

Systeme de prédiction des niveaux de compétence IT de programmeurs par machine learning

Travail de fin d'études présenté par

Amaury LEKENS

En vue de l'obtention du diplôme de

Master en Sciences de l'Ingénieur industriel orientation Informatique

Promoteur : Dr Ir. Sébastien Combéfis

Tuteur : Mr. Rémy Taymans

Année académique 2018-2019

Abstract

Ce TFE présente une exploration de modèles mathématiques et de techniques de machine learning et de data mining dans le but de développer un système de prédiction des niveaux de compétence dans des domaines de l'IT. Ces modèles prédictifs sont construits sur la base de données récupérées lors de divers challenges en IT. Ce travail présente aussi diverses techniques probabilistes et statistiques ayant pour objectif principal d'obtenir le système le plus performant possible. Enfin, une série d'approches étudiées ont été implémentées sous forme de prototype et testées.

La recherche s'est effectuée d'une part à partir de livres de référence présentant des connaissances bien ancrées dans les domaines généraux du machine learning et du data mining et d'autre part à l'aide d'articles scientifiques récents plus proche du cas d'utilisation présenté dans ce TFE.

Les intérêts d'un tel système sont multiples, le principal est d'utiliser ce dernier comme un outil d'aide au recrutement qui permettrait à un recruteur de trouver un profil idéal sur la base de compétences désirées s'accordant avec des compétences prédites. Une seconde utilisation peut s'effectuer dans un cadre pédagogique où le système permettrait d'orienter un apprenant vers des compétences non-acquises, mais dans lesquelles il semble être doué, dans le but d'élargir sa palette de talents.

Mots-clés : IA, machine learning, data mining, prédiction de compétences, IT, recrutement

Table des matières

Abstract	i
1 Introduction	1
1.1 L'entreprise : EDITx	1
1.2 Le projet	1
1.3 Objectifs concrets	2
2 Etat de l'art	3
3 Les données : accès, analyse et nettoyage	5
3.1 Accès aux données	5
3.2 Format des données	7
3.3 Analyse basique des données	8
3.4 Génération de données	10
3.5 Nettoyage des données	12
3.6 Normalisation des variables	14
3.7 Résumé des méthodes présentées	14
3.8 Choix des datasets utilisés	16
4 Algorithmes	17
4.1 Remarques	17
4.2 Régression linéaire multivariée	18
4.3 Régression polynomiale multivariée et la notion d'overfitting	26
4.4 PCA (basée sur SVD)	32
4.5 Réseau bayésien	47
5 Développement d'un framework Python	55
5.1 Accès à la base de données	55
5.2 Travail sur les données	57
5.3 Algorithmes	57
6 Conclusion	59
Table des figures	61

Table des tableaux	62
A Appendices	63
A.1 Exemple de code	63
A.2 Description des compétences liées aux datasets	64
Bibliographie	65

1 Introduction

1.1 L'entreprise : EDITx

Ce TFE a été réalisé chez EDITx, une entreprise fondée en octobre 2016 en tant que SPRL par Serge Goffin et Alexandre Dembour. L'activité principale d'EDITx est l'organisation de quiz et de challenges en informatique sur leur plateforme en ligne.

Ils ont principalement 3 cibles :

- Les étudiants et les professionnels de l'informatique : ceux-ci peuvent participer aux quiz et aux challenges pour gagner des prix, et même parfois trouver des possibilité d'emplois.
- Les professeurs en IT : ceux-ci peuvent contribuer à la plateforme en rédigeant des questions à choix multiples qui seront utilisées dans les challenges.

Pour rentabiliser cette activité, EDITx vend ses challenges à d'autres sociétés (ING, infrabel ...). Ces dernières y trouvent un double avantage:

- Elles augmentent leur visibilité dans le secteur de l'IT.
- Elles peuvent accéder aux données des utilisateurs de la plateforme et donc repérer un profil qui les intéresse.

Un challenge se déroule en 2 parties :

- Pendant une première période, le challenge est accessible à n'importe qui sur la plateforme. Elle dure 6 semaines.
- Après cette première phase, les 10 ou 20 meilleurs étudiants et professionnels sont invités dans les bureaux de la société ayant acheté le challenge pour participer à la finale. Les 3 premiers de chaque catégorie remportent un prix.

1.2 Le projet

Au fil des challenges organisés EDITx a accumulé une importante base de données de résultats d'utilisateurs pour différents challenges en IT. Le projet présenté dans ce manuscrit est lié à l'analyse de ces données.

Comme indiqué ci-dessus, les premiers clients d'EDITx sont les entreprises qui achètent

les challenges et qui en échange reçoivent un accès aux données des utilisateurs. Les données brutes sont déjà intéressantes pour une entreprise. Par exemple, si celle-ci achète un challenge en Java, elle peut directement détecter les participants qui possèdent un niveau élevé en Java via les résultats du concours. En outre, EDITx essaye d'ajouter à sa plateforme des outils d'analyse statistique pour présenter les données de la meilleure façon possible aux entreprises qui achètent les challenges.

L'augmentation constante de la taille de sa base de données et l'envie d'offrir à ses clients des outils à la pointe de la technologie pousse EDITx vers l'étape suivante, à savoir, l'utilisation de ces données dans la construction de modèles statistiques intelligents réalisant diverses tâches d'inférence (prédiction du niveau d'un utilisateur, prédiction du niveau d'une question, ...). Le but du projet est donc de construire ce genre de modèle de prédiction pour une tâche bien précise : prédire le score d'un utilisateur dans une compétence à l'aide de scores obtenus dans d'autres compétences. Pour réaliser ce travail, on prend l'avantage du fait que plusieurs utilisateurs ont réalisé plusieurs challenges et donc qu'on dispose pour ceux-ci de scores dans différentes compétences. Le travail réalisé présente donc des recherches théoriques et des applications pratiques de méthodes de détection de corrélation entre ces différents scores. L'objectif final est de pouvoir réaliser des inférences de ce type : un utilisateur bon en Java et en C++ est moyennement bon en python (c'est juste un exemple, pas un résultat).

L'utilisation de ces prédictions peut être intéressant pour les entreprises, dans le sens où elles peuvent juger le niveau d'un utilisateur dans une certaine compétence sans qu'elle ait été évaluée sur la plateforme.

1.3 Objectifs concrets

Afin d'organiser le travail en différentes étapes, des objectifs concrets sont fixés :

- Implémenter un accès propre et flexible aux données (les résultats des challenges)
- Préparer un format de données pour les algorithmes
- Réaliser une première analyse des données pour déterminer les datasets utilisables
- Trouver et implémenter une méthode pour nettoyer les données
- Trouver et implémenter différents modèles d'inférence
- Évaluer les différents modèles d'inférence

2 Etat de l'art

La prédiction du niveau d'une personne dans une compétence technique est un sujet qui a déjà été exploré. Les techniques employées sont variées, mais on peut distinguer deux écoles au niveau du choix des données utilisées. D'un coté, certaines méthodes utilisent des données personnelles classiques comme l'âge, le genre, le niveau d'éducation ou plus exotiques comme le fait de posséder ou non un téléphone mobile. D'un autre coté, certains préfèrent utiliser des scores précédemment obtenus dans d'autres compétences techniques et essayer de trouver des liens avec la compétence sur laquelle la prédiction est réalisée. Les paragraphes qui suivent résument les techniques utilisées par ces deux points de vue, bien que dans ce travail, c'est le second qui est adopté.

La méthode la plus régulièrement utilisée dans ce domaine est la régression multivariée, comme dans l'article *regression models for predicting student academic performance in a engineering course, 2010*, de Shaobo Huang et Ning Fang dans lequel un système prédit les résultats d'un examen de dynamique à partir de résultats dans d'autres cours comme la physique, le calculus ou encore la statique. Une autre méthode plus originale utilisant les régressions est présentée dans l'article *Predicting student exam's scores by analyzing social network data, 2012*, réalisé par un groupe d'étudiants de l'université de Néguev. Dans ce dernier, pour prédire le score d'un étudiant à un examen, ils utilisent certains de ces scores précédents ainsi que certains scores d'étudiants faisant parti de son cercle d'amis.

Une autre technique très utilisée et basée sur la détection de dépendances conditionnelles entre les variables est appelée réseau bayésien. C'est ce qu'ont mis en place Rafe Torabi, Parham Moradi et Ali Reza Khantaimoori dans leur article *Predict student scores using bayesian networks, 2012*. Dans celui-ci, ils prédisent le score d'un étudiant à un examen sur la base de 4 variables explicatives : son genre, l'enseignant associé au cours, le fait que l'étudiant possède ou non un emploi et le semestre en cours. Pour réaliser les prédictions, ils utilisent un réseau bayésien construit à partir de connaissances. Un autre travail basé sur les réseaux bayésiens, qui utilise aussi bien des scores précédemment obtenus que des données personnelles, a été réalisé par Rahel Bekele et Wolfgang Menzel dans leur article *A bayesian approach to predict performance of a student, 2005*.

Tous ces travaux présentent un cas d'utilisation similaire à celui qui nous concerne et semblent faire un consensus autour des méthodes utilisées. Il semble donc raisonnable

de les utiliser comme une base de travail même si les données dont on dispose sont différentes.

3 Les données : accès, analyse et nettoyage

3.1 Accès aux données

3.1.1 Etat des lieux

La plateforme EDITx a été développée à l'aide de Drupal, un CMS Open Source. C'est donc Drupal qui organise la base de données et qui crée des tables en fonction des modules activés. Cette base de données contient, à l'heure actuelle, plus de 1000 tables. Même si beaucoup de celles-ci sont des tables de cache et de backup (elles ne nous intéressent donc pas), la compréhension générale de la base de données et la récupération des données utiles sont relativement compliquées. En effet, les données intéressantes sont réparties dans plusieurs tables selon une logique propre au moteur Drupal. Pour faciliter les futurs accès aux données et pour ne pas chaque fois devoir effectuer le travail de compréhension de la logique de Drupal, il faut réaliser un script de migration vers une base de données qui répond à nos besoins.

Les données nécessaires pour réaliser le projet sont les résultats des utilisateurs aux différents challenges. Sur sa plateforme, EDITx propose 2 types de challenges :

- Des challenges d'entraînement : ils permettent aux utilisateurs de s'exercer dans une compétence et peuvent être réalisés plusieurs fois par une même personne, ce qui rend leurs résultats non-fiables pour l'analyse qui nous intéresse.
- Des challenges concours (ceux achetés par les entreprises) : ils ne peuvent être réalisés qu'une seule fois par un utilisateur. C'est donc sur la base de ces résultats que les analyses sont effectuées.

3.1.2 Nouvelle base de données relationnelle

La base de données créée contient tous les résultats des utilisateurs à des challenges de type concours. Son schéma entités-relations est illustré à la figure 31.

En observant ce schéma, on observe une table de jointure, ce qui n'est pas optimal pour 2 raisons :

- Les performances sont réduites
- Les requêtes sont moins facilement compréhensibles

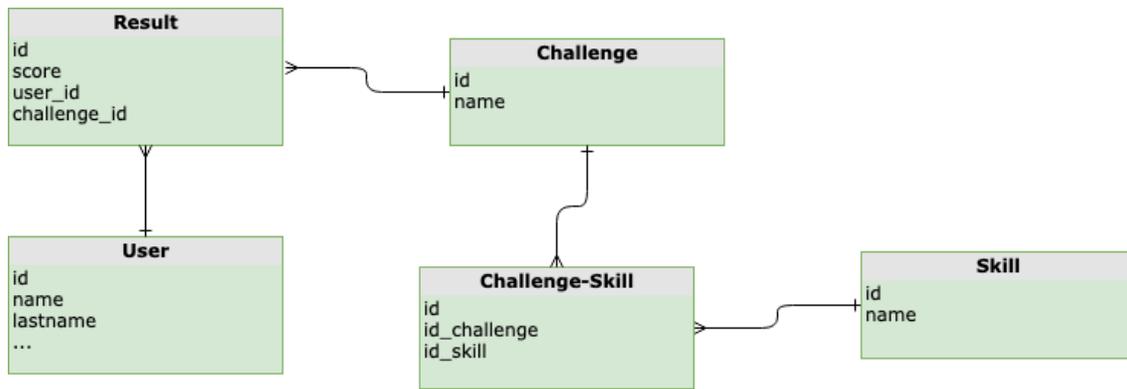


FIGURE 31 – Diagramme entités-relations

Ces points sont une partie des motivations qui poussent à migrer une nouvelle fois vers un nouveau type de base de données : une base de données graphe.

3.1.3 Nouvelle base de données neo4j

Une base de données neo4j est une base de données de type graphe. Cette nouvelle migration va apporter de nombreux avantages :

- Les requêtes sont simplifiées grâce au langage de requête CQL
- Plusieurs types de relations peuvent être créés
- Les performances seront meilleures
- La flexibilité dans la gestion des données est améliorée

La base de données graphe suit le modèle illustré à la figure 32.

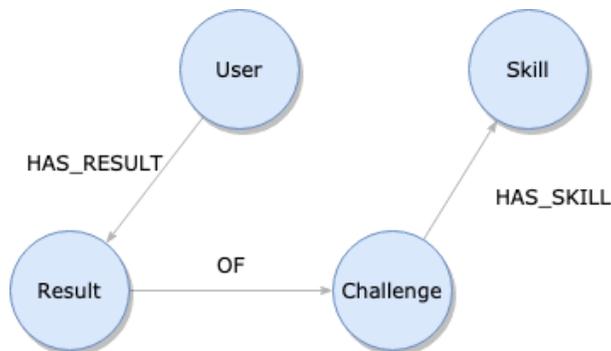


FIGURE 32 – Schéma de la base de données graphe

3.1.4 Tests de migration

Les tests de migration ont pour objectif de vérifier que la migration s'est déroulée correctement.

Pour clarifier, on effectue une double migration :

1. Drupal → base de données relationnelle : on part de la base de données Drupal vers une base de données avec un schéma défini plus propice à une exploitation des données.
2. Base de données relationnelle → base de données graphe : la migration s'effectue à partir de la base de données précédemment créée pour aller vers une base de données neo4j qui permet une réalisation des requêtes plus efficace.

Les 2 tests s'effectuent de la même manière : ils réalisent, sur chacune des 2 bases de données testées, des requêtes qui doivent produire un résultat identique. Ensuite, ils comparent l'équivalence de ces résultats.

La réalisation et les résultats des tests montrent que les migrations se sont bien effectuées. Les données sont donc considérées comme fiables pour la suite du travail. Il faut garder à l'esprit que ces tests vérifient seulement que les transferts de données d'une base à l'autre se sont bien effectués. En aucun cas, ceux-ci ne vérifient par exemple, la bonne compréhension de la base de données Drupal.

3.2 Format des données

Les algorithmes présentés dans la suite de ce travail reçoivent des données en entrée pour pouvoir produire des analyses ou des modèles. Il semble donc intéressant de définir un format standard avec lequel les données sont présentées aux algorithmes. Dans le jargon de la science des données, on parle de dataset. Tous ces algorithmes basent leur fonctionnement sur la recherche de liens entre différents résultats de challenges. Les datasets sont donc des matrices dans lesquelles une ligne représente un utilisateur et une colonne représente un challenge lié à une certaine compétence.

		<i>php</i>				...
Jean	8	15	12	20	12	...
Eric	10	5	13	10	13	...
Valerie	13	10	12	14	15	...
Nassim	7	16	13	19	8	...
Pierre	5	6	10	8	18	...
Emile	14	12	13	9	13	...
...

3.3 Analyse basique des données

Sur la plateforme EDITx, tous les utilisateurs ne participent pas à tous les challenges, on ne dispose donc pas d'un dataset général qui contient un score dans chaque challenge pour chaque utilisateur. À défaut nous sommes contraints de créer des datasets plus petits à partir d'utilisateurs qui ont participé à plusieurs challenges. Pour obtenir les plus grands datasets possibles, il faut chercher les groupes de challenges qui possèdent le plus d'utilisateurs communs.

La base d'EDITx contient pour le moment des résultats pour 16 challenges, il faut donc trouver le nombre d'utilisateurs communs pour toutes les combinaisons possibles de 2, 3 ou 4 challenges (pour des groupes plus grands ce nombre est trop petit).

La recherche de ces groupes est un problème qui croit exponentiellement quand on augmente la taille des groupes ou quand on ajoute des challenges dans la base de données. En effet, on cherche le nombre d'utilisateurs communs pour chaque combinaison possible de n_{taille} parmi $n_{challenge}$ sans permutation possible. Ce qui revient à :

$$n_{groupes} = \frac{n_{challenge}!}{(n_{taille}!)(n_{challenge} - n_{taille})!}$$

La complexité exponentielle évoquée ne pose pour le moment pas de réel problème, car la quantité de données est limitée et que par conséquent cela ne sert à rien de considérer des groupes avec un grand nombre de challenges. Cependant, si la taille de la base de données d'EDITx augmente, on devra par exemple retrouver le nombre d'utilisateurs communs pour toutes les combinaisons de 25 parmi 50 challenges, soit pour plus de 100

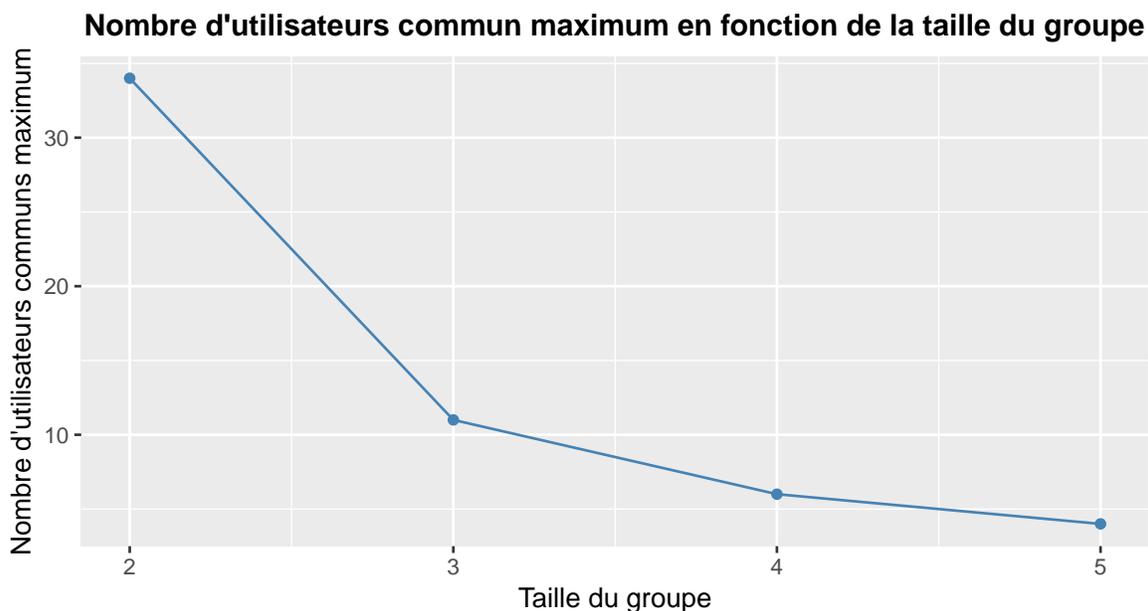


FIGURE 33 – Plus grands nombres d'utilisateurs communs.

Tops des utilisateurs communs	
Compétences	# utilisateurs
Java - Java	34
.NET - DevOps	31
Python - Java	29
Java - Java	29
IT - Business A	27

TABLE 31 – Top 5 des groupes d'utilisateurs communs

milliards de groupes.

On remarque directement sur la figure 33 que le nombre d'utilisateurs communs est très petit, même pour des groupes de 2 challenges. Le tableau 31 suivant présente le top 5 des combinaisons de 2 challenges avec le plus d'utilisateurs communs, ainsi que les compétences liées à ces challenges.

Ce peu d'utilisateurs communs engendre des datasets de petite taille qui ne permettent pas de construire des modèles assez généralistes. Une méthode destinée à résoudre ce problème de manque de données est exposée dans la suite de ce travail.

3.4 Génération de données

Comme cela a été observé ci-dessus, le manque d'utilisateurs communs sur plusieurs challenges implique des datasets peu volumineux (le dataset le plus important ne possède que 34 observations), or la recherche de modèle performant en terme de généralisation demande une quantité importante de données.

Dans le but de résoudre ce problème, on a recours à une méthode statistique qui permet d'étendre le dataset en restant le plus fidèle possible aux données de départ. Elle consiste à estimer une loi normale multidimensionnelle et de générer des observations à partir de cette loi. Pour illustrer cette méthode, on utilise les données du dataset *Python-Java* du tableau 31.

On dispose de n observations (une par utilisateur) qui possèdent chacune m variables (une par score dans un challenge).

La loi normale à l'avantage d'être paramétrée, c'est-à-dire qu'on peut trouver sa fonction de densité à partir de 2 paramètres : un vecteur de scalaire $\boldsymbol{\mu} \in \mathbb{R}^m$ qui contient les espérances mathématiques des variables et la matrice Σ qui est la matrice de covariance des différentes observations. La densité de probabilité s'écrit alors :

$$f(\mathbf{x}|\boldsymbol{\mu}, \Sigma) = \frac{1}{(2\pi)^{\frac{n}{2}} |\Sigma|^{\frac{1}{2}}} e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T \Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu})}$$

Pour estimer la loi normale d'un dataset, on calcule les paramètres grâce aux estimateurs du maximum de vraisemblance, soit :

$$\hat{\boldsymbol{\mu}} = \frac{1}{n} \sum_{i=1}^n \mathbf{x}_i$$

$$\hat{\Sigma} = \frac{1}{n-1} \sum_{i=1}^n (\mathbf{x}_i - \boldsymbol{\mu})(\mathbf{x}_i - \boldsymbol{\mu})^T$$

La figure 34 montre la densité de probabilité multidimensionnelle estimée pour le dataset *Java - Python*.

Une fois cette densité de probabilité estimée, on peut générer autant d'échantillons qu'on souhaite et tout cela est très facilement réalisé à partir d'un petit script *R*.

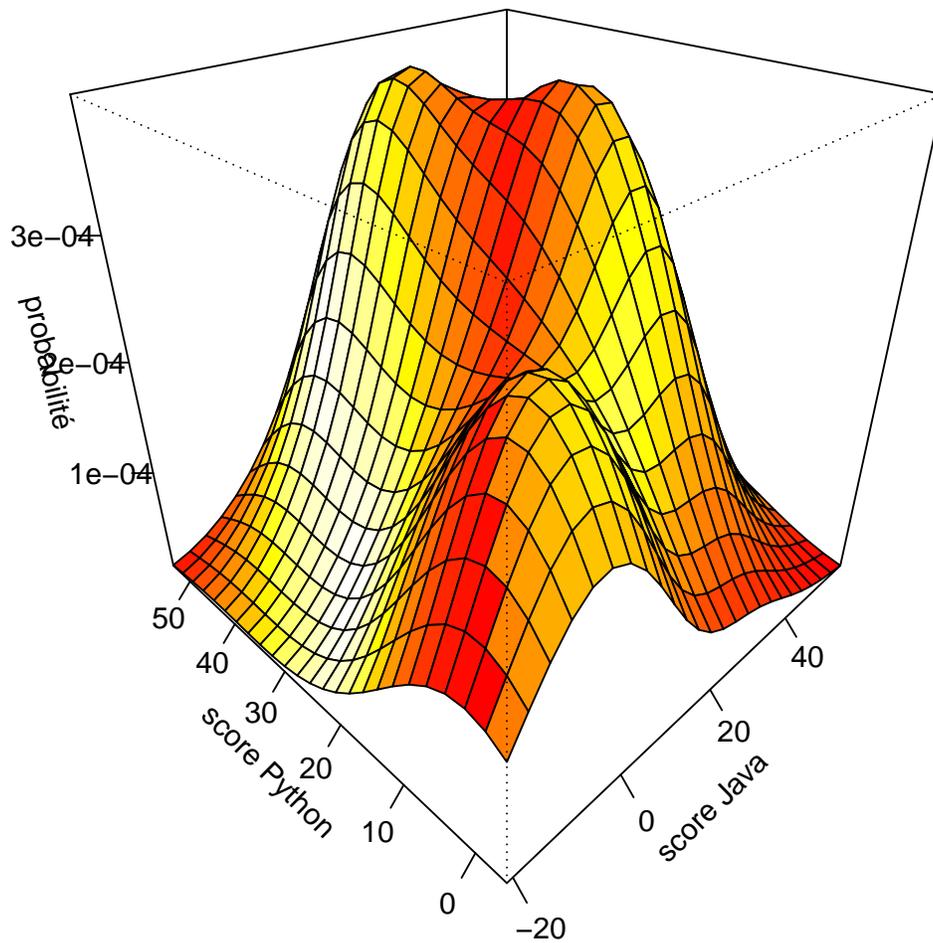


FIGURE 34 – Densité de probabilité estimée.

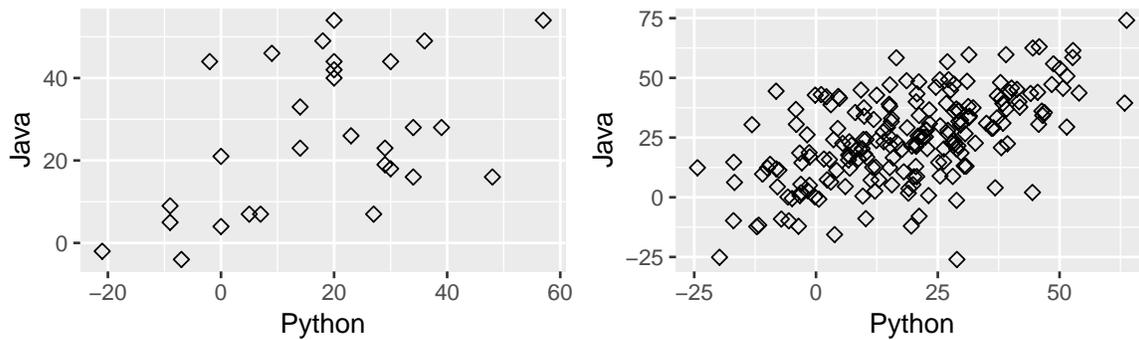


FIGURE 35 – Datasets avant et après la génération.

```
# compute the mean vector and the covariance matrix
sigma <- cov(data)
mu <- c(mean(data[,1]), mean(data[,2]))

# generate 200 new samples
samples <- mvrnorm(n=200, mu, sigma)
```

3.5 Nettoyage des données

Une des difficultés majeures dans le processus de recherche d'un modèle performant est la présence de données aberrantes dans le dataset. Elles peuvent avoir plusieurs origines: être propres au processus qui les génère ou bien être simplement le résultat d'une erreur de manipulation (par exemple un mauvais enregistrement en base de données). Ces données vont avoir un impact disproportionné sur le modèle généré: combinées à de l'overfitting, elles vont fortement augmenter la variance de celui-ci.

En prenant en compte ces considérations, il semble important de mettre en place une méthode de nettoyage des données qui va permettre aux datasets de se débarrasser des données aberrantes. La méthode envisagée est appelée .

La distance de Mahalanobis d'une observation est une mesure de sa distance dans un espace multivarié qui tient compte de la variance et de la corrélation des variables. Là où la distance euclidienne traite toutes les variables d'une observation de la même manière, la distance de Mahalanobis donne moins d'importance aux variables les plus

corrélées. Elle est formellement définie comme ceci:

$$D_M(x)_i = \sqrt{(\mathbf{x} - \boldsymbol{\mu})^T \Sigma^{-1} (\mathbf{x} - \boldsymbol{\mu})}$$

Si les variables sont non-corrélées ($\Sigma = \mathbb{I}$), la distance de Mahalanobis est égale à distance euclidienne de l'observation centrée.

Si une observation possède une distance de Mahalanobis importante, cela signifie que ses variables peu corrélées avec les autres, sont éloignées de leur moyenne. On peut assui comprendre sa valeur de manière visuelle: si un ensemble d'observations forment un nuage de points autour d'un(e) droite/plan/hyperplan représentant la corrélation entre les variables, les observations avec une distance de Mahalanobis élevée se trouvent éloignées de ce nuage de points.

Les observations aberrantes sont donc les observations dont la distance de Mahalanobis est supérieure à une distance de Mahalanobis critique.

En posant l'hypothèse que le dataset suit une loi normale de dimension m , d'espérance $\boldsymbol{\mu}$ et avec la matrice de covariance Σ :

$$X \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$$

Le carré de la distance de Malahanobis suit une loi du χ^2 à m degré de liberté:

$$D_M^2(X, \boldsymbol{\mu}) \sim \chi_m^2$$

La valeur de Malahanobis critique (élevée au carré) est la valeur qui définit le quantile $\chi_{p;0.95}^2$. On accepte donc toutes les valeurs dont la distance de Mahalanobis élevée au carré tombe dans ce quantile $\chi_{p;0.95}^2$.

Le script de la fonction qui implémente cette méthode est disponible en annexe.

Pour illustrer cette méthode, on l'applique sur le dataset *.NET - DevOps* (voir le tableau 31). Le dataset avant et après le nettoyage est affiché à la figure 36.

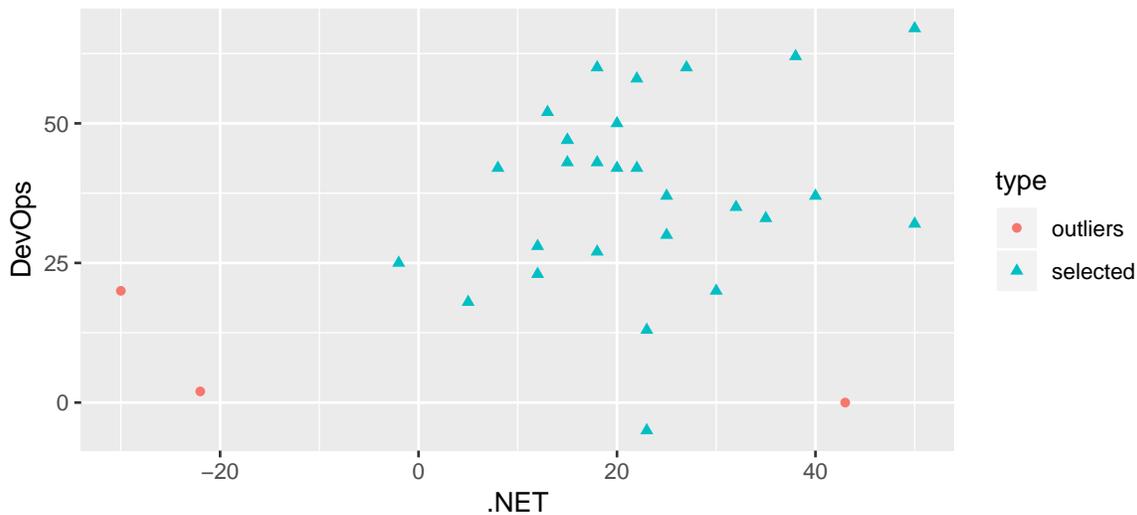


FIGURE 36 – Détection des valeurs aberrantes.

3.6 Normalisation des variables

Tous les challenges récupérés dans la base de données n'ont pas le même domaine de score possible : le challenge a peut produire des scores entre -20 et 20 et le b entre -50 et 100. Cette différence d'échelle entre les variables peut poser des problèmes :

- Certains algorithmes peuvent être ralenti
- On ne peut pas évaluer le degré des corrélations entre les variables.

Pour ne pas rencontrer ces deux problèmes, on normalise les datasets, c'est-à-dire que toutes les variables sont mises à la même échelle. La méthode classique consiste à utiliser le procédé appelé *centrage - réduction* : pour chaque variable, on soustrait la moyenne de la variable à toutes les observations et on divise par l'écart type de la variable.

$$\mathbf{x}_{\text{norm}} = \frac{\mathbf{x} - \mu}{\sigma}$$

3.7 Résumé des méthodes présentées

Cette section a exposé comment générer des données supplémentaires, comment les nettoyer pour supprimer les données aberrantes et comment les normaliser. La figure 37 montre l'évolution du dataset *.NET - DevOps* auquel on applique ces différentes techniques.

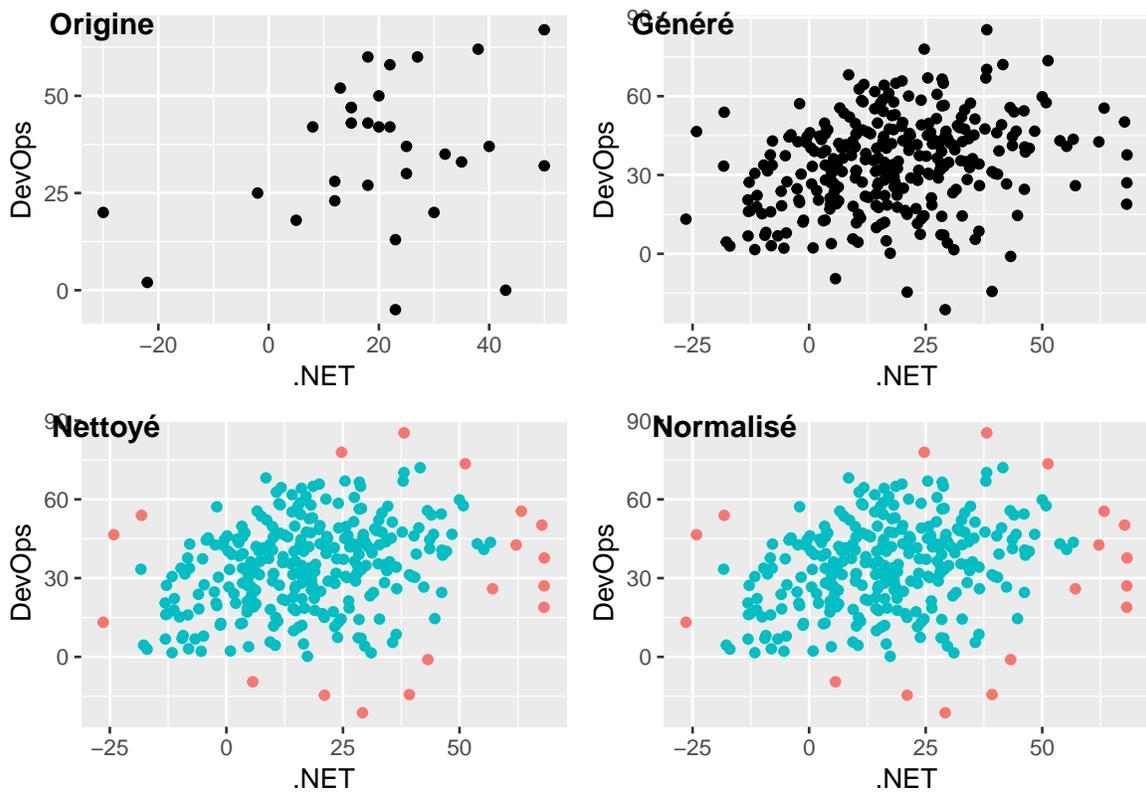


FIGURE 37 – Evolution du dataset .NET-DevOps.

Datasets EDITx générés et nettoyés			
id	Compétences	d'origine	généré nettoyé
GEN1	Python/Java	29	285
GEN2	.NET/DevOps	31	286
GEN3	C/IT/Java	10	282
Datasets ECAM nettoyés			
id	Compétences	d'origine	d'origine nettoyé
NGEN1	POO/EEE	49	46
NGEN2	Python+/POO/EEE	41	40
NGEN3	POO/EEE/TDD/Python+	29	28

TABLE 32 – Datasets utilisés

3.8 Choix des datasets utilisés

En associant toutes les techniques présentées dans cette section, on peut générer des datasets qui sont présentables aux algorithmes de la section suivante.

Comme cela a été montré, pour plus de 3 challenges, le nombre d'utilisateurs communs est très faible, tellement que des estimateurs calculés pour la génération de données ne seraient pas fiables. On se limite donc à des datasets avec maximum 3 colonnes de challenge.

Le tableau 32 présente l'ensemble des datasets qui sont testés sur les algorithmes de la section suivante. Dans la première colonne, on trouve les identifiants des datasets (GEN pour généré, NGEN pour non généré). La seconde colonne montre les compétences liées au dataset. La troisième colonne indique le nombre d'observations récupérées dans la base de données et la dernière colonne montre le nombre d'individus après la génération et le nettoyage. Le nombre de données générées dans les 3 premiers datasets n'est pas anodin, il correspond à ce qu'on pourrait retrouver dans la base de données EDITx à moyen terme.

Pour également tester les algorithmes sur des données non-générées, on utilise des données récoltés à l'ECAM et anonymisées. Ces datasets sont un petit peu plus grands que ceux d'EDITx.

4 Algorithmes

4.1 Remarques

4.1.1 Résultats

Chaque modèle présenté dans cette section va générer des résultats qui méritent de nombreuses analyses. Afin d'éviter les répétitions, les analyses complètes ne sont faites que pour certains datasets. Néanmoins, à chaque fois, un tableau récapitulatif reprend l'évaluation du modèle pour tous les datasets.

4.1.2 Évaluation des modèles

Pour évaluer un modèle, il faut mesurer sa capacité de généralisation. Généralement, on divise le dataset en 2 et on utilise une partie pour entraîner le modèle et l'autre pour l'évaluer. Étant donné que nos datasets ne contiennent pas beaucoup d'observations, on va utiliser une autre méthode : la *cross-validation*.

Dans cette méthode, on divise le dataset en q partitions, une de ces partitions est choisie comme *validation set* pendant que les autres forment le *training set* et sont utilisées pour calculer le modèle. Une fois que le modèle est calculé, on calcule la MSE (*mean square error*) sur le *validation set* pour tester les performances du modèle estimé. Cette étape est réalisée q fois de manière à ce que les q partitions jouent chacune une fois le rôle de validation set. La *MSE* de {cross-validation} est la moyenne des q *MSE* calculées. La *MSE* pour une étape de *cross-validation* se calcule avec la formule suivante :

$$MSE = \frac{1}{p} \sum_{i=1}^p p(\hat{y}_i - y_i)$$

Où p est le nombre d'observations dans le *validation set*.

Un modèle peut être considéré comme performant si sa *MSE* de *cross-validation* est inférieure à 0.2. En effet, étant donné que les données sont normalisées, cela signifierait que l'erreur moyenne de prédiction du modèle est égale à un cinquième de l'écart-type de la variable qu'il prédit.



FIGURE 41 – Cross-validation

4.2 Régression linéaire multivariée

4.2.1 Présentation

La régression multivariée est une méthode permettant de mettre en évidence les relations linéaires existant entre une variable cible et plusieurs autres variables observées.

Soit un ensemble de m observations composées de n variables $(y_i, x_{i1}, \dots, x_{i(n-1)})$, où y_i est la variable cible et les $x_i = (x_{i1}, \dots, x_{i(n-1)})$ sont les variables observées. La régression linéaire va trouver la combinaison linéaire des variables observées qui prédit le mieux la variable cible. C'est-à-dire qu'il faut trouver les coefficients β_j de la combinaison linéaire qui minimise l'erreur entre la valeur prédite par la régression et la vraie valeur de la variable cible y_i .

Cette recherche des meilleurs paramètres peut s'écrire sous la forme d'un système de m équations linéaires :

$$\begin{bmatrix} x_{01} & \dots & x_{0(n-1)} \\ \dots & \dots & \dots \\ \dots & \dots & \dots \\ x_{(m-1)1} & \dots & x_{(m-1)(n-1)} \end{bmatrix} \begin{bmatrix} \beta_1 \\ \dots \\ \beta_{(n-1)} \end{bmatrix} = \begin{bmatrix} y_0 \\ \dots \\ \dots \\ y_{(m-1)} \end{bmatrix}$$

Généralement, ces systèmes n'ont pas de solution, il faut donc une autre méthode pour trouver les paramètres β_j .

4.2.1.1 Fonction hypothèse

La fonction hypothèse est la fonction approximée. C'est une certaine combinaison linéaire des x_i :

$$h(x_i) = \beta_1 x_{i1} + \dots + \beta_{(n-1)} x_{i(n-1)}$$

Dans la plupart des applications, on rajoute un coefficient β_0 qui est appelé biais.

$$h(x_i) = \beta_0 + \beta_1 x_{i1} + \dots + \beta_{(n-1)} x_{i(n-1)}$$

4.2.1.2 Erreur individuelle et totale

L'erreur individuelle d'une observation se calcule comme la différence de la fonction hypothèse $h(x_i)$ évaluée avec la valeur de la variable cible de l'observation :

$$e_i = h(x_i) - y_i = \hat{y}_i - y_i$$

L'erreur totale est la moyenne des erreurs individuelles au carré sur l'ensemble des observations :

$$E = \frac{\sum_m (h(x_i) - y_i)^2}{m}$$

L'erreur totale est appelée fonction de coût :

$$J(\beta) = E = \frac{\sum_m (h(x_i) - y_i)^2}{m}$$

4.2.1.3 Descente de gradient

L'idée de la descente de gradient est de considérer l'erreur totale comme une fonction des paramètres $(\beta_0, \dots, \beta_{(n-1)})$. Cette fonction étant convexe, il est possible de trouver le minima global en calculant le gradient selon ces paramètres. Avec ce gradient, on calcule des nouvelles valeurs de paramètres qui diminuent le coût total jusqu'à atteindre l'optimum.

$$\beta_j = \beta_j - \eta \frac{\partial E}{\partial \beta_j} = \beta_j - \eta \sum_{i=1}^m \frac{(-x_{ij})(y_i - h(x_i))}{m}$$

L'hyperparamètre η est appelé *learning rate*, il permet de choisir la vitesse à laquelle la descente de gradient converge vers le minimum. S'il est trop élevé, on risque de rater le minimum, s'il est trop faible, la recherche de l'optimum est lente.

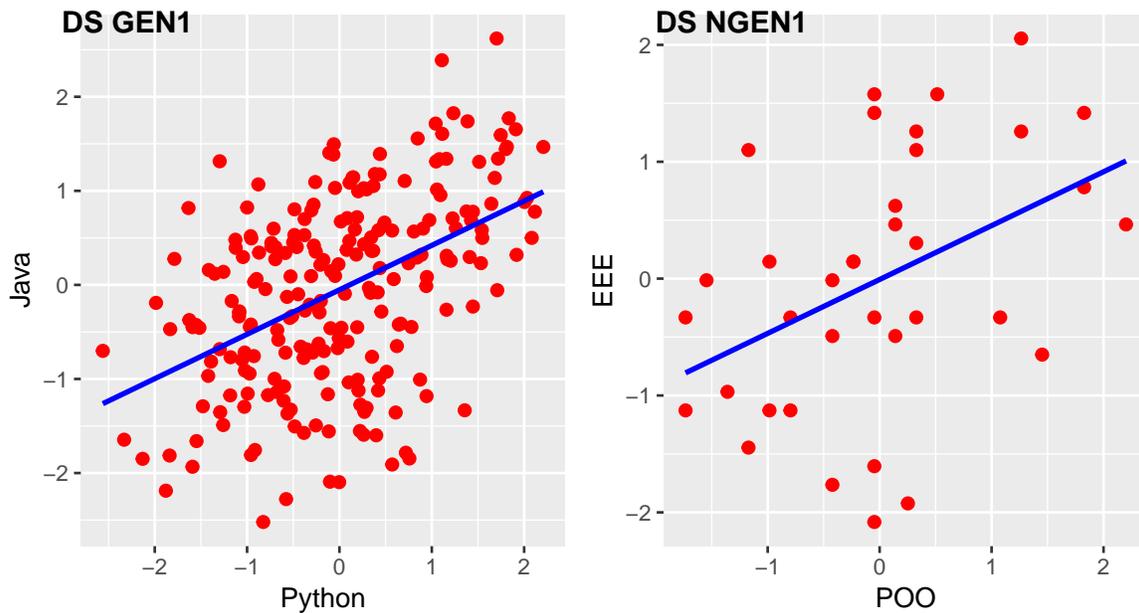


FIGURE 42 – Régression linéaire sur le dataset GEN1 et NGEN1.

4.2.2 Utilisation

Dans notre cas d'utilisation, la régression va permettre de prédire un score dans un challenge en fonction des scores dans d'autres challenges. Le désavantage de ce cas d'utilisation est que toutes les variables peuvent à la fois être des variables observées et des variables cibles. En effet, on ne doit pas prédire les scores que pour un seul challenge à partir de certains scores de challenges prédéfinis, mais pouvoir prédire les scores pour n'importe quelle variable du dataset à partir des autres variables.

En pratique, ce désavantage ne va pas poser de problème, car l'algorithme va simplement calculer les paramètres d'une fonction affine. Par une simple manipulation de cette dernière, on peut expliquer n'importe quelle variable à partir des autres.

4.2.3 Application

La représentation d'une régression linéaire dans un espace à 2 ou 3 dimensions est très explicite : elle y prend respectivement la forme d'une droite et d'un plan.

Dans cette optique, les résultats des régressions effectuées sur des datasets à 2 (*GEN1* et *NGEN1*) et 3 variables (*GEN3* et *NGEN2*) sont respectivement exposés aux figures

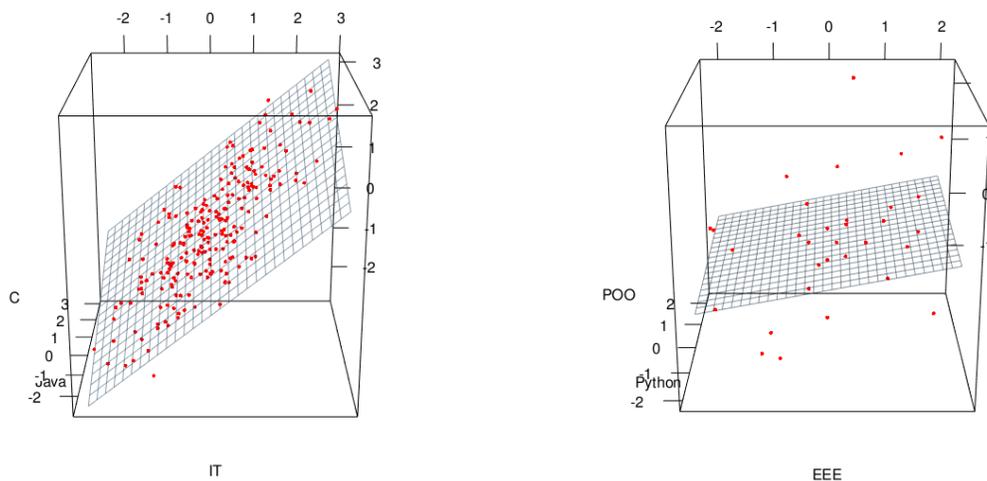


FIGURE 43 – Régression linéaire sur les datasets GEN3 et NGEN2

42 et 43. Dans ces illustrations, chaque axe représente les scores dans un challenge. Les utilisateurs qui ont chacun plusieurs scores (2 ou 3 dans les illustrations) forment un nuage de points dans l'espace formé par les différents axes. Pour rappel, les données ont été normalisées en calculant les versions centrées-réduites des variables, l'unité d'un axe est donc l'écart type de la variable lié à cet axe, calculé sur le dataset. Par exemple, pour le graphique de gauche de la figure 42, on place un utilisateur en mettant son score normalisé en *Python* sur l'axe des abscisses et son score normalisé en *Java* sur l'axe des ordonnées.

De manière générale, on voit que les nuages de points provenant de datasets générés (*GEN1* et *GEN3*) sont mieux organisés autour de la droite ou du plan de régression. En fait, ce résultat est intrinsèque au processus de génération designé (loi normale multidimensionnelle). Pour les datasets non-générés (*NGEN1* et *NGEN2*), les points sont plus dispersés et écartés de la droite ou du plan de régression. Ces écarts entre les valeurs prédites par le modèle et les vraies valeurs des observations sont appelés résidus ϵ_i et se calculent avec :

$$\epsilon_i = \hat{y}_i - y_i$$

Le langage *R* nous permet de facilement calculer une régression linéaire et d'observer

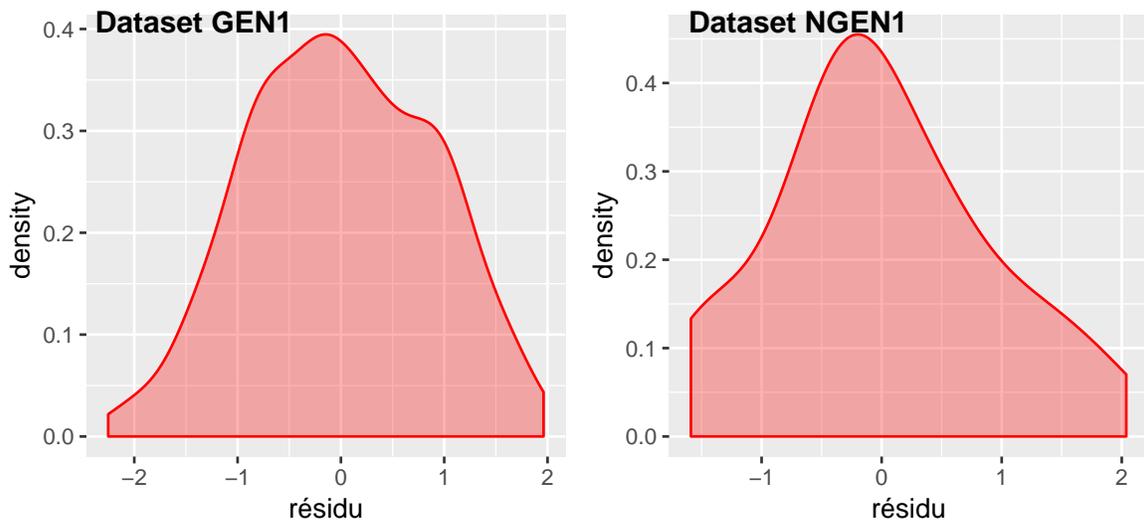


FIGURE 44 – Distribution des résidus de la régression sur le dataset GEN1 et NGEN1

les résultats qui donnent des indications précieuses sur le degré de fiabilité du modèle. En pratique, le calcul d'une régression et de ses résultats associés ne nécessite l'usage que d'une seule fonction : $lm()$.

```
# define the data
data <- ds_3$train

# compute the model
model <- lm(Python ~ Java, data=data)
summary(model)
```

4.2.3.1 Analyse des résidus

Il y a souvent une erreur entre la variable prédite par le modèle et la vraie valeur se trouvant dans le dataset, on peut donc écrire :

$$y_i = \beta_0 + \beta_1 x_{i1} + \dots + \beta_{(n-1)} x_{i(n-1)} + \epsilon_i$$

L'erreur ϵ_i est appelée résidu et est générée par un processus aléatoire qui suit une certaine distribution. L'identification de cette distribution peut d'un coté nous en apprendre plus sur la nature des résidus et d'un autre nous aider à prévoir leur comportement.

Sur la figure 44 on voit que les résidus issus des régressions sur les datasets *GEN1* et *NGEN1* suivent approximativement une loi normale. Pour le dataset *GEN1* cette observation est prévisible et c'est encore dû au processus de génération des données (voir chapitre 3) qui utilise une loi normale multidimensionnelle. Celle-ci impose que pour des valeurs fixées des variables observées, la distribution de la variable cible va être proche de la distribution normale. Ces distributions normales observées sont une bonne chose, car elle confère aux résidus un caractère aléatoire facilement modélisable.

Appliquée à notre cas d'utilisation, cette première observation tient la route, elle dit ceci : des utilisateurs obtiennent chacun le même score dans un challenge *a* (la variable observée), les scores dans le challenge *b* suivront une distribution normale conditionnée par le premier score obtenu.

Dans la pratique, le fait que les résidus soient normalement distribués est une condition nécessaire pour calculer la signification statistique des paramètres β_i du modèle calculé (la raison de cette condition est très mathématique, mais n'est pas expliquée dans ce travail).

4.2.3.2 Signification statistique des paramètres

Pour vérifier la signification statistique des paramètres, il faut vérifier la *p-value* qui leur est associée. La *p-value* est une probabilité qui en dessous d'un seuil de signification permet de rejeter une hypothèse nulle. Ici, l'hypothèse nulle est que le paramètre β_i est égal à 0 et donc que les deux variables reliées par ce paramètre sont non-corrélées. La *p-value* peut donc être vue comme la probabilité de tomber sur la valeur calculée du paramètre en partant du principe que l'hypothèse nulle est vraie. Si cette probabilité est très petite (le seuil standard est de 0.05), l'hypothèse nulle peut être rejetée et le paramètre validé.

Les *p-values* pour la régression sur le dataset *GEN1* sont :

$$\beta_0 \rightarrow 0.7492086 \qquad \beta_{Java} \rightarrow 6.063604 \times 10^{-14}$$

Les *p-values* pour la régression sur le dataset *NGEN1* sont :

$$\beta_0 \rightarrow 0.9170066 \qquad \beta_{EEE} \rightarrow 0.0102512$$

Toutes les *p-values* sont inférieures à 0.05 sauf pour les paramètres β_0 . En fait, c'est normal, β_0 est le biais du modèle, il n'indique pas une corrélation entre deux variables, c'est donc normal que l'hypothèse nulle soit validée pour lui. Les paramètres sont donc tous statistiquement significatifs.

4.2.3.3 R-carré

La mesure R^2 indique à quel point la variable cible est expliquée par le modèle. Elle se calcule avec la formule suivante :

$$R^2 = 1 - \frac{\sum_i^n (y_i - \hat{y}_i)^2}{\sum_i^n (y_i - \bar{y}_i)^2}$$

Au plus la somme des résidus est faible par rapport à la variance de la variable cible, au plus la valeur de R^2 est élevée. Une valeur de R^2 élevée indique donc que le modèle fait des bonnes prédictions sur le *training set*, mais il n'indique en rien que le modèle fera des bonnes prédictions sur des nouvelles valeurs.

Pour le dataset *GEN1* le R^2 calculé est :

$$R_{GEN1}^2 = 0.2211053$$

Cette mesure indique que les prédictions réalisées sur le *training set* ne sont pas précises, l'ajout d'une ou plusieurs variables observées pourrait améliorer cette précision, malheureusement les données disponibles pour l'instant chez EDITx ne permettent pas ces ajouts.

Pour le dataset *NGEN1* le R^2 calculé est :

$$R_{NGEN1}^2 = 0.1785367$$

Ici aussi, cette mesure est faible, mais dans ce cas, on possède des datasets avec plus de variables observées pour la même variable cible. Par exemple avec le dataset *NGEN3*, le R^2 de la régression linéaire est :

$$R_{NGEN3}^2 = 0.3878002$$

L'ajout de variables observées permet d'augmenter la valeur du R^2 . Cependant, l'ajout d'une variable n'améliore pas toujours le R^2 , elle peut aussi simplement augmenter la valeur des résidus. La valeur du R^2 doit être utilisée avec précaution, car quand elle est élevée est peut aussi indiquer une situation d'overfitting.

4.2.4 Tableau récapitulatif

Ce tableau récapitulatif reprend les performances de *cross-validation* pour les régressions contruites à partir des différents datasets.

Performances	
dataset	MSE cross-validation
GEN1	8.06e-01
GEN2	9.54e-01
GEN3	1.53e-01
NGEN1	8.71e-01
NGEN2	1.05e+00
NGEN3	7.94e-01

TABLE 41 – Performances des régressions linéaires

Le modèle construit sur le dataset *GEN3* est le seul qui présente des bonnes performances de prédiction. Mais ce dataset a été généré, pour qu'il ait des aussi bonnes performances sur de vraies nouvelles valeurs (non-générées), il faut que les données originales de ce dataset soient une bonne représentation du processus qui les crée. Pour les autres modèles, les mauvaises performances peuvent être dues à plusieurs raisons :

- Le modèle de régression linéaire n'est pas assez complexe, il faut donc essayer la régression polynomiale.
- Les variables observées ne sont tout simplement pas corrélées à la variable cible.
- Les données des datasets originaux ne sont pas suffisantes.

4.3 Régression polynomiale multivariée et la notion d’overfitting

4.3.1 Présentation

4.3.1.1 Modèle plus complexe

La régression polynomiale fonctionne comme la régression linéaire sauf qu’elle offre à la fonction hypothèse la possibilité d’être une fonction polynomiale de degré k , ce nombre k est un hyperparamètre du modèle. En fait, la régression linéaire est une régression polynomiale de degré 1. On dispose de m échantillons avec $(n - 1)$ variables observées et 1 variable cible. Dans le cas de 2 variables observées et pour $k = 2$, la fonction hypothèse s’écrit :

$$h(x_i) = b_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \beta_3 x_{i1}^2 + \beta_4 x_{i2}^2$$

Là où la régression linéaire permettait de trouver les paramètres d’une droite ($n = 1$), d’un plan ($n = 2$) ou d’un hyperplan ($n > 2$), la régression polynomiale aide à trouver les paramètres d’une courbe ($n = 1$), d’une surface ($n = 2$) ou d’une hypersurface ($n > 2$).

4.3.1.2 Overfitting

Chaque fois qu’on augmente le degré k du polynôme, la courbe/surface/hypersurface se retrouve avec un “pli” supplémentaire. En augmentant k , donc en ajoutant des “plis”, on finit par coller parfaitement aux données du *dataset*. Ce n’est pas une bonne chose, car le modèle possède alors une mauvaise généralisation et les prédictions sur des nouvelles données ne seront pas bonnes.

Dans ce genre de situation, quand le modèle colle trop aux données du *dataset* et prédit des mauvaises valeurs, on dit que le modèle se trouve en situation d’overfitting, c’est-à-dire qu’il est trop complexe.

On peut observer ce phénomène à la figure 45. Dans un premier temps, on génère des données (rouge) et on calcule des régressions pour différentes valeurs de k sur la base de ces données. Ensuite, on rajoute des nouvelles données (mauve) et on peut comparer avec les prédictions des différents modèles.

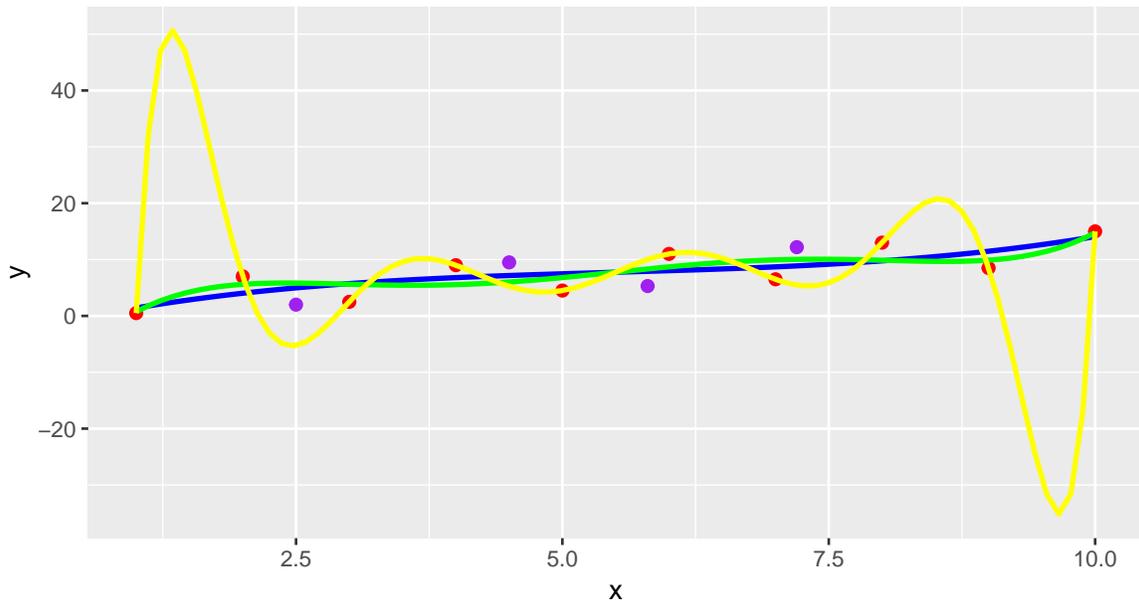


FIGURE 45 – Overfitting.

Pour résoudre le problème d'overfitting on utilise un principe appelé le compromis biais-variance. Le biais représente l'erreur totale entre les valeurs prédites et les vraies valeurs du dataset. La variance, elle, est liée à la complexité du modèle. En pratique, il faut trouver la valeur de l'hyperparamètre k qui rend le modèle ni trop simple, ni trop complexe. Pour cela, on réalise la *cross-validation* pour différentes valeurs de l'hyperparamètre k et on choisit celui qui maximise les performances (donc qui minimise la *MSE* de *cross-validation*).

4.3.2 Application

Comme pour la régression linéaire, pour se faire une idée rapide des régressions polynomiales pour divers degré k on observe leur représentation graphique. Les figures 46 et 47 représentent les régressions polynomiales respectivement calculées sur les datasets *GEN1/NGEN1* et *GEN3/NGEN2*.

Sur la figure 46 on observe des régressions pour des degrés différents. Les régressions avec un faible degré k (bleu, mauve) présente des courbes avec des plis doux. On voit ensuite qu'en augmentant trop le degré, le modèle tombe en situation d'overfitting.

Pour calculer une régression polynomiale et tous les résultats associés avec le langage

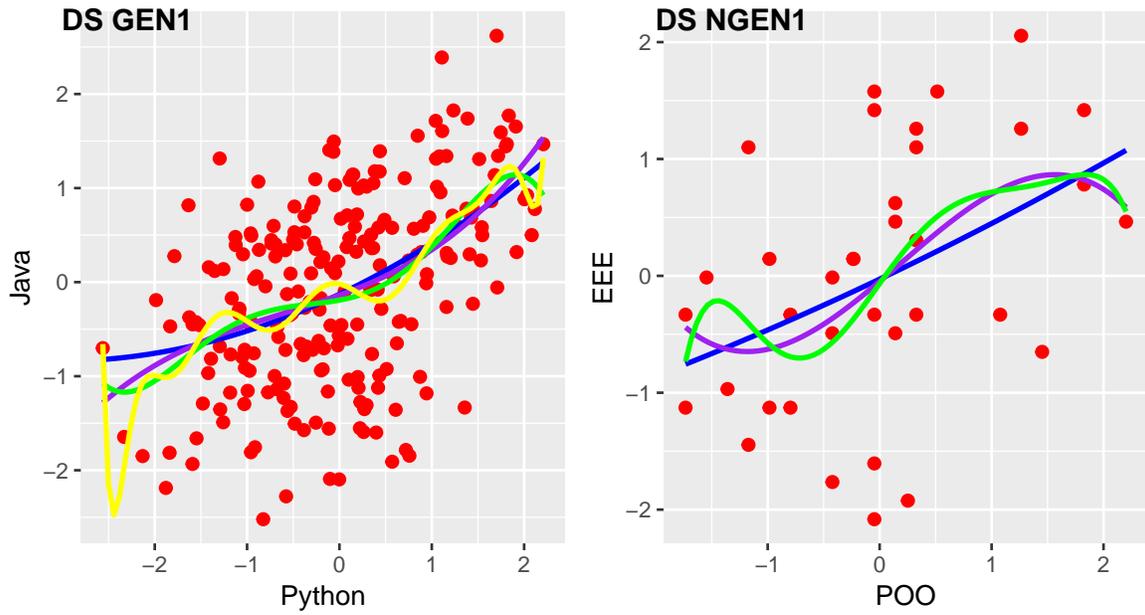
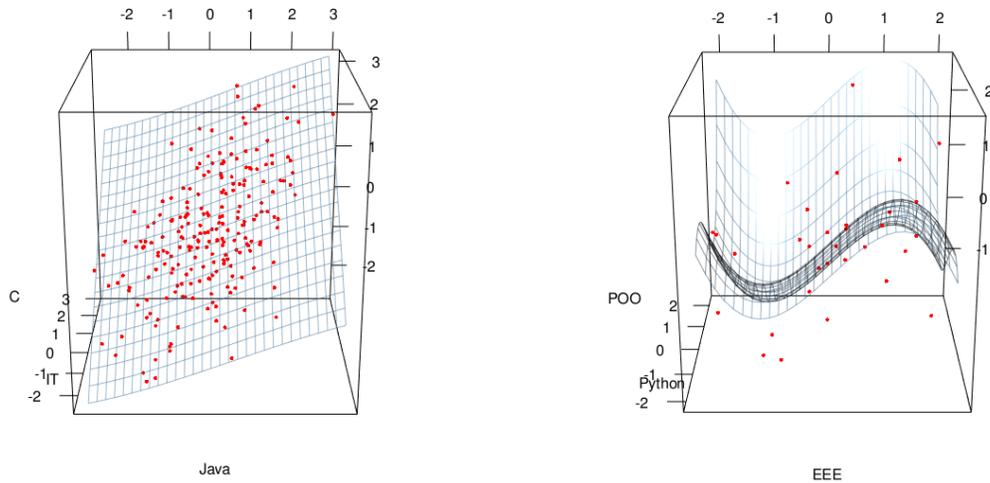


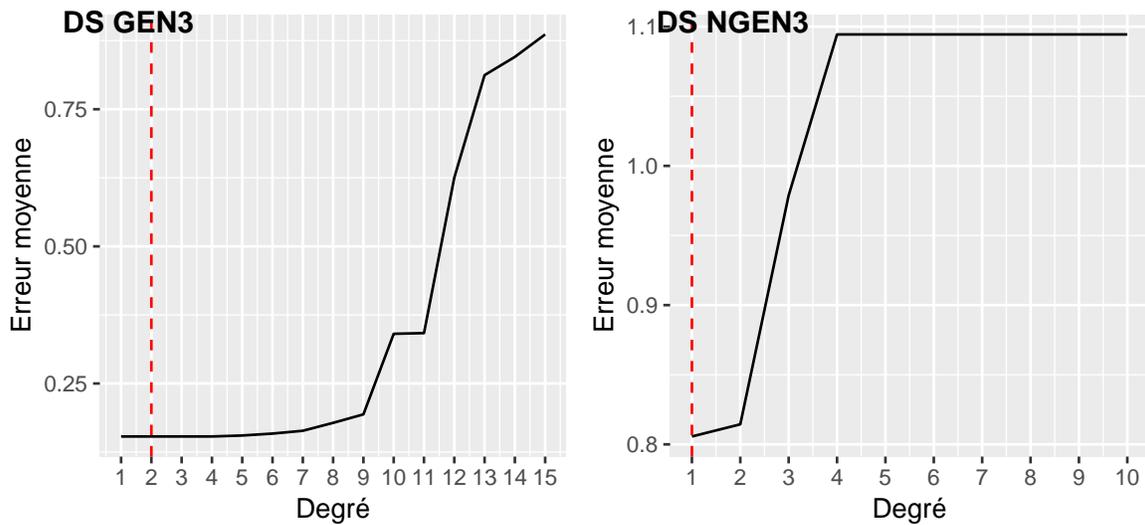
FIGURE 46 – Régressions polynomiales sur le dataset GEN1 et NGEN1.



(a) Dataset GEN3

(b) Dataset NGEN2

FIGURE 47 – Régressions polynomiales sur le dataset GEN3 et NGEN2

FIGURE 48 – Détermination du meilleur k .

R , il faut combiner la fonction $lm()$ avec la fonction $poly$.

```
# define the data
data <- ds_1_1$train

# compute the model
model <- lm(Python ~ poly(Java, degree=2), data=data)
summary(model)
```

4.3.2.1 Identification du meilleur k

Pour trouver le modèle de régression polynomial ni pas assez ni trop complexe, il faut trouver le degré k idéal. Dans cette optique, on utilise la *cross-validation* sur des modèles avec un degré k différent et on repère celui qui possède la plus faible *MSE* de cross-validation, le k associé à ce modèle est le k idéal.

La figure 48 représente l'évolution de la *MSE* de cross-validation quand on augmente le degré k pour les datasets *GEN3* et *NGEN3*. On peut directement y repérer le meilleur k . Cependant, pour le dataset *GEN3*, les degrés voisins au degré idéal produisent une *MSE* similaire. Cela est dû au fait que les régressions ont été régularisées, les “plis” ajoutés à la surface de régression par l'augmentation du degré k sont très doux, les surfaces pour ces différents degrés sont donc très similaires et les *MSE* de *cross-validation* aussi.

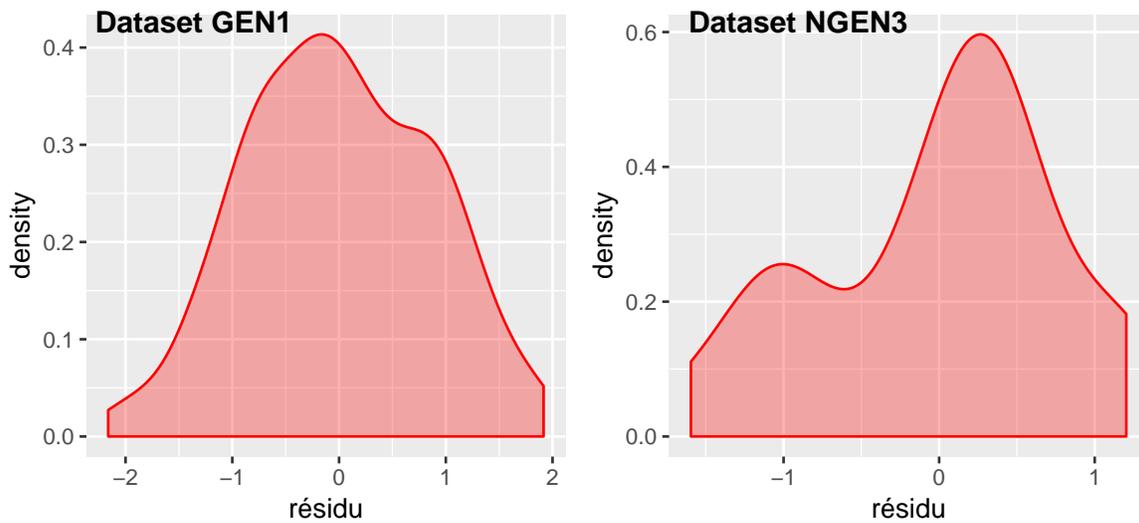


FIGURE 49 – Distribution des résidus de la régression polynomiale sur le dataset GEN1 et NGEN3.

On observe aussi sur ces graphiques que pour des degrés k trop grands, la MSE devient importante. Pour ces degrés, les modèles polynomiaux sont en situation d'overfitting, ils sont trop complexes et collent trop au *dataset*. Dans les 2 cas, cette situation arrive vite, au degré 3 pour le dataset *GEN3* et au degré 2 pour le dataset *NGEN3*.

4.3.2.2 Analyse des résidus

La figure 49 montre que comme pour la régression linéaire, les résidus des régressions polynomiales sont normalement distribués. Cette distribution normale des résidus nous permet de calculer la signification statistique des paramètres.

4.3.2.3 Signification statistique des paramètres

Pour le dataset *GEN3*, on a vu que le degré de régression optimal était le degré 2, néanmoins, on a vu que la MSE pour une régression de degré 1 est très similaire. On peut donc observer la signification statistique des paramètres.

On observe sur ce tableau que les paramètres des variables de degré 2 possèdent une p-value supérieure au seuil de 0.05, ce qui remet l'utilisation de régressions polynomiales en cause. En pratique l'analyse des significations statistiques des paramètres ne doit pas justifier l'exclusion d'un modèle, c'est la mesure de performance qui décide à la fin

Signification statistique		
-	1	2
β_{IT}	4.18e-73	9.84e-01
β_{Python}	3.10e-26	6.74e-01

TABLE 42 – Signification statistique des paramètres du modèle construit sur le dataset GEN3

quel modèle est le plus approprié.

4.3.2.4 Tableau récapitulatif

La tableau suivant présente les performances des modèles construits à partir des différents datasets.

Performances							
-	k = 1	k = 2	k = 3	k = 5	k = 6	k = 12	k = 25
GEN1	8.06e-01	8.09e-01	8.14e-01	8.07e-01	8.08e-01	1.10e+00	1.08e+06
GEN2	9.54e-01	9.52e-01	9.50e-01	9.59e-01	9.60e-01	1.00e+00	4.95e+02
GEN3	1.53e-01	1.55e-01	1.56e-01	1.67e-01	1.70e-01	4.00e+00	1.38e+05
NGEN1	8.71e-01	8.99e-01	9.39e-01	9.51e-01	1.37e+00	1.52e+04	7.85e+05
NGEN2	1.05e+00	1.08e+00	1.00e+00	2.90e+00	6.75e+00	4.04e+06	3.02e+20
NGEN3	7.94e-01	1.10e+00	1.84e+00	6.59e+01	7.17e+03	1.76e+19	1.83e+19

TABLE 43 – Performances des régressions polynomiales

Ce tableau nous montre que de manière générale, les meilleures performances surviennent pour le degré k égal à 1 et que dans tous les cas la régression polynomiale n'apporte pas d'amélioration par rapport à la régression linéaire. Comme l'ajout de complexité ne bonifie pas les performances, une autre solution pourrait être d'ajouter des variables observées. Malheureusement, la base de données d'EDITx ne permet pas ces ajouts, il faut donc pour l'instant se contenter de ces résultats pour les régressions. Les algorithmes suivants, plus complexes, abordent le problème de manière différente : par la réduction de la dimensionnalité ou par l'approximation de probabilités conjointes.

4.4 PCA (basée sur SVD)

4.4.1 Présentation

La PCA (*Principal component analysis*) est une application majeure de l'outil mathématique SVD (*Singular value decomposition*), dans le cadre de l'analyse de données. Cette méthode permet de détecter les principales variations d'un nuage de points dans un espace de grande dimension. Les directions de ces principales variations sont des vecteurs appelés composantes principales et forment un sous-espace appelé espace de sémantique. La PCA projette ensuite le nuage de points dans l'espace de sémantique. Les points sont donc représentés dans un espace de dimension inférieure à l'espace de départ.

Pour mieux comprendre les tenants et aboutissants de cet algorithme, on va poser les bases mathématiques de la SVD et de la PCA.

4.4.1.1 SVD

La SVD permet de diagonaliser une matrice A rectangulaire ($m \times n$) de rang r à partir de deux ensembles u_i et v_i de vecteurs singuliers.

Les u_i sont les vecteurs propres de AA^T ($m \times m$). Comme cette matrice est symétrique, les r premiers u_i forment une base orthonormée (avec $\|u_i\| = 1$) pour l'espace des colonnes de AA^T .

Les v_i sont les vecteurs propres de $A^T A$ ($n \times n$). Comme cette matrice est symétrique, les r premiers v_i forment une base orthonormée (avec $\|v_i\| = 1$) pour l'espace des lignes de $A^T A$.

AA^T et $A^T A$ ont les mêmes r valeurs propres positives non-nulles σ_i^2 . Les racines carrées de ces valeurs propres sont les valeurs singulières de A . Il est important de les classer dans l'ordre décroissant soit :

$$\sigma_1 > \sigma_2 > \dots > \sigma_r$$

Les deux bases orthonormées composées des vecteurs u_i et v_i diagonalisent la matrice A on a donc :

$$Av_i = \sigma_i u_i$$

Sous la forme de matrice, cela donne :

$$A_{m \times n} V_{n \times n} = U_{m \times m} \Sigma_{m \times n}$$

Comme V est une matrice composée de vecteurs formant une base orthonormée, on a :

$$A = U \Sigma V^T = \sigma_1 u_1 v_1^T + \dots + \sigma_r u_r v_r^T$$

La SVD décompose donc A en r matrices $\sigma_i u_i v_i^T$ de rang 1. Comme les σ_i sont classés par ordre décroissant, les termes à gauche dans la somme jouent un rôle important dans la matrice A , tandis que les termes les plus à droite sont quasiment insignifiants.

4.4.1.2 PCA

La PCA va utiliser la SVD, dans le sens où elle va repérer les composantes principales d'un nuage de points comme étant les vecteurs singuliers u_i de la matrice A (associée au nuage de points) possédant les valeurs singulières les plus significatives.

Pour mieux comprendre la *PCA*, on va réaliser un exemple à partir d'une matrice A_0 ($m \times n$ de rang $r = n$) qui contient 10 observations de 3 variables :

$$A_0^T = \begin{bmatrix} 10 & -3 & -7 & -5 & 11 & 9 & 10 & 1 & 9 & 11 & -9 & -12 & -6 & -7 & -2 & 10 & -10 \\ 3 & -15 & -12 & 9 & -6 & -4 & -9 & 3 & 14 & -10 & 7 & -9 & 7 & 8 & -1 & 1 & -1 \\ 0 & 1 & -1 & 2 & -2 & 1 & 0 & 0 & 0 & -1 & 2 & 1 & 0 & -1 & 1 & 1 & 2 \end{bmatrix}$$

4.4.1.2.1 Matrice de covariance

La première étape consiste à calculer la matrice centrée A . Pour cela, on soustrait à chaque entrée de la matrice, la moyenne calculée sur sa colonne :

$$a_{ij} = a_{0ij} - \mu_{0j}$$

On calcule ensuite la matrice de covariance :

$$S = \frac{A^T A}{m - 1}$$

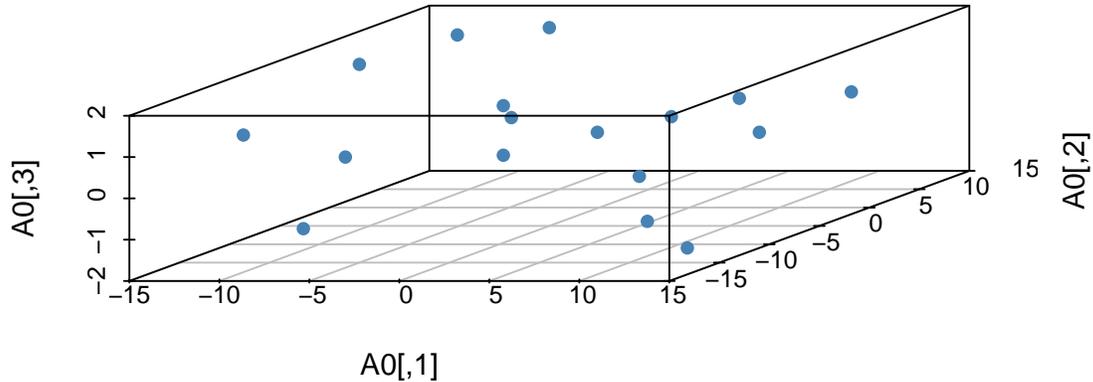


FIGURE 410 – Nuage de points de la matrice A.

$$S = \begin{bmatrix} 74.757 & -5.074 & -4.283 \\ -5.074 & 70.61 & 2.081 \\ -4.283 & 2.081 & 1.368 \end{bmatrix}$$

- $(S)_{ij}$ quand $i = j$: on a les variances s_i^2 pour les différentes variables
- $(S)_{ij}$ quand $i \neq j$: on a les covariances pour les différentes variables. Si la covariance est négative, alors les 2 variables varient de manière opposée. Si la covariance est positive, les 2 variables varient ensemble.

4.4.1.2.2 Composantes principales

On a vu grâce à la SVD que les “modes principaux” d’une matrice se calculaient à partir des valeurs et vecteurs propres des matrices AA^T et de $A^T A$. Les composantes principales de A_0 (et de A) sont donc les vecteurs propres de la matrice de covariance associés aux valeurs propres les plus élevées.

Une des propriétés de la matrice de covariance est que sa trace (somme sur sa diagonale) est égale à la somme de ses valeurs propres.

$$T = s_1^2 + \dots + s_r^2 = \sigma_1^2 + \dots + \sigma_r^2$$

Le premier vecteur propre u_1 de S montre la direction principale des données de A_0 . Cette direction vers laquelle pointe u_1 explique une fraction $\frac{\sigma_1^2}{T}$ de la variance totale des données. Ainsi en suivant l'ordre des valeurs propres de la matrice de covariance, les vecteurs propres u_i vont expliquer une fraction de plus en plus petite de la variance totale.

Le but est de sélectionner les d premières composantes principales pour former l'espace de sémantique. Dans notre exemple, l'espace de départ possède 3 dimensions, on va donc garder deux composantes principales.

Les 3 vecteurs propres de la matrice de covariance sont :

$$u_1 = \begin{bmatrix} 0.831 \\ -0.554 \\ -0.061 \end{bmatrix} \quad u_2 = \begin{bmatrix} -0.554 \\ -0.832 \\ 0.01 \end{bmatrix} \quad u_3 = \begin{bmatrix} -0.056 \\ 0.026 \\ -0.998 \end{bmatrix}$$

Et les valeurs propres associées sont :

$$\sigma_1^2 = 78.453 \quad \sigma_2^2 = 67.209 \quad \sigma_3^2 = 1.073$$

L'espace de sémantique, sera l'espace couvert par les 2 vecteurs orthonormés. u_1 et u_2 .

Le graphe de la figure 412 met en évidence qu'au plus on ajoute de composantes principales, au plus on explique la variance totale des données.

Une fois, les axes principaux sélectionnés et l'espace de sémantique identifié, il faut projeter le nuage de points sur ce dernier. Soient les colonnes de B les vecteurs formant la base de l'espace de sémantique sur lequel on projette :

$$B = \begin{bmatrix} u_1 & u_2 \end{bmatrix}$$

Pour projeter le nuage de points (défini par A_0), il faut appliquer la matrice de projection :

$$A_{\text{sémantique}} = (B(B^T B)^{-1} B^T A_0^T)^T$$

La matrice $A_{\text{sémantique}}$ contient les coordonnées du nuage de points projeté dans l'espace de sémantique. Ces points peuvent maintenant s'exprimer selon deux coordonnées, une

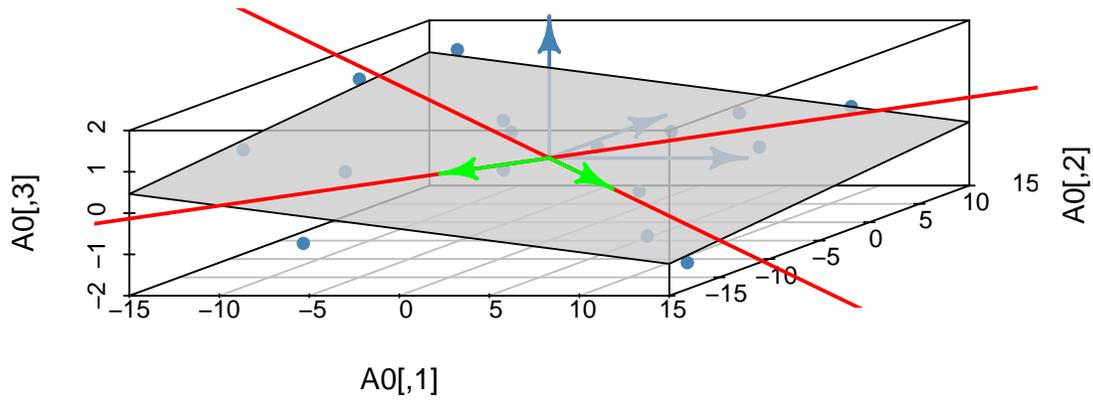


FIGURE 411 – Espace de sémantique de la matrice A.

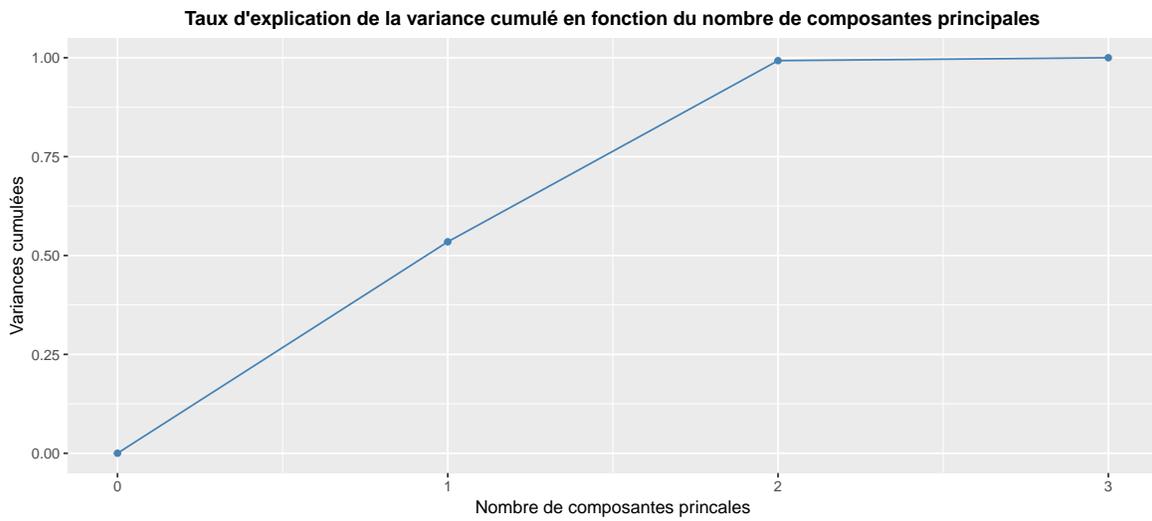


FIGURE 412 – Explication de la variance.

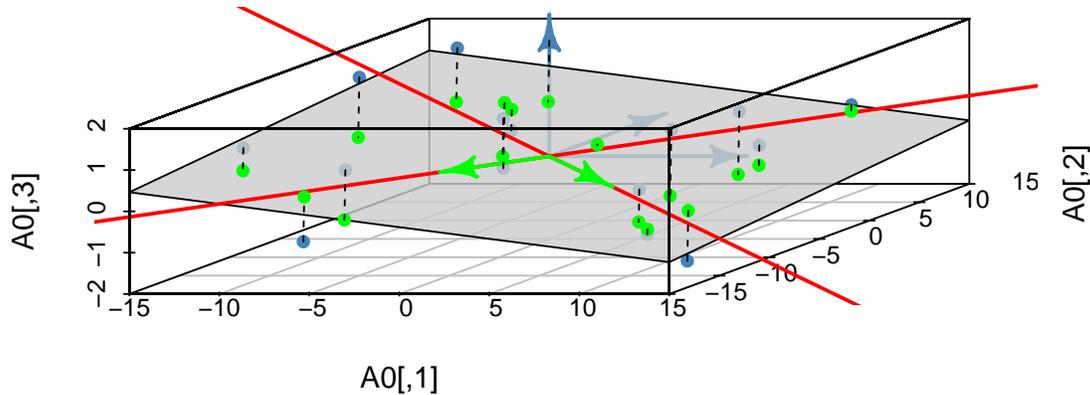


FIGURE 413 – Projection sur l’espace de sémantique.

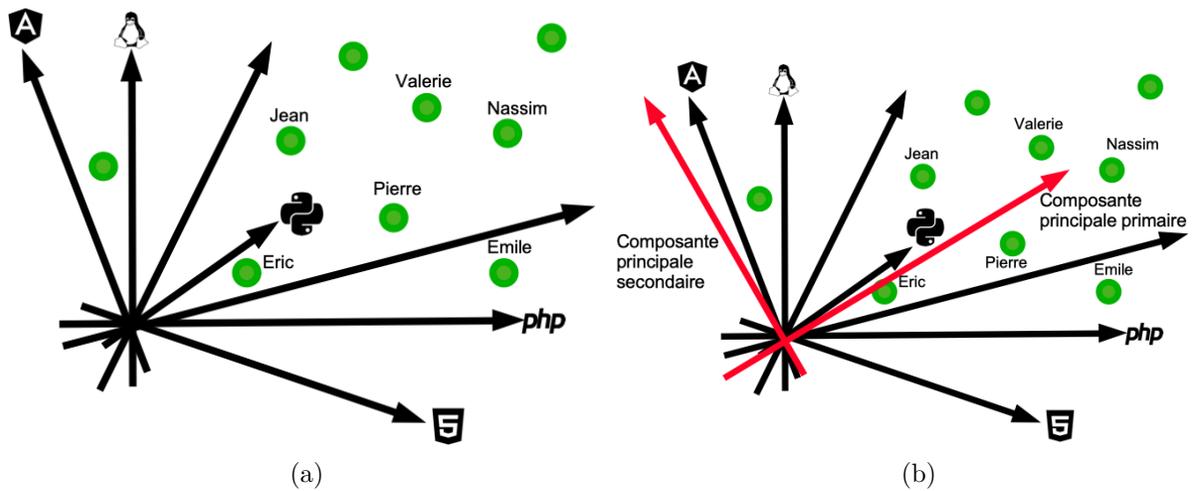
pour chaque axe principal.

En plus des points, on peut aussi projeter dans l’espace de sémantique, les vecteurs unitaires qui forment la base de l’espace de départ. Cela permet donc de placer un nouveau point dans l’espace de sémantique en ne possédant que 2 coordonnées et d’effectuer une prédiction pour la 3^{ème}. Une prédiction s’effectue en réalisant le produit scalaire du nouveau point dans l’espace de sémantique avec la projection du vecteur unitaire de l’axe lié à la variable sur laquelle on veut faire une prédiction.

4.4.2 Utilisation

Cette section décrit l’application de la *PCA* au cas d’utilisation de la prédiction de score d’utilisateur dans une compétence.

Les données utilisées par l’algorithme suivent le format défini dans le chapitre 3. Ce format est un tableau, il peut être vu comme une matrice A avec m lignes représentant chacune un utilisateur et n colonnes représentant chacune un score dans un challenge. Les n variables “score dans une compétence” sont chacune liée à un vecteur unitaire et ensembles ces vecteurs unitaires forment une base pour l’espace \mathbb{R}^n . Chaque utilisateur

FIGURE 414 – Application de la *PCA*

peut être exprimé dans \mathbb{R}^n à l'aide de cette base.

$$user_i = a_{i1} \text{Python} + a_{i2} \text{php} + a_{i3} \text{S} + a_{i4} \text{A} + a_{i5} \text{P}$$

On applique la *PCA* sur le nuage de points pour calculer les composantes principales (figure 414). Il faut souligner que ces axes principaux ne sont pas dénués de sens, ils peuvent correspondre à des concepts intuitifs. Par exemple, comme illustré à la figure 415, un axe pointant dans la direction $score \text{ en } html = score \text{ en } php$ peut être interprété comme l'axe des $score \text{ en } web$ (html et php sont deux langages qui servent à la programmation web).

Ensuite, on sélectionne les composantes principales pour former l'espace de sémantique. Les points utilisateurs et les vecteurs unitaires des axes de score dans un challenge sont projetés dans ce sous-espace qu'on nomme finalement l'espace de sémantique scores-utilisateurs. Cet espace est de dimension inférieure à l'espace de départ (on a sélectionné les composantes principale.).

Un utilisateur qui ne pouvait pas être positionné dans l'espace de départ parce qu'il lui manquait un score, peut maintenant l'être dans l'espace de sémantique et son score manquant peut être déterminé en réalisant le produit scalaire du vecteur qui le représente dans l'espace de sémantique avec la version projetée du vecteur unitaire lié au challenge du score manquant.

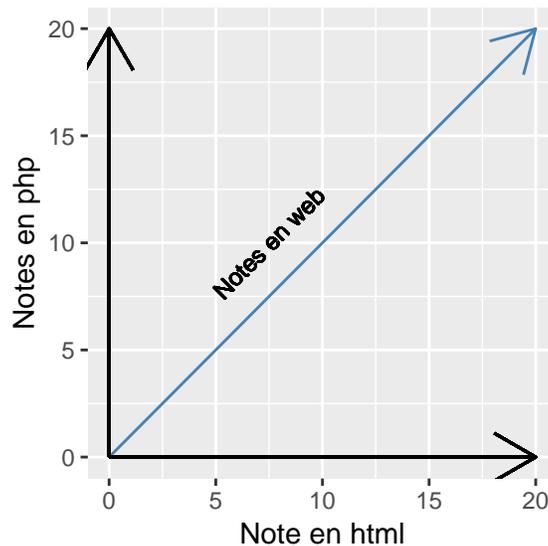


FIGURE 415 – Sens intuitif des composantes principales.

4.4.3 Application

La PCA est une technique qui fonctionne bien quand le dataset possède un nombre important de variables. On va donc la tester sur des datasets de 3 (*GEN3*) et 4 (*NGEN3*) variables.

L'implémentation de la PCA pour notre cas d'utilisation est réalisée à l'aide de R qui possède plusieurs librairies et fonctions qui réalisent la PCA sur un jeu de données. La fonction utilisée dans ce travail est la fonction *PCA()* de la librairie *FactoMineR*.

```
res_1_3.pca <- PCA(ds_1_3$train, scale=TRUE)
res_2_3.pca <- PCA(ds_2_3$train, scale=TRUE)
```

4.4.3.1 Résultats et interprétations pour le datasets *NGEN3*

Le tableau 44 présente les différents coefficients de corrélation qui existent entre les variables. Plus le coefficient est élevé, plus la relation linéaire entre les 2 variables concernées est forte.

Tous les coefficients sont positifs, cela implique que toutes les variables varient en moyenne dans le même sens, ce qui signifie qu'un score plus élevé dans un challenge implique des scores plus élevés dans les autres challenges.

Corrélation entre les variables				
-	Python	POO	EEE	TDD
Python	1	0.228	0.488	0.349
POO	0.228	1	0.126	0.596
EEE	0.488	0.126	1	0.448
TDD	0.349	0.596	0.448	1

TABLE 44 – Matrice de corrélation, dataset NGEN3

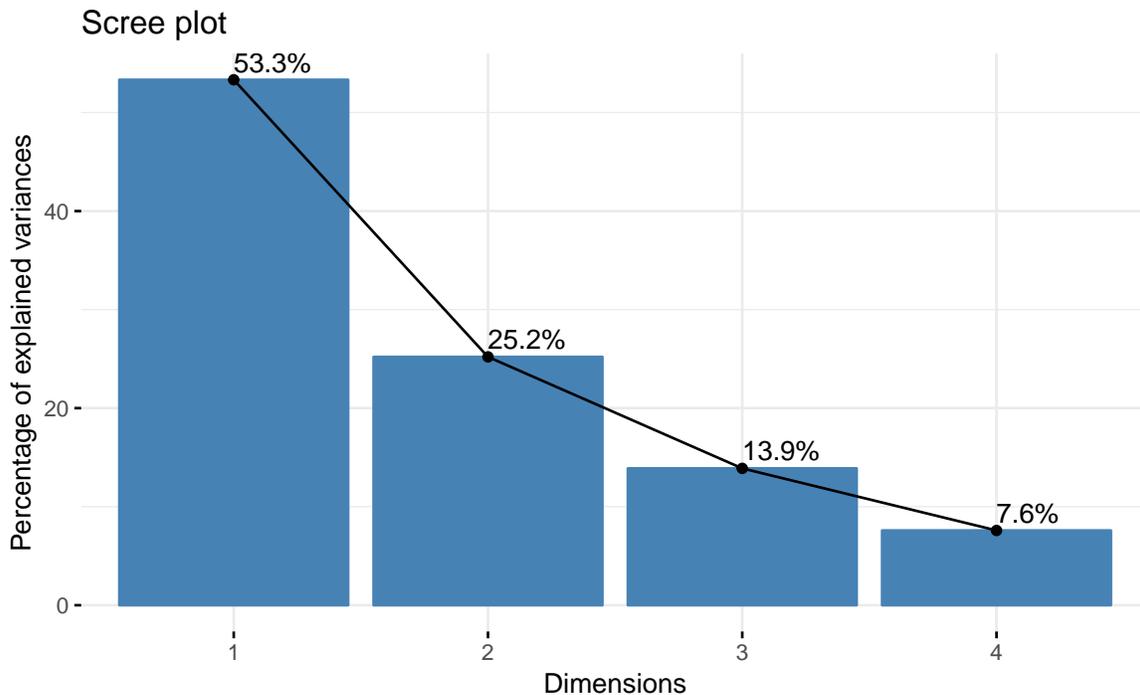


FIGURE 416 – Explication de la variance, dataset NGEN3.

La figure 416, nous montre le niveau d’explication de la variance totale du dataset qu’apporte chacun des axes principaux calculés par la PCA.

On observe que les 3 premiers axes principaux expliquent plus de 90 % de la variance des données. La perte d’information due au passage de l’espace de départ à 4 dimensions vers un espace de sémantique à 3 dimensions est minime (8 %), ce qui rend les prédictions dans ce nouvel espace plus précises. Si on projette le nuage de points des utilisateurs sur un espace de sémantique à 2 dimensions, la perte d’information est plus grande (20 %).

Le tableau 45 et la figure 417 font apparaître les résultats de la PCA sur les variables (les scores dans les différents challenges). Le tableau nous montre les coefficients de

Corrélations avec les composantes principales					
	CP 1		CP 2		CP 3
TDD	0.842	EEE	0.528	Python	0.552
EEE	0.707	Python	0.45	POO	0.165
Python	0.697	TDD	-0.293	TDD	-0.249
POO	0.662	POO	-0.664	EEE	-0.402

TABLE 45 – Corrélations des variables avec les composantes principales, dataset NGEN3

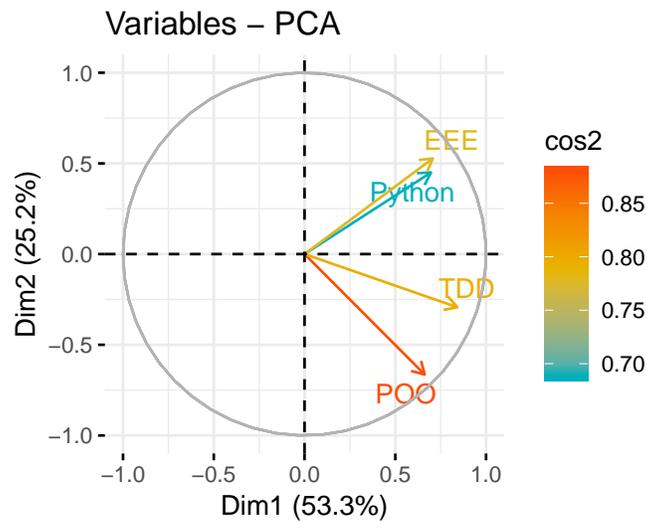


FIGURE 417 – Représentations des variables, dataset NGEN3.

corrélation entre les variables et les axes principaux calculés. Si un coefficient est élevé cela signifie que la variable et l'axe principal en question possèdent une variation similaire. La figure 417 montre la projection des vecteurs unitaires liés aux variables dans l'espace de sémantique. La légende sur le côté nous indique le coefficient \cos^2 des projections. Pour faire simple, au plus il est élevé au plus la version projetée du vecteur unitaire est proche de sa version originale.

On observe directement grâce à ce tableau et à ce graphe que le premier axe principal est positivement et significativement corrélé avec les 4 variables observées : au plus un utilisateur reçoit un score élevé pour une des variables, au plus il se trouve loin sur le premier axe principal. Derrière cet axe, se cache donc une idée d'une note globale propre aux 4 variables (au 4 scores).

Le second axe principal se comporte différemment, il est corrélé positivement avec les variables *score en python* et *score en EEE* et négativement avec les variables *score en*

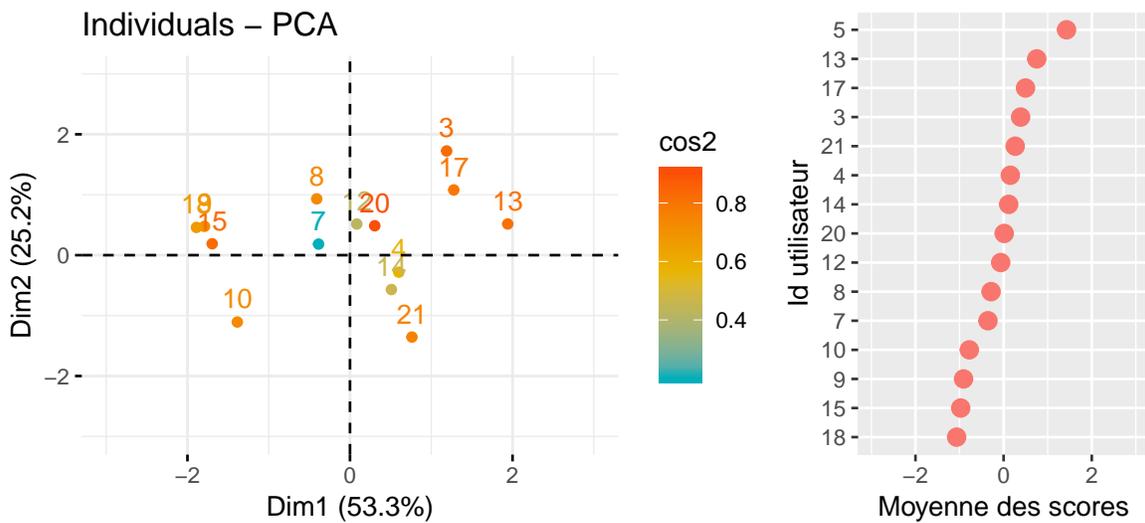


FIGURE 418 – Représentations des individus, dataset NGEN3.

TDD et *score en POO* : au plus un utilisateur possède une valeur positive élevée sur cet axe, au plus ses scores en *Python* et *EEE* sont élevés et ceux en *TDD* et *POO* sont faibles. Les variables sont donc groupées par 2 :

- Les challenges en *Python* et *EEE* demandent tous les deux des connaissances techniques d’implémentation de solution et de programmation pure.
- Les challenges en *TDD* et *POO* demandent des notions d’architecture et de qualité logicielle.

Le second axe permet donc de distinguer les utilisateurs avec un profil d’“architecte”, de ceux avec un profil d’“implémenteur”.

La figure 421 montre les résultats de la PCA sur les utilisateurs, c’est-à-dire la projection du nuage de points dans l’espace de sémantique. L’idée du premier axe principal qui représenterait un score global est confirmée par les graphiques : sur l’illustration de droite, se trouve un classement des utilisateurs (identifié par un nombre) selon leur score moyen pour les 4 variables, sur celle de gauche, on peut voir les mêmes utilisateurs dans l’espace de sémantique, si on ne tient compte que de leur coordonnée sur le premier axe principal, ils sont ordonnés de la même façon que dans le classement.

4.4.3.2 Résultats et interprétations pour le datasets GEN3

Les résultats pour ce dataset sont assez similaires, mais certaines interprétations sont

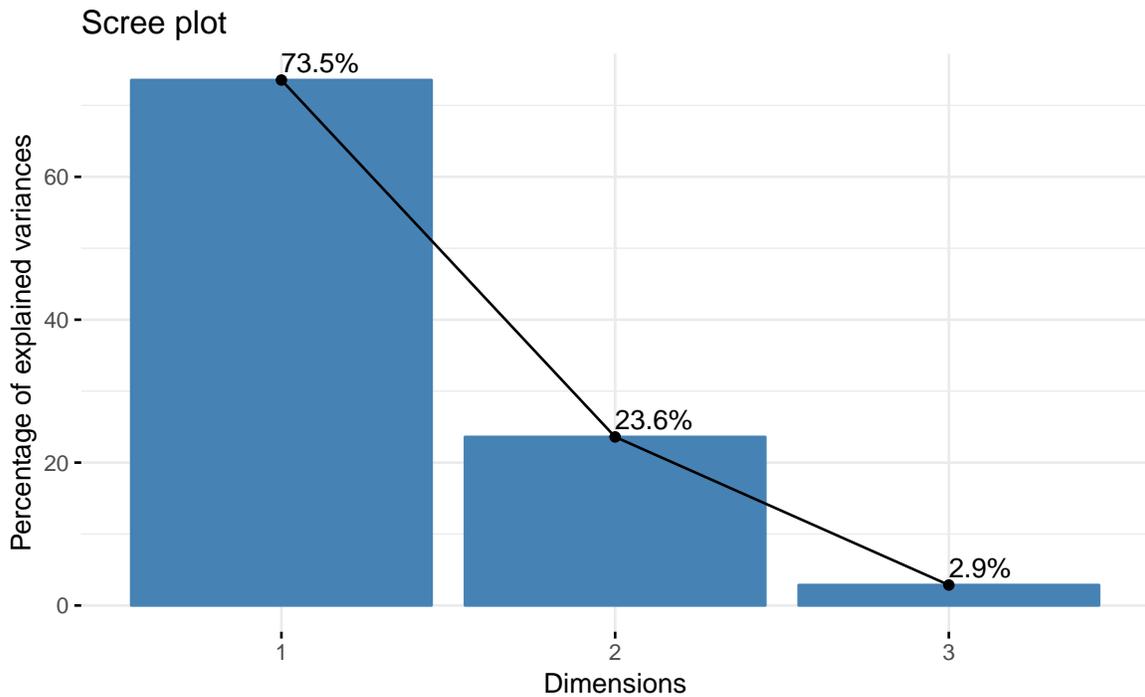


FIGURE 419 – Explication de la variance, dataset GEN3

néanmoins différentes.

Dans la figure 46, tous les coefficients de corrélation sont positifs. Les variables vont donc évoluer dans le même sens.

Corrélation entre les variables			
-	IT	C	Java
IT	1	0.866	0.322
C	0.866	1	0.579
Java	0.322	0.579	1

TABLE 46 – Matrice de corrélation, dataset GEN3

La figure 419 montre que les deux premiers axes principaux expliquent plus de 95 % de la variance du dataset *GEN3*. On peut donc projeter le nuage de points dans un espace de sémantique à deux dimensions sans perdre trop d'information.

Le tableau 47 et la figure 420 montrent comme pour le dataset *NGEN3* que l'axe principal 1 est positivement et significativement corrélé avec toutes les variables. Cet axe représente donc un score global sur les 3 variables. On observe aussi que l'axe de la

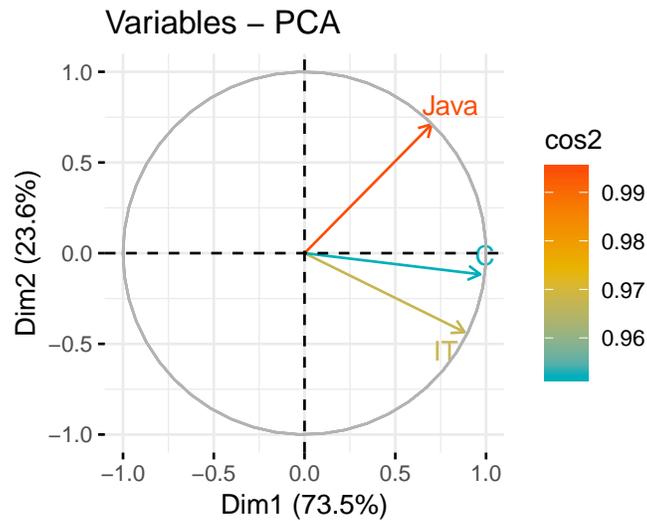


FIGURE 420 – Corrélation des variables, dataset GEN3

variable *score en C* est fortement corrélé à l’axe principal 1, donc pour un utilisateur le score obtenu en *C* est très représentatif du score global.

Le second axe principal est positivement corrélé avec la variable *score Java*, négativement avec la variables *score IT* et faiblement avec la variable *score C*, il est difficile de donner une valeur sémantique à cet axe.

Corrélations avec les composantes principales					
	CP 1		CP 2		CP 3
C	0.969	Java	0.71	IT	0.181
IT	0.882	C	-0.117	Java	0.075
Java	0.7	IT	-0.435	C	-0.218

TABLE 47 – Corrélations avec les composantes principales, dataset GEN3

La figure 421 représente la projection du dataset GEN3 sur l’axe de sémantique. On observe, comme pour le dataset *NGEN3*, que le long du premier axe principal, les utilisateurs sont organisés dans le sens ascendant de leur moyenne sur l’ensemble des variables : dans l’espace de sémantique, l’utilisateur avec le plus faible score moyen se trouve le plus à gauche et celui avec le meilleur score se trouve le plus à droite.

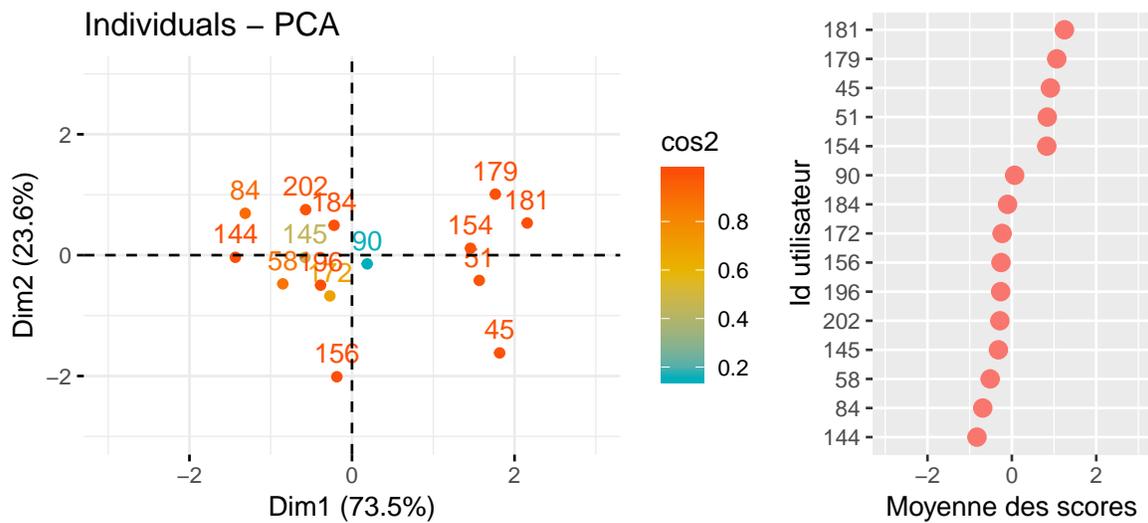


FIGURE 421 – Représentations des individus, dataset GEN3.

4.4.4 Prédictions

Une prédiction se réalise sur un utilisateur qui ne possède pas de score pour chacune des variables du dataset sur lequel on a réalisé la PCA. Par exemple, après avoir appliqué la PCA au dataset *NGEN3*, on peut faire une prédiction de score en *Python* pour un utilisateur qui possède des scores en *EEE*, en *POO* et en *TDD*. La prédiction se réalise alors dans un espace de sémantique à 3 dimensions.

On peut aussi imaginer que cet utilisateur ne possède que des scores en *POO* et en *Python*, on peut alors faire des prédictions sur les 2 autres variables, mais avec une moins bonne précision car la prédiction se fait dans l'espace de sémantique à 2 dimensions.

Afin de réaliser une prédiction, on place le point d'un utilisateur dans l'espace de sémantique grâce aux scores qu'il possède pour certains challenges. Grâce à des produits vectoriels, on peut prédire les scores pour les challenges qu'il n'a pas réalisés.

4.4.4.1 Exemple de prédiction

On réalise la *PCA* sur le dataset *NGEN3* (challenges en *POO*, *EEE*, *TDD*, *Python*) et on souhaite réaliser une prédiction pour un utilisateur qui possède les scores normalisés de 1,1 en *Python*, -1,3 en *POO* et 1,2 en *EEE*. Les coordonnées des vecteurs unitaires liés à ces challenges projetés dans l'espace de sémantique à 3 dimensions sont:

$$\vec{I}_{Python} = \begin{bmatrix} 0.641 \\ 0.492 \\ 0.588 \end{bmatrix} \quad \vec{I}_{POO} = \begin{bmatrix} 0.746 \\ -0.550 \\ 0.038 \end{bmatrix} \quad \vec{I}_{EEE} = \begin{bmatrix} 0.643 \\ 0.522 \\ -0.548 \end{bmatrix}$$

Les coordonnées de l'utilisateur dans l'espace de sémantique se calculent donc par:

$$1.1 \begin{bmatrix} 0.641 \\ 0.492 \\ 0.588 \end{bmatrix} - 1.3 \begin{bmatrix} 0.746 \\ -0.550 \\ 0.038 \end{bmatrix} + 1.2 \begin{bmatrix} 0.643 \\ 0.522 \\ -0.548 \end{bmatrix} = \begin{bmatrix} 0.507 \\ 1.882 \\ -0.060 \end{bmatrix}$$

Si on suit les interprétations faites pour les axes principaux calculés à partir du dataset *NGEN3*, cet utilisateur possède un niveau général moyen et un profil de programmeur-implémenteur.

Pour prédire son score normalisé dans le challenge en *TDD*, on réalise le produit scalaire avec le vecteur unitaire lié à ce challenge projeté dans l'espace de sémantique:

$$\begin{bmatrix} 0.507 \\ 1.882 \\ -0.060 \end{bmatrix} \cdot [0.845 \quad -0.283 \quad -0.062] = -0.1$$

4.4.4.2 Performance des prédictions

La tableau 48 reprend les résultats pour les différents datasets de plus de 2 variables. Comme pour les régressions, on calcule la *MSE* de *cross-validation*.

Performances			
-	GEN3	NGEN2	NGEN3
MSE	0.5515898	0.9428845	0.5035364

TABLE 48 – Performances des prédictions avec la PCA

4.5 Réseau bayésien

4.5.1 Présentation

4.5.1.1 Motivations

Pour avoir une représentation rigoureuse des relations probabilistes existant entre des variables aléatoires, il faudrait construire un tableau de probabilités conjointes reprenant les configurations pour chaque valeur de chaque variable. À partir de cette table, on pourrait calculer n'importe quelle probabilité conditionnelle ou marginale. Par exemple, quelle est la distribution de probabilité du score d'un utilisateur en Java, sachant qu'il a eu un 18 en python et un 12 en C.

Cette approche n'est pas praticable dans le cas où le nombre de variables aléatoires est élevé et dans une moindre mesure, dans le cas où le nombre de valeurs que peut prendre une variable aléatoire est élevé. Le nombre d'entrée de la table est :

$$n_{entry} = n_{values}^{n_{variables}}$$

Les tables trop grandes ne peuvent pas être stockées en mémoire. Par exemple, un tableau résume les probabilités conjointes de 8 variables pouvant prendre 20 valeurs. Chaque nombre est stocké sur 32 bits. La taille d'un tel tableau est :

$$n_{entry} = 20^8 = 2.56 \times 10^{10}$$

$$s_{table} = 32 * 2.56 \times 10^{10} = 8.192 \times 10^{11} bits = 8.192 \times 10^5 Mb = 819.2Gb$$

Les réseaux bayésiens vont construire des modèles probabilistes plus efficaces dans le calcul et dans le stockage.

4.5.1.2 Définition

Les réseaux bayésiens se basent sur deux domaines théoriques très importants : la théorie des graphes et la théorie des probabilités.

Un réseau bayésien est un graphe orienté (les arêtes sont dirigées) acyclique et formé par des noeuds chacun associé à une variable aléatoire du problème. Une arête représente une

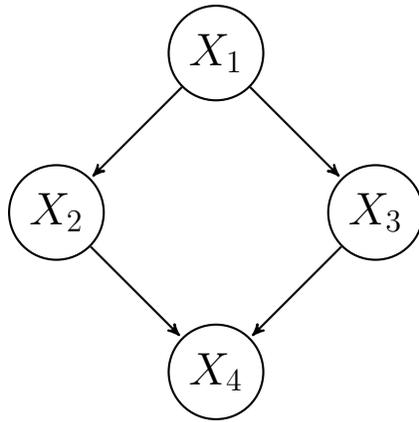


FIGURE 422 – Exemple de réseau bayésien

relation de dépendance probabiliste entre les nœuds, qui sous-entend une distribution de probabilité conditionnelle.

Si une arête va du nœud X_1 vers le nœud X_2 , X_1 influence X_2 et X_1 est appelé parent de X_2 . La notation $Parents(X_i)$ désigne l'ensemble des nœuds parents au nœud associé à la variable aléatoire X_i .

Chaque nœud est associé à une table qui reprend les différentes relations conditionnelles avec ses parents.

4.5.1.3 Calcul de probabilité plus facile

Les calculs de probabilité dans un réseau bayésien sont facilités par une règle fondamentale : chaque nœud du réseau est conditionnellement indépendant de ses non-descendants, étant donné ses parents.

$$\mathbb{P}(X_1, \dots, X_n) = \prod_{i=1}^n \mathbb{P}(X_i | Parents(X_i))$$

4.5.1.3.1 Probabilités conjointes

Un réseau bayésien simplifie la règle de chaînage classiquement utilisée pour calculer les probabilités conjointes en appliquant sa règle d'indépendance conditionnelle.

Par exemple, la probabilité conjointe du réseau illustré à la figure 422 est :

$$\begin{aligned}\mathbb{P}(X_1, X_2, X_3, X_4) &= \mathbb{P}(X_1)\mathbb{P}(X_2|X_1)\mathbb{P}(X_3|X_2, X_1)\mathbb{P}(X_4|X_3, X_2, X_1) \\ &= \mathbb{P}(X_1)\mathbb{P}(X_1|X_2)\mathbb{P}(X_3|X_1)\mathbb{P}(X_4|X_3, X_2)\end{aligned}$$

4.5.1.3.2 Probabilités marginales

Pour calculer les probabilités marginales dans un réseau bayésien, on peut ignorer les variables aléatoires associées aux nœuds dont les descendants ne sont pas les nœuds des variables aléatoires observées.

4.5.1.4 Inférences

La plupart des prédictions sont des probabilités à posteriori étant donné un événement observé. Elles sont calculées à partir de probabilités conjointes et marginales :

$$\mathbb{P}(A|B) = \frac{\mathbb{P}(A, B)}{\mathbb{P}(B)}$$

On recherche $\mathbb{P}(X, E = e)$ où X est l'ensemble des variables aléatoires pour lesquelles on souhaite obtenir une distribution, E est l'ensemble des variables aléatoires observées et e sont les valeurs observées. Y est l'ensemble des variables cachées, ni observées ni demandées.

4.5.1.4.1 Inférences par énumération

Cette méthode calcule des inférences exactes, mais demande beaucoup de calculs.

$$\mathbb{P}(X|E) = \frac{\mathbb{P}(X, E = e)}{\mathbb{P}(E = e)} = \frac{\sum_Y \mathbb{P}(X, E = e, Y = y)}{\mathbb{P}(E = e)}$$

On commence par calculer le numérateur et on normalise ensuite. Chaque terme $\mathbb{P}(X, e, y)$ est un produit de probabilités conditionnelles, mais avec les variables de E et Y instanciées. Ce produit compte n facteurs. La taille de l'ensemble X étant $|X|$, celle de l'ensemble Y étant $|Y|$ et le nombre de valeurs que peut prendre les variables aléatoires étant d , il faut calculer le terme $\mathbb{P}(X, e, y)$ $d^{|X|+|Y|}$ fois. La complexité temporelle est donc

$$\mathcal{O}(nd^{|X|+|Y|})$$

4.5.1.4.2 Inférences par échantillonnage (reject sampling)

Cette méthode, basée sur des simulations stochastiques, calcule des inférences approximatives. Elle simule des instanciations complète du réseau (en assignant des valeurs aux variables aléatoires) et estime les probabilités à partir des fréquences des observations.

$$\mathbb{P}(X|E = e) \approx \frac{freq(x, e)}{freq(e)}$$

Cette méthode est plus rapide mais moins exacte.

4.5.1.5 Construction du réseau

La construction d'un réseau bayésien demande deux éléments : la structure du réseau et les tables de probabilités pour chaque noeuds.

4.5.1.5.1 Calculer les tables de probabilités

L'apprentissage automatique des tables de probabilités se fait à partir de données :

$$\mathbb{P}(X_i|Parents(X_i) = p) \approx \frac{freq(x, p)}{\sum_{x'} freq(x', p)}$$

4.5.1.5.2 Construire la structure

La structure du graphe peut être construite manuellement ou à partir d'un algorithme qui va trouver une structure optimale à partir des données.

Les algorithmes les plus classiques de construction de réseau sont :

- *Hill Climbing*: méthode semblable à la descente de gradient qui cherche l'optimum pour la configuration d'un réseau en maximisant un score.
- *IAMB* : méthode qui recherche des dépendances conditionnelles entre les variables et qui ensuite impose des contraintes à la structure
- *Max-Min Hill Climbing* : combinaison des deux premières méthodes

4.5.2 Utilisation

Dans notre cas d'utilisation, les variables aléatoires sont évidemment les différents scores obtenues par un utilisateur dans les différents challenges. Il y a une différence importante avec la théorie exposée ci-dessus : ces variables aléatoires sont continues. Les tables de probabilités conditionnelles du réseau sont donc remplacées par des fonctions de densité et les sommes de marginalisation sont changées par des intégrales.

Le réseau est construit sur la base d'un des datasets et il peut ensuite réaliser des prédictions. Pour réaliser une prédiction, on repère le noeud qui nous intéresse sur le graphe et on fixe les valeurs de ses parents. Par exemple sur le graphe à la figure 423, on peut demander le score le plus probable en *C* d'un utilisateur sachant qu'il a obtenu un score de 35 en *Python*.

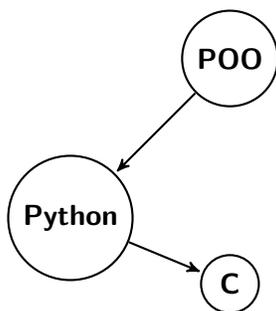


FIGURE 423 – Exemple de réseau bayésien

4.5.3 Application

Comme pour la PCA, on construit des réseaux pour des datasets de plus de 2 variables.

4.5.3.1 Résultats pour le dataset *GEN3*

Pour trouver le meilleur algorithme de construction, on génère des réseaux à partir des 3 algorithmes évoqués ci-dessus. Sur chacun de ces 3 réseaux, on mesure les performances grâce à la *cross-validation*.

La tableau 49 reprend les résultats des différentes cross-validation.

La figure 424 montre les réseaux construits par les différents algorithmes

Performances			
-	C	Java	IT
<i>Hill Climbing</i>	1.56e-01	9.24e-01	9.38e-01
<i>IAMB</i>	6.78e-01	1.03e+00	1.92e-01
<i>Max-Min Hill Climbing</i>	1.56e-01	9.20e-01	9.37e-01

TABLE 49 – Résultats de la cross-validation pour les réseaux bayésiens, dataset GEN3

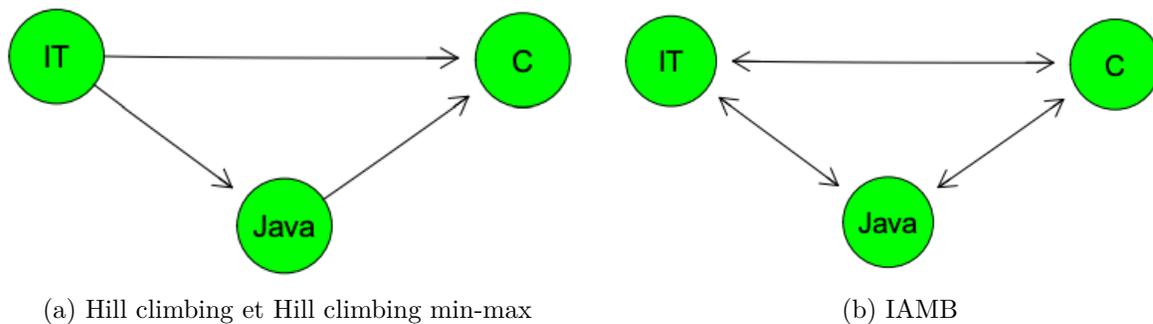


FIGURE 424 – Réseaux bayésien construits sur le dataset GEN3

On observe sur cette figure que les algorithmes *Hill Climbing* et *Hill Climbing Min-Max* génèrent la même structure, c'est donc normal que leurs performances soient similaires. L'algorithme *IAMB* crée des relations bidirectionnelles entre les nœuds : c'est pour cette raison que les prédictions de score en *IT* sont meilleures. En effet dans la première structure, on considère que la variable *score en IT* est conditionnellement indépendante des deux autres. Ces dernières ne sont donc pas utilisées pour réaliser une prédiction sur la variable *score en IT*.

Il n'y a aucun des 3 algorithmes qui se dégage en terme de performances générales et il est difficile d'en sélectionner un.

4.5.3.2 Résultats pour le dataset *NGEN3*

La tableau reprend les résultats des différentes cross-validation.

La figure 425 montre les réseaux construits pour le dataset *GEN3*. Les 3 algorithmes construisent des réseaux différents, mais avec des parties communes (par exemple le nœud de la variable *Python* est toujours nœud parent du nœud *EEE*). Ces parties communes correspondent à des corrélations entre variables déjà observées avec l'analyse

Performances				
-	Python	POO	EEE	TDD
<i>Hill Climbing</i>	9.59e-01	1.10e+00	1.04e+00	7.10e-01
<i>IAMB</i>	1.04e+00	9.03e-01	1.17e+00	9.52e-01
<i>Max-Min Hill Climbing</i>	1.10e+00	9.20e-01	1.12e+00	7.64e-01

TABLE 410 – Résultats de la cross-validation pour les réseaux bayésiens, dataset NGEN3

par la PCA.

Le tableau ne montre pas des performances impressionnantes, en effet la plus petite MSE est de 0.703 pour la prédiction d'un score en *TDD* à partir d'un réseau construit avec la méthode Hill Climbing, c'est le seul cas où un nœud possède 2 nœuds parents.

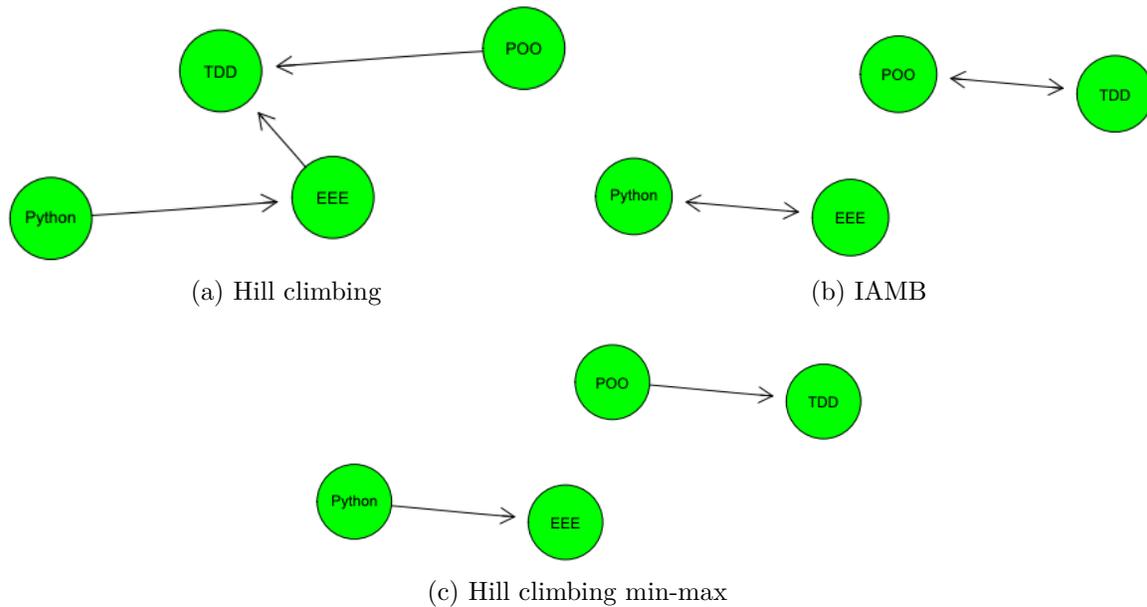


FIGURE 425 – Réseaux bayésien construits sur le dataset NGEN3

4.5.3.3 Conclusion

Aucun résultat ne nous permet pour l'instant de sélectionner une méthode de construction de réseau bayésien pour notre cas d'application, simplement parce qu'aucune ne se démarque d'un point de vue des performances. On a aussi observé que les meilleures prédictions sont faites pour les noeuds qui possèdent plusieurs noeuds parents. C'est une limitation assez importante étant donné que de tels noeuds-variables sont rares

dans les réseaux construits ci-dessus. En fait, pour surmonter ce problème, il faudrait des datasets plus importants avec plus de variables pour pouvoir construire des réseaux plus grands avec plus de relation entre les variables.

5 Développement d'un framework Python

L'idée de base du projet était d'intégrer les algorithmes de prédiction à une interface utilisateur. Malheureusement, en raison du manque de données disponibles, aucun des algorithmes n'a donné des performances qui le rendrait légitime à une utilisation commerciale. Comme la base de données d'EDITx est en constante extension, il viendra un jour où les données seront suffisantes pour construire des modèles robustes à l'aide des techniques présentées dans le chapitre 4. Afin de faciliter leur intégration dans des futures applications d'EDITx, on a décidé de développer un framework qui facilite l'utilisation des techniques explorées dans les chapitres précédents appliquées à la prédiction de niveaux de compétence d'utilisateurs.

Ce framework est un ensemble de classes *Python* qui implémentent les différentes techniques. Le diagramme de classe sur la figure 51 décrit l'architecture du framework.

5.1 Accès à la base de données

Pour garantir un accès flexible aux données, on utilise une interface *DatabaseAccess*, celle-ci définit un "contrat" à respecter pour toutes les classes d'accès aux données. Ces classes doivent contenir les 3 paramètres classiques nécessaires pour se connecter à une base de données (*host*, *username*, *password*). Elles doivent aussi implémenter 3 méthodes :

- *setConnection* : cette méthode permet de se connecter à la base de données à partir des 3 attributs.
- *getArray* : cette méthode permet de récupérer des données dans le format défini au chapitre 3 et illustré à la figure 52. Elle prend en paramètre une liste d'identifiants de challenges qu'on veut retrouver dans le dataset.
- *getMostCommonUsers* : cette méthode récupère une liste contenant les combinaisons de challenges avec le plus d'utilisateurs communs. Elle prend en paramètres le nombre entier *size* qui définit la taille des combinaisons et le nombre entier *n*) qui précise le nombre de combinaisons à renvoyer.

Pour l'instant, une seule classe implémente l'interface : la classe *Neo4jAcces*. L'utilisation d'une interface est très avantageuse dans le cas où EDITx déciderait de ne pas utiliser la base de données *neo4j*, mais par exemple, une base de données *MySQL*. Il suffit alors

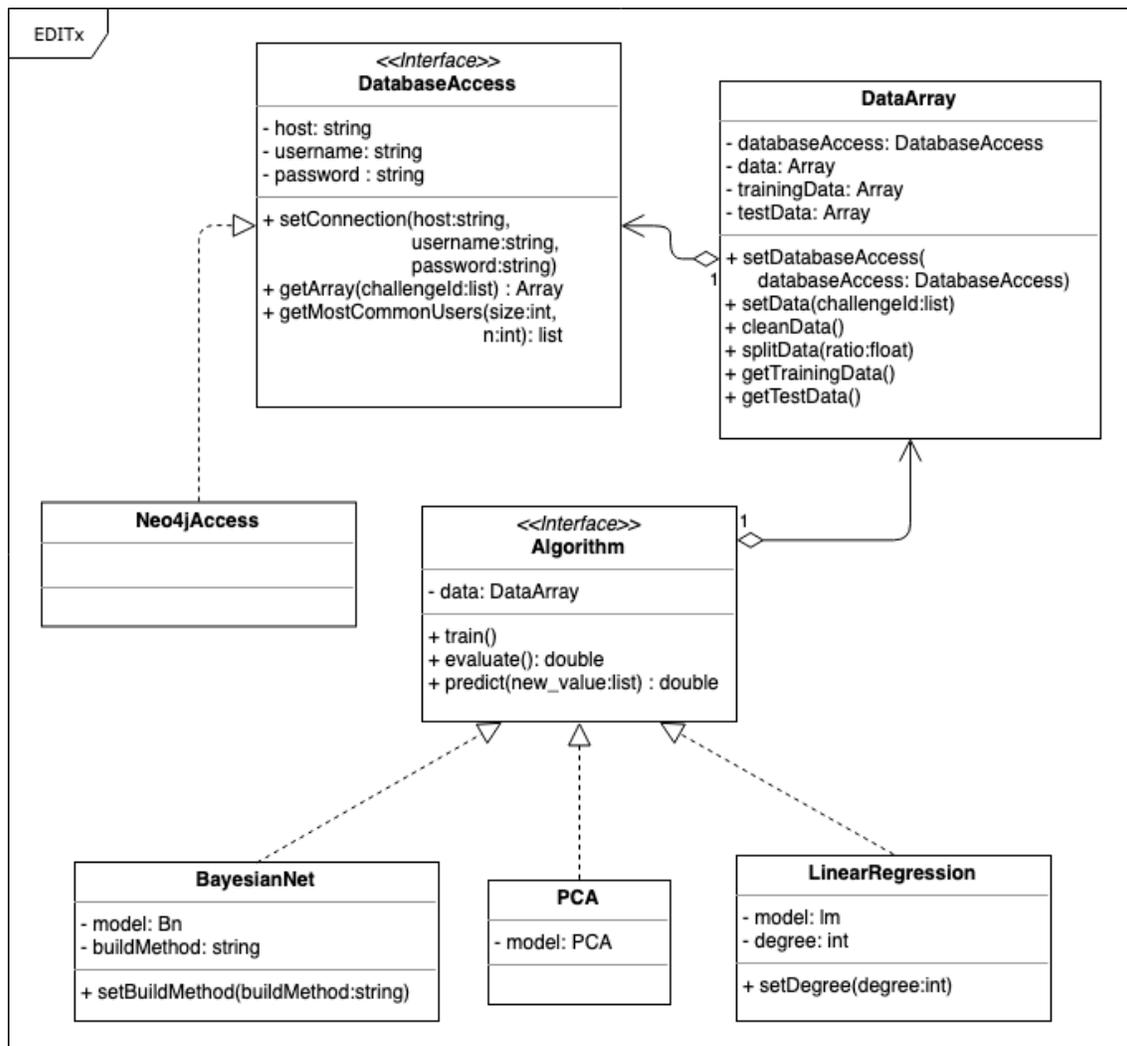


FIGURE 51 – Diagramme de classe du framework

id utilisateur	challenge 56	challenge 72	...	challenge 89
2345	56	72	...	56
2234	45	34	...	32
...
2567	41	15	...	6

FIGURE 52 – Dataset

d'implémenter une nouvelle classe *MySQLAccess* qui respecte l'interface *DatabaseAccess* et aucune autre classe de l'application ne doit être modifiée.

5.2 Travail sur les données

Les techniques de nettoyage et de division du dataset sont implémentées dans la classe *DataArray*. Comme précisé, ce *framework* a vocation à être utilisé quand EDITx possédera assez de données. Dans ce cas, il sera possible de valider les modèles avec un *test set*, c'est pour cette raison qu'une méthode de division des données est implémentée.

La classe *DataArray* possède un attribut *databaseAccess* (instance d'une classe qui implémente l'interface *DatabaseAccess*) qui lui permet de récupérer les données. Les 3 autres attributs permettent de stocker le dataset, le *training set* et le *test set*. Cette classe possède aussi plusieurs méthodes :

- *setDatabaseAccess* : la méthode permet de fixer l'attribut *databaseAccess*.
- *setData* : cette méthode permet de récupérer et stocker les données dans l'objet. Elle prend en paramètres, la liste d'identifiants de challenges qu'elle passe à la méthode *getArray* de l'attribut *databaseAccess*.
- *cleanData* : elle permet de nettoyer les données en utilisant la distance de Mahalanobis.
- *splitData* : cette méthode permet de diviser l'ensemble de données en un *training set* et en un *test set* avec un ratio passé en paramètre.
- *getTrainingData* et *getTestData* : ces 2 méthodes permettent respectivement de récupérer le *training set* et le *test set*.

5.3 Algorithmes

Pour qu'on ait aussi une certaine flexibilité au niveau des algorithmes utilisés, on définit une interface *Algorithm*. Toutes les classes qui veulent être appelées à travers cette interface doivent implémenter les 3 méthodes suivantes :

- *train* : méthode qui entraîne le modèle.
- *evaluate* : méthode qui évalue le modèle avec le *test set*.
- *predict* : méthode qui prédit un score à partir de nouvelle données.

L'avantage d'utiliser une interface est qu'on ne devra pas modifier le code existant du framework, si on désire ajouter un nouvel algorithme. Chaque classe qui implémente l'interface *Algorithm* peut ajouter ses spécificités. Par exemple, la classe *BayesianNet*

ajoute une méthode *setBuildMethod* qui permet de préciser avec quel algorithme sera construit le réseau bayésien.

6 Conclusion

Pour conclure ce travail, on vérifie dans un premier si les objectifs fixés dans le chapitre 1 ont été atteints. Le premier objectif était d'implémenter un accès propre et flexible aux données. Ce premier objectif est atteint, car d'une part, des migrations vers des bases de données plus appropriées à l'exploitation des données ont été réalisées et d'autre part, le framework développé offre un accès simple à ces nouvelles bases de données. Néanmoins, les bases de données conçues ne contiennent que les données nécessaires à la réalisation de ce travail et donc leur exploitation pour d'autres applications demandera une extension qui ne devrait pas être difficile à réaliser avec la flexibilité de *neo4j*. Le second objectif proposait de préparer un format de données standard pour les algorithmes, ce format a été défini et prend la forme d'un tableau contenant les scores de divers utilisateurs dans différents challenges. Il fallait ensuite trouver une méthode pour trouver les meilleurs datasets possibles. La méthode implémentée cherche simplement les groupes de challenges avec le plus d'utilisateurs communs possibles. On a relevé qu'à l'avenir si la base de données devenait trop volumineuse, cette méthode ne serait pas envisageable car elle possède une complexité exponentielle. Une fois les datasets définis, le quatrième objectif nous demandait de trouver un procédé permettant de les nettoyer. La méthode choisie est la distance de Mahalanobis et le framework permet de l'utiliser facilement. L'avant-dernier objectif demandait de trouver des algorithmes permettant de réaliser les prédictions de scores d'utilisateurs. 4 approches ont été implémentées : la régression linéaire multivariée, la régression polynomiale, l'analyse en composantes principales et les réseaux bayésiens. Le dernier objectif consistait à évaluer les modèles pour les comparer, c'est ce qui a été fait grâce à la *cross-validation*. Les objectifs ont tous été remplis, néanmoins certains problèmes font que les résultats de ce travail ne sont pour l'instant pas utilisables pour une application commerciale.

Les évaluations des différents modèles ont tous montré des performances peu satisfaisantes. En effet, aucun modèle ne fait de prédictions assez précises que pour pouvoir être utilisé dans une application. Le manque de données disponibles a été identifié comme étant une source potentielle de ces faibles performances. En effet d'un côté, on ne peut pas utiliser beaucoup de variables dans nos algorithmes (maximum 4), car le nombre d'utilisateurs communs à des groupes de plus de 4 challenges est très faible. D'un autre côté même pour un petit nombre de challenges, le nombre d'utilisateurs communs n'est pas énorme. Ce manque de données rend les modèles entraînés mauvais

en terme de généralisation et trop dépendant du dataset. La génération de données a été une solution envisagée, mais elle ne résout qu'une partie du problème. En effet, elle ne permet que d'ajouter des utilisateurs communs, elle ne permet pas d'augmenter le nombre de variables.

Même si les performances sont faibles pour l'instant, on peut quand même penser que l'utilisation des techniques présentées est justifiée. Premièrement, comme l'a montré l'état de l'art, ces méthodes ont déjà été utilisées pour des cas d'utilisations similaires à celui de ce travail. Ensuite, certains résultats sont quand même encourageant :

- En observant certaines régressions, on observe que les données s'organisent bien autour d'une droite ou d'un plan.
- Les axes principaux calculés par la *PCA* possèdent une valeur sémantique qui fait sens.
- Les réseaux bayésiens détectent des indépendances conditionnelles cohérentes avec les résultats des autres algorithmes.

EDITx organise constamment des challenges et donc leur base de données devient de plus en plus importante. Il faut que les techniques présentées dans ce travail soient facilement intégrable à des applications, une fois que les données seront suffisantes. Dans cette optique, EDITx dispose maintenant du présent manuscrit qui peut servir de bases théoriques et du framework documenté qui permet d'utiliser facilement toutes les techniques présentées.

Plusieurs pistes d'amélioration peuvent être envisagées. D'autres techniques de génération et de nettoyage de données existent, il serait intéressant de les explorer. La méthode de prédiction utilisée avec l'analyse en composante principale n'est pas la méthode classique. Généralement, on combine la *PCA* avec des régressions ou des arbres de régression.

Table des figures

31	Diagramme entités-relations	6
32	Schéma de la base de données graphe	6
33	Plus grands nombres d'utilisateurs communs.	9
34	Densité de probabilité estimée.	11
35	Datasets avant et après la génération.	12
36	Détection des valeurs aberrantes.	14
37	Evolution du dataset .NET-DevOps.	15
41	Cross-validation	18
42	Régression linéaire sur le dataset GEN1 et NGEN1.	20
43	Régression linéaire sur les datasets GEN3 et NGEN2	21
44	Distribution des résidus de la régression sur le dataset GEN1 et NGEN1	22
45	Overfitting.	27
46	Régressions polynomiales sur le dataset GEN1 et NGEN1.	28
47	Régressions polynomiales sur le dataset GEN3 et NGEN2	28
48	Détermination du meilleur k.	29
49	Distribution des résidus de la régression polynomiale sur le dataset GEN1 et NGEN3.	30
410	Nuage de points de la matrice A.	34
411	Espace de sémantique de la matrice A.	36
412	Explication de la variance.	36
413	Projection sur l'espace de sémantique.	37
414	Application de la <i>PCA</i>	38
415	Sens intuitif des composantes principales.	39
416	Explication de la variance, dataset NGEN3.	40
417	Représentations des variables, dataset NGEN3.	41
418	Représentations des individus, dataset NGEN3.	42
419	Explication de la variance, dataset GEN3	43
420	Corrélation des variables, dataset GEN3	44
421	Représentations des individus, dataset GEN3.	45
422	Exemple de réseau bayésien	48
423	Exemple de réseau bayésien	51
424	Réseaux bayésien construits sur le dataset GEN3	52
425	Réseaux bayésien construits sur le dataset NGEN3	53

Table des figures

51	Diagramme de classe du framework	56
52	Dataset	56

Liste des tableaux

31	Top 5 des groupes d'utilisateurs communs	9
32	Datasets utilisés	16
41	Performances des régressions linéaires	25
42	Signification statistique des paramètres du modèle construit sur le dataset GEN3	31
43	Performances des régressions polynomiales	31
44	Matrice de corrélation, dataset NGEN3	40
45	Corrélations des variables avec les composantes principales, dataset NGEN3	41
46	Matrice de corrélation, dataset GEN3	43
47	Corrélations avec les composantes principales, dataset GEN3	44
48	Performances des prédictions avec la PCA	46
49	Résultats de la cross-validation pour les réseaux bayésiens, dataset GEN3	52
410	Résultats de la cross-validation pour les réseaux bayésiens, dataset NGEN3	53

A Appendices

A.1 Exemple de code

A.1.1 Netoyage des données

```
clean_data <- function(data){  
  
  # compute de mahalanobis distance for each sample  
  mean <- colMeans(data)  
  S <- cov(data)  
  d <- mahalanobis(data,mean,S)  
  
  # compute the trigger to detect outlier  
  trigger <- qchisq(0.95,df=ncol(data))  
  
  # remove outlier and return  
  data <- data[which(d < trigger),]  
  return(data)  
}
```

A.1.2 Génération des données

```
generate_data <- function(original_data, n){  
  # compute the mean vector and the covariance matrix  
  sigma <- cov(original_data)  
  mu <- colMeans(original_data)  
  
  # generate new data and return  
  new_data <- mvrnorm(n=300, mu, sigma)  
  
  return(new_data)  
}
```

A.2 Description des compétences liées aux datasets

Python : langage de programmation

Java : langage de programmation

.NET : framework de Microsoft

DevOps : combinaison de développement logiciel et d'administration système

C : langage de programmation

IT : compétences générale en programmation, architecture logicielle, algorithmes

POO : programmation orientée objet

EEE : électronique embarquée

TDD : test-driven development ou développements pilotés par les tests, ensemble de bonnes pratiques à utiliser lors du développement d'un logiciel

Bibliographie

- Aivazyan, S. (2011). Multi-dimensional statistical analysis. URL https://www.encyclopediaofmath.org/index.php/Multi-dimensional_statistical_analysis
- Baccini, A. (2010). Statistique Descriptive Multidimensionnelle. URL <https://www.math.univ-toulouse.fr/~baccini/zpedago/asdm.pdf>
- Bekele, R. & Menzel, W. (2005). A Bayesian Approach to Predict Performance of a Student (BAPPS): A Case with Ethiopian Students. p. 189-194.
- Biernat, E. & Lutz, M. (2015). *Data science : fondamentaux et études de cas*. Eyrolles.
- Distance de Mahalanobis. (2018). URL https://fr.wikipedia.org/wiki/Distance_de_Mahalanobis
- Fire, M., Katz, G., Elovici, Y., Shapira, B. & Rokach, L. (2012). Predicting Student Exams Scores by Analyzing Social Network Data. p. 584-595.
- Frost, J. (2019). How to Interpret P-values and Coefficients in Regression Analysis. URL <https://statisticsbyjim.com/regression/interpret-coefficients-p-values-regression/>
- Horny, M. (2014). *Bayesian Networks*. Boston university.
- Huang, S. & Fang, N. (2010). Regression models of predicting student academic performance in an engineering dynamics course. *ASEE Annual Conference and Exposition, Conference Proceedings*.
- Larochelle, H. & Kabanza, F. (2013). Intelligence Artificielle : Réseaux bayésiens. URL <https://www.youtube.com/watch?v=bXyTTtEeVTs>
- Loi normale multidimensionnelle. (2019). URL https://fr.wikipedia.org/wiki/Loi_normale_multidimensionnelle
- Murphy, K.P. (2001). *An introduction to graphical models*. The university of British Columbia.
- Observations aberrantes. (2019). URL <https://support.minitab.com/fr-fr/minitab/18/help-and-how-to/modeling-statistics/regression/supporting-topics/model-assumptions/unusual-observations/>

- Pearl, J. (1985). *Bayesian networks: a model of self-activated memory for evidential reasoning*. University of California.
- Prabhakaran, S. (2017). Linear Regression. URL <http://r-statistics.co/Linear-Regression.html>
- Rego, F. (2015). Quick Guide: Interpreting Simple Linear Model Output In R. URL <https://feliperego.github.io/blog/2015/10/23/Interpreting-Model-Output-In-R>
- Rosenmai, P. (2013). Using Mahalanobis Distance to Find Outliers. URL <https://eurekastatistics.com/using-mahalanobis-distance-to-find-outliers/>
- Scutari, M. (2017). *Understanding Bayesian Networks with Examples in R*. University of Oxford.
- Strang, G. (2016). *Introduction to Linear Algebra*. MIT; Wellesley-Cambridge Press.
- Torabi, R., Moradi, P. & Khanteymooori, A. (2012). Predict Student Scores Using Bayesian Networks. *Procedia - Social and Behavioral Sciences*, **46**, 4476-4480.
- Walpole, R.E., Myers, R.H., Myers, S.L. & Ye, K.E. (2014). *Probability and Statistics for Engineers and Scientists*. Pearson.
- WikiStat. *Scénario: de la SVD à l'ACP*. Université de Toulouse.

Cahier des charges relatif au travail de fin d'études de
Amaury Lekens, inscrit en 2^{ème} Master, orientation informatique

! Année académique : 2018-2019

! Titre provisoire :

Identification automatique de profil utilisateur sur base de compétences IT à l'aide de technique de data mining

! Objectifs à atteindre :

- Design d'un outil d'aide au recrutement qui détecte des profils utilisateurs sur base de compétences IT

L'outil permettra :

- d'interagir avec le recruteur pour que ce dernier définisse un profil recherché
- d'assigner des niveaux de compétences aux utilisateurs sur base des réponses à des questions de challenges IT
- de faire le rapprochement entre le profil recherché et les compétences attribuées

! Principales étapes :

- accès aux données
- analyse des données
- choisir un modèle
- entraîner le modèle
- documenter

Fait en trois exemplaires à Woluwe-st-Lambert, le 19/11/2018

L'étudiant

Amaury Lekens

Signature :



Le tuteur

Rémy Taymans

GEI

Signature :



Le promoteur

Sébastien Combéfis

EDITx

Signature :

