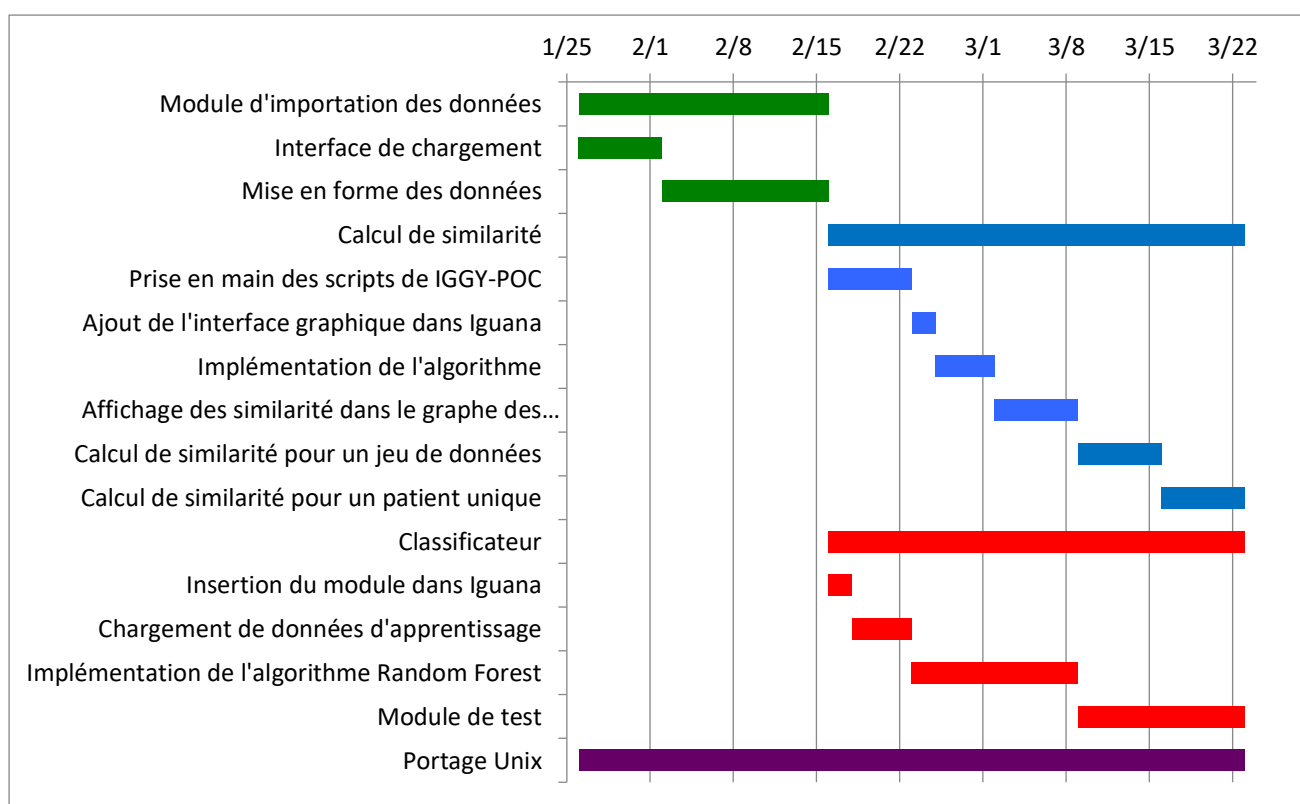
Portage vers plateforme Unix (MacOS ou Linux).

Objectif : Rendre l'application Iguana entièrement opérationnelle sur ces OS tout en conservant la compatibilité Windows

Tâches à réaliser :

- Choix de la plateforme au démarrage
- Possibilité de modifier la plateforme si besoin
- Conserver le fonctionnement des différents modules (à valider avec les graphes d'essai)

3. Répartition des tâches et organisation du projet



Annexe 4 : Bibliothèque et logiciels

Bibliothèques Python :

- cx_Freeze (5.1.1)
- networkx (1.10)
- numpy (1.13.3)
- pandas (0.21.1)
- psutil (5.4.2)
- py2cytoscape (0.6.2)
- pydot (1.2.3)
- python-igraph (0.7.1.post6)
- python-qt5 (0.1.10)
- scipy (1.0.0)
- sklearn (0.0)
- xgboost (0.7)

Logiciels :

- QtDesigner
- Cytoscape 3.5.1
- Clingo 4.0

Annexe 5 : XGBoost

Quelques informations supplémentaires sur XGBoost :

- <http://xgboost.readthedocs.io/en/latest/model.html>
- <https://en.wikipedia.org/wiki/Xgboost>
- <https://machinelearningmastery.com/gentle-introduction-xgboost-applied-machine-learning/>

Annexe 6 : Distribution des données

Distribution des scores de quelques composants :

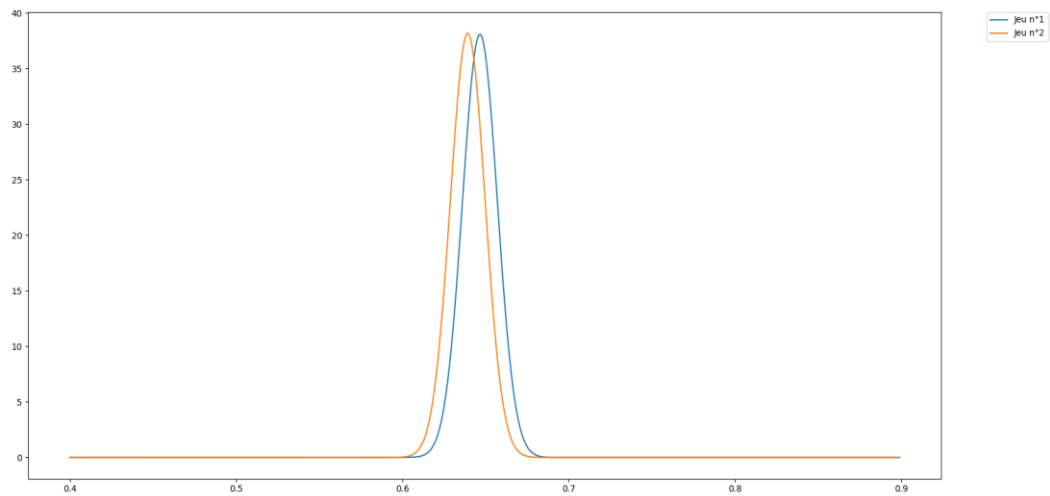


Figure 35 : Distribution des scores du composant 9

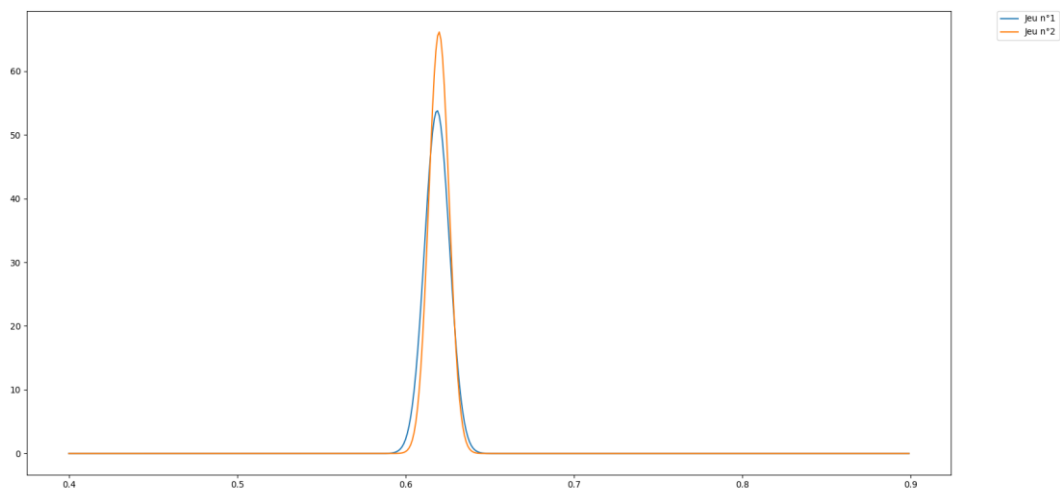


Figure 36 : Distribution des scores du composant 15

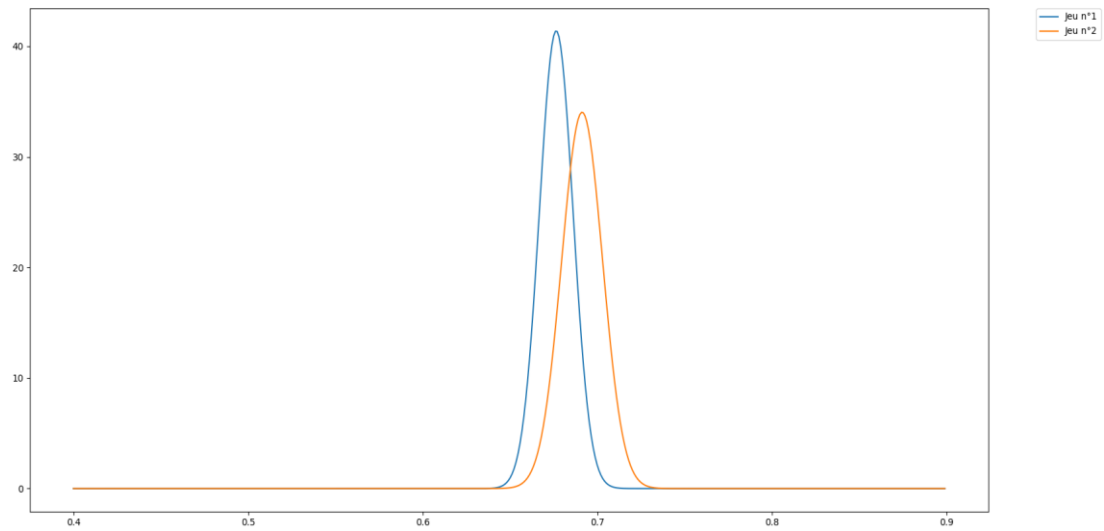


Figure 37 : Distribution des scores du composant 19