École Centrale des Arts et Métiers

Insertion professionnel

Rapport de stage

Auteurs :
M. AmauryLekens

Encadrants :
Maitre de stage : Mr. Combéfis
Superviseur : R. Taymans

Table des matières

Chapitre 1

Présentation de l'entreprise

Activités

Le stage 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'nformatique : ceux-ci peuvent participer aux quiz et aux tests pour gagner des prix et même parfois trouver des possibilités d'emploi.
- 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 fait sponsoriser les challenges par d'autres sociétés (ING, infrabel . . .). Ces dernières trouvent un double avantage dans ce sponsoring :

- 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 le plateforme. Elle dure environ 4 semaines.
- Après cette première phase, les top 10 ou 20 des étudiants et des professionnels sont invités dans les bureaux de la société sponsorisante pour partiper à la finale. Les 3 premiers de chaque catégorie remportent un prix.

Répartition des rôles

EDITx est une PME et possède donc un nombre limité d'employés.

- Directeur financier et des ressources humaines : Serge Goffin. Il est en charge de toutes les transactions financières de l'entreprise (salaires, paiements des clients, paiements publicitaires . . .). Il s'occupe aussi du recrutement.
- Directeur commercial : Alexandre Dembour qui est assisté par Julien Carlier. Leur rôle est de trouver des entreprises intéressées par les challenges. Il gère aussi tout le processus d'un challenge et le promeut principalement sur des réseaux sociaux et forums.

— Directeur IT : Nicolas Forêt. Il a construit et gère la plateforme EDITx. Aujourd'hui, il est assisté par Olivier Mourisco qui gère la plateforme au quotidien. C'est principalement dans ce département que je suis inséré pendant le stage.
— Directeur académique : Sébastien Combéfis. Il est en charge des relations avec le monde académique et apporte parfois des conseils techniques.

Chapitre 2

Objectifs du stage

Contexte

Le stage étant lié au TFE, les objectifs de celui-ci sont pour la plupart des objectifs préparatoires et intermédiaires au résultat final du TFE. Comme dit ci-dessus, EDITx organise des challenges en informatique qui sont sponsorisés par d'autres entreprises. Un des objectifs de ce sponsoring est de détecter et de recruter différents profils dans les domaines de l'informatique. C'est sur ce point que se concentre le stage et le TFE, en effet l'objectif final est de développer à l'aide d'algorithme de *machine learning* et de *data mining* un outil de recrutement qui facilite la détection de profils précis pour les entreprises sponsorisantes.

Dans cette optique le stage servira, entre autres, à se former aux techniques de *machine learning* et à préparer le terrains pour le TFE au niveau des données, des technologies ...

Objectifs fixés

Les objectifs concrets fixés avec le maitre de stage sont les suivants :

- Découverte des principaux modèles de *machine learning*
- Apprentissage du langage de programmation R
- Préparer un accès au données plus simple que les données brut de *Drupal* (travail sur la base de données)
- Réalisation d'un dashboard pour faciliter la recherche de profils d'utilisateurs (pour les recruteurs)
- Réalisation d'un système d'assignation de badges de compétence pour les utilisateurs

Chapitre 3

Réalisations

Travail sur les données

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. La base de données de EDITx, à l'heure actuelle contient plus de 1000 tables. Même si beaucoup de celles-ci sont des tables de cache et de backup (et 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 futures accès aux données et pour ne pas chaque fois devoir refaire 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.

Réalisation de la base de données relationnelle

La première étape est de lister les différents besoins de la future base de données. Au niveau des entités, il faut pouvoir enregistrer :

- Les utilisateurs : nom, prénom, âge, email, genre, ...
- Les questions : contenu de la question, le nombre de points associé
- Les réponses : contenu de la réponse
- Les compétences : nom de la compétence

Au niveau des relations qu'on doit pouvoir réaliser (qui dans la pratique sont des containtes), il faut identifier le type de relation :

- Les questions auquelles un utilisateur a répondu : un utilisateur peut répondre à plusieurs questions et plusieurs utilisateurs peuvent répondre à une même question, il s'agit d'une relation *n-n*.
- Les réponses qu'un utilisateur a donné : un utilisateur peut donner plusieurs réponses et une réponse peut être donnée pas plusieurs utilisateurs, c'est une relation *n-n*.
- La question à laquelle appartient une réponse : une question possède plusieurs réponses (c'est une question à choix multiple) et une réponse est liée qu'à une seule question. Néanmoins on peut imaginer qu'une réponse pourrait être réutilisée dans plusieurs choix multiples (ce qui éviterait certaines redondances), il s'agit donc aussi d'une relation *-n-n*
- La compétence liée à une question : une compétence peut être assignée à plusieurs questions et une question peut avoir plusieurs compétences (ce n'est pas le cas actuellement mais c'est une bonne idée d'extension)

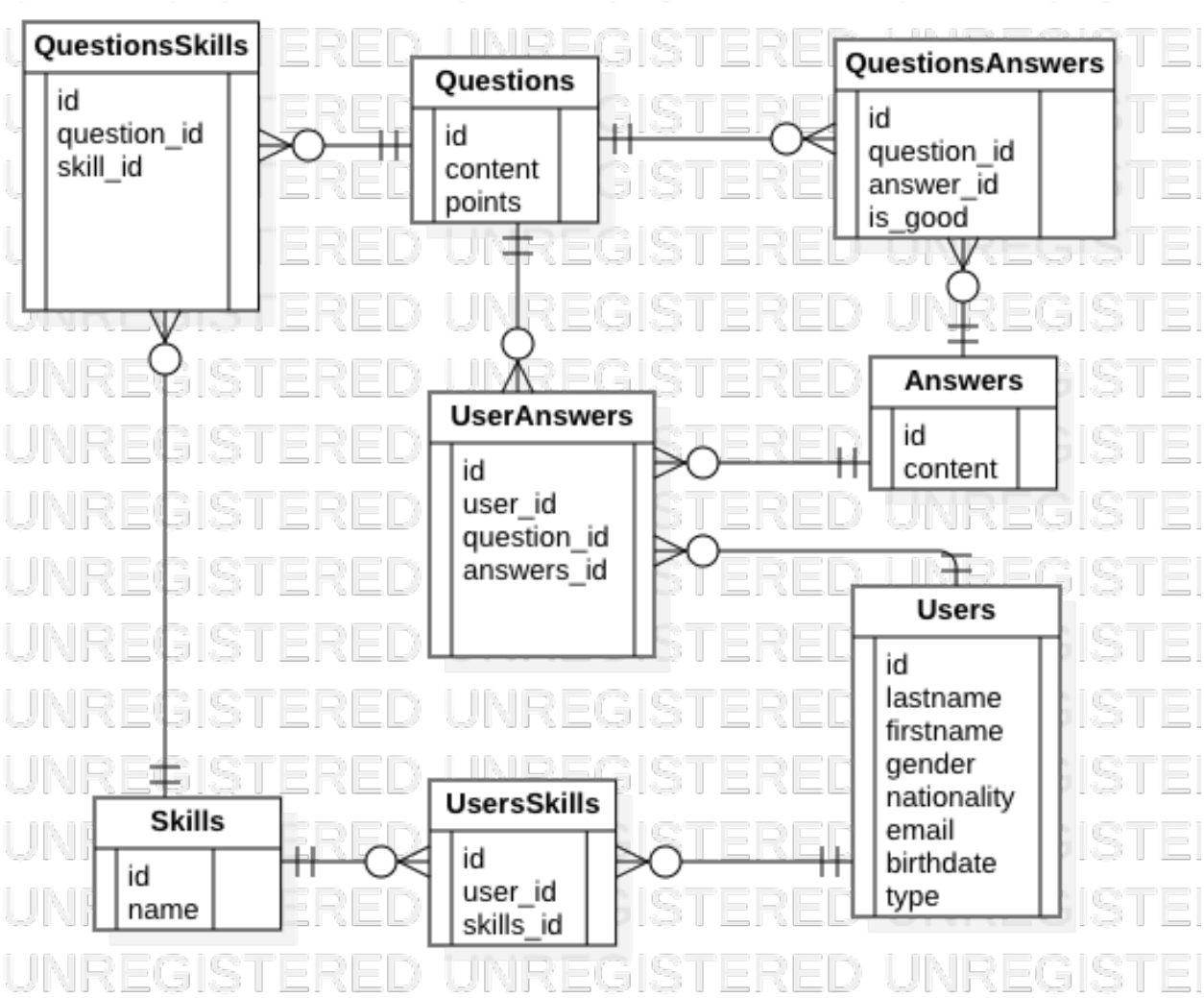


FIGURE 3.1 – Schéma entités-relations

— Les compétences liées à un utilisateur : cette relation est prévue pour le travail d'assignation de badges de compétence.

Les besoins étant énoncés on peut réaliser un diagramme entités-relations (figure 3.1) qui va définir la structure de la base de données.

La dernière étape est la réalisation d'un script de migration qui va créer la base de données décrites sur la figure 3.1, récupérer les données dans la base de données Drupal et insérer ces données correctement dans la nouvelle base de données. Concrètement le script est réalisé en python et utilise le module *mysql_connector*.

Evaluation du modèle relationnel

En observant le schéma entité-relation, on remarque qu'il y a beaucoup de tables de jonction. Cette abondance n'est pas optimale au niveau de la qualité logicielle . Premièrement, les requêtes SQL sont lourdes et difficilement compréhensibles, ce qui impacte sur la maintenabilité des logicielles accédant à la base de données. Par exemple, la requête pour sélectionner le nombre de fois qu'on a répondu à la question avec l'id "27026", requiert pas moins de 6 "INNER JOIN" :

```sql
SELECT COUNT(Answer.content) AS COUNT
FROM Answer
INNER JOIN UserAnswer
 ON Answer.id = UserAnswer.answer_id
 INNER JOIN User
   ON UserAnswer.user_id = user.id
   INNER JOIN QuestionAnswer
     ON Answer.id = QuestionAnswer.answer_id
     INNER JOIN Question
       ON QuestionAnswer.question_id = Question.id
       INNER JOIN QuestionSkill
         ON Question.id = QuestionSkill.question_id
         INNER JOIN Skill
           ON QuestionSkill.skill_id = Skill.id
WHERE UserAnswer.question_id = '27026';
```

Table 3.1: 1 records

COUNT
207

Ensuite, les performances des requêtes peuvent être affectées par le grand nombre de tables de jonction. Cela signifie qu'au fur et à mesure que la base de données grandit, la vitesse de réalisation des requêtes diminue.

Pour ces raisons, il est intéressant d'envisager un autre modèle de base données.

Passage vers une base de données orientée graphe

La nouvelle base de données envisagée est une base de données NoSQL. Le NoSQL regroupe un ensemble de modèles de base de données qui s'éloignent du paradigme relationnel. Au sein du NoSQL, il y a plusieurs familles de SGBD : orienté clé-valeur, colonnes, documents et graphe pour citer les principaux. Le modèle qui semble être la plus appropriée pour notre projet est le modèle orienté graphe. Dans ce genre de base de données, le stockage se fait sous la forme d'un graphe contenant des noeuds (entités) et des arrêtes (relations) qui peuvent tous les deux posséder un ou plusieurs labels. L'utilisation du modèle orienté graphe apporte divers avantages au projet.

Premièrement, les requêtes sont simplifiées car les jointures ne sont plus nécessaires. En effet la plupart des moteurs orientés graphe possède un langage de requête, le CQL, qui permet de récupérer des données en décrivant un pattern de combinaison entités-relations.

Deuxièmement, on peut créer différents types de relation (ce qui n'est pas possible dans le modèle relationnel). Dans notre base de données, on a par exemple une relation entre un utilisateur et une question qui sera différente de celle entre une question et une compétence. Dans un gaphe, on peut respectivement les nommer *MAKE_QUESTION* et *HAS_SKILL*.

En plus, les performances seront bien meilleures comparées à celles d'une base de données relationnelle parce qu'on utilise des données qui ont beaucoup de liens entre elles. Contrairement aux bases de données relationnelles où les requêtes contenant beaucoup de jointures se détériorent quand la taille de la base de données augmente, les performances des requêtes dans une base de données orientée graphe tendent à rester constantes vis à vis de la taille du graphe. Cette propriété est dûe au fait que les requêtes sont localisées dans une petite partie du graphe.

Enfin, on gagne en flexibilité par rapport au modèle relationnel. On peut facilement ajouter des noeuds, des arrêtes ou des labels sans perturber le fonctionnement de l'application cliente de la base de données. En effet,

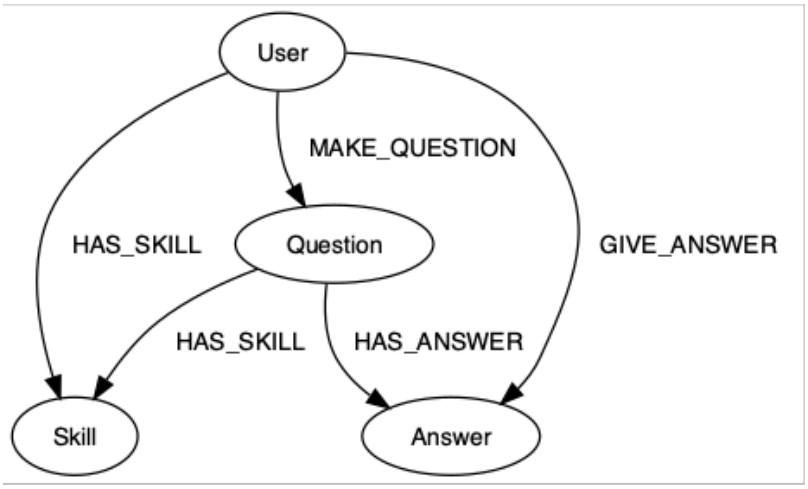


FIGURE 3.2 – Structure du graph

les requêtes que celle-ci effectue sont toujours valides même après l'ajout de noeuds, arrêtes ou labels.

Choix du moteur NoSQL orienté graphe

Maintenant que le modèle est choisi, il faut désigner un de ses moteurs (un moteur est un logiciel qui implémente le modèle de base de données). L'application utilise le moteur Neo4j.

Premièrement Neo4j utilise des transactions ACID (atomicité, cohérence, isolation, durabilité), elles sont donc fiables. Cette fiabilité est importante en raison des nombreuses connexions entre les entités, la correction d'erreur est parfois difficile.

Ensuite ce moteur est le plus populaire des orientés-graphe, ce qui signifie qu'il existe une communauté Neo4j importante. C'est toujours utile pour la prise en main du moteur et lorsqu'on rencontre des difficultés.

Neo4j fournit aussi un client avec interface graphique : le Browser Neo4j. Il facilite grandement le développement car d'une part il permet d'effectuer des requêtes directement sur la base de données. D'autre part, il offre une représentation graphique de l'état actuel de la base de données ou du résultat d'une requête.

En plus Neo4j possède un langage de requête spécifique : Cypher Query Language (CQL). Celui-ci permet de faire très facilement des requêtes, à priori complexes, sur les noeuds et les arrêtes de relations.

Enfin il existe un driver pour utiliser neo4j directement depuis un script r (RNeo4j) ou depuis un script python (neo4j-driver).

Construction de la base de données orientée graphe

Pour construire la base de données orientée graphe à partir de la base de données relationnelle, on transforme simplement le contenu des tables de données en entités (noeuds) et le contenu des tables de jonction en relations (arrêtes) différenciables grâce à un nom (il y a plusieurs types de relation) (figure 3.2).

La construction de cette nouvelle base de données à partir de la base de données relationnelle se fait via un script python qui utilise le module neo4j-driver.

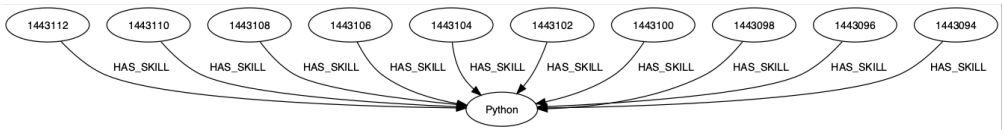


FIGURE 3.3 – Résultat de la requête

Exemples de requête

Pour illustrer la puissance de Neo4j, on va réaliser la requête CQL analogue à la requête SQL effectuée ci-dessus.

```
MATCH (q:Question)<-[]-(u:User)-[g:GIVE_ANSWER]->(a:Answer)<-[]-(q)
WHERE q.id = '27026'
RETURN COUNT(DISTINCT g)
```

```
## +--------------------+
## | COUNT(DISTINCT g) |
## +--------------------+
## | 207                |
## +--------------------+
##
## 1 row available after 17 ms, consumed after another 0 ms
```

On peut relever quelques observations sur la requête et le résultat.

Premièrement le résultat correspond à celui de la requête SQL, c'est un bon départ pour valider le script de migration même si d'autres tests doivent évidemment être effectués (ces tests n'ont pas été réalisés pendant le stage, il ne seront donc plus évoqués dans ce rapport).

Ensuite les requêtes sont formidablement simplifiées par rapport au SQL. En fait dans la base de données SQL, chaque fois qu'on "traverse" une table de jonction, cela requiert 2 "INNER JOIN" et comme dit précédemment les tables de jonction sont remplacées dans le graphe par une relation. Donc dans une requête CQL, le symbole "-[]->" remplace 2 "INNER JOIN", ce qui est le cas dans notre exemple, étant donné qu'il y 6 "INNER JOIN" pour 3 symboles "-[]->"

On peut réaliser encore une autre requête et montrer le résultat sous la forme d'un graphe (figure 3.3) : on veut récupérer 10 questions python.

```
MATCH (q:Question)-[]->(s:Skill)
WHERE s.name = 'Python'
RETURN DISTINCT q.id AS q_id, s.name AS skill
LIMIT 10
```

Commentaires sur les scripts de migration

Afin d'obtenir une qualité logicielle satisfaisante pour les deux scripts de migration, une attention particulière a été portée sur les 2 points suivants :

- Les deux scripts sont écrits sous la forme de classe avec des méthodes qui divisent proprement le travail de migration.
- Les deux scripts respectent entièrement les conventions de codage officielles de python : *pep8*. Il existe une commande qui permet de vérifier le respect de ces conventions directement sur un fichier.

```
pycodestyle file.py
```

Réalisation d'un dashboard

Objectifs du dashboard

L'objectif principal du dashboard est d'offrir aux entreprises clientes d'EDITx un outil de détection de profil ergonomique et interactif. Concrêtement le dashboard est séparé en 2 parties distinctes.

La première partie affiche des classements et des statistiques basés sur les résultats des différents utilisateurs. Cette partie permet donc au recruteur d'avoir des informations générales sur l'ensemble des utilisateurs : moyennes des scores, répartitions des scores ...

La deuxième partie sert d'interface graphique au système d'attribution de badges de compétence (qui est expliqué ci dessous). Concrètement, il permet d'afficher de manière interactive les différents groupes de niveau des utilisateurs.

Shinydashboard

Pour développer la dashboard, on a fait le choix d'utiliser Shiny qui est une librairie r open source et qui permet de réaliser facilement des applications web d'affichage de résultats scientifiques et statistiques. Plusieurs éléments ont motivé ce choix.

Premièrement, Shiny permet d'inclure facilement, la plupart des widgets d'interaction (checkbox, bouton, slider, champ de texte ...). Par exemple pour inclure un *slider* il suffit d'appeler la fonction adéquate de Shiny :

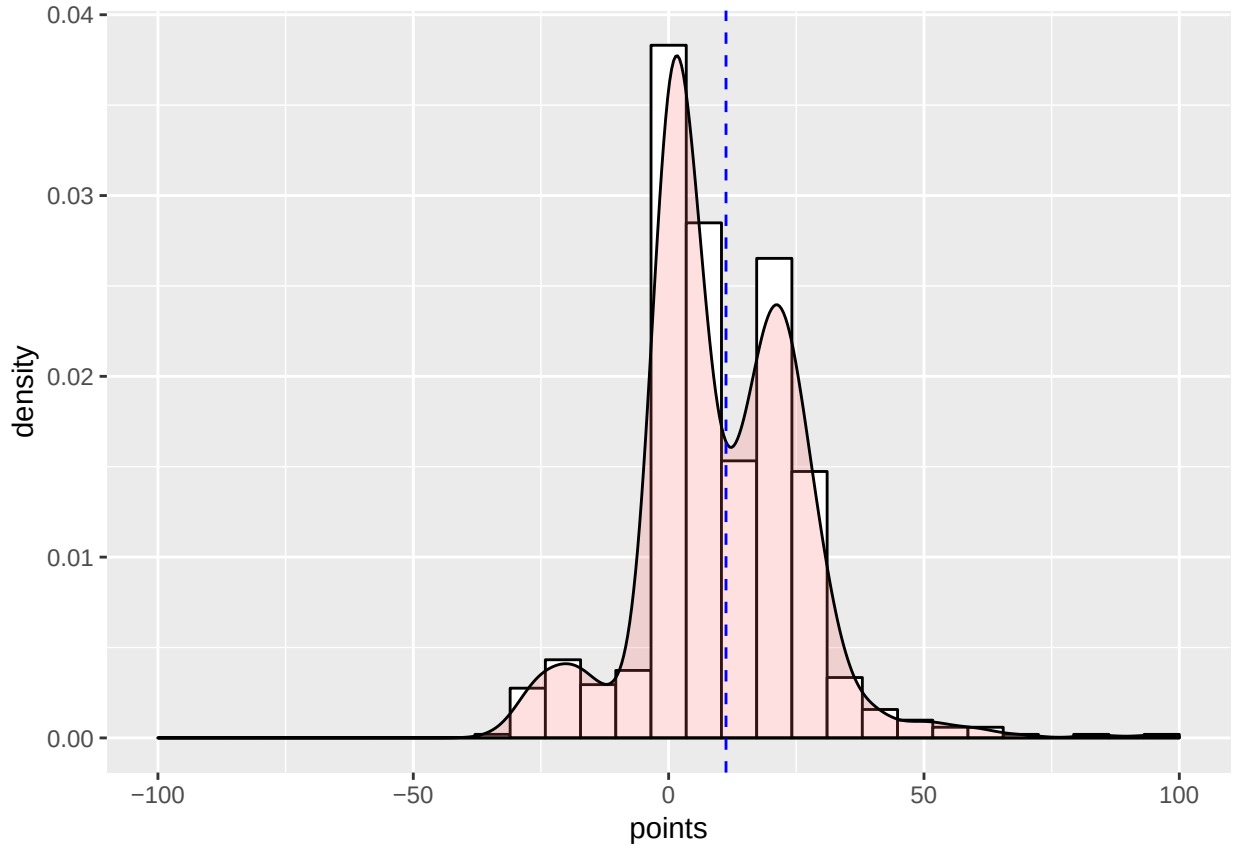
```
library(shiny)

sliderInput("slider", label = h3("Slider"), min = 0,  max = 100, value = 50)
```

Ensuite, Shiny gère la plupart des bibliotèques d'affichage de données, ce qui permet d'ajouter très facilement des graphiques. Par exemple, pour afficher une distribution des scores des utilisateurs, on utilise la librairie *ggplot2*. Il suffit alors d'appeler la bonne fonction de *ggplot2* :

```
library(ggplot2)

x <- ranking$points
ggplot(data = ranking, aes(x=points)) +
  geom_histogram(mapping = aes(y = ..density..), color="black", fill = "white", bins=30) +
  geom_density(alpha=.2, fill="#FF6666") +
  xlim(-100, 100) +
  geom_vline(data=ranking, aes(xintercept=mean(points)),linetype="dashed", color="blue")
```



Enfin, si c'est nécessaire Shiny peut être étendu avec du contenu HTML en utilisant diverses fonctions. Par exemple :

```
withTags(div(h1("Title"), div("content")))
```

est équivalent à :

```
<div>
  <h1> Title </h1>
  <div> content </h1>
</h1>
```

Shiny est donc une technologie qui fonctionne avec r de manière simple et flexible.

Présentation du dashboard

Partie 1 : affichage de statistiques

Le but est que l'affichage des statistiques s'adapte au critères imposés par l'utilisateur dans la barre latérale (figure 3.4). Pour cette première étape du projet ces critères sont les suivants :

- La compétence : l'utilisateur doit pouvoir directement trouver les statistiques dans la compétence qui l'intérèsse.
- Le pays : l'utilisateur doit pouvoir restreindre l'affichage des statistiques à un pays.
- Le genre : l'utilisateur doit pouvoir directement afficher les statistiques pour un genre particulier.
- L'âge : l'utilisateur doit pouvoir sélectionner les statistiques propres à une plage d'âges.

Les différentes statistiques affichées (figures 3.5 et 3.6) sont les suivantes :

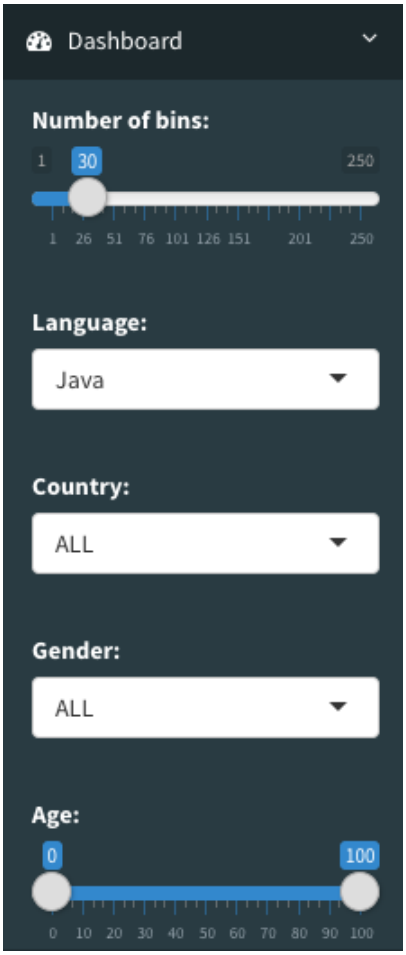


FIGURE 3.4 – Barre latérale de la partie 1

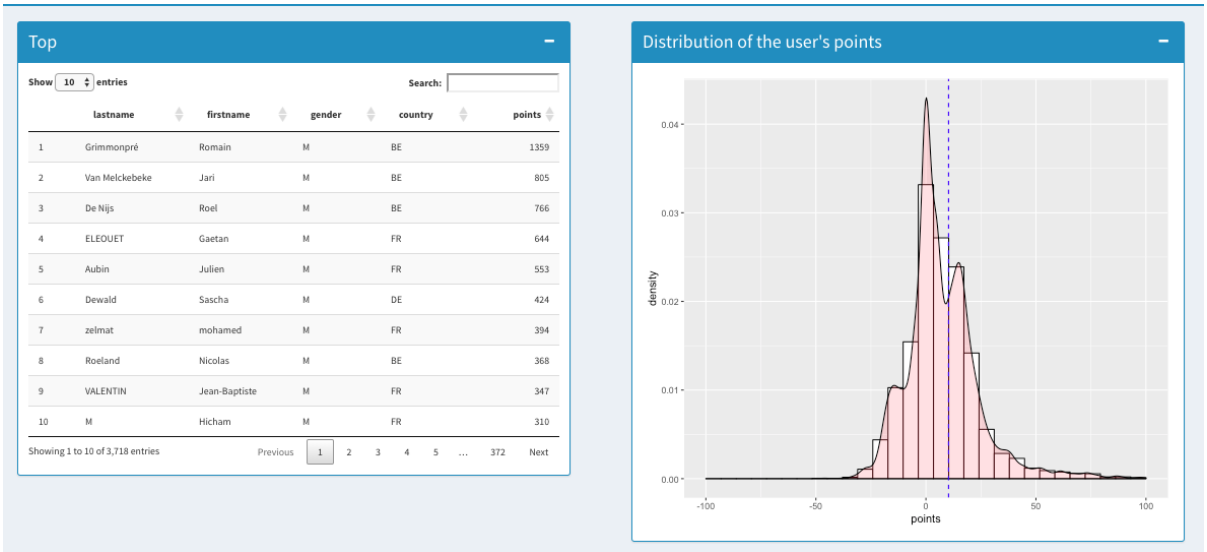


FIGURE 3.5 – Classement et distribution

— Un classement basé sur les résultats des réponses aux questions
— Une distribution des résultats des réponses aux questions avec affichage de la moyenne.
— Une répartition des utilisateurs par pays (qui est inutile quand un pays particulier est sélectionné)

Partie 2 : affichage des badges de compétence

Le système d'assignation de badges classe les utilisateurs sur 10 niveaux selon une compétence, le niveau 1 étant le plus haut niveau. L'utilisateur peut donc sélectionner 2 paramètres (figure 3.7) :

— La compétence
— Le niveau

La page affiche donc les différents utilisateurs appartenant au groupe du niveau et de la comptétence sélectionnés (figure 3.8).

Système d'assignation des badges de compétence

Objectifs et présentation du problème

L'objectif de cette partie du projet est de développer un système intelligent qui classifie les utilisateurs en différents niveaux selon une compétence bien précise. De cette manière, l'outil permet aux recruteurs de trouver tous les utilisateurs qui sont dans le groupe de tel niveau pour telle compétence.

Ce système est basé sur du *machine learning* est plus particulièrement l'algorithme *k-means*. En *machine learning*, les algorithmes sont en fait des modèles mathématiques génériques (le modèle fait corresspondre une ou des sorties à une ou des entrées) qui sont entrainés grâce à des données existantes afin qu'ils disposent ensuite d'un pouvoir de prédiction sur des nouvelles données. Les problèmes en *machine learning* sont divisés principalement selon 2 critères :

— Les sorties du modèle mathématique sont-elles continues ? : on parle d'un problème de régression quand les sorties sont continues et d'un problème de classification quand elles sont discontinues (dans ce cas, les sorties sont des labels).

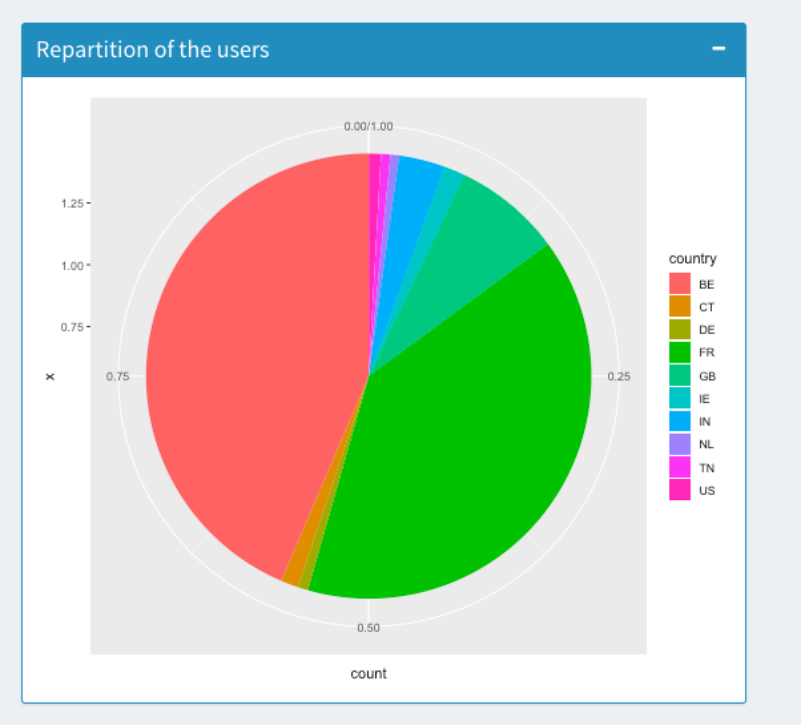


FIGURE 3.6 – Répartition par pays

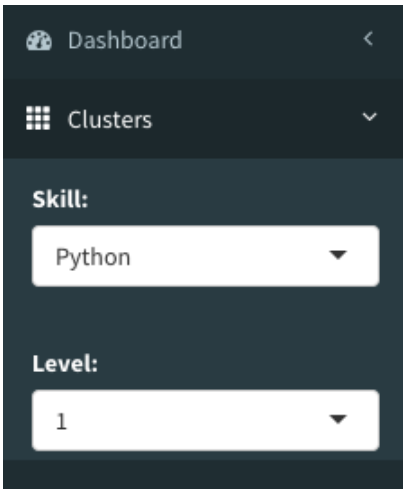


FIGURE 3.7 – Barre latérale de la partie 2

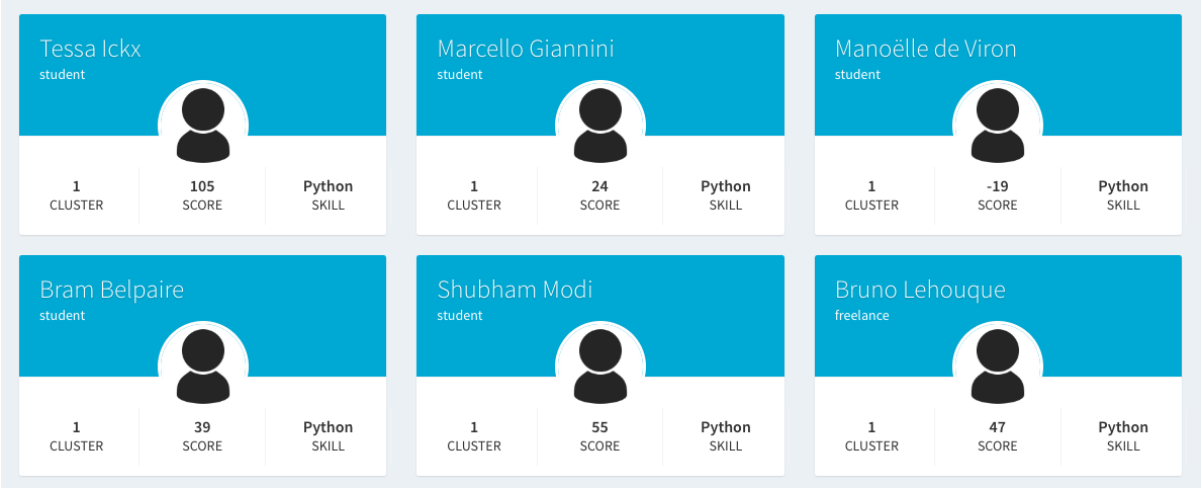


FIGURE 3.8 – Affichage du groupe sélectionné

— Dispose-t-on des sorties dans le set d'entraînement ? : on parle de problème supervisé quand on possède les sorties et de problème non-supervisé dans le cas contraire.

Dans notre cas la sortie de l'algorithme sera un chiffre de 1 à 1O qui représente un niveau de compétence et on ne dispose pas des niveaux de compétence des utilisateurs avant l'application de l'agorithme. Il s'agit donc d'un problème de classification non-supervisé. L'algorithme doit donc créer différents groupes et leur associer un label sur base des données existantes à propos des utilisateurs de la plateforme. Dans le langage technique, ces groupes sont appelés des clusters. Etant donné que les clusters sont basés sur des données exclusivement tirées de la plateforme EDITx, il faut faire attention à l'interprétation qu'on donne à la classification. Il faut plus voir le "cluster 1 en Java" comme celui regroupant les meilleurs utilisateurs de la plateforme en Java que comme celui regroupant les utilisateurs possédant un niveau expert en Java.

Architecture

Dans le but de garantir une certaine qualité logicielle, il faut définir une architecture pour le système d'assignation de badges. Il faut donc définir les différentes parties du système :

— Un composant qui récupère les données
— L'algorithme de classification
— Un composant qui donne les niveaux aux clusters
— Un composant qui enregistre dans la base de données l'appartenance d'un utilisateur à un cluster
— Un affichage graphique des clusters (il s'agit de la partie 2 du dashboard)

Le but principal de cette architecture est d'assurer au maximum une indépendance de l'algorithme de classification utilisé, avec le reste du système. On utilise l'algorithme de classification *k-means* mais il en existe beaucoup d'autres et c'est ici que l'utilité de l'indépendance de l'algorithme saute aux yeux : celle-ci permet au système de fonctionner avec n'importe quel algorithme de classification en ne devant modifier aucun autre de ses composants que l'algorithme de classification lui-même. La figure 3.9 reprend l'architecture imaginée.

Le schéma est un diagramme de classe car le système est construit en orienté objet. La programmation orientée objet sous R se réalise grâce à *S4*. *S4* est la quatrième version du langage de programmation S sur lequel est basé *R*. Par rapport à ces versions précédantes *S4* ajoute des fonctionnalités qui permettent à *S* d'être considéré comme un langage orienté objet, et donc à *R* aussi. Concrètement, *S4* désigne par abus de langage la programmation orientée objet avec *R*.

Le diagramme est composé de 4 classes :

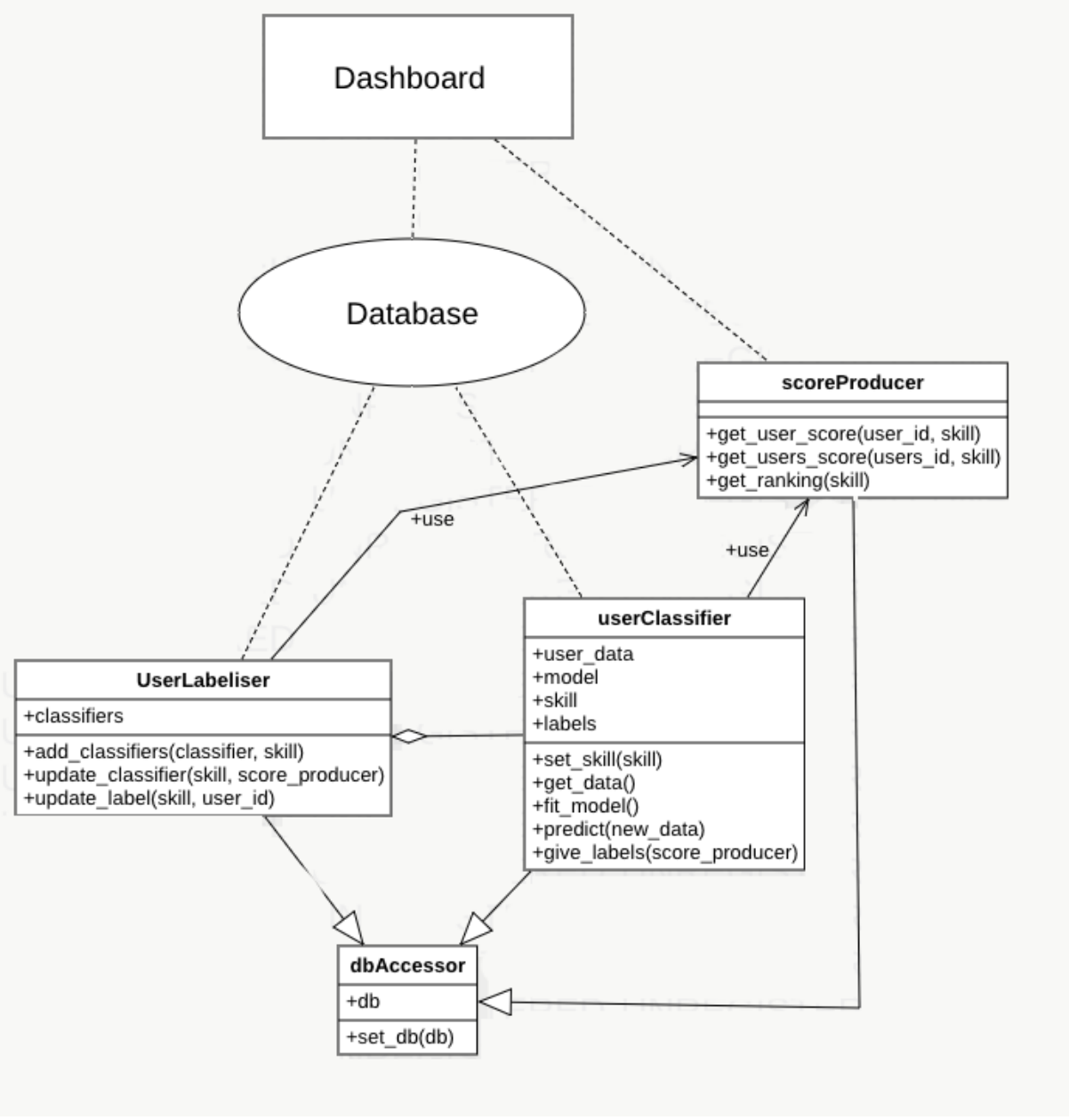


FIGURE 3.9 – Architecture du système d'assignation de badges

— *UserClassifier* : c'est cette classe qui va classifier les utilisateurs. La méthode *set_skill* permet d'assigner à l'objet la compétence sur laquelle il doit classifier les utilisateurs. La méthode *get_data* récupère les données nécessaires pour la classification en base de données. La méthode *fit_model* contient l'algorithme de classification. La méthode *predict* détermine le cluster d'un nouvel utilisateur. La méthode *give_labels* permet de donner des niveaux aux clusters sur base des résultats des utilisateurs qu'ils contiennent.
— *UserLabeliser* : c'est la classe qui écrit dans la base de données graphe l'appartenance des utilisateurs à leurs clusters. L'instance de *UserLabeliser* contient plusieurs instances de *UserClassifier* (normalement une instance de *UserClassifier* par compétence), la relation d'agrégation entre ces deux classes se justifie par le fait qu'une instance de *UserClassifier* continue à exister sans l'instance de la classe *UserLabeliser* qui la possède. La méthode *add_classifier* permet d'ajouter une instance de *UserClassifier*. La méthode *update_classifier*, recalcule les clusters d'un classifieur particulier et met à jour la base de données en suivant ces nouveaux clusters. La méthode *update_label* permet de mettre à jour le cluster d'un utilisateur particulier relatif à une compétence particulière. Cette méthode est utile pour que les utilisateurs puissent évoluer ou régresser dans leurs niveaux de compétences.
— *dbAccessor* : cette classe offre simplement un accès à la base de données. Les 3 précédantes classes l'étende car elles doivent toutes accéder à la base de données.
— *ScoreProducer* : cette classe offre des méthodes utiles aux autres classes, par exemple elle peut récupérer le score d'un utilisateur pour une certaine compétence. Elle peut être vue comme une classe statique même si ce concept n'existe pas dans *S4*.

Le reste du schéma montre les entités qui accèdent à la base de données et la fait que le dashboard utilie la classe *ScoreProducer* pour calculer ses classements.

Algorithme des k-means

L'algorithme des *k-means* est un des algorithmes les plus basiques de classification supervisée. Dans notre cas il divise n utilisateurs en k clusters.

Les données

Pour qu'il puisse former les clusters l'algorithme a besoin qu'on lui fournisse des données. Celle-ci sont organisées sous la forme d'un tableau semblable à celui ci-dessous.

```
##   q_1 q_2 q_3 q_4 q_5 q_6 q_7 q_8 q_9 q_10 q_11 q_12 q_13 q_14 q_15
## 1   0   1   0   1   0   0   1   0   1    0    0    0    1    0    1
## 2   1   1   1   1   1   1   0   1   1    1    1    0    0    1    1
## 3   1   1   1   1   1   1   0   1   1    1    1    0    0    1    1
## 4   1   0   1   1   0   1   0   1   1    1    1    0    0    1    1
## 5   0   0   0   1   1   1   0   0   1    1    1    1    0    1    0
## 6   1   0   1   1   0   0   0   1   1    1    1    0    0    0    1
## 7   1   1   0   1   1   0   1   1   1    0    1    0    0    0    1
## 8   0   1   0   1   1   0   1   0   1    1    1    0    0    0    1
```

Chaque colonnes correspond à une question qui concerne la compétence pour laquelle on construit le classifieur. Chaque ligne correspond à un utilisateur. Un *1* signifie que l'utilisateur a bien répondu à la question et un *0* signifie le contraire. Par exemple, l'utilisateur *2* a bien répondu à la question *q_2* et a mal répondu à la question *q_7*.

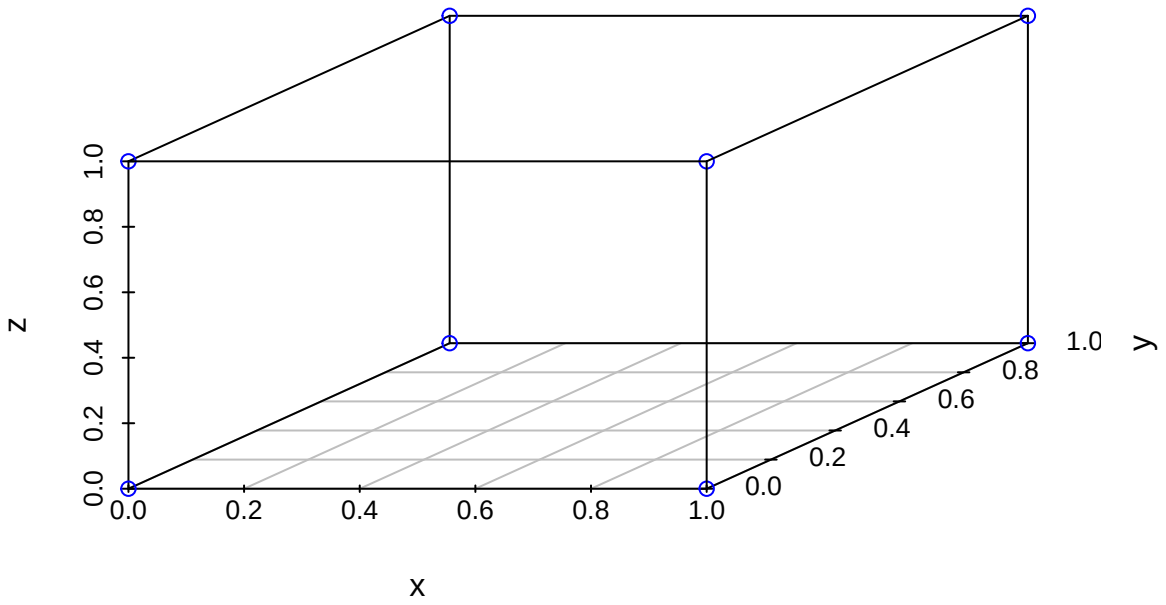
Fonctionnement de l'algorithme

Dans un premier temps l'algorithme va projeter les n points (utilisateurs) dans un espace à m dimensions où m est le nombre de questions dans le set de données qui est fourni à l'algorithme. Pour simplifier, imaginons

que le set contient 3 questions, les points sont donc projeter dans un espace à 3 dimensions. L'algorithme construit des sphères, selon des règles fixées, qui englobent ces points pour former les clusters.

```
## Warning in title(main, sub, ...): "fill" n'est pas un paramètre graphique

## Warning in plot.xy(xy.coords(x, y), type = type, ...): "fill" n'est pas un
## paramètre graphique
```



Le but de l'algorithme est de partager les n points $(x_1, x_2, ..., x_n)$ en k ensembles $S = \{S_1, S_2, ..., S_n\}$ en minimisant la somme des distances euclidiennes entre le barycentre du cluster et ses points. Cette distance totale peut s'exprimer comme ceci:

$$C(\mu_i) = \sum_{i=1}^{k} \sum_{x_j \in S_i} \|x_i - \mu_i\|^2$$

Le $\sum_{i=1}^{k}$ somme les clusters tandis que $\sum_{x_j \in S_i}$ somme les distances pour chaque point à l'intérieur d'un cluster. La distance euclidienne est élevée au carré pour que les termes négatifs pénalisent aussi la fonction.

En machine learning ce genre de fonction est appelée fonction de coût et le but de l'algorithme est de la minimiser. Il faut bien comprendre que la fonction de coût ne se trouve pas dans le même espace que celui dans lequels les utilsateurs sont projetés. C'est à dire que les réponses aux questions (les données d'entrées fournies à l'algorithme) sont des paramètres et non des variables de la fonction de coût. Ce sont les barycentres qui sont les variables de la fonction de côut. Le problème revient donc à trouver les k barycentres qui minimisent la fonction de coût.

Ce problème peut être résolu analytiquement (il s'agit d'une fonction convexe), en dérivant la fonction de coût et en trouvant la racine. Seulement, la méthode utilisée par les ordinateurs est une méthode itérative.

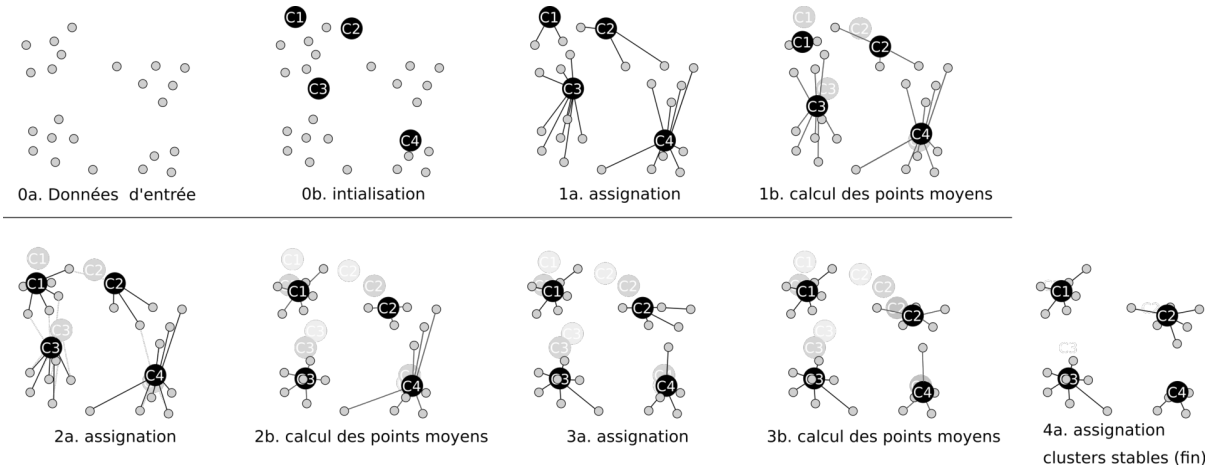


FIGURE 3.10 – Itération de l'algorithme k-means (Wikipédia)

A la première itération, l'agorithme choisit k centres au hasard, il assigne ensuite chaque point au centre le plus proche. A la fin de cette étape, on obtient k clusters S_i tel que :

$$S_i = \left\{ x_j : \left\| x_j - \mu_i^{(t)} \right\| \leq \left\| x_j - \mu_{i^*}^{(t)} \right\| \forall i^* = 1, ..., k \right\}$$

Enuite on recalcule les centres selon la formule :

$$m_i^{(t+1)} = \frac{1}{\left| S_i^{(t)} \right|} \sum_{x_j \in S_i^{(t)}} x_j$$

On passe ensuite à l'itération suivante.

L'algorithme se poursuit jusqu'à la convergence (quand les mêmes clusters apparaissent sur 2 itérations d'affilée) ou jusqu'à une condition d'arrêt (souvent un nombre d'itération maximum). La figure 3.10 illustre bien le fonctionnement des itérations.

Organisation d'un challenge à L'ECAM

Dés le début du stage, il a été convenu qu'il serait intérressant d'organiser un challenge à l'ECAM. D'une part, pour participer au processus d'organisation d'un challenge et d'autre part, parce que la récolte de données et une part importante du travail d'un projet de *machine learning*.

Pendant une après midi, nous sommes donc allés à l'ECAM avec Julien Carlier, Julien Kessels et Sébastien Combéfis faire passer à une trentaine d'étudiants un des challenges qui étaient en cours sur la plateforme EDITx : *IT Student of the Year 2018 by AG Insurance*.

L'expérience était intéressante par plusieurs aspects :

— L'ajout de 30 participants supplémentaires n'est pas négligeable pour EDITx.
— C'est intéressant de temps en temps de quitter son bureau et son ordinateur pour aller sur le terrain.

Chapitre 4

Conclusions

Objectifs

Des objectifs ont été fixés au début du stage, il faut vérifier à quel point ceux-ci ont été remplis.

Découverte des principaux modèles de *machine learning*

Cet objectif a été réalisé tout le long du stage. En effet, on a accordé un place importante à l'autoformation à travers des lectures de livre de référence qui ne se limitait pas seulement au machine learning. Voici les principaux livres qui ont été parcourus.

- un livre qui reprend les principaux modèles de machine learning du point de vue mathématique
- un livre qui aborde les bases de données graphe, ses avantages, son utilisation ...
- un livre de référence en statistique et en probabilité. Cette lecture est utile car les probabilités sont avec l'algèbre linéaire les maths derrière le *machine learning*

Les titres et auteurs de ses livres sont repris dans la bibliographie.

Apprentissage du langage de programmation R

Cet objectif n'est pas entièrement rempli. Afin de découvrir le langage, on a lu le livre *R for Data Science*, mais ensuite il n'y a pas spécialement eu le temps de pratiquer car les scripts de migration ont pris beaucoup de temps et ceux-ci sont réalisés en python.

Préparer un accès au données

Cet objectif est entièrement rempli. 2 scripts de migration ont été réalisés pour construire des bases de données contenant les informations nécessaires au projet. Une étape supplémentaire serait de vérifier la fiabilité de ces nouvelles bases de données via une série de tests.

Réalisation d'un dashboard

Cet objectif est rempli puisque d'une part, il propose un affichage ergonomique et interactif de statistiques et d'autre part, il offre au système d'assignation de badges une interface graphique. En outre, le développement de ce dashboard a permis un apprentissage de la technologie Shiny.

Réalisation d'un système d'assignation de badges

Cet objectif est en partie rempli car d'un coté, le système dispose d'une architecture relativement flexible au niveau du choix de l'algorithme de classification, mais d'un autre coté l'algorithme de classification n'est pas encore très performant, c'est à dire qu'il n'y a eu encore aucune personnalisation des métaparamètres de l'agorithme k-means (nombre de clusters, choix du type de distance ...) et aucune comparaison avec un autre algorithme de classification.

Pistes pour le TFE

Comme expliqué au début du rapport, le stage était préparatoire au TFE. Il est donc intéressant à la fin de celui-ci d'établir des pistes de travail pour le TFE.

— Travail sur le problème de prédiction pour les nouveaux utilisateurs : si un nouvel utilisateur arrive sur la plateforme et réalise un challenge ne contenant que des nouvelles questions qui n'ont pas servi lors de l'élaboration des clusters, celui-ci ne pourra évidemment pas être assigné au cluster adéquat, étant donné que la méthode de prédiction ne disposera pas des données nécessaires. Une piste pour résoudre ce problème est de créer des catégories de questions et classifier les utilisateurs selon leur maitrise ou non-maitrise de ces différentes catégories.
— Réaliser des tests de fiabilité des nouvelles bases de données : Pour vérifier que les bases de données construites grâce aux scripts de migration sont valides, ils seraient intéressant de mettre en place une batterie de tests. Un test de base serait de réaliser des requêtes équivalentes dans les différentes bases de données et vérifier qu'on obtient les mêmes résultats.
— Trouver le meilleur algorithme de classification : il existe beaucoup d'autres algorithmes de classification (Random Forest, SVM ...). Il serait intéressant de les comparer sur base de leur performance, vitesse d'exécution ...

Bibliographie

CRAN. 2018. « cran project ». 2018. https://cran.r-project.org/.

Datastorm. 2017. « visNetwork, an R package for interactive network visualization ». 2017. https://datastorm-open.github.io/visNetwork/.

Genolini, Christophe. 2013. *Petit Manuel de S4 : Programmation Orient´e Objet sous R*.

Ian Robinson, Jim Webber & Emil Eifrem. 2015. *Graph Databases*. OReilly.

MacQueen, J. 1967. « Some methods for classification and analysis of multivariate observations ».

Michel Lutz, Eric Biernat &. 2016. *Data Science : fondamentaux et études de cas*. Eyrolles.

Ronald E. Walpole, Sharon L. Myers & Keying Ye, Raymond H. Myers. 2012. *Probability & Statistics for Engineers & Scientists*. Pearson.

Studio, R. 2017. « Shiny ». 2017. https://shiny.rstudio.com/.

Wickham, Hadley, et Garrett Grolemund. 2016. *R for Data Science*. OReilly.

Wikipédia. 2017. « k-means clustering ». 2017. https://fr.wikipedia.org/wiki/K-means_clustering.

———. 2018. « NoSQL ». 2018. https://fr.wikipedia.org/wiki/NoSQL.